es:saar

Testing Embedded Systems

Qualitätssicherung in der Entwicklung eingebetteter Systeme

Aufwand identifizieren und reduzieren durch bessere **Prognosen**, **Risikominimierung** und bedarfsgerechte **Kapazitätsplanung**

Unser Ziel ist es, den Aufwand in der Entwicklung eingebetteter Systeme zu reduzieren.

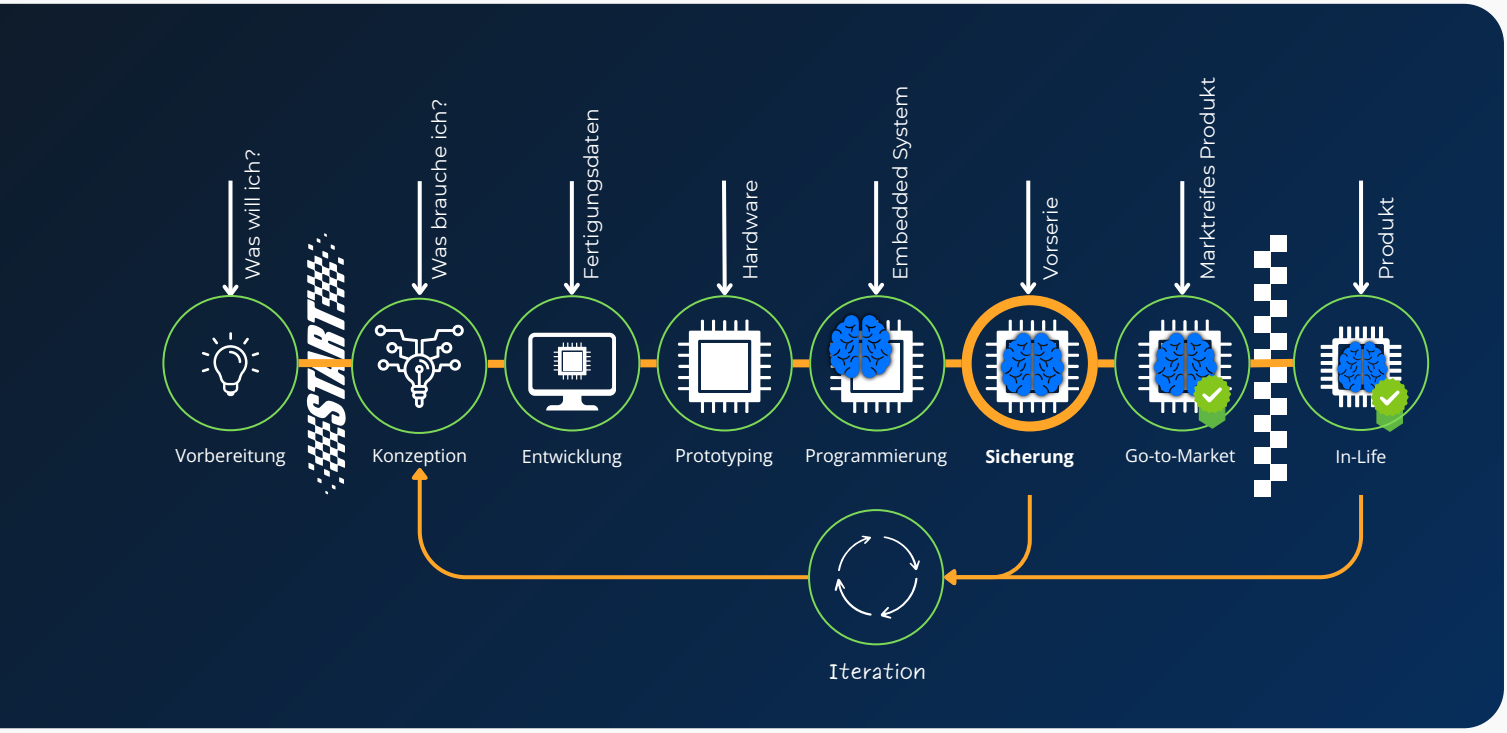
Whatsapp Business

kontakt@essaar.de

+49 (0) 176 814 561 07

essaar.de/kontakt

Einleitung



Qualitätssicherung der Entwicklung

Aspekte des Aufwands identifizieren, untersuchen und optimieren

Qualitätssicherung ist in der Elektronikentwicklung unerlässlich. Gleichzeitig ist sie aber oft mit einem hohen und schwer vorhersehbaren Aufwand verbunden. Die in diesem Artikel beschriebenen Erfahrungen und Erkenntnisse basieren auf mehrjährigen Beobachtungen und haben uns geholfen, den Aufwand für Funktions- und Performancetests* zu **identifizieren**, zu **analysieren** und schließlich zu **reduzieren**. Dadurch konnten wir den Aufwand der Kosten und Zeit in unserer Entwicklung optimieren, eine verbesserte Time-to-Market erreichen und eine höhere Produktqualität gewährleisten.

Um diese Erkenntnisse zu teilen, geben wir zunächst eine grundlegende und verallgemeinerte Darstellung des Qualitätssicherungsprozesses in der Elektronik-entwicklung. Die Aspekte der **Prognostizierbarkeit**, **Risikominimierung** und der **bedarfsgerechten Kapazitätsplanung** werden anschließend einzeln untersucht. Mit einer Anregung zur qualitativen Analyse des Bedarfs und der Kapazität der eigenen Qualitätssicherung kann abgeschätzt werden, ob und inwieweit der Aufwand optimiert werden kann. Im Kern geht es dabei um die Frage, wie gut die Prozessanforderungen durch die vorhandenen Kapazitäten gedeckt sind. Abschließend wird anhand eines Business Case gezeigt, welche konkreten Optimierungen erreicht werden können. Im Zusammenhang des Dokuments wird die Software es:scope® vorgestellt. Im Appendix ist ein Überblick über die Literatur zum Testen von Embedded Systems zusammengefasst.

* Compliance Tests sind ein eigenes Thema

Aus der Vogelperspektive

Qualitätssicherung der Entwicklung

Versteckten Aufwand identifizieren, untersuchen und reduzieren

In der Entwicklung werden Anforderungen definiert, umgesetzt und geprüft - das V-Modell des VDE ist der zugrundeliegende Standard. In einer idealen Welt würde die Implementierung alle definierten Anforderungen direkt erfüllen, so dass eine Prüfung überflüssig wäre. Doch selbst bei präzise formulierten Anforderungen und sorgfältig durchgeführten Berechnungen und Simulationen zeigt die Qualitätssicherung oft Abweichungen vom gewünschten Verhalten. Das liegt daran, dass Berechnungen und Simulationen immer nur idealisierte Annahmen über die Realität treffen. Faktoren wie Materialunterschiede, Umwelteinflüsse oder unvorhergesehene Wechselwirkungen im System führen dazu, dass reale Tests notwendig werden. Warum ist das so? Weil die Komplexität realer Systeme die Genauigkeit von Modellen übersteigt. Messungen sind daher unerlässlich - um Modelle an die Realität anzupassen, um die Funktionalität der entwickelten Systeme zu überprüfen und um Abweichungen vom gewünschten Verhalten zu korrigieren.

Während der Qualitätssicherung kann es vorkommen, dass trotz Anpassungen die gewünschten Ergebnisse nicht erreicht werden. In solchen Fällen ist möglicherweise eine Design-Iteration notwendig, um die Mängel zu beheben. Gleichzeitig kann es vorkommen, dass bestimmte Funktionen im Validierungsprozess als überflüssig erkannt werden. Diese Funktionen können in einer leicht umsetzbaren Iteration vereinfacht oder sogar ganz weggelassen werden, um das System kosteneffizienter zu gestalten.

Begrifflichkeiten

Begriffe wie „testen“ und „anpassen“ sind oft zu ungenau, um die spezifischen Prozesse im Entwicklungsablauf präzise zu beschreiben. Damit Missverständnisse vermieden werden können, sollen die wesentlichen Konzepte erläutert werden, die in der Entwicklung eine zentrale Rolle spielen: Verifizierung, Validierung, Kalibrierung und Tuning.

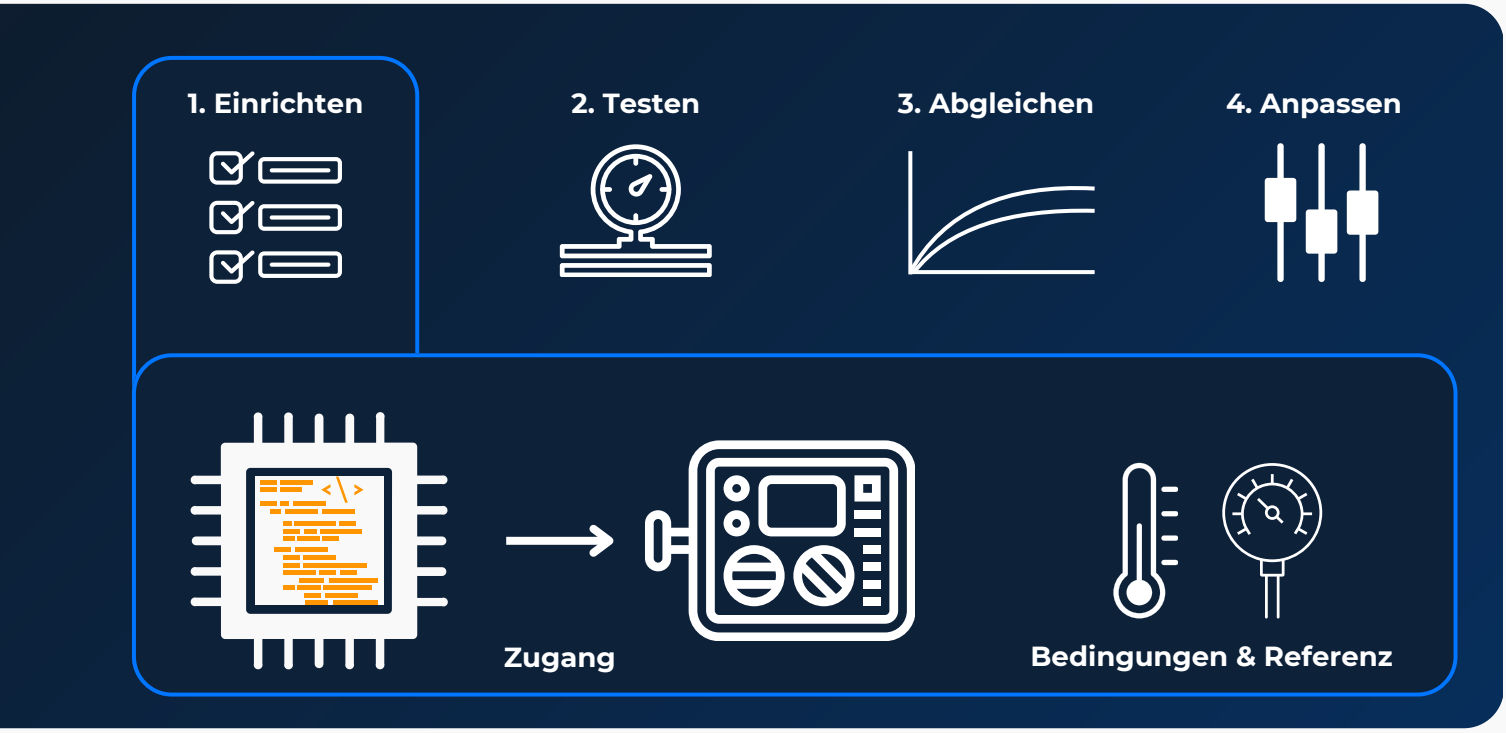
Verifizierung:
Die Verifizierung bezieht sich auf das quantitative Überprüfen, ob festgelegte Anforderungen erfüllt werden. Es handelt sich um eine objektive Messung, bei der eine spezifizierte Genauigkeit eingehalten werden muss – beispielsweise darf eine Messabweichung nicht mehr als 5 % betragen. Das Ziel der Verifizierung ist es, sicherzustellen, dass das entwickelte System innerhalb der definierten Toleranzgrenzen funktioniert.

Validierung:
Die Validierung geht über die Verifizierung hinaus und beschäftigt sich damit, ob das richtige Produkt entwickelt wurde, um die angestrebte Lösung bereitzustellen. Hier steht die Frage im Mittelpunkt, ob die entwickelten Funktionen den tatsächlichen Bedarf erfüllen – etwa ob wirklich fünf USB-Anschlüsse an einem Laptop nötig sind oder ob diese Funktion überdimensioniert ist. Die Validierung stellt sicher, dass die erstellten Eigenschaften in der realen Anwendung sinnvoll und notwendig sind.

Kalibrierung:
Kalibrierung bedeutet die Anpassung von Systemparametern, um systematische Abweichungen zu korrigieren. Ein typisches Beispiel ist die Vorverarbeitung eines Sensorsignals, um einen Offset auszugleichen und die Messgenauigkeit zu verbessern. Diese Anpassungen erfolgen in der Regel softwaregestützt durch Neueinstellung von Kalibrierwerten oder -parametern.

Tuning:
Tuning geht über die Kalibrierung hinaus und bezieht sich auf die Feinabstimmung von Parametern, um die Systemleistung zu optimieren. Ein typisches Beispiel ist das sanfte Anlaufverhalten eines Motors, das durch präzise Anpassung der Steuerungsparameter erreicht wird. Tuning setzt eine erfolgreiche Kalibrierung voraus und hat zum Ziel, die Leistung des Systems unter den gegebenen Bedingungen zu optimieren.

Der Ablauf der Qualitätssicherung 1/4



1. Einrichten

Messzugang, Testbedingungen, Referenzmessung

Um Tests durchzuführen, müssen sowohl das physische System als auch ein Messsystem verfügbar und korrekt eingerichtet sein. Dabei ergeben sich mehrere Herausforderungen:

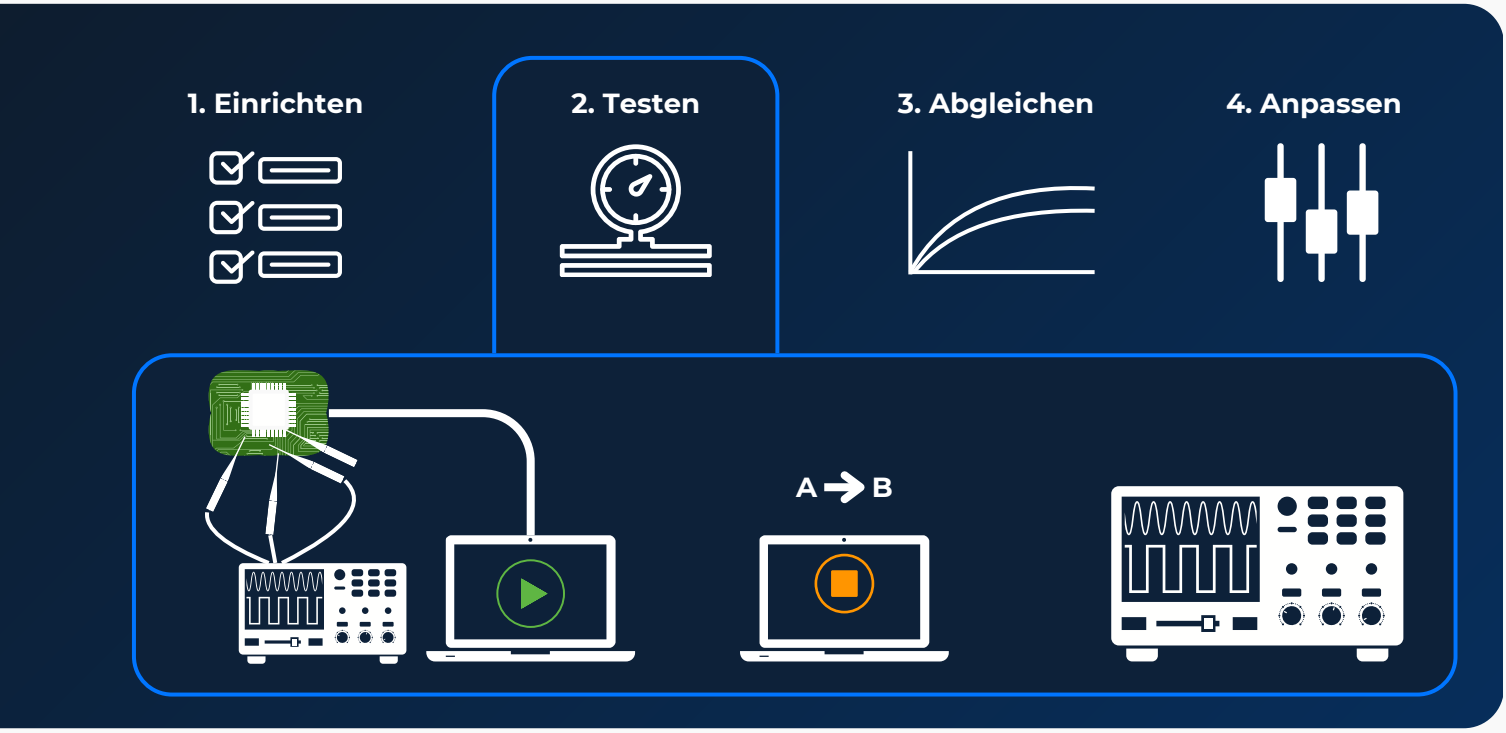
- **Zugang**: Die zu messenden Größen müssen für das Messsystem zugänglich gemacht werden. Interne Signale eines Mikrocontrollers müssen beispielsweise nach außen geführt werden, was bei begrenzter Rechenleistung und einer limitierten Anzahl an Ausgängen Schwierigkeiten verursachen kann.

- **Testbedingungen**: Besondere Bedingungen wie extreme Temperaturen (z. B. Tests in einem Ofen) können die Durchführung von Tests weiter verkomplizieren.

- **Referenzmessung**: Eine zuverlässige Referenzmessung ist notwendig, um die Testergebnisse mit einem Wert vergleichen zu können.

- **Systemressourcen**: Begrenzter Speicherplatz und eingeschränkte Kommunikationskanäle können die Datenerfassung erschweren. Wenn die Daten über Kommunikationsprotokolle wie CAN oder UART gesendet werden, müssen oft spezielle Lösungen für die Datenübertragung entwickelt werden.

Der Ablauf der Qualitätssicherung 2/4



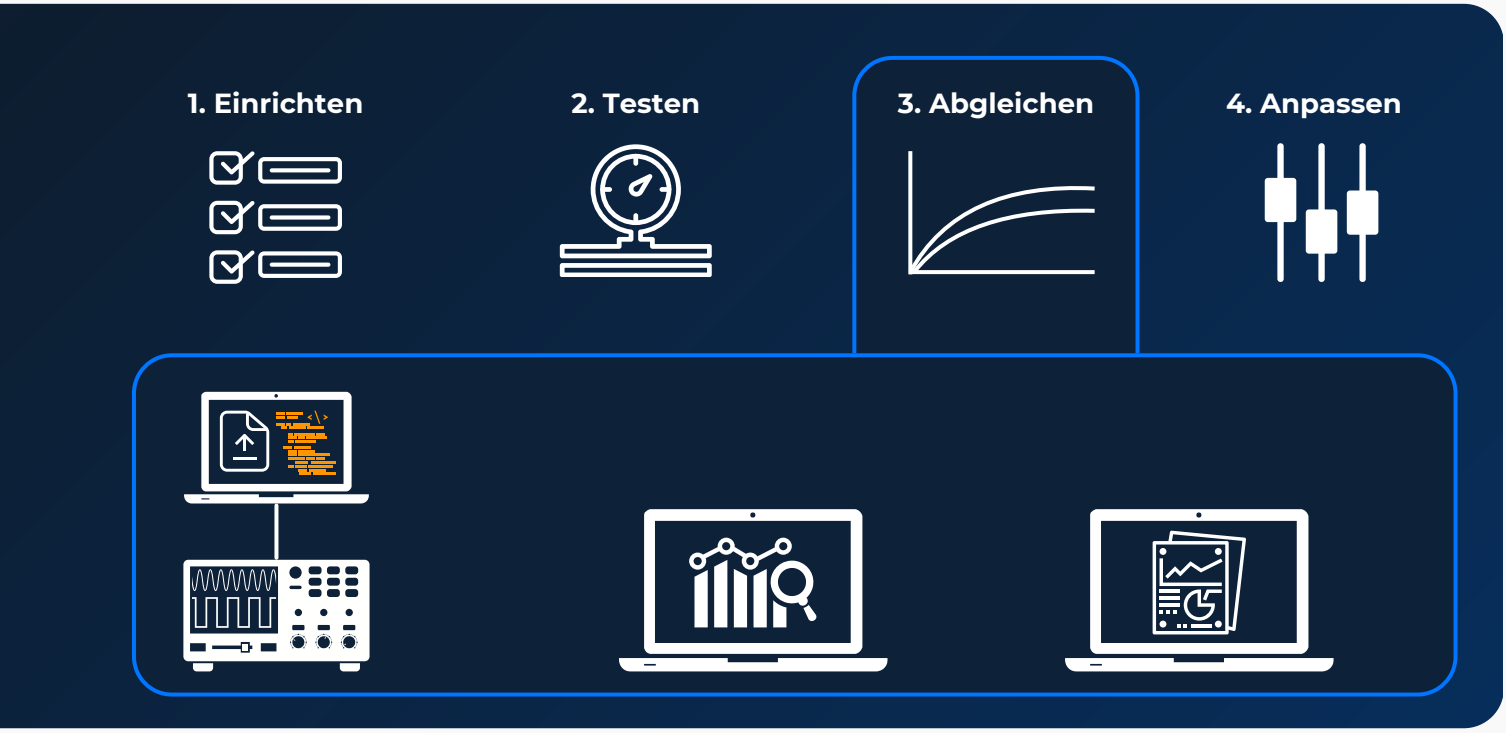
2. Testen

Betriebsstart, Zustandswechsel, Überwachung

Nach der Einrichtung beginnt der Test, der je nach Komplexität und Anforderungen wenige Minuten bis hin zu mehreren Tagen dauern kann. Während dieses Schrittes müssen verschiedene Herausforderungen gemeistert werden:

- **Betriebsstart**: Das System wird unter den definierten Testbedingungen gestartet und die zugänglichen Signale mit dem Testsystem überwacht oder aufgezeichnet.
  - **Datenaufzeichnung**: Die relevanten Messgrößen werden erfasst.
  - **Synchronisierung:** Unsynchronisierte Daten führen zu Fehlinterpretationen.
- **Unterbrechungen für Zustandswechsel**: Tests können durch Störungen oder bewusste Änderungen im Betriebszustand unterbrochen werden, was eine flexible Überwachung erfordert.
- **Datenrate**: Hohe Datenraten können die Übertragung und Speicherung erschweren, insbesondere bei eingeschränkten Systemressourcen.
- **Datenqualität**: Die Verwertbarkeit der Messergebnisse hängt von der Präzision des Messsystems und der Synchronität der Daten ab.
- **Testüberwachung**: Fehlkonfigurationen werden oft erst nachträglich erkannt, da eine Echtzeitüberwachung nicht immer möglich ist.

Der Ablauf der Qualitätssicherung 3/4



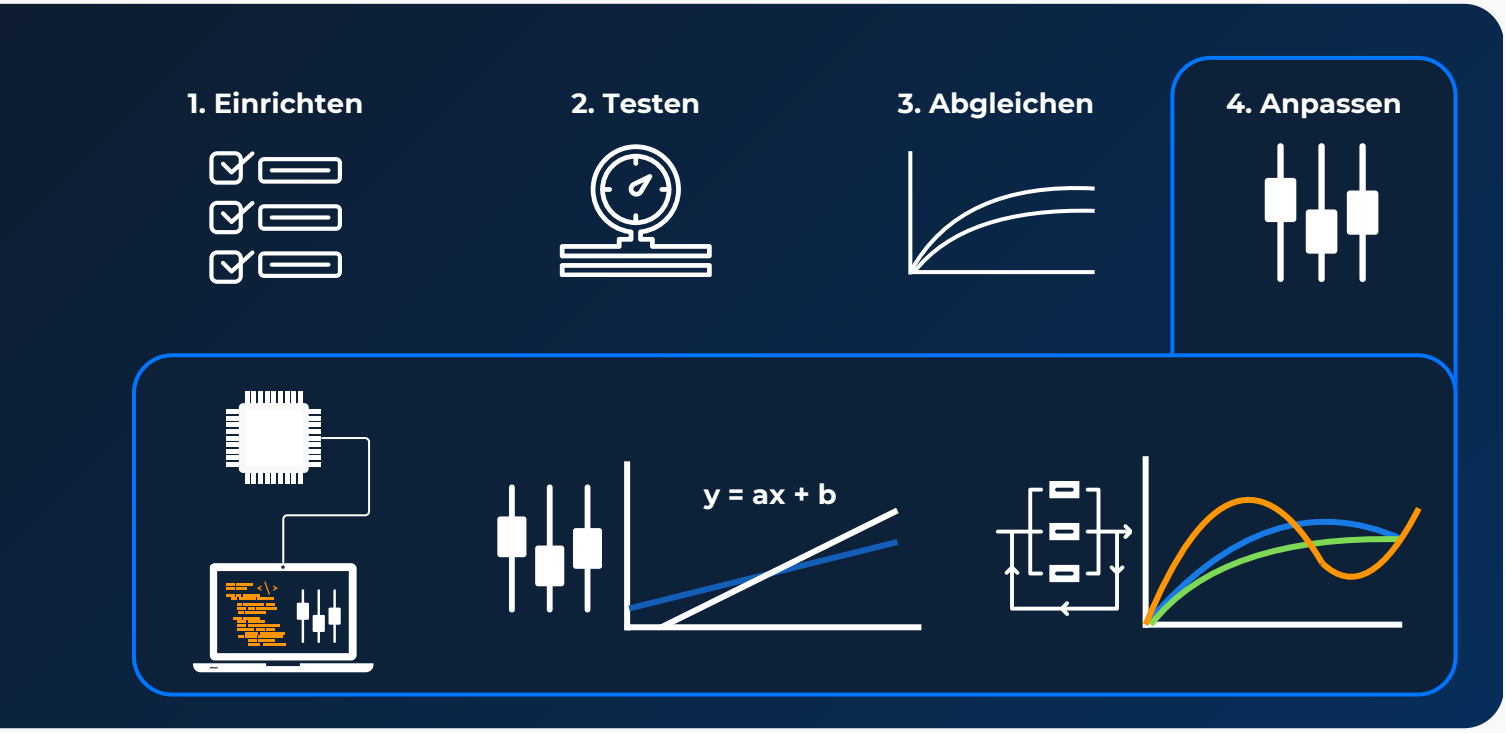
3. Abgleichen:

Exportieren, Analysieren, Auswerten

Nach dem Test werden die erfassten Daten ausgewertet. Dieser Schritt umfasst:

- **Datenexport und Analyse:** Die Testdaten werden exportiert und in einer Analyse-Software (z. B. MATLAB) verarbeitet.
- **Verifizierung**: Das Systemverhalten wird mit den erwarteten Werten und den Referenzmessungen verglichen, um sicherzustellen, dass die Anforderungen erfüllt sind.
- **Validierung**: Die Messdaten werden genutzt, um zu bewerten, ob das entwickelte System den gestellten Anforderungen entspricht und die gewünschten Funktionen erfüllt.
- **Fehleranalyse**: Bei Abweichungen wird untersucht, ob diese durch Messfehler, Modellungenauigkeiten oder systembedingte Probleme verursacht wurden.
- **Abgleich und Auswertung**: Abweichungen werden durch den Vergleich mit bestehenden Modellen und Referenzdaten identifiziert, was zu Erkenntnissen über die benötigten Anpassungen der Systemparameter führt.

Der Ablauf der Qualitätssicherung 4/4



4. Anpassen:

Anpassen, Kalibrieren, Tunen

Basierend auf den Erkenntnissen aus der Analyse werden die Systemparameter kalibriert und getunt, um Abweichungen zu korrigieren und das System zu optimieren. Dieser Schritt umfasst:

- **Softwareanpassungen und Hardwaremodifikationen:** Je nach Art der Abweichungen können sowohl Softwareanpassungen (z. B. Neueinstellung von Parametern) als auch Hardwaremodifikationen (z. B. Austausch von Bauteilen) notwendig sein.

- **Kalibrieren**: Die Systemparameter werden neu eingestellt, um systematische Abweichungen zu korrigieren und eine präzise Funktionsweise sicherzustellen.

- **Tunen**: Nach der Kalibrierung erfolgt eine Feinabstimmung der Parameter, um die Leistung des Systems unter realen Betriebsbedingungen zu maximieren.

Iterativer Zyklus: Der gesamte Ablauf wird wiederholt, bis das System in der Validierung alle Anforderungen erfüllt. Die Anzahl der Iterationsschleifen und Unterbrechungen kann je nach Komplexität des Systems stark variieren.

Prognostizierbarkeit der Qualitätssicherung

Unsere Erfahrung in der Entwicklung eingebetteter Systeme hat gezeigt, dass die Berücksichtigung aller Faktoren, die die Qualitätssicherung beeinflussen, entscheidend für die Optimierung der Projektplanung ist. Werden bei der Projektplanung nicht alle Kosten- und Zeitaspekte berücksichtigt, wird der tatsächliche Aufwand fast immer unterschätzt.

Fehlende Transparenz im Projektplan: Es ist wichtig, dass die Qualitätssicherung als **eigenständiger und detaillierter Posten im Projektplan** aufgeführt wird. Häufig wird der Testprozess pauschal als Teil der allgemeinen Entwicklungsaufgaben betrachtet, was dazu führt, dass der tatsächliche Aufwand unterschätzt wird. Eine einfache Lösung hierfür ist, separate Zeitpläne und Budgets für spezifische Qualitätssicherungsaktivitäten (z.B. Kalibrierung, Prototypentests, Softwareverifikation) zu erstellen. Auf diese Weise kann sichergestellt werden, dass jeder Testprozess realistisch berücksichtigt wird und das Risiko von Budgetüberschreitungen verringert wird.

Berücksichtigung der iterativen Prozessnatur: Eine wichtige Erkenntnis bei der Planung der Qualitätssicherung ist, dass sie aufgrund ihres iterativen Charakters auch als eigenständiger Punkt im Projektplan schwer vorhersehbar bleibt. Pufferzeiten und flexible Testzyklen haben sich bewährt, um auf unvorhergesehene Probleme reagieren zu können. Prozessunterbrechungen und wechselnde Verantwortlichkeiten können die geplante Projektlaufzeit verzögern. Auch die tatsächliche Anzahl der Wiederholungen von Anpassungsschleifen ist schwer vorhersehbar. Im folgenden Business Case wird gezeigt, wie wir mit es:scope® die **Anzahl der Unterbrechungen und Iterationen reduziert** haben.

Bedarfsgerechte Daten-Qualität: Die Zuverlässigkeit der getroffenen Entscheidungen hängt stark von der Qualität der verfügbaren Daten ab. Wenn Messungen ungenau, asynchron oder unvollständig sind oder nicht alle relevanten Informationen erfasst werden, können wichtige Abweichungen übersehen oder fehlinterpretiert werden. Solche Probleme können wiederum zu Qualitätsmängeln im Endprodukt führen. Eine **bedarfsgerechte Investition in hochwertige Mess- und Prüfsysteme** zahlt sich hier langfristig aus.

Risikoabwägung der Strafkosten bei unzureichender Qualität: Mängel in der Qualitätssicherung, die zu Fehlern in der Serienproduktion führen, bergen erhebliche finanzielle und Reputationsrisiken. Anders als bei Software, wo ein Update möglich ist, sind Nachbesserungen bei physischen Produkten wesentlich aufwändiger. Langfristig überwiegen die Vorteile einer gründlichen Qualitätssicherung die kurzfristigen Einsparungen. Es gilt, klare Prioritäten zu setzen und eine informierte **Balance zwischen Geschwindigkeit und Qualität** zu finden, um spätere Probleme zu vermeiden.

Risikominimierung durch TDD

In der Entwicklung von Anwendungssoftware ist Test-Driven Development (TDD) eine bewährte Methode, bei der Tests vor dem eigentlichen Code entstehen. Mit der Testabdeckung kann dann sichergestellt werden, dass keine Probleme nach Veröffentlichung des Codes auftauchen. In der Entwicklung eingebetteter Systeme ist TDD zwar prinzipiell möglich, aber deutlich komplexer. Die Gründe dafür sind:

Hardwareabhängigkeit: Die Entwicklung eingebetteter Systeme ist eng mit der zugrunde liegenden Hardware verbunden. Während Emulatoren wie QEMU es ermöglichen, Hardwareumgebungen zu simulieren, sind bestimmte Hardwarespezifika wie elektrische Signale oder physikalische Wechselwirkungen nur schwer oder gar nicht emulierbar. Kritische Aspekte wie die Genauigkeit der Signalübertragung und deren Timing spielen in Echtzeitsystemen eine wichtige Rolle und erfordern Tests auf realer Hardware. Diese Abhängigkeit von physischer Hardware schränkt die Anwendbarkeit von TDD ein, insbesondere wenn es darum geht, das Zusammenspiel von Software und Hardware in unterschiedlichen Umgebungen zu testen.

Individuelle Entwicklungsprozesse und -umgebungen: Eingebettete Systeme sind oft sehr spezifisch und vielfältig, was die Standardisierung von TDD erschwert. Im Gegensatz zur Anwendungssoftware, wo es etablierte Toolchains gibt, sind die Werkzeuge für eingebettete Systeme oft weniger ausgereift oder schwer zugänglich. Unterschiedliche Hardware-Plattformen, kundenspezifische Anforderungen und unterschiedliche Toolchains erschweren die Implementierung eines standardisierten TDD-Prozesses. Die Vielfalt der Entwicklungsumgebungen führt zu einem erhöhten Anpassungsaufwand, um TDD in den Prozess zu integrieren.

Ressourcenbeschränkungen: Embedded-Systeme sind oft durch begrenzten Speicher, Rechenleistung und I/O-Möglichkeiten eingeschränkt. Dies stellt eine Herausforderung dar, da viele TDD-Tests auf umfangreiche Ressourcen angewiesen sind, um die Code-Abdeckung und Funktionalität zu validieren.

Echtzeit-Anforderungen: Viele eingebettete Systeme arbeiten in Echtzeit, d.h. sie müssen strenge Zeitanforderungen erfüllen. Das Testen von Echtzeitanforderungen stellt eine besondere Herausforderung dar, da Testumgebungen dieses Verhalten oft nicht mit der erforderlichen Genauigkeit nachbilden können. Selbst minimale Verzögerungen in simulierten Umgebungen können zu Ergebnissen führen, die stark von der Realität abweichen und damit die Validität der Tests beeinträchtigen.

Black, Grey and White box testing

Die Testmethoden Black Box-, Grey Box- und White Box Tests unterscheiden sich stark darin, wie tief die Tester in die internen Details der Software eindringen können. Bei eingebetteten Systemen kommen zusätzliche Herausforderungen durch die enge Verzahnung von Hard- und Software hinzu - es gibt Software, die von der Hardware abhängig ist.

Black Box Testing konzentriert sich auf die externen Schnittstellen und die sichtbare Funktionalität des Systems. Der Tester hat keinen Einblick in die internen Abläufe oder den Quellcode. Dies ist besonders nützlich für dynamische Tests, bei denen die Reaktion des Systems auf verschiedene Eingaben während der Laufzeit getestet wird (u.a. “Fuzzing”). Beispielsweise kann man das Verhalten eines Sensors testen, indem man ihn verschiedenen Umgebungsbedingungen aussetzt und die Reaktion des Systems misst. Der Vorteil besteht darin, dass der Tester das System wie ein Endbenutzer testet, aber der Einblick in interne Fehler oder zeitkritische Probleme ist begrenzt.

White Box Testing hingegen bietet einen detaillierten Einblick in den Quellcode und die Hardware-Implementierung und ermöglicht sowohl statische als auch dynamische Tests. Statische Tests untersuchen den Code, ohne ihn auszuführen, um Fehler in der Logik oder mögliche Schwachstellen zu identifizieren. Statische Analysewerkzeuge durchsuchen den Quellcode nach Sicherheitslücken oder nicht erfüllten Bedingungen. Dynamische Tests im White Box Testing konzentrieren sich auf die laufzeitabhängigen Aspekte, wie die Überprüfung von Interrupthandlern oder Registerzugriffen im Mikrocontroller. Hier spielt auch die Emulation eine große Rolle, um Tests durchführen zu können, bevor die reale Hardware verfügbar ist. QEMU (Quick Emulator) wird häufig eingesetzt, um die Hardwareumgebung zu simulieren und damit das Verhalten der Software zu testen, insbesondere in frühen Entwicklungsphasen. Die Emulation stößt jedoch bei hardwarekritischen Aspekten wie Signalgenauigkeit oder Echtzeitverhalten an ihre Grenzen, so dass abschließende Tests auf echter Hardware erforderlich sind.

Grey Box Testing kombiniert die Ansätze von Black und White Box Testing und ist besonders nützlich in Szenarien, bei denen der Tester teilweise Kenntnis über den Quellcode und die Hardware hat. Hier können Tests durchgeführt werden, um kritische Module oder Schnittstellen gezielt zu testen. In eingebetteten Systemen wird Grey Box Testing häufig genutzt, um spezifische Komponenten oder Kommunikationsprotokolle wie UART oder SPI zu testen, bei denen sowohl die externe Funktionalität als auch die interne Logik relevant sind. Dynamische Tests können hierbei sicherstellen, dass das System unter verschiedenen Betriebsbedingungen korrekt funktioniert.

Die Kombination aus dynamischen und statischen Tests innerhalb der drei Testmethoden ist in eingebetteten Systemen unerlässlich, da die enge Kopplung von Software und Hardware spezielle Testansätze erfordert. Während Black Box Tests sicherstellen, dass das System funktional den Anforderungen entspricht, helfen White und Grey Box Tests, Fehler auf tieferer Ebene zu erkennen, die durch die enge Abhängigkeit von Hardware und Echtzeitanforderungen entstehen können.

Bedarf & Kapazität

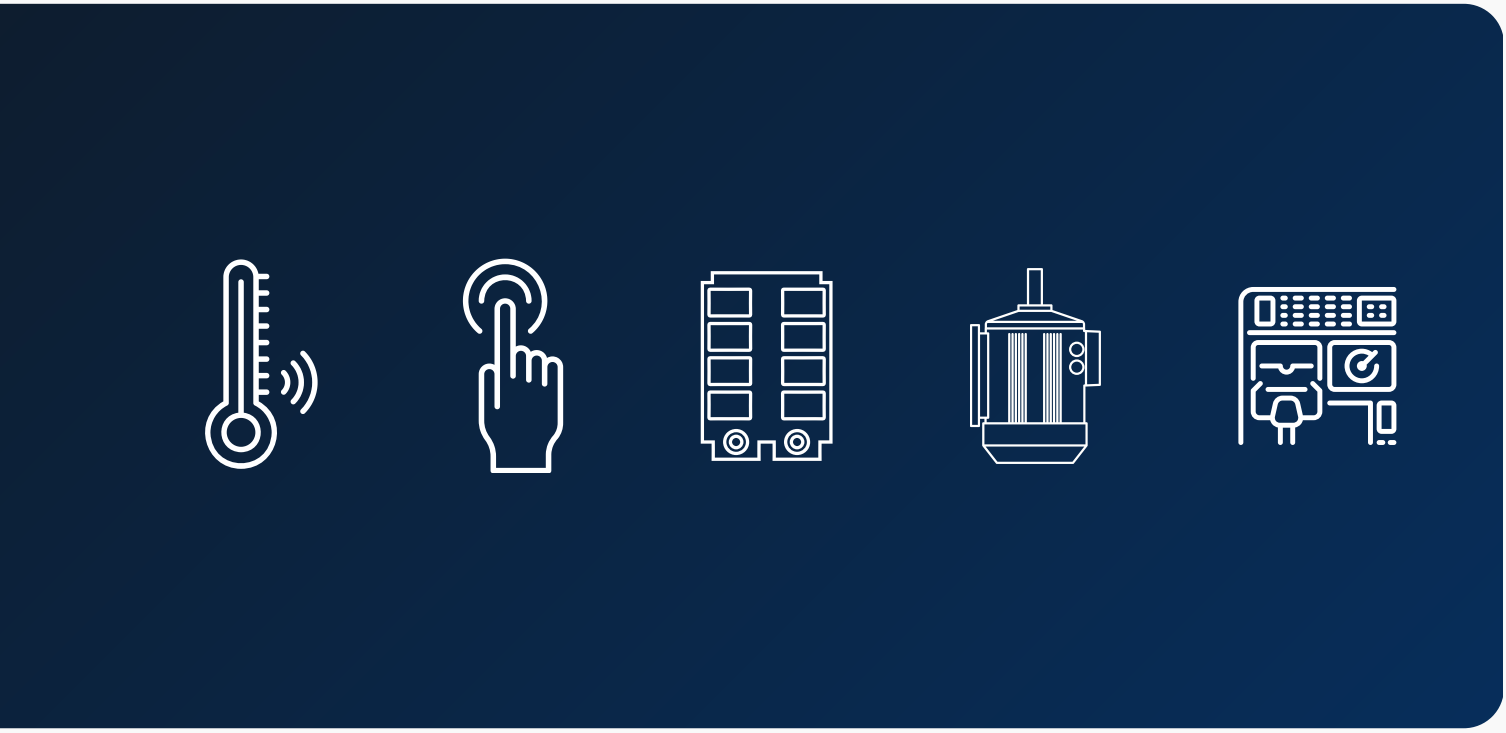
Bedarf der Qualitätssicherung ermitteln

Aspekte des Bedarfs untersuchen

Der Bedarf an Qualitätssicherung hängt stark von den **Produktanforderungen** und der **Unternehmensstrategie** ab. Eine langfristige Optimierung der Qualitätssicherung lohnt sich, wenn sie nachhaltig zur Verbesserung der Produktqualität und der betrieblichen Effizienz beiträgt. Die Entscheidung, welche Qualitätssicherungsmaßnahmen sinnvoll sind, hängt direkt von der Art des Produktes, der Einsatzumgebung und den langfristigen Zielen des Unternehmens ab. Die Bedarfsermittlung basiert auf einer **Kosten-Nutzen-Analyse**, bei der Aufwand und Ertrag der Prüfmaßnahmen sorgfältig gegeneinander abgewogen werden, um eine **bedarfsgerechte Kapazität** für die Qualitätssicherung zu schaffen.

Ein Unternehmen, das selten neue Produkte entwickelt, könnte in eine grundlegende, aber kostengünstige Qualitätssicherung investieren, während Unternehmen mit hohem Durchsatz und individuellen Kundenlösungen umfassendere, automatisierte Tests und dynamischere Qualitätssicherungsmaßnahmen benötigen, um der Vielzahl von Varianten und Anforderungen gerecht zu werden. Für weniger komplexe Systeme, wie z.B. einen Temperatursensor, reicht eine minimalistische Teststrategie aus, die auf einfache Funktionstests abzielt, während komplexe Systeme, wie z.B. das Energiemanagement in einem Elektrobagger, eine robuste Qualitätssicherung erfordern, die verschiedene Szenarien, Lastwechsel und Extrembedingungen abdeckt.

Bedarf der Qualitätssicherung

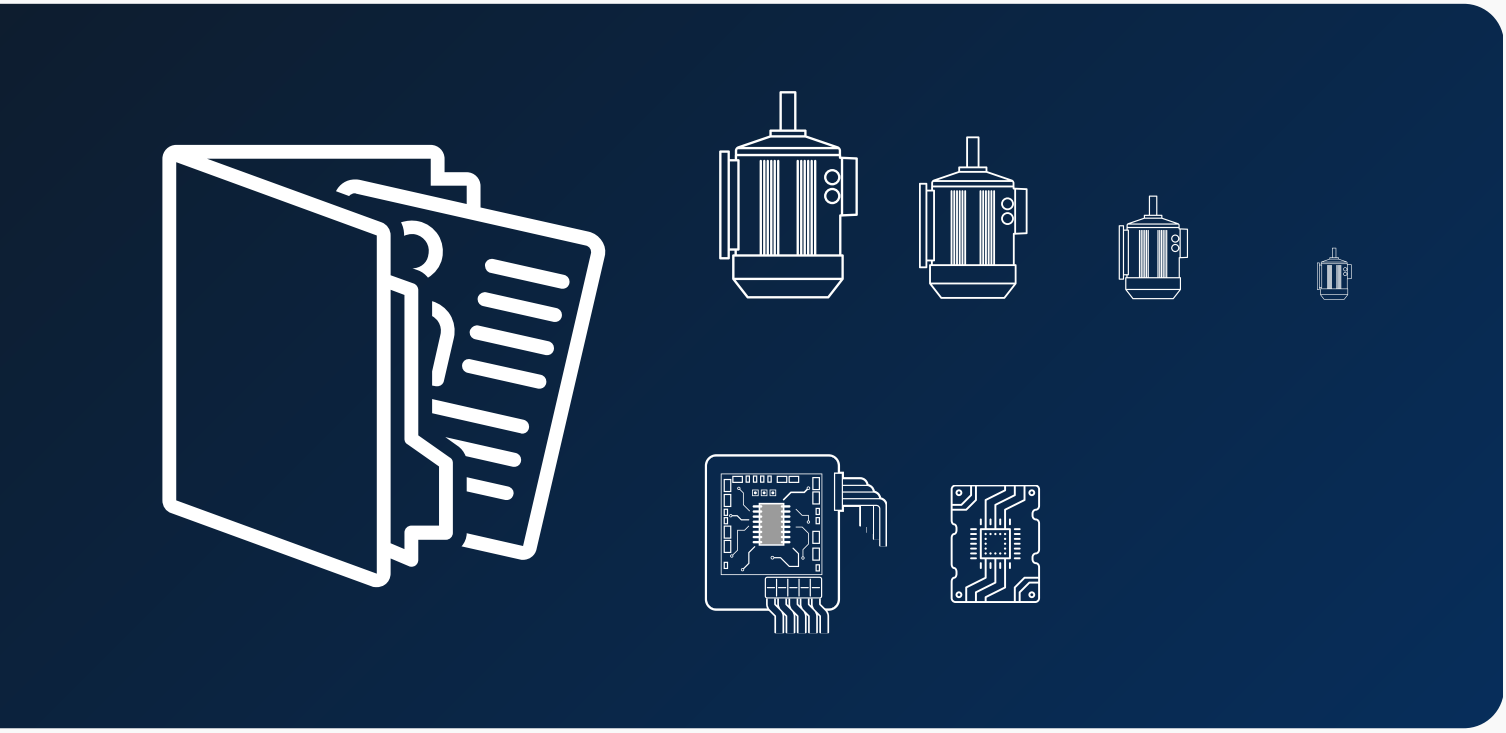


1. Projektbezogene Anforderungen

Technische Anforderungen des Produkts

- **Anzahl der Anforderungen:** Je mehr Anforderungen erfasst werden müssen, desto größer wird auch der Aufwand in der Erfassung und Auswertung dieser ebendieser Größen. Die Korrelation zwischen den Parametern und deren unterschiedliche Komplexitäten beeinflussen den Testaufwand erheblich.
- **Anzahl der Geräte und Testzyklen:** Die Anzahl der Geräte, die getestet werden soll wirkt sich auf den Aufwand für die Durchführung der Tests aus - insbesondere, wenn Testkonfigurationen an einem Prüfstand einzurichten sind. Mehrere Iterationen zur Qualitätssicherung, die oft erforderlich sind, erhöhen den Aufwand erheblich, insbesondere wenn jedes Testgerät neu konfiguriert werden muss.
- **Schnittstellenvielfalt**: Die Vielzahl an Kommunikationsschnittstellen (z. B. CAN, SPI, UART) kann zu einem erhöhten Testaufwand führen, da jede Schnittstelle spezifisch geprüft werden muss.
- **Software-Komplexität:** Je komplexer die Firmware, desto mehr Einfluss hat dies auf den Aufwand, da komplexe Programme oft mehr Edge-Cases aufweisen, die getestet werden müssen.
- **Datenrate**: Die Anforderungen an die Datenrate können den Umfang der Qualitätssicherung stark beeinflussen. Müssen beispielsweise sehr kurze Systemreaktionszeiten oder Stromspitzen bei 10 kHz gemessen werden, erfordert dies spezielle Messsysteme und Testverfahren.
- **Echtzeitanforderungen:** Wenn das Produkt bestimmte Echtzeitfähigkeiten erfüllen muss, müssen auch die Testsysteme entsprechend ausgelegt sein, um diese Anforderungen präzise zu prüfen.
- **Ressourcenbeschränkungen**: Hohe Datenraten oder umfangreiche Testdaten können durch begrenzte Kommunikationskanäle, Speicherplatz oder Rechenleistung erschwert werden. Dies erfordert spezifische Maßnahmen zur Überwindung dieser Einschränkungen.
- **Betriebsmodi und Einsatzumgebungen:** Die Anzahl der Betriebsmodi sowie besondere Einsatzbedingungen (z. B. extreme Temperaturen) beeinflussen den Aufwand für die Qualitätssicherung erheblich. Tests unter realen Einsatzbedingungen sind oft aufwendiger und erfordern spezielle Ausrüstung.

Bedarf der Qualitätssicherung



2. Portfoliobezogene Anforderungen

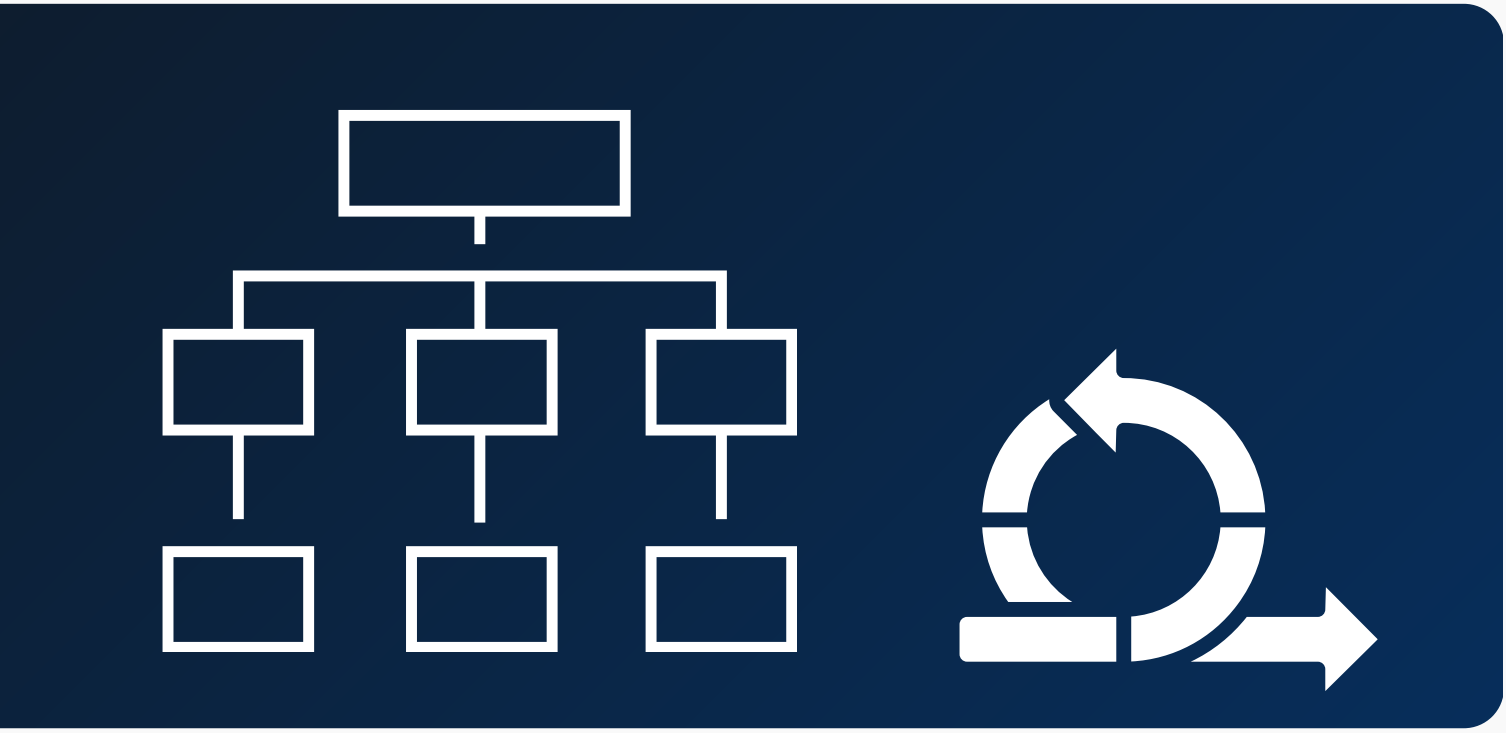
Produktmanagement- und Geschäftsanforderungen

Das Produktportfolio eines Unternehmens hat ebenfalls einen großen Einfluss auf den Bedarf an Qualitätssicherung:

- **Hochspezialisierte Produkte:** Wenn das Portfolio nur ein einziges Produkt mit sehr hohen technischen Anforderungen umfasst, kann es akzeptabel sein, wenn die Qualitätssicherung hier ineffizient ist, da die Frequenz der Tests gering ist.

- **Produkt- vs. Projektgeschäft:** Unternehmen, die im Produktgeschäft tätig sind, führen in der Regel weniger häufig Qualitätssicherungsdurchläufe durch als Unternehmen im Projektgeschäft, die ständig neue, maßgeschneiderte Lösungen entwickeln.

Bedarf der Qualitätssicherung



3. Strategische und operative Anforderungen

Unternehmensstruktur und Prozesse

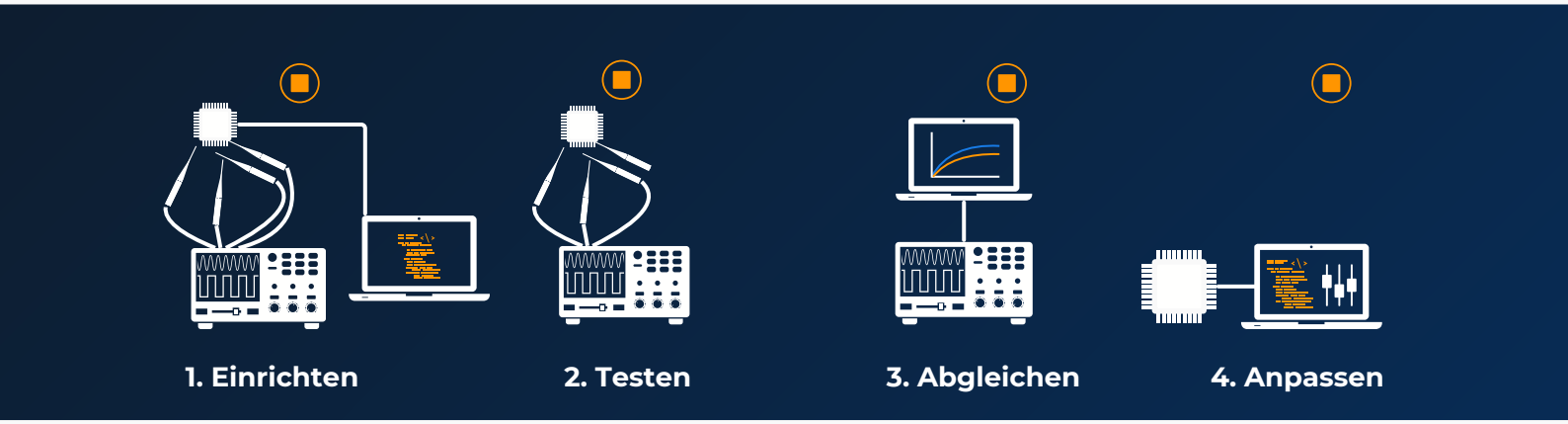
Die Organisationsstruktur beeinflusst ebenfalls, wie effizient die Qualitätssicherung gestaltet werden kann. Eine abteilungsübergreifende Prozesslandschaft kann eine besondere Herausforderung darstellen.

Wenn die Entwicklung eines Produkts auf mehrere Abteilungen verteilt ist, z. B. Hardware, Software und Validierung, können Iterationsschleifen besonders problematisch sein. Beispielsweise kann es vorkommen, dass ein Entwickler in der Validierungsabteilung eine Anomalie entdeckt und das Produkt an den Programmierer zurückgibt. Dies führt oft zu zusätzlichen Verzögerungen, insbesondere wenn diese Abteilungen unterschiedliche Prioritäten oder parallele Projekte haben. In einem solchen Szenario ist die Qualitätssicherung ein Prozess, der sowohl gut strukturiert als auch agil sein muss, um effizient zu sein.

Kapazitätsbewertung

Qualitätssicherung von der Bastelei bis zu Raumfahrt

Die Effizienz des Tuning-Prozesses hängt von den Prozessen, den verfügbaren Werkzeugen und der Personalstärke ab. Jedes Unternehmen hat unterschiedliche Kapazitäten. Wir haben hier **fünf arbiträr definierte Levels zur Einordnung der Kapazität der Qualitätssicherung.**



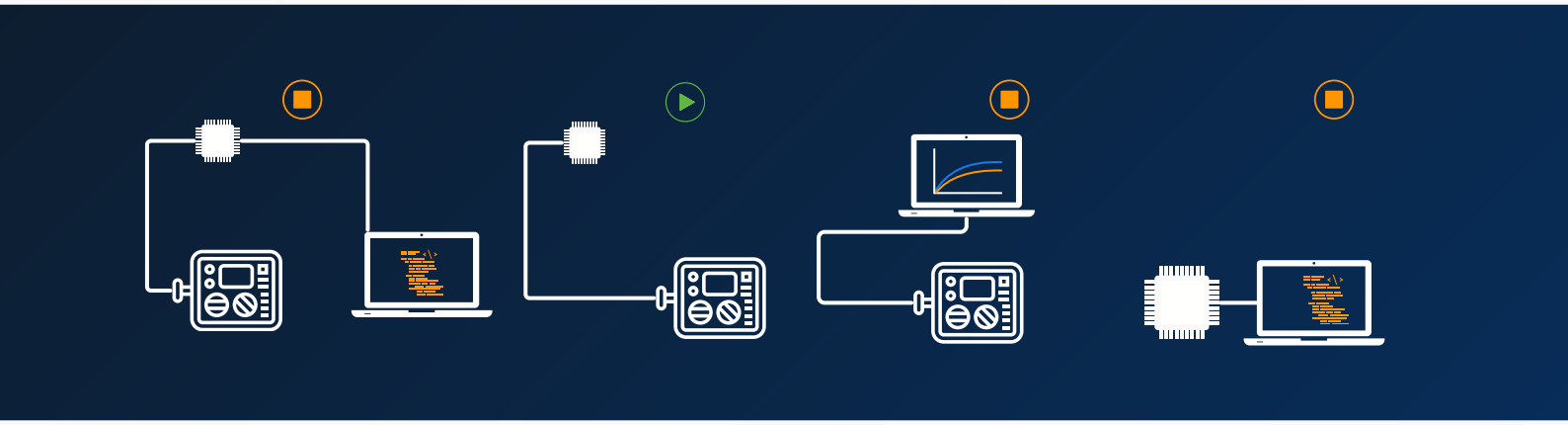
Level 1 (Basic): Black Box, Anpassung offline, asynchrone Erfassung

Beschreibung: Labormessequipment, wie ein Oszilloskop, wird zur Datenerfassung an den Pins des Microcontrollers oder an Testpunkten genutzt.

Merkmale: Die Daten werden im Anschluss eines Tests exportiert und ausgewertet. Anpassungen werden manuell einprogrammiert. Die Daten des Labormessgeräts sind nicht mit den Eingaben oder anderen Systemparametern synchronisiert. Dieser Ansatz ist sehr zeitaufwändig und iterativ, da Anpassungen erst nach der Systemabschaltung vorgenommen und die Daten manuell ausgewertet werden. Jede Anpassung erfordert einen erneuten Testlauf, was zu langen Entwicklungszyklen führt. Die Black-Box Betrachtung mit asynchronen Daten erlaubt Korrekturen an groben Abweichungen zur Messdatenkalibrierung, hilft beim Tuning von Reglern aber wenig.

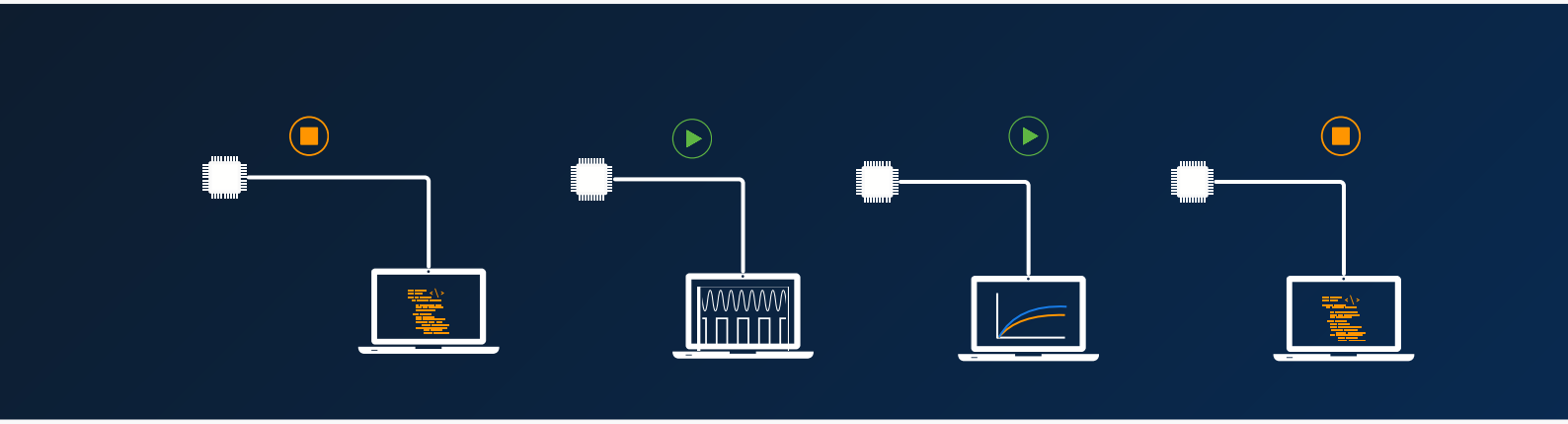
Einsatzszenarien: Geeignet für einfache Systeme mit sehr niedrigen Anforderungen, bei denen nur gelegentliche Anpassungen nötig sind.

Beispiel: Ein Temperatursensor, der seriell alle 10 Minuten einen gerundeten Temperaturwert ausgibt.



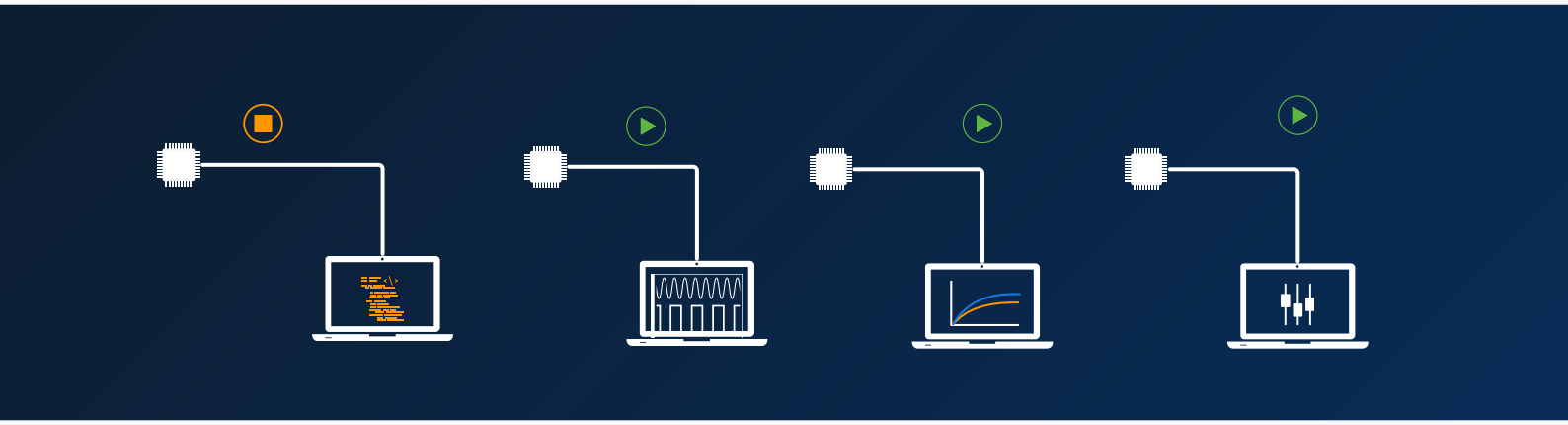
Level 2 (Sync): Black Box, Anpassung offline, **synchronisierte** Erfassung

- **Beschreibung**: Die Datenerfassung erfolgt mittels eines synchronisierten Datenloggers, an den interne Daten über eine Kommunikationsschnittstelle ausgegeben werden.
- **Merkmale**: Nach dem Testlauf werden die Daten exportiert und analysiert, um notwendige Anpassungen zu bestimmen. Die Abtastrate ist hoch genug, um die allgemeine Systemleistung zu überwachen, aber bei schnellen Systemveränderungen könnten Details verloren gehen. Die Datenerfassung mit einem Datenlogger verbessert die Effizienz im Vergleich zu Level 1, jedoch bleibt die Offline-Natur der Anpassung, die zu vielen Iterationen führt. Einfache Tuning Aufgaben sind hiermit möglich.
- **Einsatzszenarien:** Einfache Sensor-Anwendungen
- **Beispiel**: Das Kriechverhalten eines kapazitiven Drucksensors unter verschiedenen Einsatzbedingungen testen und auswerten, um dies in einer Look-up-Table zu berücksichtigen.



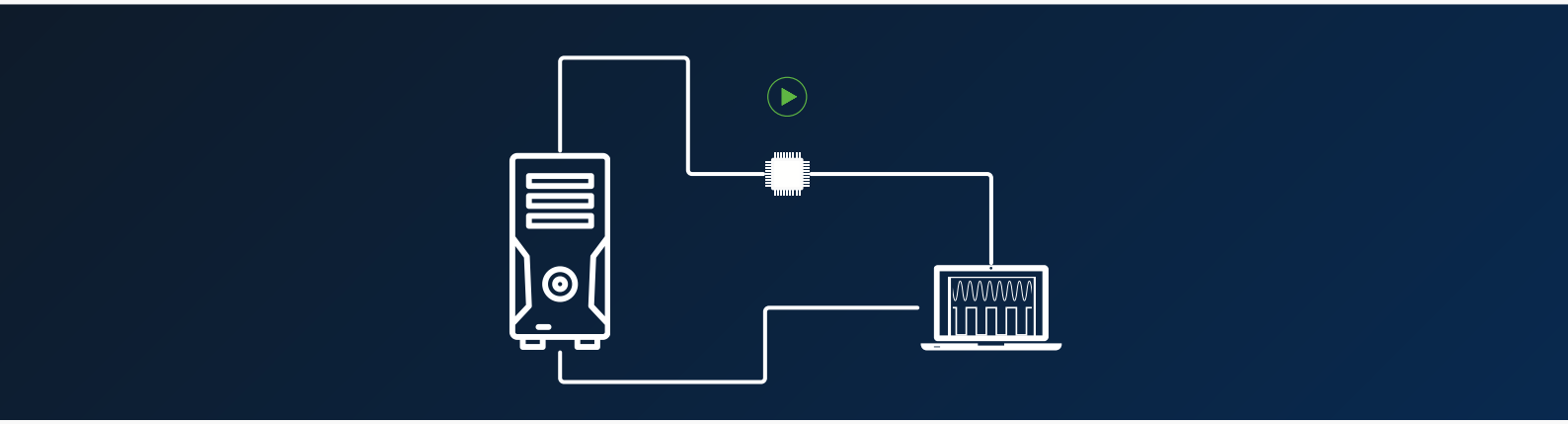
Level 3 (Echtzeit): Grey Box, Anpassung offline, synchronisierte **Echtzeit** Erfassung

- **Beschreibung**: Die Erfassung und Aufzeichnung ausgewählter Variablen erfolgt während diese in Echtzeit und während des laufenden Systembetriebs angezeigt und überwacht werden.
- **Merkmale**: Es können direkt im Verlauf des Tests Rückschlüsse auf das Systemverhalten ermittelt werden und im Anschluss ohne gesonderte Analyse Parameteranpassungen in der Software einprogrammiert werden. Mehrere Daten können direkt miteinander verglichen werden. Es gibt weniger Unsicherheit und Unterbrechungen als bei einer Erfassung ohne Echtzeit-Anzeige.
- **Einsatzszenarien**: Tuning von Reglern, Validierung bei einer Vielzahl an Signalen, hohe Testabdeckung durch Unit Testing, Systeme, die hohe Anforderungen und einen hohen Qualitätsanspruch haben
- **Beispiel:** FOC Motorsteuerung eines BLDC Controllers



Level 4 (Online-Actions): Grey Box, Anpassung **online**, synchronisierte Echtzeit-Erfassung

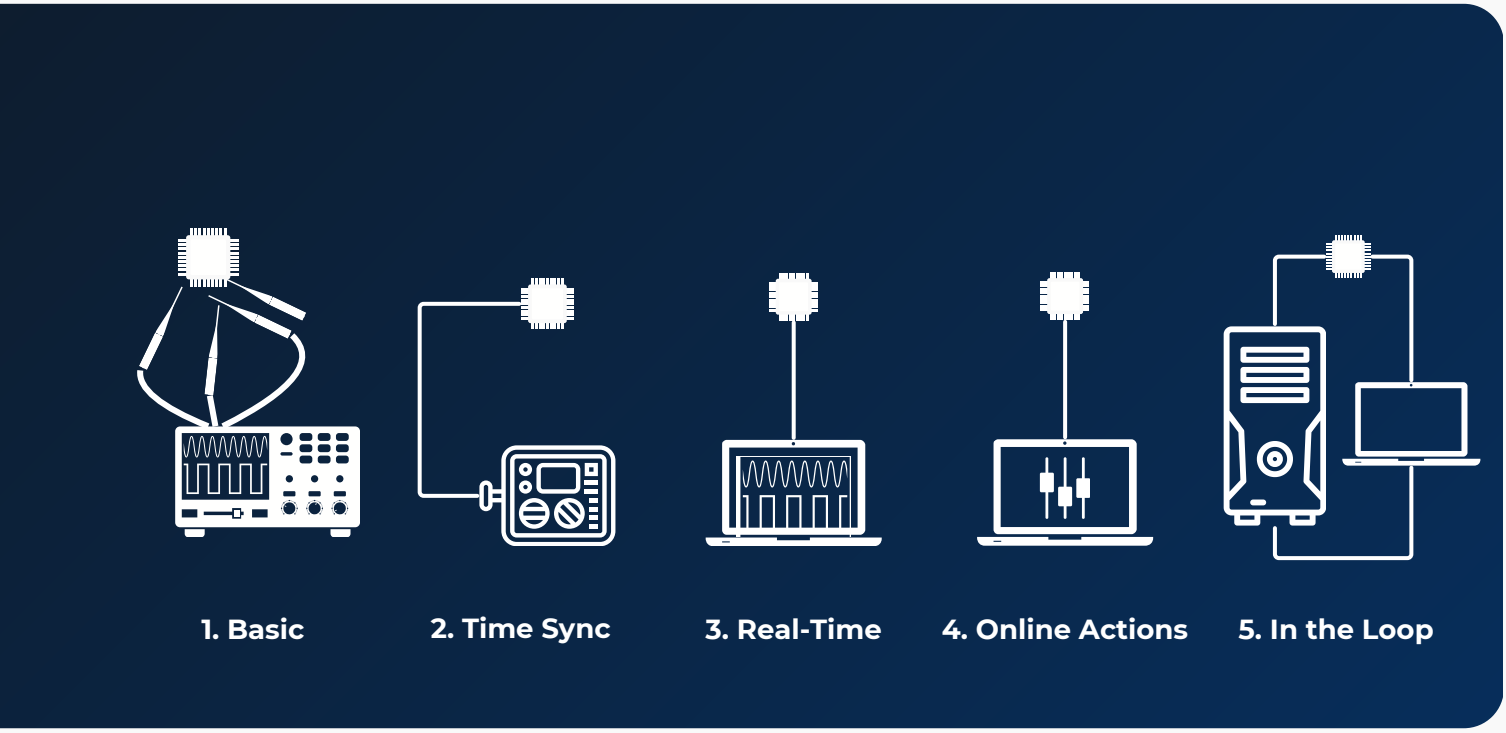
- **Beschreibung**: Gleiche Bedingungen wie bei Level 3, jedoch mit der Möglichkeit, Parameter während der Laufzeit einzustellen, Befehle zu senden und Event-Trigger einzurichten. Diese Trigger aktivieren die Datenerfassung bei bestimmten Ereignissen oder Anomalien, sodass kritische Systemzustände gezielt erfasst werden können.
- **Merkmale**: Anpassungen zur Laufzeit reduzieren die Anzahl der Iterationen und Unterbrechungen. Die hohe Abtastrate ermöglicht eine sehr präzise Überwachung des Systems. Durch Event-Trigger lassen sich seltene oder kritische Ereignisse des Systems erfassen, was den Tuning-Prozess effizienter und gezielter macht.
- **Einsatzszenarien**: Besonders nützlich für Systeme, die eine hohe Testabdeckung benötigen, die unter verschiedenen Umgebungsbedingungen betrieben werden müssen, oder deren Leistung stark optimiert werden muss.
- **Beispiel:** Sensorlose FOC Motorsteuerung eines BLDC Controllers



Level 5 (In the Loop): White Box Testing mit Hardware-in-the-Loop (HIL)

- **Beschreibung**: Der Tuning-Prozess ist in diesem Level vollständig automatisiert. Ein HIL-System (Hardware-in-the-Loop) ermöglicht es, Parameter in Echtzeit automatisch anzupassen und sofortige Optimierungen vorzunehmen.
- **Merkmale**: Automatische Anpassungen und kontinuierliche Echtzeit-Überwachung ermöglichen das Testen von komplexen Szenarien, in denen das System in Echtzeit auf simulierte Umgebungen reagiert. Der Einrichtungsaufwand ist hoch, sowohl in Bezug auf die Kosten als auch auf die technische Implementierung.
- **Einsatzszenarien**: Optimal für sicherheitskritische Anwendungen (z.B. Luftfahrt, Medizintechnik), bei denen Fehler schwerwiegende Folgen haben könnten.
- **Beispiel**: Ein autonomes Fahrzeugsteuerungssystem kann getestet werden, um zu simulieren, wie das Fahrzeug in verschiedenen Situationen (Regen, Schnee) reagiert.

Kapazitätsbewertung



Fünf Levels der Kapazität

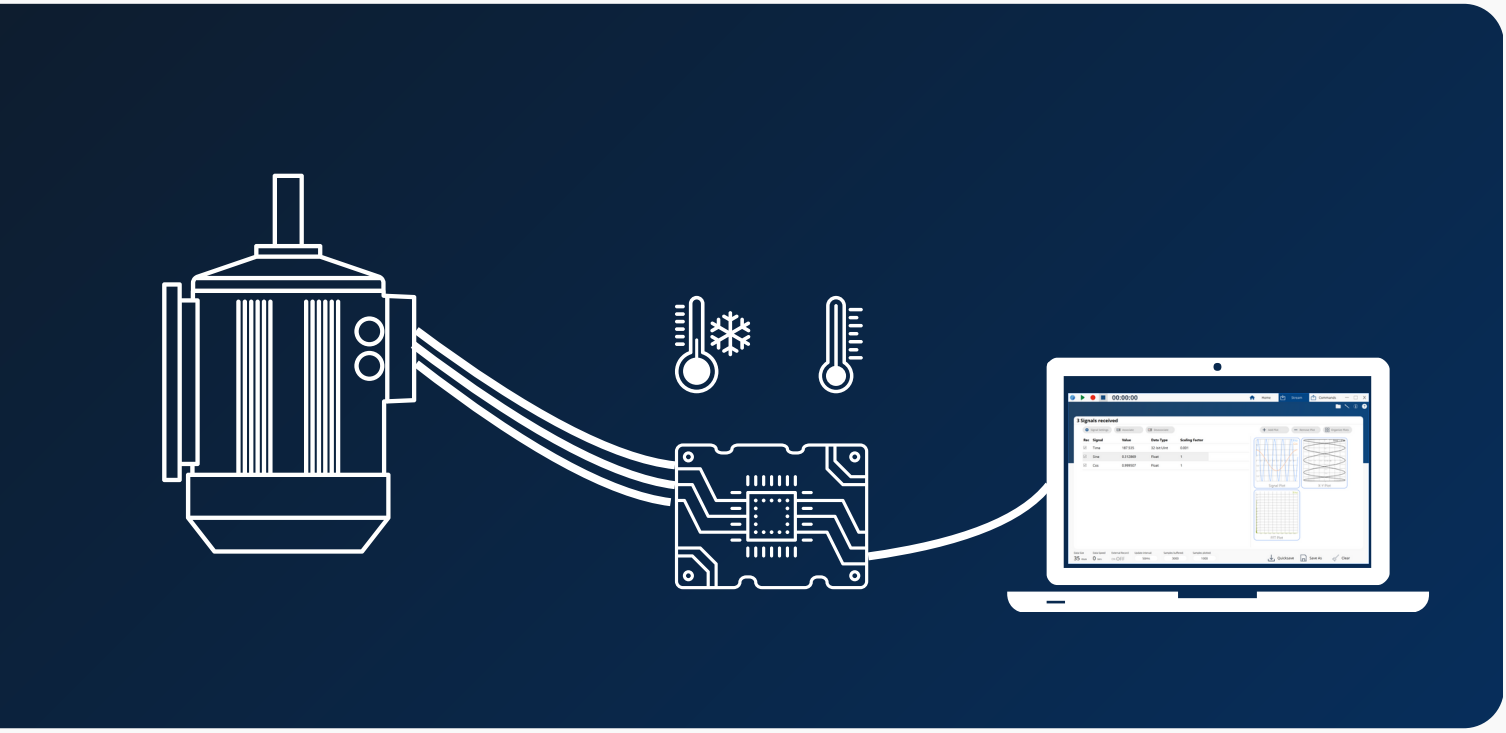
Qualitätssicherung von der Bastelei bis zu Raumfahrt

Level 1 oder 2: Für einfache, unkritische Systeme ausreichend, jedoch mit der Unsicherheit, dass der Sicherungsaufwand schwer abschätzbar und im schlimmsten Fall groß werden kann. Der Tuning-Prozess ist hier relativ ineffizient, sodass sich der Aufwand schnell erhöhen kann, wenn viele Iterationen notwendig sind.

Level 3 oder 4: Für komplexe, anspruchsvolle Systeme in zeitkritischen Projekten essenziell. Der Einsatz von Echtzeit-Daten und synchronisierten Anpassungen sorgt für eine effizientere Prozesssteuerung und eine bessere Vorhersagbarkeit des Sicherungsaufwands. Hiermit lässt sich die Entwicklungszeit und die Qualitätssicherung erheblich verbessern.

Level 5: Aufgrund der hohen Kosten und des Aufwands nur sinnvoll für sicherheitsrelevante und extrem komplexe Anwendungen. Solche Systeme müssen oft unter einer Vielzahl von Einsatzbedingungen und Umgebungsfaktoren funktionieren, was eine umfassende automatische Echtzeitüberwachung und Anpassung notwendig macht.

Der Business Case: es:scope®



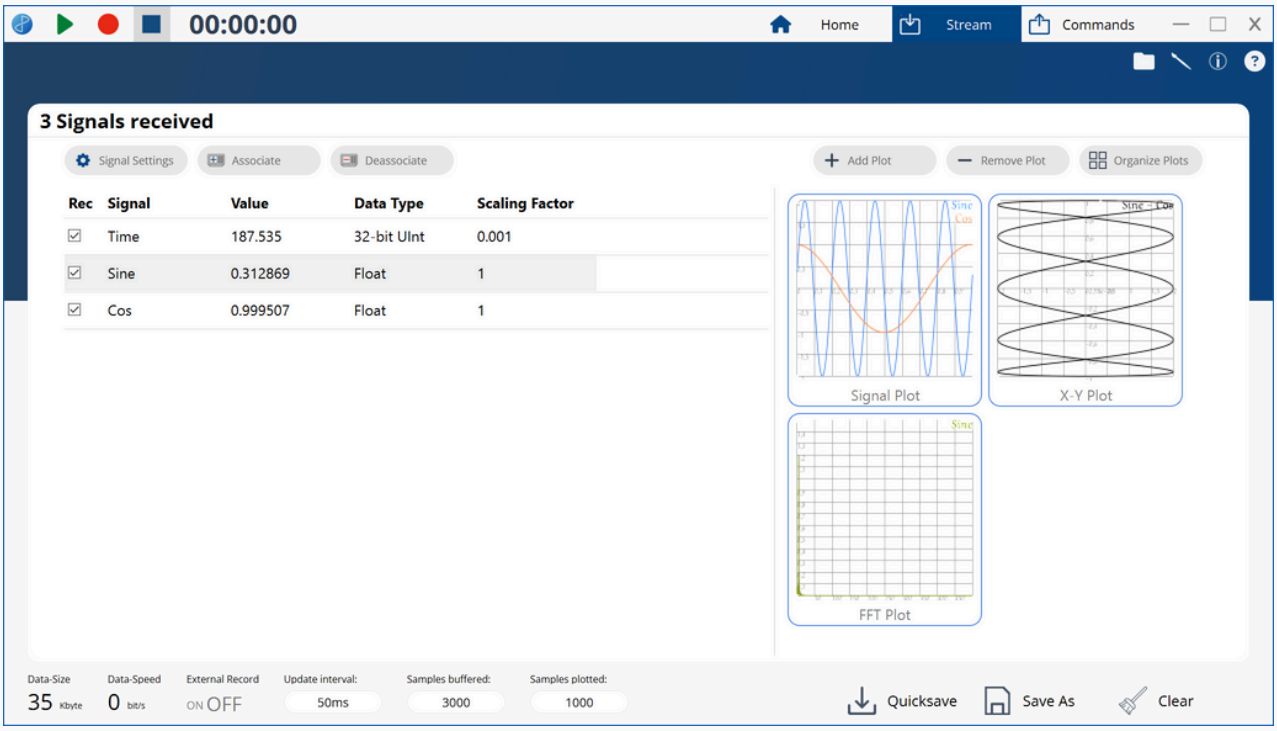
Die Ausgangssituation

Neue Motorsteuerung in Mittelstandsunternehmen entwicklen

Ein mittelständisches Unternehmen entwickelt eine neue FOC-Motorsteuerung für einen BLDC-Motor mit strengen Echtzeitanforderungen und begrenzten Systemressourcen. Der bisherige Entwicklungsprozess wird als effizient genug betrachtet, aber durch die mehrfachen Iterationen und manuellen Anpassungen ist er dennoch zeitintensiv. Die Daten, die zur Validierung genutzt werden, werden als ausreichend betrachtet - auch wenn es keine hohe Testabdeckung gibt.

- **Technische Anforderungen:**
  - FOC-Motorsteuerung für BLDC-Motor
  - 20 Parameter in der Anforderungsliste
  - Hohe Datenrate (20 kHz bei der Ansteuerung, 1 Mbps im CAN-Bus)
  - Minimale Systemressourcen
  - Umweltbedingungen -20°C bis 60°C
- **Produktportfolio:**
  - Antriebssysteme wie Umrichter, Motoren, Motorsteuerungen und Hall-Sensoren, mit typischer Laufzeit von 8-12 Jahren
- **Unternehmensstruktur:**
  - Mittelständisches Unternehmen mit 160 Mitarbeitern
  - Trennung zwischen Software- und Hardwareentwicklung; Validierung durch Elektroingenieure
- **Aktuelle Qualitätssicherungs-Kapazität:**
  - Nutzung eines Datenloggers zur Datenerfassung
  - Nach jeder Iteration müssen neue Parameter manuell programmiert und getestet werden
  - Simulationen unterstützen die Parameterauswahl, aber das System muss regelmäßig abgeschaltet werden

Der Business Case: es:scope®



Der Ansatz

es:scope® im bestehenden Entwicklungsprozess einführen und testen

Ziel ist es, es:scope® in den bestehenden Entwicklungsprozess zu integrieren, um die Effizienz zu steigern und den Nutzen auch für nachfolgende Projekte zu evaluieren. Eine Software-Ingenieurin und ein Hardware-Ingenieur erhalten eine Lizenz, um den besten Weg zur Implementierung und Nutzung der Software zu ermitteln.

Ablauf:

- In einer kurzen Einweisung wird es:prot in den bestehenden Microcontroller-Code integriert.
- Es werden relevante Variablen für die Messgrößen und Parameter zur Kalibrierung, sowie die Kommunikationsparameter ausgewählt.
- Über eine beliebige serielle Schnittstelle können nun Echtzeit-Daten auf dem Laptop verfolgt und Parameter angepasst werden. Auch während das Gerät sich im Temperaturprüfschrank befindet.
- Sobald eine gültige Konfiguration gefunden ist, wird der Code final angepasst, um die Entwicklung abzuschließen.

Vorteile:
Für den Elektroingenieur: Echtzeit-Analyse der Messsignale und der Regler-Performance, sowie Kalibrierung und Tuning während des Tests.
Für die Programmiererin: Unit-Tests der hardwareabhängigen Software, die zuvor nicht möglich waren.

1. Zeitaufwand & Lohnkosten

Durch den Einsatz von es:scope® können Iterationen reduziert und die Durchlaufzeit erheblich verkürzt werden. Bisherige manuelle Anpassungen und Systemabschaltungen entfallen, da es:scope® eine Echtzeit-Analyse und Kalibrierung ermöglicht.

- **Echtzeit-Analyse und Kalibrierung:** Anpassungen erfolgen während des Systembetriebs, wodurch die Notwendigkeit für Systemabschaltungen entfällt.
- **Effiziente Parameterauswahl:** Messparameter lassen sich durch einfache Code-Änderungen festlegen, was Einrichtungs- und Anpassungszeiten minimiert.
- **Universelle Schnittstellen**: es:scope® kann mit verschiedenen Protokollen kommunizieren, was aufwändige Anpassungen überflüssig macht und die Einrichtung beschleunigt.
- **Plug-and-Play:** Vorkonfigurationen für die Anzeige von Signalen erleichtern die Inbetriebnahme.
- **Keine Neustarts notwendig:** Betriebsmodi können direkt durch Befehle aus es:scope® geändert werden.
- **Anzahl der getesteten Geräte und der Testdurchläufe:** Durch die softwarebeschriebene Testkonfiguration kann eine Vielzahl an Geräten per Plug-and-Play analysiert werden.

Beispiel:

Aspekt	Vorher	Nachher
Einrichtung des Messsystems	4 Stunden	1 Stunde
Lohnkosten der Einrichtung	320€	80€
Iterationen	8 Iterationen	2 Iterationen
Zeitaufwand pro Iteration	8 Stunden	2 Stunden
Gesamtlohnkosten pro Iteration	640 € (8h * 80 €/h)	160 € (2h * 80 €/h)
Gesamtkosten	**5440€**	**400€**

Bessere Planbarkeit: Durch die Reduzierung der Iterationen und die schnellere Einrichtung fallen keine Unterbrechungen durch andere Aufgaben an, was eine exakte Projektplanung ermöglicht.

Prozessbeschleunigung: 2 Wochen

Lohnkostenersparnis: 5080€

2. Equipmentkosten

Resultat: Der Beschaffungsbedarf kann mit es:scope® reduziert werden

Durch den Einsatz von es:scope® kann der Beschaffungs- und Wartungsbedarf für spezialisierte Messtechnik erheblich reduziert werden. In vielen Fällen werden teure Systeme wie Hardware-in-the-Loop (HIL)-Systeme, spezialisierte Datenlogger oder andere Testgeräte überflüssig. Es werden vorhandene Hardware und Software genutzt, um die nötige Testgenauigkeit und Datenqualität zu erreichen.

Reduzierung des Messgeräte-Bedarfs
es:scope® ermöglicht es, bestehende Infrastruktur optimal zu nutzen, was den Bedarf für teure Neuinvestitionen in Messtechnik minimiert:

- **Weniger Hardware:** Statt Datenlogger, Digitaloszilloskope oder HIL-Systeme wird die Echtzeit-Datenerfassung direkt auf einem Computer ermöglicht, wodurch die Abhängigkeit von benötigter physischen Messgeräten sinkt.
- **Weniger Software:** Mit dem Protokoll es:prot, das mit es:scope® zusammenarbeitet ist es nicht erforderlich, zusätzlichen Microcontroller-Code zur Signalbereitstellung, für Messprotokolle oder spezielle Software für die Datenformatierung zu entwickeln. Dies spart nicht nur Entwicklungszeit, sondern vermeidet auch mögliche Fehlerquellen bei der Erstellung eigener Softwarelösungen.
- **Weniger Peripherie: es:scope®** ermöglicht es, das bestehende System zur Datenerfassung zu nutzen, ohne dass zusätzliche externe Messtechnik oder Kommunikationsperipherie erforderlich ist. Da es:prot direkt in die Embedded-Software integriert werden kann, bietet es große Flexibilität bei der Systemarchitektur und eliminiert den Bedarf für zusätzliche physische Schnittstellen oder Geräte.
- **Weniger Wartung:** Anders als bei physischem Equipment entfallen bei es:scope® die Wartungskosten für Hardware.

Beispiele:

Um das Einsparungspotenzial durch den Einsatz von es:scope® zu verdeutlichen, hier eine beispielhafte Kalkulation:

- **Kosten eines Datenloggers:** 1.000 € bis 5.000 €.
- **Entwicklungskosten für eigene Microcontroller-Software zur Signalbereitstellung**: 40 Stunden * 80 €/h = 3.200 €.

Gesamteinsparungen:
Durch den Einsatz von es:scope® werden die Anschaffungskosten für Messequipment reduziert, da es:scope® eine kostengünstige Alternative zur herkömmlichen Messtechnik bietet. Ein zentrales Merkmal ist die Schnittstellenunabhängigkeit, wodurch Anpassungen des Systems und Prüfständen für spezifische Tests weniger aufwendig werden. Materialkosten, die normalerweise für solche Anpassungen anfallen, entfallen oder werden minimiert. Auch die Entwicklungskosten für eigene Test-Software können gesenkt werden, da es:scope® eine umfassende, sofort nutzbare Test- und Analyseumgebung bereitstellt.

3. Qualitätsrisiko

Resultat: Durch die höhere Testsicherheit und Testabdeckung mit es:scope® kann die Sicherheit erhöht werden, dass Design-Fehler vor der Produktion entdeckt werden. Die programmierbaren Trigger und die hohe Abtastrate von es:scope® ermöglichen es, Fehler, Anomalien und Ausfallszenarien zu identifizieren und zu dokumentieren.

- **Anomalie-, Fehler- und Ausfallerkennung**: Mit der hohen Abtastrate von es:scope® können selbst kleinste Abweichungen vom gewünschten Systemverhalten erkannt werden. Kritische Zustände und Peaks, die bei niedrigen Abtastraten unentdeckt bleiben könnten, werden frühzeitig erfasst. es:scope® ermöglicht außerdem Langzeittests, bei denen kritische Systemzustände erfasst werden. Mit programmierbaren Triggern können gezielt Anomalien verfolgt und die genauen Bedingungen dokumentiert werden, unter denen sie auftreten. Mit Befehlen kann die Reaktion des Systems überprüft werden.
- **Feintuning und Optimierung:** Durch das Echtzeit-Feedback der Messgrößen und die Online-Parameteranpassungen können verschiedene Einstellungen am realen Gerät schnell ausprobiert werden. So kann eine Parameterkonfiguration für eine optimierte Systemperformance gefunden werden.
- **Testabdeckung:** Mit es:scope® lässt sich hardware-abhängige Software in verschiedenen Prüfszenarien analysieren. Über Kommandos kann das Systemverhalten auf bestimmte Eingaben überprüft werden. Durch die flexible Parameterauswahl, das befehlsbasierte Ändern des Systemverhaltens und die Echtzeit-Analyse wird die Wahrscheinlichkeit verringert, dass Bugs oder Fehler erst nach der Produktion entdeckt werden.

Beispiel:
Um das Einsparungspotenzial zu verdeutlichen, hier Beispiele typischer Kostenpunkte:

- **Kosten für Nachbesserungen:** Typische Nachbesserungskosten liegen normal im fünfstelligen Bereich, insbesondere bei Serienproduktionen. Solche Kosten umfassen nicht nur Material und Arbeitszeit, sondern oft auch zusätzliche Tests und ggf. den Rücktransport defekter Einheiten.
- **Kosten für fehlerhaft produzierte Hardware:** Wenn eine Nachbesserung technisch nicht möglich ist, fallen volle Material- und Produktionskosten ohne Entschädigung an. Dies betrifft speziell komplexe Embedded-Hardware, bei der Fehler oft irreversibel sind.
- **Potenzielle Ausfallkosten:** Produktionsausfälle können leicht zu Kosten von mehreren 10.000 € pro Tag führen, je nach Volumen und Dringlichkeit der Produktion.
- **Produktionsverzögerungen:** Verzögerungen aufgrund unerwarteter Fehler können die Markteinführung verzögern, was zu Umsatzverlusten führt.

Quellen, die die wirtschaftlichen Auswirkungen der Testabdeckung behandeln:

- Jovanović, Z., & Živković, D. (2006). "Testing of Embedded Software". Journal of Automatic Control, 16(1), 9–14.
- National Institute of Standards and Technology (NIST) (2002). The Economic Impacts of Inadequate Infrastructure for Software Testing.
- Zhu, H., Hall, P.A.V., & May, J.H.R. (1997). "Software Unit Test Coverage and Adequacy". ACM Computing Surveys, 29(4), 366–427.
- Vahid Garousi et al. Testing embedded software: A survey of the literature, 2018

Fazit

Aufwand identifizieren & reduzieren

Bessere Prognosen, Risikominimierung und bedarfsgerechte Kapazitätsplanung

In diesem Dokument wurde ein umfassender Überblick über den Ablauf der Qualitätssicherung in der Entwicklung gegeben. Um den Aufwand in Bezug auf Zeit und Investitionen zu reduzieren, wurden Prognostizierbarkeit, Risikominimierung und die bedarfsgerechte Kapazitätsplanung der Qualitätssicherung detailliert untersucht.

Mit es:scope® können Validierungs- und Kalibrierprozesse deutlich verbessert, Kosten gesenkt und die Time-to-Market verkürzt werden. In der Regel rechnet sich der Einsatz von es:scope® bereits ab dem ersten Projekt. Langfristig steigen die Einsparungen mit jedem weiteren Projekt, so dass der Nutzen kontinuierlich wächst. Sowohl Embedded Software-Ingenieure als auch Elektroingenieure profitieren von der Möglichkeit, ihre Arbeitsabläufe zu optimieren und die Testabdeckung zu verbessern.

Die in diesem Dokument diskutierten Aspekte des Aufwands in der Qualitätssicherung können durch den Einsatz von es:scope® gezielt adressiert werden. Dennoch sollte im Einzelfall geprüft werden, wie die spezifischen Anforderungen an die Qualitätssicherung aussehen und in welchem Umfang eine Kapazitätserweiterung sinnvoll ist. Gerne unterstützen wir Sie bei der Erstellung eines maßgeschneiderten Business Case, um den optimalen Einsatz von es:scope® für Ihr Unternehmen zu evaluieren.

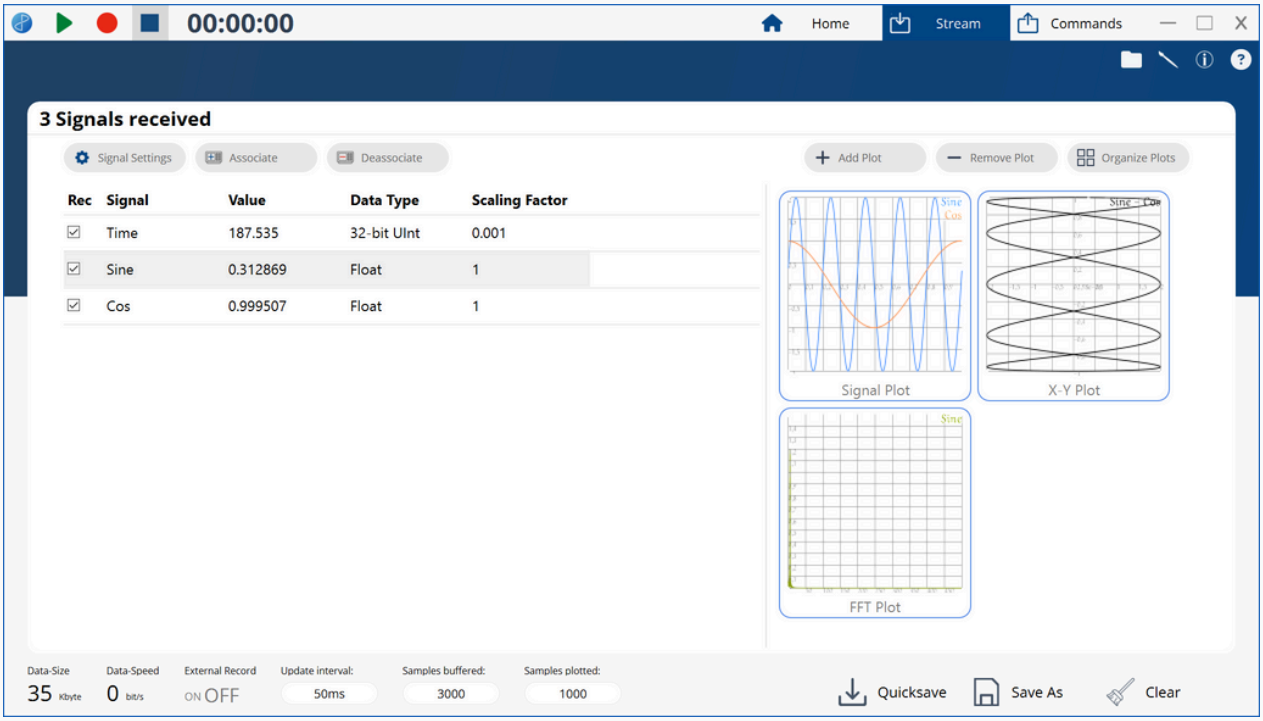
Kontakt

https://wa.me/4917681456107

kontakt@essaar.de

+49 (0) 176 814 561 07

essaar.de/kontakt



es:scope® Testversion jetzt anfordern

essaar.de/esscope

Appendix A: Testen von Embedded Software

Die Publikation "**Testing Embedded Software: A Survey of the Literature" von Garousi et al.** (2018 [https://doi.org/10.1016/j.infsof.2018.06.016] bietet einen umfassenden Überblick über den Stand der Technik im Testen von Embedded Software. Aus dieser Studie lassen sich mehrere wichtige Erkenntnisse zum Stand der Qualitätssicherung und den Herausforderungen beim Testen von Embedded Systems ableiten:

1. **Spezifische Herausforderungen beim Testen von Embedded Software:**
   - Hardware-Software-Integration: Embedded Systeme sind durch die enge Verzahnung von Hardware und Software gekennzeichnet, was das Testen komplexer macht.
   - Ressourcenbeschränkungen: Begrenzter Speicher, Rechenleistung und Energie beeinflussen die Wahl der Testmethoden.
   - Echtzeitanforderungen: Zeitkritische Funktionen erfordern spezielle Testansätze, um Timing- und Synchronisationsprobleme zu identifizieren.
   - Sicherheitskritische Anwendungen: In Domänen wie Automotive, Medizin oder Luftfahrt sind hohe Zuverlässigkeitsanforderungen zu erfüllen.
2. **Vielfalt der Testmethoden:**
   - Modellbasiertes Testen: Einsatz von Modellen zur Generierung von Testfällen und zur Simulation von Systemverhalten.
   - Hardware-in-the-Loop (HIL) Testing: Testen mit realer Hardware in einer kontrollierten Umgebung, um das Zusammenspiel von Hardware und Software zu prüfen.
   - Software-in-the-Loop (SIL) und Processor-in-the-Loop (PIL) Testing: Simulation von Hardwarekomponenten, um Softwaretests unabhängig von der physischen Hardware durchzuführen.
   - Formale Verifikation: Anwendung mathematischer Methoden zur Beweisführung von Eigenschaften der Software.
   - Statische und dynamische Analyse: Analyse des Quellcodes ohne (statisch) und mit (dynamisch) Ausführung, um Fehler frühzeitig zu erkennen.
3. **Automatisierung und Werkzeugunterstützung:**
   - Testautomatisierung: Notwendig, um die Effizienz zu steigern und menschliche Fehler zu reduzieren. Automatisierte Tests ermöglichen häufigere und gründlichere Prüfungen.
   - Spezialisierte Tools: Entwicklung und Einsatz von Werkzeugen, die auf die spezifischen Anforderungen von Embedded Systems zugeschnitten sind.
4. **Testabdeckung und Qualitätsmetriken:**
   - Erhöhte Testabdeckung: Eine höhere Abdeckung führt zu einer besseren Fehlerentdeckung, ist jedoch aufgrund der Komplexität von Embedded Systems oft schwer zu erreichen.
   - Metriken: Verwendung von Metriken zur Bewertung der Testeffektivität und zur Identifikation von Bereichen mit hohem Risiko.
5. **Herausforderungen und offene Forschungsfragen:**
   - Skalierbarkeit: Umgang mit der steigenden Komplexität und Größe von Embedded Software.
   - Automatisierte Testfallerstellung: Bedarf an Methoden zur automatischen Generierung von Testfällen, insbesondere für seltene oder kritische Szenarien.
   - Integration von Tests in den Entwicklungsprozess: Förderung von Continuous Integration und Continuous Testing Praktiken.

Schlussfolgerungen zum Stand der Technik:

- Notwendigkeit integrierter Ansätze: Die effektive Qualitätssicherung in Embedded Systems erfordert integrierte Teststrategien, die Hardware und Software gemeinsam betrachten.
- Frühes Testen: Shift-Left-Prinzipien, bei denen Tests früh im Entwicklungszyklus beginnen, sind entscheidend, um Fehler frühzeitig zu erkennen und zu beheben.
- Kombination von Methoden: Eine Kombination aus verschiedenen Testmethoden (statisch, dynamisch, formal) bietet die besten Ergebnisse.
- Ressourcenbewusstsein: Tests müssen unter Berücksichtigung der Ressourcenbeschränkungen von Embedded Systems gestaltet werden.