5 Fehler

die Dein Produkt unzuverlässig und instabil machen

und wie Du sie vermeidest

8tronix
make a long story smart

5 Fehler die Dein Produkt unzuverlässig und instabil machen und wie du sie vermeidest

„Eines unserer Bauteile wurde abgekündigt und die Mitarbeitenden, die das System entwickelt haben, arbeiten nicht mehr bei uns.“

„Unser Produkt stürzt immer wieder ab, wir bekommen aber nicht heraus, ob es an der Hard- oder Software liegt.“

So oder ähnlich sind die Aussagen vieler unserer neuen Kund:innen und häufig fehlen einfach die Zeit oder die Ressourcen, diese Probleme zu lösen.

In dieser PDF zeigen wir Dir fünf häufige Fehler, mit deren Vermeidung Du Dein System robust und zuverlässig entwicklen kannst.

Darin steckt unser Wissen aus über 70 Projekten – direkt von unseren Hard- und Softwareexpert:innen.

Fehler 1: Schlechte Anforderungen

Einer der kritischsten Fehler bei der Entwicklung eingebetteter Systeme sind schlecht definierte Anforderungen.

„Wir haben bislang nur im internen Team garbeitet, da brauchten wir keine Anforderungen."

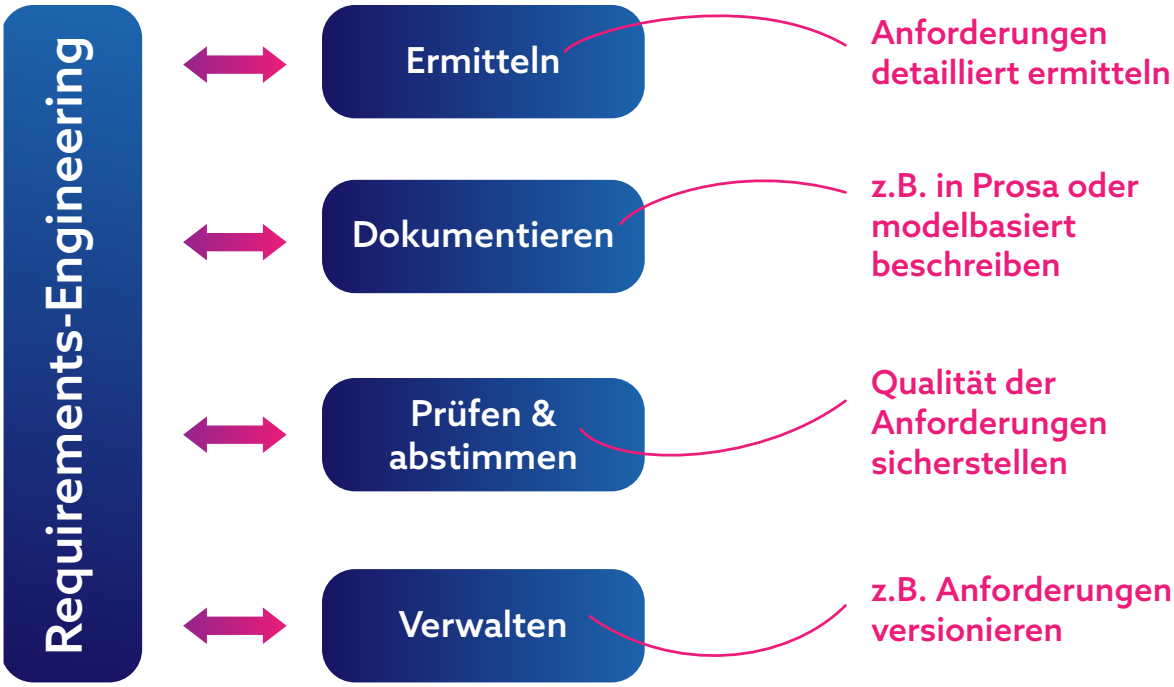
Falsch! Mit gut definierten Anforderungen sicherst Du Dir klare Erwartungen, minimierst Missverständnisse innerhalb Deines Teams und ersparst Dir damit Fehler im finalen System.

Gut herausgearbeitete Anforderungen führen zu einer höheren Systemqualität, verkürzen Entwicklungszeiten und reduzieren Kosten durch weniger Nacharbeit.

Um diesen Fehler zu vermeiden, solltest Du also einen systematischen Prozess der Ermittlung, Dokumentation, Prüfung, Abstimmung sowie das Verwalten der Anforderungen befolgen.

„Muss ich das wirklich machen? Requirement Engineering ist so mühsam!"

Ja! Requirement Engineering ist eine Fleißaufgabe und kann anstrengend sein. Doch wir garantieren Dir, dass es sich lohnt. In den letzten Jahren mussten wir immer wieder feststellen, dass Projekte ohne Requirment Engineering lange nicht so erfolgreich waren, wie solche, bei denen die Anforderungen detailliert geschrieben und gepflegt wurden. Hier gibt es keine Abkürzung – also ran an die Arbeit, wir unterstützen dabei auch gerne.



Fehler 2: Unzureichende Tests

Ein weiterer großer Fehler in der Entwicklung eingebetteter Systeme besteht darin, das System gar nicht oder erst am Ende der Entwicklungsphase zu testen.

Beim Testen wird überprüft und validiert, ob das System die Anforderungen erfüllt und wie erwartet funktioniert.

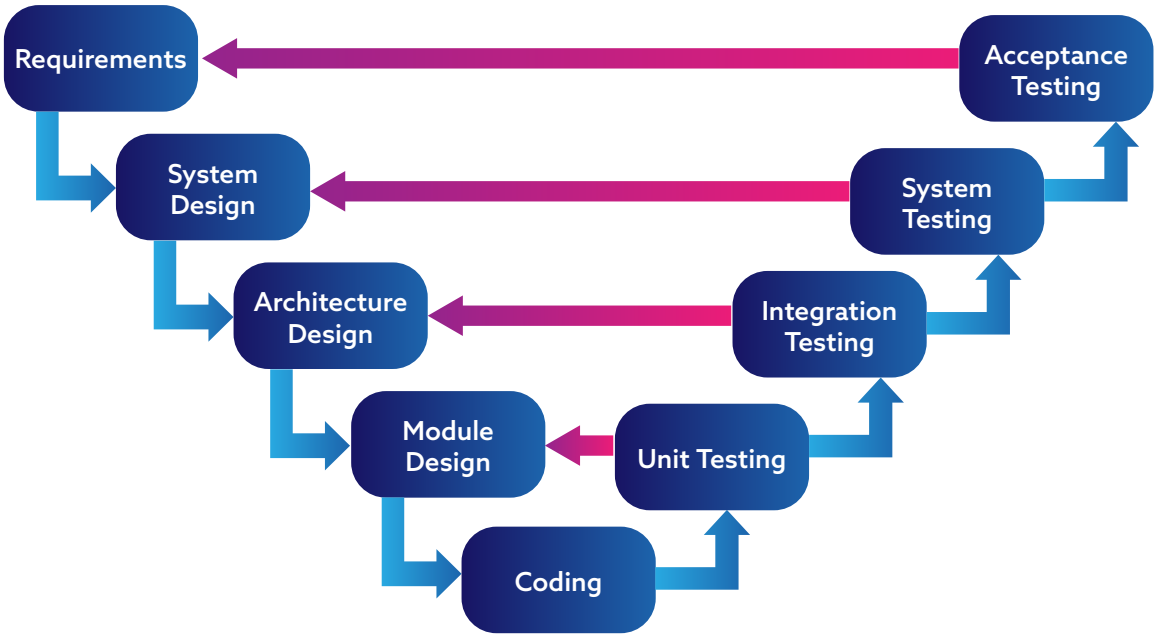
Wie Du merkst, musst Du Fehler 1 berücksichtigen, um Fehler 2 vermeiden zu können.

Keine oder unzureichende Tests können zu unentdeckten Fehlern, Defekten oder Schwachstellen führen, die die Funktionalität, Zuverlässigkeit oder Sicherheit des Systems beeinträchtigen können.

Kurzum: Der Ruf Deines Produktes steht damit auf dem Spiel.

Um diesen Fehler zu vermeiden, solltest Du eine umfassende Teststrategie planen und ausführen, die die Aspekte wie Funktionalität, Leistung und Robustheit Deines Systems abdeckt. Dafür kannst Du Tools und Methoden verwenden, wie z.B. Jenkins, diverse Debugging-Tools und Testsoftware wie Ceedling oder CMock.

All diese Dinge werden Dir beim Schreiben und Durchführen von Unit-, Intregration- und Systemstests während der Entwicklungsphase helfen.



Fehler 3: Schlechte Programmierstandards

Die Qualität und Wartbarkeit Deiner Embedded Software wird erheblich von Deinen Programmierstandards beeinträchtigt.

Zu den häufigsten Problemen gehören:

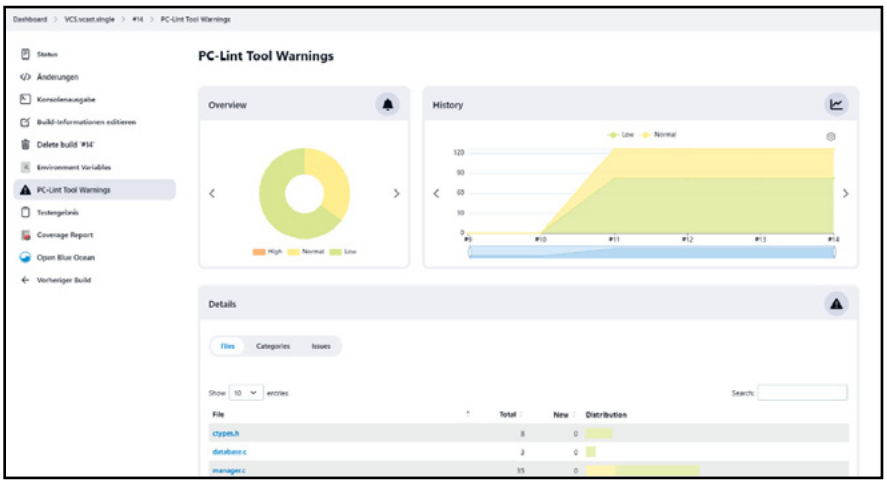
- ✗ Inkonsistente oder unklare Namenskonventionen
- ✗ Komplexer oder redundanter Code
- ✗ Verletzung von Codierstandards oder -richtlinien
- ✗ Hartcodierte Werte ohne
- ✗ Erklärung (Magic Numbers)

Solche Probleme kannst Du verhindern, indem Du Dir angewöhnst etablierte Programmierstandards konstant zu verwenden.

Code-Reviews oder Pair Programming sind gute Methoden, um die Qualität und Deine eigenen Programmierskills auf das nächste Level zu bringen. Zusätzlich gibt es Tools für statische Codeanalysen, die ein absoluter Game Changer für Deine Programmiergewohnehiten sind.

Codeanalysen weisen Dich auf potenzielle Fehler und Codeschwächen hin, noch bevor der Code ausgeführt worden ist.

Das Tool PC-lint bietet eine umfassende und detaillierte Analyse, die für komplexe Projekte und strenge Codierungsstandards wie z.B. MISRA C: 2012 nützlich ist. Die kostenfreie Option Cppcheck lässt sich leicht in verschiedene Entwicklungsumgebungen integrieren und ist besonders für kleinere bis mittelgroße Projekte besonders gut geeignet.



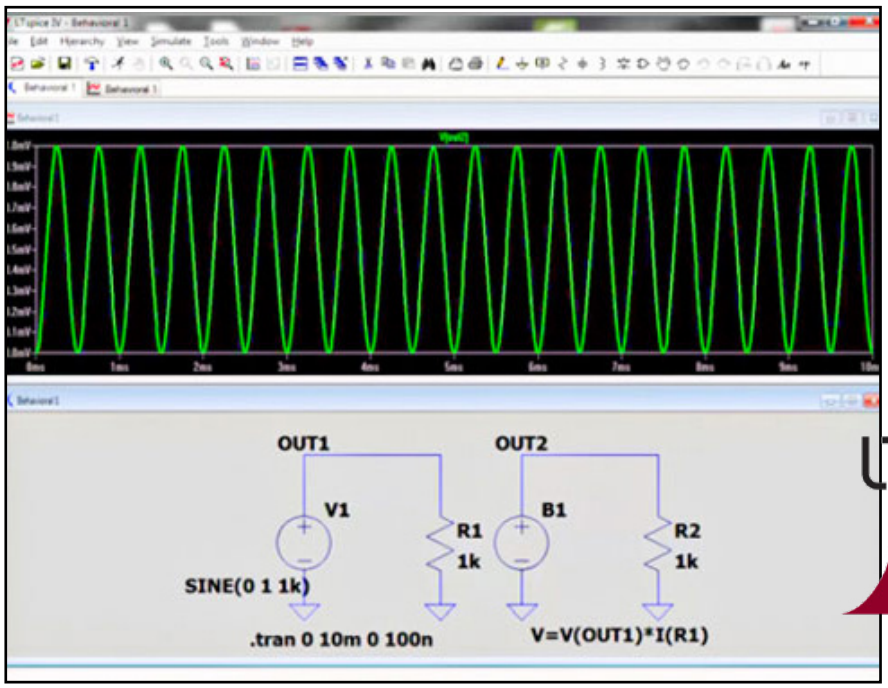
Fehler 4: Schaltungen ohne Simulation entwerfen

Endlich hast Du Deine Hardware fertig entwickelt, doch bei der Inbetriebnahme erkennst Du, dass gewisse Schaltungen nicht so funktionieren, wie Du dir das vorgestellt hast. Klar baust Du Deine Schaltungen vorher auf einem Breadboard auf und testest Sie, aber machst Du das auch nach kleinen Änderungen wieder und wieder und wieder?

An dieser Stelle kann ich Dir nur Simulationstools wie z.B. LTspice empfehlen.

Mit LTspice kannst Du Deine Schaltungen analysieren, bevor Du in die physische Umsetzung gehst. Das Tool bietet detaillierte Einblicke in Spannung, Strom und Frequenzverhalten, ermöglicht eine Optimierung des Designs und minimiert das Risiko von Fehlfunktionen.

Es versteht sich von selbst, dass Du dadurch nicht nur Zeit und Kosten sparst, sondern auch die Qualität deiner Schaltung verbesserst.



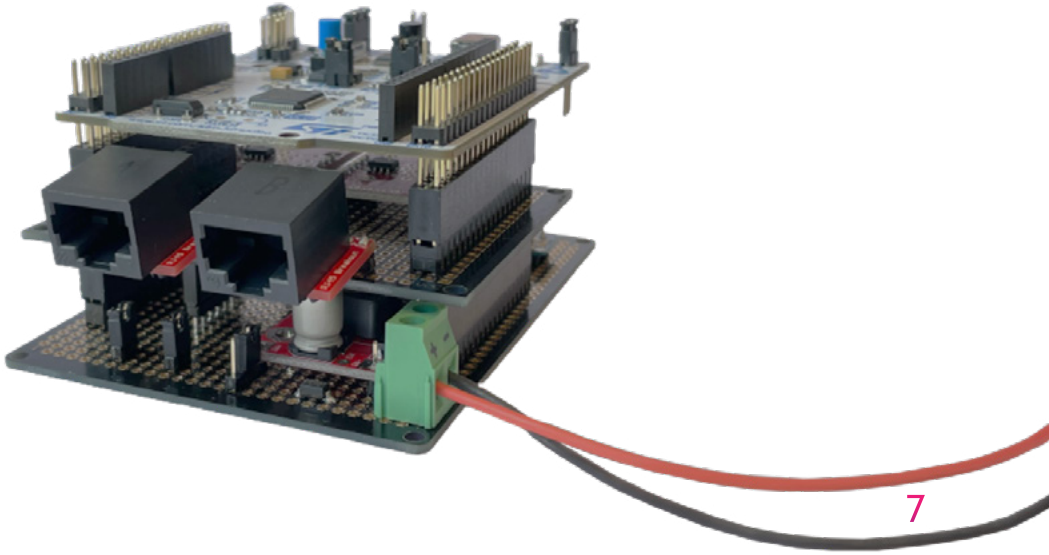
Fehler 5: Die Hardware in einem Guss entwickeln

Ein monolithisches Design kann zu Herausforderungen bei Fehlerdiagnosen, Anpassungen und der Integration neuer Funktionen führen. Wer versucht, im ersten Schritt alles in einem Guss zu entwickeln, geht einen überaus steinigen Weg.

Der Weg muss aber gar nicht steinig sein, wenn Du hoch stapelst.

In der Entwicklungsphase von Embedded Hardware erweist sich das Stapeln von Platinen als überaus vorteilhaft. Dadurch kannst Du Schaltungen schrittweise entwickeln und sukzessive zusammenzufügen, was zu einem flexibleren und effizienteren Entwicklungsprozess führt.

- ✓ Platzersparnis: Kompakte Designs durch gestapelte Platinen nutzen den verfügbaren Raum optimal, was besonders für IoT-Geräte und tragbare Technologien vorteilhaft ist.
- ✓ Robustheit: Die Verwendung einer fest verdrahteten oder eigens designeten Platine erhöht die Robustheit des Prototyps im Vergleich zu Jumper- oder Breadboard-Lösungen.
- ✓ Modularität: Einzelne Boards können unabhängig getestet, angepasst und ersetzt werden, was Zeit und Kosten spart, da Probleme gezielt behoben werden können.
- ✓ Skalierbarkeit: Durch das Hinzufügen zusätzlicher Boards lässt sich das System einfach erweitern und an neue Anforderungen anpassen.
- ✓ Kosteneffizienz: Langfristig werden Material- und Produktionskosten gesenkt, da die Wiederverwendbarkeit von Modulen und reduzierte Neugestaltungen wirtschaftliche Vorteile bieten.



BONUSFEHLER: Immer Deine eigene Lösung programmieren

Weil dieser Tipp einfach so unfassbar wichtig ist und er uns persönlich am Herzen liegt, möchten wir Dir noch einen kleinen Bonus an die Hand geben.

Ein einfach vermeidbarer Fehler ist es, eine eigene unstruktierte Lösung zu entwickeln. Programmieraufgaben und -lösungen sind repetativ und ähneln sich. Du brauchst das Rad nicht jedes Mal neu zu erfinden. Verwende Design-Patterns.

Was sind Design-Patterns?

Design-Patterns sind allgemein wiederverwendbare Lösungen für häufig auftretende Probleme bei der Softwareentwicklung und -gestaltung. Sie bieten eine Vorlage zur Lösung bestimmter Problemtypen und helfen Entwickler:innen, den Code so zu strukturieren, dass er modularer, wartungsfreundlicher und skalierbarer wird.

Vorteile von Design-Patterns:

- ✓ Klarheit und Struktur: Patterns bieten eine klare Struktur und wiederverwendbare Lösungen, die den Code lesbarer und wartbarer machen.
- ✓ Konsistenz: Durch Design-Patterns wird eine konsistente Herangehensweise in der gesamten Codebasis gewährleistet. Dies verbessert die Zusammenarbeit im Team und reduziert die Lernkurve für neue Entwickler:innen.
- ✓ Wiederverwendbarkeit: Design-Patterns fördern die Wiederverwendbarkeit von Code, da bewährte Lösungen in verschiedenen Projekten oder Modulen wieder eingesetzt werden können.
- ✓ Flexibilität: Design-Patterns bieten flexible Lösungen, die leicht an neue Anforderungen angepasst werden können.
- ✓ Schnellere Problemlösung: Mit Design-Patterns können Entwickler:innen auf bewährte Lösungen zurückgreifen, die bereits getestet und optimiert sind. Dies beschleunigt den Entwicklungsprozess und reduziert die Anzahl von Fehlern.

Wenn Du bisher nicht regelmäßig Design-Patterns verwendet hast, dann wird dieser Tipp die Art und Weise, wie Du programmierst, völlig verändern.

Jetzt bist Du dran!

Wenn Du ein robustes und zuverlässiges Embedded System haben willst, dann

buche einen kostenlosen Termin!

Solch ein unverbindlicher Ersttermin war bei vielen unserer Kund:innen der Anfang einer großartigen Zusammenarbeit mit dem Erfolg einer individuell zugeschnittenen, zuverlässigen und robusten Lösung.

„Bei der Entwicklung des levo hooks haben wir eng mit der 8tronix GmbH zusammengearbeitet. Wir sind mit dem Engagement, der Umsetzung sowie mit den Produktergebnissen mehr als zufrieden."

Johannes Knafl
pewag austria GmbH

Christian Kluge
D+H Mechatronik AG

„Wir haben als Firma D+H Mechatronik mit 8tronix zusammen ein System aus Hardware und Software entwickelt. Eine strukturierte Vorgehensweise und frühe Demoaufbauten schafften qualitativ hochwertige und abgesicherte Resultate"

„Wir haben ein sehr umfangreiches Projekt mit 8tronix erfolgreich zum Abschluss gebracht und sind mit ihrer Professionalität und der Qualität ihrer Arbeit äußerst zufrieden. Die Zusammenarbeit war reibungslos und effizient. Wir können 8tronix uneingeschränkt weiterempfehlen."

Oliver Jöhnck
paratus electronic GmbH

Sven Höppner
Werner Wirth GmbH

„Mehrere Projekte mit 8tronix durchgezogen. Das 8tronix-Team arbeitet kompetent, transparent und kundenorientiert. Für entsprechende Projekte ist 8tronix immer im Anfrageverteiler."


8tronix
make a long story smart