ABER WIE?

AGILE HARDWARE

EIN ERFAHRUNGSBERICHT

Das agile Manifest hat unser Vorgehen bei der Entwicklung von Software nachhaltig zum Besseren verändert. Agil durchgeführte Software-Projekte kommen objektiv schneller zum Ziel und schränken das Risiko von Budgetüberschreitungen stark ein. Was für den überwiegenden Anteil unserer heutigen Software gilt, stimmt für den Bereich der hardwarenahen Software bestenfalls bedingt.

Aber warum ist das so? Als Hauptgrund wird meist die enge Verzahnung mit der parallel zu entwickelnden Hardware genannt. Diese wird traditionell sequenziell entwickelt, das Erzeugen von Prototypen kostet schließlich viel Zeit und Geld. Auch kann eine einmal hergestellte Platine nicht in Inkrementen erweitert oder in Iterationen verfeinert werden.

Doch ist das wirklich so? Würde man nicht auch hier von kürzeren Zyklen profitieren? Könnte Hardware agil entwickelt werden, stünde einer agilen Entwicklung von Embedded Software-Systemen ebenfalls nichts mehr im Wege.

Fallbeispiel: Entwicklung eines medizinischen Messgeräts

Um die Praxistauglichkeit von Ideen überprüfen zu können, muss man diese in einem konkreten Projektkontext erproben. Unser Fallbeispiel: die Entwicklung eines neuartigen medizinischen Messgerätes. Ein vergleichbares, aber zu diesem Zeitpunkt bereits abgeschlossenes Projekt diente als Referenz. Dort waren zunächst Lastenheft und Pflichtenheft ausgearbeitet worden. Anschließend entwickelte ein erfahrenes Team die Hardware und übergab sie für die Integration an den Auftraggeber. Bis zu diesem Zeitpunkt waren bereits sechs Monate vergangen, bis zum ersten erfolgreichen funktionalen Test sollte etwas mehr als ein Jahr vergehen.

ANFORDERUNGEN ÄNDERN SICH

Während einer so langen Zeit hatten sich die zu Beginn niedergeschriebenen Anforderungen mehrmals geändert. Einige Anforderungen stellten sich als übertrieben heraus, andere entfielen und neue kamen hinzu. Die hierdurch erzeugte „Unruhe“ schlug sich in etlichen Fehlerbildern nieder, die das Team erst durch sehr zeitaufwändige weitere Revisionen unter Kontrolle bringen konnte.

Üblicherweise versucht man, alle an die Hardware gestellten Anforderungen mit möglichst wenigen PCBs (Printed Circuit Boards) umzusetzen. So ist es auch in unserem Referenzprojekt geschehen. Für die spätere Herstellung des Produktes ergeben sich daraus viele Vorteile, zum Beispiel niedrige Herstellungskosten. Nur müssen dafür schon zu Beginn alle Anforderungen bekannt sein. Ist dies nicht der Fall, wird man versuchen, mehrere Optionen oder doch zumindest die Option mit den meisten Funktionen auf dem PCB unterzubringen. Alternativ wird das System für den größtmöglichen Einsatzbereich ausgelegt. Eine Temperaturkontrolle wird dann zum Beispiel mit einem Messbereich von 0°C bis 120°C und einer Auflösung von 0,1°C spezifiziert. Werden im späteren Produkt tatsächlich aber nur ein Bereich von 30°C bis 50°C und eine Auflösung von 0,5°C benötigt, würde man im Kontext agiler Software-Entwicklung von „Gold-Plating“, also „Waste“ sprechen. Bevor wir zeigen können, wie sich dies

Könnte Hardware agil entwickelt werden, stünde einer agilen Entwicklung von Embedded Software-Systemen ebenfalls nichts mehr im Wege.

vermeiden lässt, ist es notwendig die einzelnen Arbeitsschritte der Hardware-Entwicklung kurz vorzustellen.

AUFWÄNDE IN DER HARDWARE-ENTWICKLUNG

Die Entwicklung eines PCB durchläuft stets eine festgelegte Abfolge von Phasen. Als Erstes wird ein Entwurf erstellt, das sogenannte Blockschaltbild. Darin werden die wesentlichen Leistungsmerkmale modelliert. Anschließend werden Funktionsblöcke, wie etwa Spannungsversorgung, analoges Front-End oder Motoransteuerung, im Schaltplan implementiert. Der Schaltplan ist eine Abstraktion, die benötigte elektronische Bauteile und ihre Verbindungen untereinander beschreibt. Danach geht es an die Erstellung des Layouts. Grob kann dieser Arbeitsschritt in Bauteilplatzierung und Entflechtung unterteilt werden. Letzteres ist das „Ziehen" von Kupferbahnen zwischen den Bauteilen. Aus dem Layout werden anschließend die Fertigungsdaten der Platine erstellt und in die Fertigung übergeben. Im nächsten Arbeitsschritt wird die Platine bestückt, d.h. die Bauteile werden an den vorgegebenen Positionen aufgelötet. Das fertig bestückte PCB durchläuft abschließend eine visuelle und elektrische Kontrolle.
Interessanterweise sind die Aufwände für Fertigung und Bestückung nahezu unabhängig von der Designkomplexität. Die Aufwände für Schaltplanentwurf und Layout reagieren allerdings umso empfindlicher. Leider gibt es keine allgemein gültige Kennzahl für die Bewertung der Komplexität eines Designs. Die Zahl der Bauteile, der Anschlusspins oder der elektrischen Verbindungen sind in unseren Augen aber hilfreiche Größen. Die Auswertung dieser Zahlen für uns bekannterw Designs ergab, dass der Aufwand für die Entflechtung nichtlinear wächst. Und er wächst umso stärker, je kleiner die zur Verfügung stehende Fläche ist. Welche Rückschlüsse lassen sich daraus bereits ableiten? So wird z.B. offensichtlich, dass eine Überspezifikation das Aufwandsproblem verschärft, weil mehr Bauteile und Netze berücksichtigt werden müssen, als am Ende wirklich gebraucht werden. Die Reduktion von Komplexität birgt daher Potenzial.

Design	# Parts	# Nets	# Pins
Very complex	>1000	>1000	>2000
Complex	750	544	2250
Moderate	315	383	1162
Simple	121	101	350
Very Simple	<100	<70	<200

Tabelle 1: Typische Größen für die Anzahl von Bauteilen, elektrischen Verbindungen und Anzahl zu verbindender Pins der Bauteile. Je größer die Zahlen werden, umso mehr Aufwand, vor allem für das Leiterplattenlayout, entsteht.

ITERATIVE HARDWARE-ENTWICKLUNG

In der agilen Software-Entwicklung hat die Bereitstellung von (Teil-) Funktionalität zum frühestmöglichen Zeitpunkt einen hohen Stellenwert. Dies darf jedoch nicht zu Lasten der Qualität gehen – daher die Beschränkung auf wichtige und klar verstandene Anforderungen. Übertragen auf ein Hardware-Projekt könnte also zunächst nur eine einfache Ansteuerung für eine Pumpe realisiert werden. Zu diesem Zeitpunkt noch wenig spezifische Anforderungen, wie beispielsweise die Ansteuerung einer Lichtquelle oder einer Heizung, werden in einer nachfolgenden Iteration realisiert. In der Praxis resultiert ein solches Vorgehen in mehreren kleinen statt einem großen multifunktionalen PCB. In der Software entspricht das dem Auflösen einer „Monster"-Klasse in viele kleinere mit jeweils einer einzigen Verantwortlichkeit. Während die entworfenen Software-Klassen bereits in einem Software-Produkt genutzt werden könnten, wird hingegen niemand ein Produkt aus zehn oder

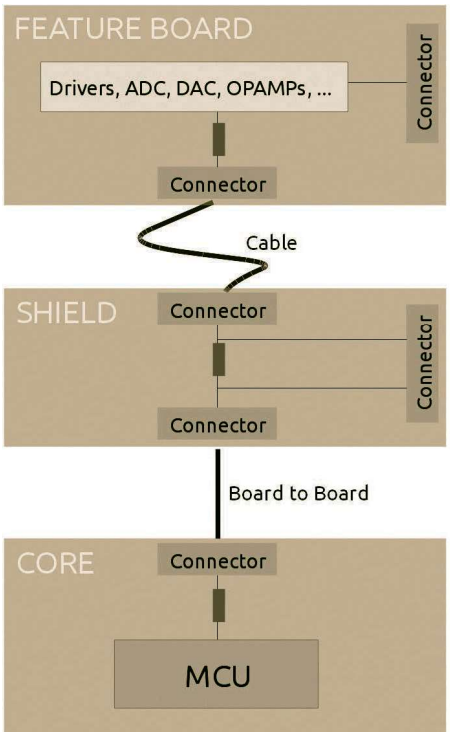

Abbildung 1: Rig Design: Mikrocontroller und Module sind durch ein Shield miteinander verbunden und können unabhängig voneinander genutzt werden.

mehr Platinen als final akzeptieren. Es bedarf also einer Methode, die aus den vielen kleinen PCBs wieder ein einziges macht. **Dafür haben wir vier Techniken definiert:**

Rig Design

Die Menge aller für ein Projekt benötigten PCBs bezeichnen wir als Rig. Im Rig dürfen/sollen in der Entwicklung PCBs ausgetauscht und verändert werden. Ein Rahmenwerk für elektrische und mechanische Schnittstellen sowie weitere Design-Richtlinien bilden das Rig Design. In unserem Projekt wurden die PCBs in Core, Shield und Module unterschieden. Das Core beherbergte die MCU sowie eine USB-Schnittstelle und konnte ohne weitere Hilfsmittel direkt mit einem PC in Betrieb genommen werden. Wesentliche Teile der Produkt-Firmware konnten damit bereits erstellt werden. Ohne starre Festlegung auf die Peripherie

stellt das Core I/O-Pins der MCU auf einem Board-to-Board Verbinder zur Verfügung. Dessen Gegenseite ist das Shield-PCB. Dieses vermittelt zwischen dem Core und den Modulen. Ein Modul-PCB realisiert dann z.B. die Ansteuerung eines Motors. Modul-PCBs dürfen dabei stets nur mit dem Shield, nicht aber untereinander verbunden sein.

Clean Code

In der Hardware-Entwicklung geht viel Zeit durch mangelhafte „Styleguides" verloren. Die Folge sind schlecht lesbare Schaltpläne, die (wenn überhaupt) nur vom ursprünglichen Autor ohne Gefahr geändert werden können. In der Software-Entwicklung gibt es ähnliche Probleme und die dort gefundenen Lösungen sind auf die Hardware-Entwicklung übertragbar. Auch wenn in Hardware nicht wirklich Code geschrieben wird, so bezeichnet der Begriff „Clean Code" doch am besten, worum es geht: les- und wartbare Designs. In unserem Projekt bestanden die ersten Schritte in einem verbindlichen Styleguide und der Einführung eines Versionskontrollsystems (Git).

Automation

Die Verwendung eines Versionskontrollsystems hat noch einen weiteren Vorteil: Bei der Qualitätssicherung kann ein Continuous-Integration Server helfen. Designchecks werden automatisiert und Probleme oder Regelverstöße teilweise deutlich früher erkannt. Insbesondere ist es sinnvoll, die Erzeugung der Fertigungsdaten einem reproduzierbaren CI-Job zu überlassen.

Rig Reduction

Bliebe am Ende noch zu klären, wie aus dem Rig wieder ein einzelnes Produkt-PCB wird, ohne nochmals von vorne anfangen zu müssen. Die technischen Details aussparend, gilt es zunächst aus den vielen Einzelschaltplänen einen einzigen zu erstellen. Dabei werden nicht mehr nötige Bauteile entfernt (zum Beispiel Steckverbinder zwischen den PCBs). Die Einhaltung der Rig-Design Konventionen vorausgesetzt, geschieht das automatisiert; es entstehen also keine zusätzlichen Aufwände. Für diesen Schaltplan wird ein neues Layout benötigt, das der Komplexität entsprechend mehr Zeit in Anspruch nehmen wird. Allerdings wird die Schaltung ihre Aufgabe erfüllen und die Software binärkompatibel auf der finalen Hardware ausgeführt werden können.

FAZIT

Die vorgestellte Methodik ermöglichte uns im Beispielprojekt neue Hardware-Funktionalität in zwei bis drei Wochen umzusetzen. Nach weiteren zwei Wochen konnte diese durch ebenfalls agil entwickelte Software angesprochen werden. Damit gelang es dem Team neue Features schnell und produktnah zu realisieren. Die Ableitung der Produktplatine nahm am Ende des Projektes nur einmal sechs Wochen in Anspruch. Darüber hinaus konnten die entstandenen Rig-PCBs in Nachfolgeprojekten wiederverwendet bzw. einfach angepasst werden.

Dem hehren Ziel einer agilen Hardware-Entwicklung sind wir damit ein gutes Stück nähergekommen.

Dr. Tobias Kästner ist promovierter Physiker und hat mehrjährige Erfahrung in der Entwicklung vernetzter Medizingeräte. Als Expert Engineer IoT kümmert er sich seit 2016 bei der Method Park Engineering GmbH darum, das Internet of Medical Things Wirklichkeit werden zu lassen.

Mario Blunk, Inhaber und Geschäftsführer der Firma Blunk electronic, hat 25 Jahre Erfahrung in der Entwicklung elektronischer Baugruppen und Systeme. Als Selbstständiger legt er den Schwerpunkt auf agile Hardware-Entwicklung, Test und Fertigung. Die dabei gesammelten Erfahrungen bietet er darüber hinaus als Berater und Trainer an. Sein besonderes Interesse gilt sicherheitsrelevanten Anwendungen in den Bereichen Medical, Transportation und Avionics.