WHITEPAPER

Status Quo Agile Hardware-Entwicklung

Viele Hardware-Entwicklungen folgen noch immer linearen Prozessen – in einer Welt, die längst vernetzt, digital und schnell ist. Dieses Whitepaper zeigt, warum klassische Vorgehensweisen an ihre Grenzen stoßen und wie agile Prinzipien helfen, schneller zu lernen, besser zu liefern und näher am Kunden zu entwickeln.

Wir analysieren typische Hindernisse – kulturell, organisatorisch und technisch – und geben konkrete Impulse, wie Unternehmen agil starten können: mit gemischten Teams, iterativer Planung, modularer Architektur und mehr Transparenz.

Unser Fazit: Agile Hardware ist kein Nice-to-have – sondern ein entscheidender Erfolgsfaktor für zukunftsfähige Produktentwicklung.

WHITEPAPER

Die Autoren

Christoph Schmiedinger
Executive Consultant bei borisgloger consulting, einem der führenden Beratungsunternehmen Deutschlands für alles rund um Agilität.

christoph.schmiedinger@borisgloger.com

Dr. Tobias Kästner
Solution Architect Medical IoT bei inovex GmbH, einem führenden Dienstleister für die digitale Transformation mit Full-Stack Technologiekompetenz von Cloud bis Deep Embedded.

tobias.kaestner@inovex.de

Gregor Groß
Geschäftsführer alpha-board gmbh, einem Berliner Dienstleister für agile Hardware-Entwicklung, PCB-Design und Fertigungsservice.

gregor.gross@alpha-board.de

WHITEPAPER

Agile Hardware-Entwicklung

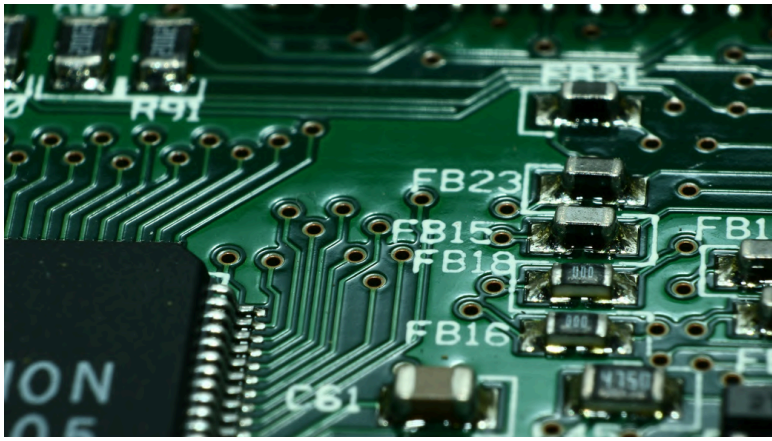
Autoren: Christoph Schmiedinger, Dr. Tobias Kästner und Gregor Groß

Intro

Agile Arbeitsweisen waren die letzten Jahre in aller Munde, ob positiv oder negativ. Anfangs stark aus der Softwareentwicklung kommend, wurden agile Werte & Prinzipien zunehmend auch in anderen Bereichen der Produktentwicklung eingesetzt, z.B. in der Entwicklung elektronischer Hardware. Damit meinen wir die Entwicklung von Mechanik, Mechatronik, Elektronik und Software für elektronische Produkte aller Art, von der intelligenten Waschmaschine bis hin zur hochkomplexen Medizintechnik für den Operationsaal. In diesem Artikel fokussieren wir dabei auf die integrierte Entwicklung von Hard- und Software und weniger auf Mechanik und Gehäusedesign.

Doch trotz dieses Überschwappen der agilen Trends müssen wir beim Blick in viele Unternehmen feststellen, dass wir in vielen Hardware-Produktentwicklungen weiterhin in alten Verhaltensweisen und verstaubten Entwicklungsprozessen festhängen. Das ist insbesondere problematisch, als die schnelle und zuverlässige Entwicklung innovativer, hochdigitalisierter und intelligenter Produkte einer der Eckpfeiler der deutschen Wirtschaft sein sollte.

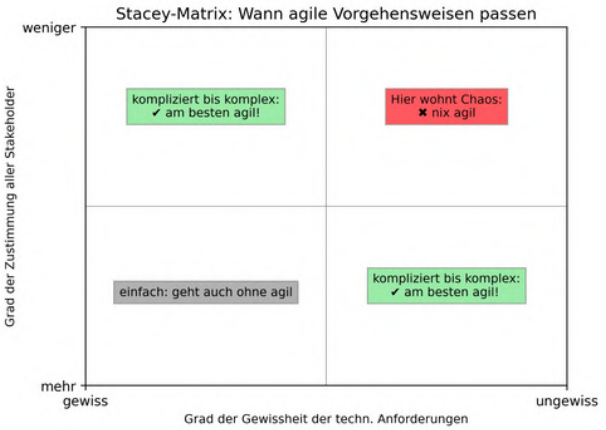
Grund genug für uns Autoren, den Status Quo zu analysieren und Wege aus dem Dilemma aufzuzeigen. Wer wir sind: ein Trio mit gemeinsam 60+ Jahren Erfahrung in der Entwicklung innovativer mechatronischer Produkte und Vertreter dreier unterschiedlicher Perspektiven: der technologischen, der unternehmerischen und der prozessual-organisatorischen. Wir waren mitten drin in zahlreichen solchen Produktentwicklungsprojekten, wir haben sie als Consultants begleitet, haben einen Teil selbst entwickelt und haben die Projekte dabei von innen und außen erlebt.



Agilität in Hardware - braucht es das überhaupt?

Während sich Agilität ab 2008 in der Softwareindustrie rasend schnell verbreitete, ging die Adoption in der Hardware-Entwicklung deutlich gemächlicher voran. Selbst 2025 nutzt nur ein kleiner Bruchteil der Hardware-Teams agile Denkmodelle als Basis für ihre Arbeitsweise oder auch nur ein agiles Tool, um die eigene Arbeit zu organisieren. Viele solcher Teams lehnen agile Vorgehensweisen quasi grundsätzlich ab: es käme aus der Software-Industrie, wo alles soft sei, Entwicklungszyklen seien andere, warum ändern, was doch gut lief?

In einer Welt, in der Hardware für sich (fast) alleine stand, haben die lange genutzten Arbeitsweisen ihren Job gemacht. Doch kommt heute selten eine Hardware ohne zugehörige Software aus. Heute neu entwickelte Hardware meldet im Minimum gesammelte Daten an andere Hard- und Software, mittlerweile in der Regel bis in die Cloud. Aber was bringt Agilität in der Software-Entwicklung, wenn die dazugehörige Hardware erst Monate später verfügbar ist und dann kaum noch angepasst werden kann? Ein großer Teil der Agilität geht nämlich verloren, wenn Hard- und Software sequentiell entwickelt werden.



Stacey-Matrix

Wie in dieser Abbildung der Stacey-Matrix zu sehen (nach Ralph Douglas Stacey), eignen sich insbesondere zwei Situationen für agile Vorgehensweisen:

- wenig Gewissheit über Zustimmung aller betroffenen Stakeholder (z.B. Endanwender) und damit über deren Bedürfnisse und Wünsche
- technischen Anforderungen sind ebenso wenig klar, z.B. weil der technische Lösungsraum für die optimale Lösung exploriert werden muss

Der einfache Fall, wo alle Stakeholder der Idee zustimmen und alle technischen Anforderungen bekannt sind, ist ohne Agilität lösbar. Einfach machen ist die Devise! Die beiden Quadranten in grün, hohe Ungewissheit technischer Anforderungen oder kaum vorhandene Zustimmung aller Stakeholder, sind am besten agil anzugehen.
Warum ist agile genau dafür gut geeignet? Weil agile Vorgehensweisen iterativ sind, um schnell Neues zu bauen und am besten an Endanwender:innen zu testen, und daraus dann zu lernen:

"Agile is a way of working that harnesses change as a competitive advantage, rather than a liability. It does this by operating with rapid-learning and fast-decision cycles, making use of networks of teams with a people-centered culture. Agile can be used in hardware product development as well as in system development, which combines hardware and (embedded) software."

pg 2 of "It's coming home: the return of agile hardware product development", McKinsey

Das im Zitat genannte System Development, integriertes Entwickeln von Hard- und Software, ist heute ob der Komplexität vieler Produkte kaum mehr wegzudenken. Logischerweise sollten dann beide Aufgabenbereiche weitestgehend parallel statt sequentiell abgearbeitet werden. Dafür sind einige Umstellungen im Ablauf der Hardware-Entwicklung nötig, aber vor allem Transparenz und Austausch zwischen Hard- und Software-Leuten.

Technische Möglichkeiten für Hard- und Software entwickeln sich rasend schnell weiter, ebenso die Bedürfnisse von Kunden und Endanwender:innen. Deren Ansprüche haben sich verändert, zwischen Hardware und Software wird kaum noch unterschieden: Kunden kaufen ein Produkt, welches einen Mehrwert bietet. Auch der Wettbewerb ist auf Grund der Globalisierung härter. Schnelligkeit zählt, denn ein Alleinstellungsmerkmal kann große Umsätze bringen, bevor der Wettbewerb überhaupt auf dem Markt ist.

Das alles führt zu unserer Conclusio, dass agile Arbeitsweisen in der Hardware-Entwicklung mehr als nice to have sind. Ist es heute noch ein Wettbewerbsvorteil, wird es in Zukunft sogar für das Bestehen am Markt immer essenzieller sein: was heute einzelne Kunden begeistert, wird künftig Basismerkmal von Hardware-Entwicklung sein (vgl. KANO-Modell).

Kuriosität am Rande: Agilität hatte seine Ursprünge übrigens in physischer Produkt-entwicklung – in einer Zeit, in der Software kaum Anteil an modernen innovativen Produkten hatte. Gerade das agile Framework Scrum baut in großen Teilen auf den Erkenntnissen einer wissenschaftlichen Studie von Nonaka und Takeuchi aus 1986 auf („The New New Product Development Game " im Harvard Business Review). Die in dieser Studie analysierten Produktentwicklungen waren allesamt physische Produkte, unter anderem ein Auto von Honda, eine Fotokamera von Canon oder ein Drucker von Fuji-Xerox. Auf Grund der Herausforderungen in großen Softwareentwicklungsprojekten in den 90er Jahren wurde das Framework dann nichtsdestotrotz zuallererst stark in diesem Kontext eingesetzt.

Teil 1 - Der Status Quo: Wo stehen wir in 2025?

Blicken wir zunächst auf die Hardware-Entwicklung eines typischen mittelständischen deutschen Unternehmens, wie wir es im Arbeitsalltag hundertfach erlebt haben. Diese ist von allerlei Herausforderungen geprägt - viele davon leider auch hausgemacht. Die folgenden Punkte treffen wir dabei immer wieder an:

1. Arbeitsteilung nach Fachlichkeit
2. Lineares Hardware-First-Entwicklungsmodell
3. Proprietäre und preisintensive Werkzeuge
4. Kultivierung hochspezialisierter Experten
5. Mangelnde Unterstützung an den Schnittstellen
6. Kostenkontrolle als Maß aller Dinge
7. Werksverträge/Pflichten- und Lastenhefte mit voll spezifiziertem Umfang

Arbeitsteilung nach Fachlichkeit

Wie sieht die Zusammenarbeit zwischen Hard- und Software-Entwicklern aus? Zumeist werden Hard- und Software noch immer in getrennten Teams bearbeitet. Wissensaustausch findet nur sporadisch durch direkte Kommunikation statt, falls sich dafür jemand aus den dedizierten Teams bereit erklärt. Zudem arbeiten Hardware- und Software-Entwickler nicht selten auch noch in getrennten Werksverträgen oder basierend auf verschiedenen Pflichtenheften. Das führt oft zu folgenden Problemen:

- Fehlende Abstimmung zwischen Pflichtenheften
- Software-Entwickler:innen legen Parameter und Schnittstellen spät fest
- Für Hardware-Prototypen liegen anfangs keine Software-Tests vor, die Erstellung von geeigneter Test-Software stellt eine (oftmals ungeplante) Zusatzbelastung für das SW-Team dar
- Die eigentliche Softwareentwicklungs-Arbeit wird erst kurz vor Serienfertigung auf der echten Hardware getestet, dann entdeckte Hardware-fehler zu beheben ist entsprechend teuer
- Für Hardware-Tests gibt es keinen Zugang zu Endanwendern und somit fast ausschließlich Feedback nur von anderen Hardware-Expert:innen
- Software-Teams brauchen zuätzliche (und zeitintensive) Unterstützung, wo sie relevante Informationen über die Hardware finden können (z.B.Versorgungsspannungen, Datenbusse,
- Adressen, PIN-Belegungen etc.). Dasselbe gilt auch für Leiterplatten-Layout und -bestück-informationen (z.B. wo genau ist diese LED?)

Lineares HW-First Entwicklungsmodell

Die bereits beschriebene Aufteilung nach Fachlichkeit ist darüberhinaus auch als Ursache zu verstehen, warum Hardware und Software in fast allen uns bekannten Unternehmen sequentiell erstellt werden. Dies führt in der Praxis immer wieder zu den gleichen, die Entwicklung lähmenden, Effekten:

- viele definieren up-front die Hardware bis hinunter zum MCU-Modell dann wird die voll-umfängliche HW realisiert und monatelang in Betrieb genommen
- Tests bei der Inbetriebnahme erfolgen von Hand und sind entsprechend zeitintensiv bzw. in ihrer Aussagekraft äußerst unvollständig
- die SW-Unterstützung für die neue HW entsteht mit massiven Zeitverzögerungen und HW-Regressionen werden entsprechend spät entdeckt

Um es noch einmal deutlich zu sagen: Die Zwangslinearisierung eines eigentlich inhärent iterativen Entwicklungsprozesses führt zu folgendem dead-lock problem: Hardware braucht Software fürs Testen, Software braucht Hardware für das Implementieren, Ausprobieren und Experimentieren.

Wie bereits gesagt sind in hochkomplexen Szenarien Iterationen unvermeidbar. Ein darauf nicht ausgelegter Entwicklungsprozess macht die unvermeidlichen Iterationen unverhältnismäßig teuer und langwierig.

In vielen Unternehmen stellen wir überdies eine sich daraus begründbare Konditionierung fest: Weil alle Stakeholder um die langen Laufzeiten der Hardware-Entwicklung wissen, werden diese versuchen, möglichst viele ihrer Wünsche gleich im ersten (Hardware-)Release unterzubringen,weil niemand zweimal lange warten will. Damit wird zusätzliche Arbeitslast erzeugt und die Fertigstellung der ersten funktionalen Prototypen noch einmal massiv verzögert - ganz genau wie bereits von allen erwartet.

Proprietäre und preisintensive Werkzeuge
Gängige im industriellen Umfeld anzutreffende EDA-Tools wie z.B. Altium Designer oder Mentor Expedition sind leistungsfähige Werkzeugsuiten, die in den Händen von Expert:innen eindrucksvolle Hardware-Designs ermöglichen. Allerdings hat das auch (wie alles) eine Kehrseite:

Die Software, die wir für Stromlaufplan und Leiterplattenlayout benutzen, ist sehr gewöhnungsbedürftig und teuer. Die Designdaten sind in proprietären Dateiformaten eingeschlossen, die zumeist nur das entsprechende EDA-Tool selber lesen kann. Weitere Nachteile sind:

- teure Lizenzen zwingen Unternehmen zu maximaler Auslastung einzelner Expert:innen (für die eine Lizenz angeschafft wurde), es entstehen Bottle Necks und wenig interdisziplinäre Zusammenarbeit
- Arbeitsergebnisse sind oftmals nur mit diesem Tool und durch Experten sichtbar, d.h. Information werden nicht so schnell verteilt, wie sie entstehen
- Feedback von anderen Team-Mitgliedern entsteht erst spät, wenn überhaupt.
- Mangelnde Offenlegung der Datenformate behindert die automatisierte Weiterverabeitung der Daten (z.B. durch Automations-Skripte)
- die meisten dieser Werkzeuge verfügen nur über eine grafische Benutzeroberfläche, verhindern also den Zugriff auf die Designdaten über die Kommandozeile (um Automatisierungen wie automatische Builds zu erlauben)

Kultivierung eines hochspezialisierten Expertentums
Hohe Kosten der zum Einsatz kommenden Werkzeuge begünstigen eine weitere Charakteristik vieler Entwicklungsabteilungen: Um die Investitionen bestmöglich zu verwerten, sind Mitarbeitende angehalten, ausschließlich Teiltätigkeiten des Designprozesses durchzuführen, was sie über die Zeit natürlich zu erstaunlichen Expert:innen macht. Diese Arbeitsteilung wirkt sich aber auch auf das Selbstverständnis der einzelnen Personen aus. Der eigene „Wert" wird in Einheiten des erarbeiteten Expertenwissen gemessen und der jeweils zu verantwortende Teilschritt wird vom Mittel der Zielerreichung zum Wert an sich.

Die Folge sind u.a. Besitzstandswahrung und Ablehnung von Änderungen auf der Ebene einzelner Mitarbeiter:innen. Das hat Konsequenzen: Oft ist der Hardware-Entwurf immer noch in der Verantwortung einer einzelnen Person, wenn auch manchmal eine Aufteilung nach Schaltplan und Layout erfolgt. Das erzeugt ein Gefühl der Notwendigkeit und des Gebrauchtwerdens. Es ist aber auch Ausdruck von Misstrauen, dass man im Zuge einer Effizienzsteigerung ersetzt werden könnte, wenn auch andere die gleiche Arbeit machen könnten.

Fehlendes Augenmaß in anderen Unternehmensprozessen
Wir haben allzu oft beobachet, dass die Unterstützung für agile Entwicklung an den Schnittstellen zu weiteren Abteilungen und Einheiten der von uns besuchten Unternehmen bröckelt. Klassisches Beispiel ist der zentrale Einkauf in größeren Unternehmen, der oft zu weit weg ist, um die ganze Herausforderung des Vorhabens – auch abseits der Leistungsmerkmale der zu beschaffenden Bauteile – erfassen zu können.

Ein typischer Fall in der Entwicklung ist, dass auf Grund einer kurzfristigen Änderung ein Bauteil/Werkzeug schnell bestellt werden muss, um weitere Tests damit durchzuführen. Der Einkauf hat jedoch gar keine Motivation, das Bauteil/Werkzeug schnellstmöglich zu bestellen, wird er doch in der Regel daran gemessen, Teile günstig zu beschaffen und schnell ist in der Regel nicht günstig. Dass im Zweifelsfall jedoch anschließend das gesamte Team mehrere Tage blockiert ist, wird übersehen. Das Problem entsteht, weil das Credo der Mengenrabattierung durch einen zentralen Einkauf von stets gleichen und bekannten Gütern auf den Prototypenbau mit in der Regel neuartigen Teilen in geringen Stückzahlen übertragen wird.

Ein Teil des Problems liegt also in einer isolierten Incentivierung der Prozessbeteiligten, die die unterschiedliche Bedürfnisse eines agilen Entwicklungsprozesses mit gemischten Teams bis hin zur späteren Serienproduktion nicht abbildet.

Die Schnittstellenproblematik wie in unserem Beispiel nur auf den Einkauf zu reduzieren, wäre aber zu kurz gesprungen. Allgemeiner ließe sich von einer zu wenig ausgepägten systemischen Optimierung der Unternehmensprozesse sprechen.

Kostenkontrolle als Maß aller Dinge
Viel zu oft herrscht in den Unternehmen die Mentalität, Herstellungskosten um jeden Preis möglichst gering zu halten. In einer Welt, in der Veränderungen eher die Ausnahme bilden, ist das stimmig., Denn es gilt bei weitestgehend austauschbaren Produkten kostenmäßig vorne zu liegen und so den eigenen Ertrag zu sichern.

Dieser Fokus auf Kosten nimmt uns jedoch Flexibilität, wenn z.B. ein später erweiterter Funktionsumfang das Risiko birgt, eine neue Hardware zu erfordern, welche dann für Produkte im Feld schwer zu tauschen ist. Beispiele hierfür sind eine knapp bemessene, kaum vorausschauende Dimensionierung des Mikrocontrollers oder keine Kapazitätsreserven beim verfügbaren Speicher oder des Daten-Busses.

Umgekehrt wollen wir aber auch warnen, alles überzudimensionieren. Denn hier würden wir uns selbst der Kostenvorteile berauben und hätten das Risiko, ein zu teures Produkt auf den Markt zu bringen. Es geht um die richtige Balance zwischen Kostendruck und Flexibilität.

Noch ein Beispiel: die Fertigung der Elektronik. Statt lokale Lieferanten früh ins Projekt einzubinden, um von ihrer Erfahrung zu profitieren und dafür zu sorgen, dass sich auch dort Mitarbeiter mit dem neuen Produkt identifizieren und zur Lösung beitragen, wird oft lange vermieden, sich festzulegen. Statt Vertrauen und enger Zusammenarbeit wird auf austauschbare Lieferanten gesetzt.

Voll spezifizierter Umfang von Anfang an
Aufträge für Hardware-Entwicklung werden meist mit vollständig beschriebenen Lasten- und Pflichtenheften (und bei externer Unterstützung: Werksverträgen) organisiert. Dies führt oft zu langen Phasen des Zusammenstellens dieser Informationen. Daraus entstehen dann die bereits erwähnten riesigen Arbeitslasten vor dem ersten Prototyp.

Mit von Anfang an vollständig spezifiziertem Funktionsumfang zu starten führt außerdem dazu, das erst spät getestet werden kann, wie das Produkt bei Endanwender:innen ankommt – und dann festzustellen, dass einige Funktionen nicht so gebraucht werden, wie zu Beginn angenommen, ist teuer bezahltes Wissen um etwas, das nicht gebraucht wird.

Teil 2 - Woran hapert es? 5 Thesen zur langsamen Verbreitung agiler Arbeitsweisen

Wie aus diesen Beobachtungen ersichtlich wird, arbeiten viele Unternehmen auch heute noch mit Überzeugungen, Methoden und Werkzeugen einer vergangenen Zeit in einem Umfeld, das sich stark verändert. Unsere Beobachtungen verdeutlichen, wie wichtig es wäre, sich den neuen Bedingungen der Märkte agil zu nähern. Doch woran scheitert es? Es folgt eine Situationsanalyse in fünf Thesen:

1. Missing Sense of Urgency
2. Kaum (bekannte) Erfolge
3. Fehlender Glaube an iterative Hardwareentwicklung
4. Unsere Werkzeuge behindern agiles Arbeiten
5. Alles nur mit halbem Herzen

Missing Sense of Urgency
In vielen Unternehmen bemerken wir, dass es noch gar kein Bewusstsein für mögliche Veränderungen in Art und Weise der Produkt-Entwicklung gibt. Viele Unternehmen sind in ihrer Domäne Marktführer. Eine Position, die sie sich über viele Jahre hart erarbeitet haben. „Es läuft doch alles“ ist ein Satz, den wir häufig hören.

Wir streiten gar nicht ab, dass die bisherige Art der Entwicklung zweifelsohne erfolgreich war. Doch Vorsicht: viele prominente Beispiele der letzten Jahre zeigen, dass selbst vermeintlich starke Marktführer schneller zu Fall kommen als gedacht. Wir reden hier nicht das Alte schlecht. Vielmehr sollten wir uns ständig mit Neuerungen beschäftigen, um auf dem aktuellen Stand der Technik zu bleiben. Und das gilt auch für Projektmanagement und Entwicklungsprozesse.

Kaum (bekannte) Erfolge
Auch das Fehlen einer breiten Basis an erfolgreichen dokumentierten Anwendungsfällen erkennen wir an. Es nimmt zu, aber es ist immer noch sehr wenig im Vergleich zu Erfahrungen mit Agilität in der Softwareentwicklung. Das verunsichert: ist es bei Hardware überhaupt möglich, mit modernen agilen Methoden erfolgreicher als bisher zu sein? Ist das Risiko nicht zu groß, das bekannte Terrain zu verlassen? Was, wenn sich die Veränderungen negativ auf die Produktivität auswirken? Alles berechtigte Fragen.

Doch ein Vorteil wirklich gelebter Agilität ist, dass die Einführung der Arbeitsweise durch kürzere Iterationen engmaschig beobachtet werden kann. Etwaige negative Auswirkungen werden rasch entdeckt, können in Retrospektiven thematisiert und oft auch gelöst werden. Übrigens gibt es auch viele Beispiele für nicht erfolgreiche Projekte, die ganz ohne agile Vorgehensweisen schief liefen. Diesen fehlten dann aber oft diese Korrektivprozesse agiler Vorgehensweisen.

Fehlender Glaube an iterative Hardwareentwicklung

Ein potentiell auslieferbares Produkt alle zwei Wochen? Unmöglich! Denn im Gegensatz zu reiner Software, die virtuell und damit schnell und fast ohne zusätzliche Kosten „gebaut" werden kann, braucht es für physische Produkte meist kapitalintensive Maschinen, eigens angefertigte Werkzeuge und das Investment diverser Herstellkosten. Es ist nicht nur schwieriger, sondern auch teurer, jede Iteration ein potentiell auslieferbares Produkt bereit zu stellen. Das streiten wir gar nicht ab.

Doch abgesehen davon, dass durch den Einsatz von immer besseren Prototypen-Werkzeugen die Kosten hierfür kontinuierlich sinken, trägt auch digitale Modellierung dazu bei, dass für das Kernziel iterativer Arbeit – das Einholen eines ersten Feedbacks – nicht immer eine wirklich fertig produzierte Hardware vorliegen muss.

Unsere Werkzeuge behindern agiles Arbeiten

Egal, ob in gemischten Teams oder bei der Kommunikation zwischen dedizierten Hardware- und Software-Teams, die am selben Produkt arbeiten: wichtig ist weitestgehende Transparenz. In der Softwareentwicklung beispielsweise liegen die meisten Inhalte, sprich Programmcode, in Textform vor. Text hat dabei folgende Vorteile:

- menschen- wie maschinenlesbar
- durchsuchbar
- einfach zu versionieren
- auch ausserhalb des ursprünglichen Werkzeugs modifizierbar und weiterverwertbar

In der Hardware-Welt ist das bei weitem nicht so. Dort haben wir, wie schon gesehen, oft proprietäre, binäre Dateiformate, um Design-Informationen abzulegen. Und es wird eine teure Lizenz pro Arbeitsplatz benötigt, um diese Dateien überhaupt lesen zu können.

Die Bedienung dieser Programme ist auf Expert:innen ausgerichtet, d.h. die Werkzeuge lassen sich von Laien, und sei es nur zum Nachschlagen von Informationen, in der Regel nicht bedienen. Die Datei-Formateder Hardware-Welt sind nicht menschenlesbar und lassen sich somit auch nicht auf Ordner-Ebene durchsuchen. Damit sind die Dateien auch nur bedingt versionierbar.

Wozu führt das? Informationen sind schwerer einsehbar. Teams stellen sie dann gerne doppelt bereit: beispielsweise der Stromlaufplan/die Bauteilbibliothek in einem Elektronik-Design-Tool für die Hardware-Leute, und in exportierten PDF-Dateien für alle anderen. Nur: dieselbe Information an zwei Stellen führt über kurz oder lang auch zu Fehlern. Wie wird z.B. sichergestellt, das für jede Änderung am Stromlaufplan im Design-Tool auch ein aktualisiertes PDF ausgeleitet und zur Verfügung gestellt wird?

Alles nur mit halbem Herzen

Was wir auch oft feststellen: An irgendeiner Stelle ist die Umsetzung agiler Vorgehensweisen bestenfalls halbherzig. Dies führt früher oder später zum langsamen Tod jeglicher Agilität, weil nicht so umfassend agil vorgegangen wird, dass sich das Team selber ständig in seiner Arbeit und Effizienz verbessern kann. Diese Halbherzigkeit kann leider an mehreren Stellen zu wuchern anfangen:

- auf Unternehmensebene, wenn das Management nicht wirklich an agil glaubt
- auf Teamebene, wenn Teammitglieder nicht konsequent in die neue Arbeitsweise einsteigen
- im Kopf einzelner Mitarbeiter:innen, die agil – warum auch immer – ablehnen

Oft tritt diese Halbherzigkeit zutage, wenn das Projekt auf erste große Probleme trifft. In den allermeisten Fällen sind dies heraufziehende Konflikte zwischen der wachsenden Autonomie des agiler werdenden Teams und den nach wie vor starren auf Zuständigkeit und Kontrolle ausgerichteten Unternehmensprozessen. Fast genauso oft traten aber auch noch viel größere Probleme auf, als man schließlich den agilen Pfad verliess und wieder mit konventionellen Vorgehensweisen weitermachen wollte. Wie diesen Absätzen zu entnehmen ist, gibt es zahlreiche Hindernisse auf dem Weg, sich moderner aufzustellen und an dynamische Märkte anzupassen. Unsere Überzeugung aber bleibt: es lohnt sich!

Teil 3 - Wie starten, um von der agilen Entwicklung zu profitieren?

Nach all diesen potenziellen, systemischen Hindernissen stellt sich die Frage: wie können wir uns auf den Weg machen, um inder Hardware-Entwicklung auch von agilen Arbeitsweisen zu profitieren? In einer Welt, die nach schnellerer, effizienterer Entwicklung von physischen Produkten verlangt, bei gleichzeitig hohem Anspruch an Innovation und Qualität. Bevor wir auf einzelne konkrete Praktiken eingehen, wollen wir die Bedeutung der richtigen Rahmenbedingungen hervorheben. Denn so wie Pflanzen einen guten Boden benötigen, um zu gedeihen, aber auch die richtige Temperatur, Luftfeuchte und Windstärke, und vor allem ganz viel Sonnenlicht, so brauchen auch moderne Arbeitsweisen die richtige Umgebung, um eine wirkliche Verbesserung zu garantieren.

Die Rahmenbedingungen

Wichtig ist, dass es im und um das Team herum Lust und Motivation für die Veränderung der Arbeitsweise gibt. Egal, ob auf Grund von aktuellen Herausforderungen oder aus Interesse der Mitarbeiter:innen: Agilität sollte nur eingeführt werden, wenn die beteiligten Führungskräfte und Mitarbeiter es auch wirklich wollen.

Dann besteht der erste Schritt darin, ein einheitliches Verständnis zu schaffen. Zwar sprechen oft alle Beteiligten über Agilität, verstehen tun jedoch alle was anderes darunter. Gefährliches Halbwissen kursiert, wie z.B. die erwähnten zwei Wochen als Dauer für einen Iterations-Takt. Einige Projekte werden als “agil” dargestellt, obwohl sie es nicht wirklich waren. Ein sauberes Grundlagen-Training über 2-3 Tage für alle schafft gemeinsames Verständnis und legt die Basis für den späteren Erfolg.

Wir empfehlen ein auf Hardwareentwicklung und das Unternehmen angepasstes Training, da viele Standardschulungen stark auf Softwarekontexte ausgelegt sind.

Wir befürworten weiter, dass ein erfahrener Scrum- oder Agile Coach den gesamten Prozess begleitet. Anfangs vielleicht dichter und enger gestaffelt, später seltener und weiter gefasst. Wichtig ist erfahrene Moderation vor allem beim Einholen von Feedback, also z.B. beim Testen der Funktionalität durch Endanwender, bei teaminternen Reviews und vor allem bei Retrospektiven. Moderation ist auch dann sehr nützlich, wenn es im und ums Team zu Konflikten kommt. In diesem Zusammenhang muss immer bedacht werden, dass alle Teams durch die Teambuilding-Phasen Forming, Storming, Norming müssen, bis sie bei Performing ankommen.

Am besten startet man mit wirklich passenden Projekten, um die vollen Vorteile agiler Arbeits-weisen zu erleben. Ideal sind Vorhaben, in denen Neuland betreten wird und die eine gewisse Komplexität aufweisen. Oder Vorhaben, deren Lösung nicht offensichtlich zu Grunde liegt oder vielleicht sogar schon vollständig beschrieben ist. Auch sollten Endanwender zumindest bekannt und verfügbar sein, um die Tests eng an dieser Zielgruppe auszurichten.

Alle involvierten Mitarbeiter:innen müssen umdenken. Operative Ingenieure lösen sich von sequentiellen Abläufen wie dem V-Modell. Experimentierfreudigkeit, die typischerweise in dieser Gruppe sehr stark ausgeprägt is, für neue Werkzeuge und digitale Hilfsmittel wird gefördert. Alle akzeptieren, dass sich neben Technologie auch die Arbeitsweisen ändern müssen und diese Neuerungen auszuprobieren sind. Und alle verstehen die Bedeutung von Transparenz.

Insbesondere Führungskräfte müssen umdenken, weg von reiner Kosteneffizienz hin zum Wert- schätzen erhöhter Flexibilität (die etwas kostet) als Chance für den wirtschaftlichen Erfolg eines Unternehmens in einer volatilen Welt. Volatilität erzeugt Unsicherheit. Und das ist in Ordnung, denn dafür gibt es agile Arbeitsweisen, die helfen, sich in diesem Umfeld sicher Schritt für Schritt nach vorne zu bewegen.

Führungskräfte brauchen Zuversicht in die Fähigkeiten der Selbstorganisation ihrer Teams und Vertrauen darin, dass die Teams ihr Bestes geben. Solche Teams brauchen Zeit für Retrospektiven ihrer eigenen Arbeit und für das Erlernen von Neuem (z.B. für neue Technologien oder aber neue Wege der Selbstorganisation etc.). Wird den Teams diese Zeit gewährt, gestalten sie die eigene Arbeit zunehmend effizienter, z.B. durch das Nutzen neuer Tools. Nur so organisieren sich Teams immer besser, bis sie schließlich performen. Nicht ganz „auf Vollast" zu laufen, bringt zudem weitere Vorteile:

- bei Engpässen im Team helfen Teammitglieder aus
- Zeit zum Lernen neuer Methoden, Tools u.ä.
- Zeit für das Implementieren von Ideen, um effizienter zu sein

Quick Wins

Veränderungsprozesse brauchen neben der tiefgreifenden Veränderung auch schnelle sichtbare Erfolge, um Motivation für den oft anstrengenden Weg zu erzeugen. Daher wollen wir bewusst Raum für jene Dinge schaffen, die leicht in der Umsetzung und groß in ihrer Wirkung sind.

Agil Arbeit organisieren

Agile Organisationen orientieren sich an diesem Prinzip:

> *„Bring work to the people (team) not people to the work."*

Traditionell werden Fachabteilungen gebildet, die nacheinander die verschiedenen Aufgaben in einem Projekt bearbeiten (move people to the work). Agile Organisationen bilden stattdessen interdisziplinäre Teams, die an jeweils einem Projekt von Anfang bis Ende (move work to the people) arbeiten. Die Idee dabei ist, verschiedenste Perspektiven auf die anzugehende Herausforderung zu nutzen, um die beste Lösung zu identifizieren. Agile Arbeitsorganisation heißt, als kleinste Einheit ein Team und nicht einzelne Mitarbeitende zu sehen. Das fängt mit einer breiten Aufstellung des Teams an. Wir müssen uns von Spezialisten-Teams für Hardware, Mechanik oder Software verabschieden. Vielmehr brauchen wir für komplexe Aufgabenstellungen gemischte, eben cross-funktionale Teams, die Skills aus allen Disziplinen beinhalten.

Solche cross-funktionalen Teams eint ihre gemeinsame Mission, beispielsweise sich um die Blutzuckermessung innerhalb eines neuen Insulinpumpensystems zu kümmern. Eine Mission, die wirklichen Mehrwert für den Endanwender stiftet, im Vergleich z.B. zur reinen Entwicklung irgendeines Gehäuses. Neben der Kombination verschiedenster Perspektiven sorgt dieses enge, disziplinübergreifende Arbeiten auch für wesentlich weniger Übergaben und damit "Liegezeit" im System. Alle Beteiligten lernen die Werkzeuge und Methoden der anderen kennen und nutzen. Das fördert gegenseitiges Verständnis, baut Berührungsängste ab, weitet persönliche Perspektiven und schafft mit der Zeit einen Raum, in dem echte Kollaboration möglich wird.

Wichtig ist dabei, dass das Team sich auf ein Projekt konzentriert und seine Effizienz nicht unter switching costs leidet. Darüber hinaus sollte das Team, um nicht immer wieder den unvermeidbaren Forming-Storming-Norming-Performing-Zyklus zu durchlaufen, am Besten für mehrere aufeinanderfolgende Projekte intakt bleiben. Das bedeutet aber nicht, dass in cross-funktionalen Teams alle alles können und machen müssen. Und es bedeutet ebensowenig, dass stabile Teams über Jahre oder gar Jahrzehnte in ihrer Zusammensetzung unverändert bleiben müssen.

Arbeit in Iterationen

Für komplexe Herausforderungen ist detailliertes Planen einzelner Arbeitspakete Monate im Voraus nicht machbar. Besser ist, sich in kleinen Schritten voran zu tasten und so über die Zeit herauszufinden, wie der eigentliche Lösungsweg aussehen wird. Es bietet sich an, einen iterationsgetriebenen Ansatz wie Scrum zu verwenden. In ein- bis vierwöchigen Zyklen (Sprints oder Takte gennant) setzen sich Entwicklungsteams kleine überschaubare Ziele, planen diese aus und setzen sie dann direkt um. Wenn man so will, wird aus einem großen Pflichtenheft eine Folge kleinerer Pflichtenhefte für den jeweiligen Takt.

Die Kernidee: am Ende der Iteration haben wir etwas, was uns Feedback gibt. Dies mag in der Hardware-Entwicklung nicht immer ein wirklich fertiges physisches Bauteil sein - nichts desto trotz zählt der Wille, so nah wie möglich an diese ideale Lieferung heranzukommen und etwas zu liefern, was von Endanwendern testbar ist.

Frameworks wie Scrum folgen dabei dem PDCA-Zyklus (Plan, Do, Check und Act):

- Plan: Planung einer Iteration (eines Arbeitstaktes): dafür gibt es eigene Meetings wie das Sprint Planning
- Do: Arbeiten und dabei als Team jeden Tag verifizieren, wie weit wir gekommen sind und ob das Ziel noch erreichbar ist, im sogenannten 15-minütigen Daily Scrum
- Check: Am Ende der Iteration checken wir, ob das Ergebnis den geplanten Erfolg erzielt, in dem wir einerseits Kunden und Anwender im Sprint Review um Feedback bitten und andererseits uns als Team in der Sprint Retrospektive selbst fragen, wie wir uns weiter verbessern können
- Act: Einsichten daraus fließen wiederum in die nächste Iteration

Begleitet wird dieser stringente Prozess durch eine eigene Facilitation-Rolle, den Agile Coach, der dafür sorgt, dass die Teams dem Prozess treu bleiben und sich kontinuierlich verbessern.

Arbeitsplanung mit User Stories

Um sich auf Bedürfnisse von Endanwendern zu fokussieren, sind User Stories ein gutes Hilfsmittel. Im Gegensatz zu reinen Requirements, die sich stark auf das WAS? und die Funktionalität fokussieren, reichern User Stories dieses WAS? um zusätzlichen Kontext an. User Stories werden häufig in der folgenden Syntax geschrieben:

"Als <Rolle> möchte ich folgende <Funktionalität>, um folgenden <Nutzen> zu erreichen"

Die Rolle sagt aus, WER diese Funktionalität benötigt, also um welchen Endanwender:in es sich handelt. Diese divergieren hinsichtlich ihrer Erwartungen und Bedürfnisse, aber auch ihres Wissenstands und Umgang mit dem Produkt oft gewaltig.

Ein Beispiel:

- Labormitarbeiter:in, der/die mit dem System interagiert, um eine Probe zu analysieren
- Systemtechniker:in, der/die regelmäßig Kalibrierungen des Systems durchführt

Je nachdem, welche dieser beiden Anwender-Typologien eine bestimmet Funktionalität benötigt, braucht es eine kontextbezogene Implementierung und Design der Umsetzung.

Der zweite Part der User Story, die Funktionalität, beschreibt WAS? durch den Anwender gewünscht ist. Der Teil, der noch am ehesten der klassischen Anforderung entspricht. Die letzte Komponente der User Story enthält Hinweise auf das WOZU: was will der Anwender mit Hilfe dieser Funktion erreichen - eine Information, die zusätzliche Hinweise für die Implementierung liefert.

Idealerweise ist eine User Story so formuliert, dass diese auch durch den Endanwender getestet werden kann. Der Vorteil von User Stories ist, dass diese leichter gegeneinander priorisiert werden können, da schon beim Lesen klar ist, welcher Nutzen erzielt werden soll. Sie helfen auch, stärker über jede Story nachzudenken: für wen bauen wir das eigentlich und wofür?

Solche User Stories transportieren, geschrieben als Narrativ, den Kontext und somit auch Emotionen. Beides Dinge, die in technischen Anforderungsheften naturgemäß ausgeblendet werden, für das Erreichen unserer kreativen Gehirnhälfte aber essentiell sind.

Ein paar Beispiele:

- Als Biochemiker:in kann ich am Prototyp eine Pumpe vorwärts und rückwärts laufen lassen, um die Probe zwischen Mischkammer und Messkammer hin- und herzubewegen.
- Als Biochemiker:in kann ich die Pumprate einstellen, um die Probe erst schnell in die Mischkammer fliessen zu lassen und dann langsam zurück in die Messkammer.
- Als Biochemiker:in kann ich die Temperatur der Probe mit einer Auflösung von einem Grad messen, um die Temperatur bei der Berechnung der finalen Konzentration berücksichtigen zu können.

Den drei genannten Beispielen ist dabei gemeinsam, das sie tatsächlich die Story einer Person, mit der wir uns identifizieren können, erzählen.

Ausdrücklich warnen wollen wir vor „Pseudo-Stories" und anderen allzuoft anzutreffenden Anti-Patterns:

- User Stories mit Pseudo-Usern: Als „Turbine“ möchte ich ….
- Als Arbeitsaufgaben verkleidete Stories: „Als Projektleiter möchte ich xyz umgesetzt haben, um …“
- Und damit verwandt: Das Schneiden von Stories entlang der Aufgaben im V-Modell von der Requirements-Spezifikation über das Design bis hin zur Implementierung.

Gute Stories sind zudem weitestgehend unabhängig voneinander. Nur dann können diese nämlich von Endanwendern oder Product Ownern priorisiert werden und das Team kann seine Energie auf die wichtigsten Features verwenden.

Mehr Transparenz

Transparenz klingt einfach, ist es aber oft nicht. Mehr Transparenz hat den Vorteil, dass involvierten Kolleg:innen alle Informationen zur Verfügung stehen, um selbst eigene sinnvolle Entscheidungen treffen zu können. Ein priorisierter (Product) Backlog von User Stories ist Kern dieser Transparenz.

Darüber hinaus ist Klarheit über die mittelfristige Planung wichtig, zum Beispiel über eine Roadmap – damit für Team und Außenstehende klar ist, in welche Richtung wir navigieren. Das gleiche gilt für die kurzfristige Sicht. Hier helfen Boards für die Teams, die den aktuellen Arbeitsstand widerspiegeln und jedem Involvierten schnell einen guten Überblick ermöglichen.

Einen Schritt weiter ginge diese Transparenz mit Partnern und Kunden, um diese stärker in den eigenen Entwicklungsansatz zu integrieren. Dabei geht es nicht darum, Geschäftsgeheimnisse zu teilen, sondern jene Informationen, die Zusammenarbeit erleichtern. Was spricht gegen frühzeitiges Teilen erster Zeichnungen mit dem Vermerk, dass es sich um keine finale Version handelt? So können Lieferanten die Komplexität ihrer nachfolgenden Arbeit genauer abschätzen und sich vorbereiten. Sie können auch schon Feedback und Hinweise geben, die uns helfen, unser Produkt später besser fertigen oder zertifizieren zu können.

Agile Architektur und Design

Bei nur teilweise bekannten Anforderungen, die sich zudem dynamisch und aufgrund der in vorangegangen Iterationen gewonnenen Erkenntnisse ändern, müssen auch Architektur und Design entsprechend anders gedacht und umgesetzt werden.

Die Aufgabe der Architektur ist es, mittels Flexibilität frühe Festlegungen unnötig zu machen. Denn diese haben ein vielfach höheres Risiko, sich im späteren Verlauf als falsch zu erweisen, weil zu einem frühen Zeitpunkt ja kaum Informationen vorliegen. Wird zum Beispiel gleich zu Beginn ein spezieller Mikrocontroller ausgewählt und stellt sich später heraus, dass wichtige Schnittstellen fehlen oder nicht ausreichend Speicher vorhanden ist, wiegt das umso schwerer, wenn niemand inHardware noch in Software für einen solchen Wechsel vorgedacht hat.

Wir stellen uns ideale Design-Entscheidungen als Plug and Play vor, das heißt Designentscheidungen sind jeweils in einem Modul (z.B. in einer Platine) gekapselt. Die Module können dann miteinander in fast beliebiger Weise kombiniert werden. Das begrenzt Auswirkungen notwendiger Designänderungen auf jeweils ein Modul und unterbindet gefürchtete Kaskadeneffekte.

Haben wir uns dann Modul für Modul alle Designentscheidungen erarbeitet und getestet, können später die einzelnen Module miteinander verschmolzen werden. Hier spielen ein weiteres Mal offene Dateiformate ihren Vorteil aus, denn oft kann dieser Schritt mittels eigener Skripte automatisiert werden.

Weitere Vorteile:

- diese Reduktion kann kontinuierlich erfolgen und etwaige Probleme können frühzeitig erkannt und behoben werden
- simpler und weniger fehleranfällig, weil einzelne Modul einfacher umzusetzen sind
- schneller, weil weitgehend unabhängige Module parallel von eigenständigen Teams realisiert werden können (oder früher bereits realisiert wurden)

Natürlich werden einzelne Module selten von allein zueinander passen. Vielmehr erkennen wir hier eine weitere wesentliche Aufgabe agiler Architekturarbeit: Es braucht ein durchgängiges Framework, dass Spielregeln festlegt, nach denen Module integriert werden können.

Teil 4 - Weitere Ideen für noch agilere Entwicklung von Hardware

Im letzten Absatz widmen wir uns tiefgreifenderen Ideen, die für ihre Implementierung etwas mehr Zeit benötigen. Diese sollen als Inspiration dazu dienen, wo mögliche Verbesserung nach der Einführung der Quick Wins liegen.

Continuous Integration im Hardware-Design nutzen

Im Abschnitt zum Agilen Design haben wir das Thema Automatisierung bereits angesprochen. In der (agilen) Software-Entwicklung ist Continuous Integration (CI) praktisch flächendeckend angekommen und erlaubt überhaupt erst die Beherrschung der Komplexität heutiger Entwicklungsprojekte. Es lohnt sich also, zu testen, ob das auf Hardware-Entwicklung ebenso übertragen werden kann, mit Augenmerk auf die zeitfressenden, aber automatisierbaren Aufgaben.

Die Autoren haben in der Vergangenheit zum Beispiel sehr erfolgreich gängige CI-Systeme wie Jenkins oder Github für die tägliche oder wöchentliche

- Durchführung von Electrical Rule Checks (ERC) und Design Rule Checks (DRC)
- Erzeugung von CAM-Daten
- Erzeugung von PDFs des Schaltplans oder des Bestückungsdrucks
- Erzeugung der technischen Dokumentation von Baugruppen
- Abfrage von Informationen zur Verfügbarkeit von Bauteilen

genutzt. Die Automatisierung agiert dabei jedes Mal gleich, das heißt reproduzierbar. Probleme werden frühzeitig angezeigt. Insbesondere wird Transparenz im Team gesteigert, da Designartefakte jederzeit in einer für alle lesbaren Form vorliegen.

Konfigurationsmanagement richtig einsetzen

Änderungen an einem Design sind idealerweise kleinschrittig, was die Zahl der Änderungen bis zu einem finalen Stand deutlich erhöht. Eine manuelle Vergabe von Revisionsnummern im Zeichnungsblatt kommt damit schnell an ihre Grenzen. Auch hier weist die Software-Entwicklung den Weg.

Wir haben sehr gute Erfahrungen mit der Software git gemacht, dem de-facto Standard für Versionskontrolle bei Software. Einmal daran gewöhnt, wollte niemand mehr die Bequemlichkeit missen, mit einem einzigen Blick auf die Git-SHA1 im Silkscreen einer Platine sofort das genau passende Schaltplan-PDF aufzuschlagen. Es passiert nämlich öfters als gedacht, dass sich Designs mit der angeblich gleichen Versionsnummer in subtilen Details unterscheiden, schlicht weil die Anpassung des entsprechende Textfeldes im Zeichnungsrahmen vergessen wurde.

In einen Hardware-Design Styleguide investieren

Auch Style Guides sollten wir von Software-Entwicklung übernehmen. Beim Bearbeiten von Quelltexten, aus denen Software besteht, kommen praktisch immer sogenannte Style Guides zum Einsatz. Die Idee ist, Lesbarkeit und letztendlich das Verstehen des Quelltextes durch einheitliche und wiederkehrende Idiome zu erhöhen.

Solche Style Guides haben wir bereits mehrfach erfolgreich in die Hardware-Welt übertragen. Was am Ende im Style Guide auftaucht, muss das Team entscheiden. Hier ein paar Ideen:

- Schaltpläne in A4 sind wesentlich besser für Ausdrucke und Review geeignet
- Daten fließen am besten in Leserichtung (also von links nach rechts) und Versorgungsspannungen von oben nach unten
- Komplexe Bauteile auf verschiedene Symbole aufteilen
- Einheitliche Symbole für eine Bauteilart (Widerstand, LED, ...) verwenden
- Ambivalente Designatoren vermeiden, also z.B. Konnektoren mit X und Jumper mit J auseinanderhalten
- Ausrichtung im Bestückungsdruck in Leserichtung bzw. von oben nach unten
- Bei mehrpinnigen Bauteilen Pin 1 immer kennzeichnen
- Jedes Netz bekommt einen Testpunkt
- Für eine verpolungssichere Signalbelegung bei Steckverbindern sorgen
- Jumper an Stelle von Lötbrücken bevorzugen

So ein Styleguide kann dann schon einige Seiten füllen. Es hilft, wenn mittels Continuous Integration Abweichungen vom Style Guide automatisch erkannt werden können. Der Lohn dieser Mühe sind qualitativ hochwertige Schaltpläne, die sowohl von Firmware-Entwickler:innen einfacher zu lesen als auch anderen Hardware-Entwickler:innen einfacher abzuwandeln sind.

Mit Open-Source Tools experimentieren

Bei allen Experimenten, die Hardware-Welt agiler zu machen, haben wir immer wieder festgestellt, dass vor allem die Automatisierung an den bekannten kommerziellen Werkzeugen scheitert. Diese sind nicht darauf ausgelegt, in Eigenregie die erstellten Designdateien weiter zu verarbeiten. So sehr die Werkzeuge für die bekannten Workflows optimiert sind, so kläglich versagen sie, will man eigene Workflows gestalten. Bei einem agilen Vorgehen kommt es aber genau auf diesen Spielraum an. Werkzeuge müssen einfach erweiterbar und anpassbar sein, und die erzeugten Daten müssen auch außerhalb des ursprünglichen Tools interpretierbar sein.

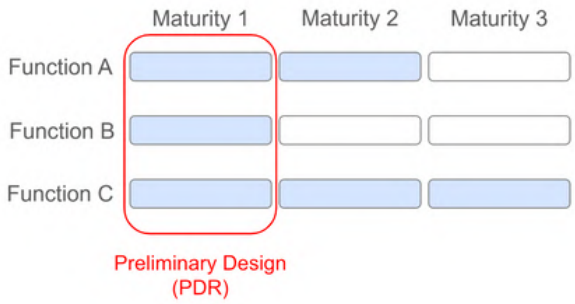
Berücksichtigten wir diese Kriterien bei der Auswahl unserer Werkzeuge, dann landen Open-Source Werkzeug oftmals weit vorne. Die namensgebende Quelloffenheit erlaubt uns genau die Änderungen vorzunehmen, die wir unter Umständen benötigen. Die Wahrscheinlichkeit, dass irgendwo auf der Welt jemand eine fast identische Änderung bereits gemacht hat und erneut als Open-Source zur Verfügung stellt, ist zudem hoch. Das gilt insbesondere für die Verzahnung verschiedenster Werkzeuge miteinander. Ein Beispiel: Die Integration der ECAD-Toolsuite Kicad mit dem MCAD-Werkzeug Freecad ist ebenfalls frei verfügbar und erlaubt die Platzierung der Bauteile einer Platine im MCAD-Werkzeug bei gleichzeitig stattfindender Bauraumprüfung. Eine weitere Intgration erlaubt den Export von Kicad-Designdaten für die Berechnung von EMC-Eigenschaften in openEMS.

Open-Source Werkzeuge haben an vielen Stellen sicher noch immer Defizite. Dafür kann ihr Einsatz niederschwellig erfolgen, das heißt alle im Team haben Zugang zum Werkzeug und Anpassungen am Werkzeug können im Team für die teamspezifischen Umstände umgesetzt werden. Wir haben so zum Beispiel in der Vergangenheit die Software-Expertise in unserem Team genutzt, um bei der automatischen Erstellung der Produktionsdaten aus unseren Designfiles mittels eines eigenen Skriptes die Git-SHA1s in Schaltplan und Silkscreen automatisch einzufügen.

Reifegradmodelle hinterfragen

Etwas, was wir ebenso oft in Entwicklungsprojekten sehen, sind Entwicklungsprozesse nach klassischen Stage-Gates, die Meilensteine entlang des traditionellen V-Modells vorsehen, um so den Fortschritt der Produktentwicklung zu messen. Während eine grobe Übersicht damit gegeben ist, werden diese Ansätze im Detail heutigen Anforderungen nicht gerecht. Selten erleben wir Projekte, in denen das Konzept wirklich abgeschlossen ist, bevor wir das Design beginnen und diesen Prozess so fortsetzen, dass das finale Design vor dem Start der Implementierung vollständig abgesegnet ist.

Wenn dem so ist, ist die Aussagekraft des geschafften Meilensteins 2a (welcher beispielsweise die Finalisierung des System Designs vorsieht) von schwindender Bedeutung. Häufig werden in Unternehmen die Meilensteine mit einer Reihe von Abweichungen passiert, da die eine oder andere Komponente doch noch nicht fertig spezifiziert und designed ist.



Moderner Ansatz für das Reifegradmanagement

Es ist sinnvoll, sich von den Reifegradmodellen, die sich zu sehr am V-Modell orientieren und daher dokumentengetrieben agieren, ein Stück weit zu verabschieden.

Stattdessen empfehlen wir, den Reifegrad im Projekt funktionsgetrieben zu messen. Ein Beispiel wäre hierfür das Tracking sämtlicher Funktionen (idealerweise aus Anwendersicht) in deren Reifegrade. Die Reifegrade müssten hier in weiterer Folge standardisiert werden, sodass zu jeder Zeit transparent ist, welcher Reifegrad welchen Stand der Entwicklung meint. Über den Lauf des

Projekts können einzelne Funktionen bereits im finalen Reifegrad vorhanden sein (Stichwort “Langläufer”), während andere wiederum noch in früheren Reifegraden sind, um hier noch Flexibilität für kostengünstigere Änderungen zuzulassen. Haben alle Funktionen einen bestimmten Reifegrad erreicht, können wir damit gegebenenfalls einen klassischen Meilenstein erreichen, wie beispielsweise das Preliminary Design. Diese Art der Fortschrittskontrolle erfordert ein modulares Design, in dem einzelne Module Funktionen unabhängig voneinander realisieren können, mehr dazu in unserem Absatz zur agilen Architektur und Design.

Fazit

Unsere Bestandsaufnahme zur Verbreitung von agilen Arbeitsweisen in der Hardwareentwicklung fällt mehr als 20 Jahre nach dem Agilen Manifest noch immer verhalten aus. Die von uns beobachteten Symptome die sich aus der Nicht-Passung der etablierten sequenzorientiertenEntwicklungsprozesse und heutiger Marktbedingungen ergeben, sprechen hingegen eindeutig für die Notwendigkeit sich auch in der Hardwareentwicklung in Zukunft stärker auf agile Methoden einzulassen. Hierzu gibt es es zwar weniger Erfahrungsberichte und Wissen als für die Softwareentwicklung, dennoch sind genügend Informationen vorhanden, um – den eigenen Willen vorausgesetzt – mit Agiler Hardware starten zu können. Die Autoren werden in Zukunft ihre Ideen und Erfahrungen in ihre nächsten Projekte einbringen und freuen sich ebenfalls über jede Nachricht, die sie zu einem weiteren agilen Hardwareprojekt erreicht.

Unsere Experten:

Christoph Schmiedinger

Executive Consultant bei borisgloger consulting, einem der führenden Beratungsunternehmen Deutschlands für alles rund um Agilität.

Dr. Tobias Kästner

Solution Architect Medical IoT bei inovex GmbH, einem führenden Dienstleister für die digitale Transformation mit Full-Stack Technologiekompetenz von Cloud bis Deep Embedded.

Gregor Groß

Geschäftsführer alpha-board gmbh, einem Berliner Dienstleister für agile Hardware-Entwicklung, PCB-Design und Fertigungsservice.