Scrum für Embedded-Software

Gut – aber aus anderen Gründen, als Ihr Manager glaubt

Dr. Joachim Schlosser, Martin Hillbrand, Elektrobit Automotive GmbH
<joachim.schlosser|martin.hillbrand>@elektrobit.com

Agil, was ist das eigentlich?

Agile Entwicklung, das hat was von Leichtigkeit. Tatsächlich trägt agile Entwicklung dazu bei, schneller bessere Ergebnisse erzielen zu können. Es gilt aber auch: der Prozess ist strikter, als Sie ihn heute wahrscheinlich leben. Scrum ist strikter gegenüber dem Management und erfordert einen funktionierenden Integrations- und Testprozess, vor allem in Embedded Systemen.

Der erste agile Wert lautet „Individuals and interactions over processes and tools". Eben um im täglichen Arbeiten den Menschen und Interaktionen eine höhere Bedeutung einräumen zu können, ist es so wichtig, dass funktionierende Prozesse und Tools vorhanden sind, die ganz natürlich genutzt werden.

Agil klingt innovativ, ebenso wie Scrum mit seinen User Stories, dem Zusammenarbeiten.

Das agile Prinzip, Änderungen willkommen zu heißen, lässt Kunden und das höhere Management in Entwicklungsorganisationen in freudiger Schnappatmung erzittern. Verheißt dies doch die Möglichkeit, ohne sauer dreinblickende Projektmanager auch zwischendrin mal das Ruder rumzureißen, und das Ganze auch noch mit dem Segen des Agilen Manifests und mit einem Framework namens Scrum. Echt jetzt? Wenn Ihre Organisation mit Scrum wirklich erfolgreich sein will, dann gilt es, etwas tiefer zu blicken. Der Prozess ist mit hoher Wahrscheinlichkeit deutlich strikter, als Sie ihn heute leben. Scrum ist außerhalb des Teams auch strikter gegenüber dem Management. Außerdem ist Scrum zu Beginn der Einführung anstrengender als Ihr heutiger Integrations- und Testprozess, vor allem für Embedded Systeme.

Mindset

Über das agile Mindset wurde bereits viel geschrieben, etwa im Agilen Manifest [1]. Die Prinzipien hinter Scrum sind dieselben wie im agilen Manifest beschrieben:
„Wir erschließen bessere Wege, Software zu entwickeln, indem wir es selbst tun und anderen dabei helfen. Durch diese Tätigkeit haben wir diese Werte zu schätzen gelernt:

- Individuen und Interaktionen mehr als Prozesse und Werkzeuge
- Funktionierende Software mehr als umfassende Dokumentation
- Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung
- Reagieren auf Veränderung mehr als das Befolgen eines Plans

Das heißt, obwohl wir die Werte auf der rechten Seite wichtig finden, schätzen wir die Werte auf der linken Seite höher ein." [2]

Oft finden wir gute Ansätze und Motivation vor, dabei werden jedoch ebenso oft in der Rezeption in Teams nur die vier „mehr-als"-Prinzipien in den Mittelpunkt gerückt. Zusätzlich kann es bei der Einführung von Scrum zu Verwirrung, doppelter Arbeit und Unsicherheit kommen, vor allem während der Selbstfindung des Teams,

oder bei der Zusammenarbeit mit Kunden, die selbst noch nicht „agil“ vorgehen, wie in [3] zu lesen.

Die Bedürfnisse von Embedded Software Engineering, eben die starke Bindung an Hardware, mit sehr preissensitiven elektronischen Komponenten und langen Vorlaufzeiten, sind teilweise anders.

Die Probleme beginnen, wenn Teams den Nachsatz „obwohl wir die Werte auf der rechten Seite wichtig finden“ übersehen. Denn die agilen Prinzipien bedeuten eben *nicht*, dass Prozesse, Werkzeuge und Dokumentation jetzt unwichtig sind. Die Präposition im englischen Original hilft weiter: „Individuals and interactions *over* processes and tools”. *Over*, also *über*.

Wenn Individuen und Interaktionen *über* Prozessen und Werkzeugen stehen, dann sind diese *darunter* und bilden somit *die Basis*. Individuen sind uns wichtiger, damit das aber Raum hat, brauchen wir hilfreiche Prozesse und Werkzeuge.
Wenn uns funktionierende Software wichtiger ist, dann brauchen wir *darunter* eben gerade genug Dokumentation, damit jeder weiß wie die bestehende Software und die Entwicklung laufen.

Wenn uns die Zusammenarbeit mit dem Kunden wichtiger ist, dann brauchen wir sinnvolle Vertragsverhandlungen, um eben diese Zusammenarbeit zu haben und gestalten zu können. Wer bei Embedded Software Engineering von automobilen Systemen allen Anforderungen funktionaler Sicherheit gerecht werden will, muss festhalten, bei wem die Zuständigkeiten liegen.

Gerade wenn wir schnell auf Veränderung reagieren wollen hilft als Grundlage ein Plan, in dem die ursprünglichen Idee und daraus folgenden Zielsetzungen festgehalten werden. In der Entwicklung von Embedded Systemen gibt es Rahmenbedingungen, die am besten Platz in einem Plan finden.

Planung ersetzt Zufall durch Irrtum. Und aus Irrtum kann das Team lernen [1].

Das agile Manifest wird oft als Waffe gegen jegliche klassische Methode benutzt. Das agile Manifest jedoch ist viel mehr: Zeigt es doch einen Weg auf, über all den klassischen Vorgehensweisen noch mehr Fokus auf das Gelingen zu legen, als Folge der Zusammenarbeit von Menschen in einer veränderlichen Welt.

Infrastruktur 1 – DevOps
Scrum besagt, das cross-funktionale Team soll quasi autark agieren und sich seine Prozesse und Methoden so definieren, wie es für das Projekt am besten ist. Und ja, mit absolut konstanten Teams kann das tatsächlich funktionieren. In der Realität sehen sich Firmen jedoch einer Fluktuation gegenüber, so dass immer wieder neue Teammitglieder hinzukommen. Oft sind Projekte so groß, dass mehrere Teams daran arbeiten, was die Wahrscheinlichkeit neuer Teammitglieder zusätzlich erhöht.

Wenn das Team regelmäßig funktionierende Software liefern möchte, dann wird es nicht umhin kommen, die Funktionsfähigkeit der Software oft zu prüfen. So rücken „Continuous Integration“ und „Nightly Builds“ in den Fokus. Sie beziehen in einer

entsprechenden Infrastruktur automatisch Code und andere Artefakte aus einem Versionierungssystem, bauen die Software und testen sie ausgiebig.

Obwohl sich nach Scrum jedes Team selbst organisieren und die Infrastruktur selbst bestimmen darf, sollte unbedingt sichergestellt sein, *dass* jedes Team eine funktionierende Build-Infrastruktur nutzt. Teams, die sich keine eigene Infrastruktur geben können oder möchten, sollte eine bestehende zur Verfügung stehen.

Das Erstellen einer Projekt- und Build-Infrastruktur ist nichts, was ein Entwickler eben mal an einem Nachmittag erreicht. Zusammen genommen ergibt die Build-Infrastruktur mit allem, was dazu gehört, die *Development Operations*, kurz DevOps [4]. Ohne Anspruch auf Vollständigkeit benötigt diese:

1. **Versionskontrollsystem**, z.B. Subversion [5], Git [6]
2. **Ticket-System**, z.B. Jira [7], Trello, Leankit, Kanbanize
3. **Build-Server**, z.B. Jenkins [8]
4. **Deployment-Automatisierung**, z.B. GoCD [9]
5. Tool für **Statische Codeanalyse**, z.B. Polyspace [10], Klocwork [11], die sowohl auf dem Build-Server als auch lokal für die Entwickler laufen.
6. Tool für **Code Coverage**, z.B. Bullseye [12]
7. **Testautomatisierung**, Framework je nach verwendeter Technologie
8. **Branching**-Strategie, damit neue Feature-Entwicklungen risikofrei entwickelt werden können [13]
9. **Vollautomatisiertes Reporting**. Niemand sollte manuell Statistiken und Stati zusammentragen müssen.
10. Mehrstufiges **Integrations- und Merge-Verfahren** auf dem Build Server, um Branches zusammen zu führen [14]
11. **Sandbox-System** z.B. Docker [15], damit jeder reproduzierbar dieselbe Umgebung hat
12. **Rollendefinition** für Test Manager, Integration Manager, Release Manager, Build Manager.

Neben den funktionierenden Werkzeugen ist es wichtig, dass das Teammitglied mit dem Wissen über oben genannte Systeme und Verfahren im Projekt bleibt, um Änderungen angemessen adaptieren zu können. Hier wird die Scrum-Forderung nach cross-funktionalen Teams wichtig: Die betreffende Person muss Teil des Projekts sein und bleiben, sonst kommt es immer wieder zu Verzögerungen. Für große Projekte mit mehreren Teams kann ein DevOps-Team tatsächlich die hilfreichste Lösung sein. Um den Problemen der Fluktuation entgegen zu wirken wird so das notwendige Wissen durch ein ganzes Team „Wissender“ sichergestellt. Dieses Dev-Ops-Team muss aber direkt in die Entwicklungsteams eingebunden sein. Denn auch klar ist: Die Infrastruktur muss zuverlässig laufen, es dient als Produktivsystem.

Denn in Scrum gilt: Es gibt keine Testphase. Es gibt nur eine Entwicklungsphase, die Testaktivitäten einschließt.“ [3, p. 228] Selbiges trifft auf Releases der Systemsoftware zu: „Releasing muss zu einem Standardprozess werden, der völlig geräuschlos abläuft.“ [3, p. 230]

Infrastruktur 2 – Embedded Hardware
Eine weitere Herausforderung bei der Erstellung der Infrastruktur wird durch die Frage deutlich, wie wir so schnell wie möglich auf die Zielplattform kommen. Denn „funktionierende Software“ im agilen Manifest muss im Kontext von Embedded Systems eigentlich heißen: „funktionierendes System.“
Embedded Hardware ist etwas spröde in der Handhabung, und das liegt nicht am Material der Platinen. Wer mechatronische Hardware entwickelt, denkt in Vorlaufzeiten für Platinen-Layouts, Lieferzeiten und Verfügbarkeit aktueller Prozessoren.

Ein funktionierendes System muss nicht unbedingt die real existierende Einheit aus Software, Hardware und Mechanik bedeuten. Das funktionierende System kann auch als Simulation aufgebaut sein, mit deren Hilfe sehr frühzeitig der tatsächliche Kundennutzen geprüft wird [16]. Gerade Frameworks wie AUTOSAR helfen dabei, Funktionalitäten im Rahmen einer Simulation zunächst von Hardware getrennt betrachten zu können [17]. Dabei gilt es, Vorsicht vor unüberlegter Modellierung und unbedachter Simulation walten zu lassen. Das Aufsetzen von Simulationen, die keine notwendigen Erkenntnisse bieten, kann viel Aufwand und Zeit verschlingen [18].

Sukzessive werden dann der reinen Simulation in Software Methoden Processor-in-the-Loop (PiL) und Hardware-in-the-Loop (HiL) zugefügt, um konkrete Effekte von Prozessorarchitektur, Steuergerätehardware und mechatronischer Umgebung mit in die automatisierten Tests einzubeziehen. Alle Erfordernisse der DevOps-Infrastruktur gelten auch weiterhin: Nur was automatisiert getestet werden kann, ermöglicht es dem Team, in einem festen Takt einen Stand eines Systems auszuliefern.

Die Architektur des Systems kann dabei ihr übriges tun. Sauber geschnittene Module – sei es als Microservices [19] oder AUTOSAR-Komponenten – erlauben es, diese für sich zu entwickeln, zu testen und auszuliefern.
All diese Frameworks und Verfahrensweisen schränken scheinbar die Freiheit der Teams ein, geben tatsächlich jedoch mehr Freiheit in der Lösung der eigentlichen Aufgabenstellung, vorausgesetzt die Rahmenbedingungen sind bekannt und bleiben relativ stabil.

Manager und Einführung von Scrum
Scrum betrifft nicht nur das oder die Entwicklungsteams. Scrum betrifft die ganze Organisation. Auch wenn nur Teile einer Organisation tatsächlich nach Scrum arbeiten, so sollten doch alle, die mit diesen Teams interagieren, verstehen, was Scrum bedeutet.

Dies gilt auch und vor allem für Führungskräfte. Scrum sieht explizit vor, dass in Sprints gearbeitet wird, also Entwicklungsabschnitten von wenigen Wochen Dauer, an deren Ende jeweils ein lauffähiges System an den Kunden geliefert wird.

Nach jedem abgeschlossenen Sprint werden dann die Anforderungen für den nächsten geplant. Wichtig ist: *Während* des Sprints *bleiben die Anforderungen stabil*. Es kommt also kein Manager herein und gibt den Entwicklern plötzlich etwas anderes zu tun. Gibt es während eines Sprints neue Anforderungen, werden diese in

der Regel nicht in den laufenden integriert, sondern kommen ins Product Backlog. Häufen sich die Fälle, dass für den aktuellen Sprint geplante Anforderungen plötzlich obsolet werden, ist das ein impliziter Auftrag an Scrum Master und Product Owner, sich Gedanken über bessere Anforderungen und deren Abstimmung mit den Kunden zu machen.

Die vornehmliche Aufgabe des Scrum Masters wird damit, das Team von organisatorischen Interrupts von außen zu schützen. Die Führungsriege einer Organisation sollte diese Bemühungen tunlichst unterstützen, indem sie eben *nicht* ins laufende Getriebe eingreift, sondern den Sprint-Takt nutzt, um geänderte Situationen und Anforderungen einzuspeisen.

Die Einführung von Scrum allein über Team-Schulungen reicht damit nicht aus. Es bedarf explizit auch der Schulung der Führungskräfte in angrenzenden Organisationseinheiten.
Agil bedeutet, dass wir die gesamte Organisation verändern müssen, hin zu einer Struktur, die das Projektergebnis ermöglicht und auf Wertschöpfung ausgerichtet ist. Das bedeutet nicht, dass jede Abteilung agile Methoden verwenden muss, doch sollten sie verstehen, was ihre Rolle in der agilen Organisation ist [20].

Die Einführung von Scrum hat einen Anfang, aber durch das Wesen der Selbstorganisation kein Ende. Veränderte Rahmenbedingungen und Projektumfelder werden sich in veränderten Umsetzungen von Scrum niederschlagen. Ihr Manager wird sich daran gewöhnen müssen, dass er keinen statischen Entwicklungsprozess unter sich hat, den er einmal aufgezeichnet bekommt und der dann ewig so bleibt.

Zusammenfassung
Agile Vorgehensweisen im Embedded Software Engineering sind kein Selbstläufer. Den Teams durch gute Infrastruktur ermöglichen, regelmäßig aufwandsarm zu liefern, und ihnen Methoden an die Hand geben, die Belange von Embedded Systems zu berücksichtigen, ist eine gute Basis. Wenn dann auch noch die Manager die Teams in Ruhe arbeiten lassen, garantiert das schon fast den Erfolg.

Wir bei Elektrobit haben sowohl große Teile unserer eigenen Entwicklungsorganisation, als auch namhafte Kundenorganisationen in Embedded Software Engineering erfolgreich durch die Transition auf Agile Vorgehensweisen begleitet und lernen immer noch ständig dazu. Sprechen Sie uns an.

Literaturverzeichnis

[1] J. Sutherland and J. Sutherland, Scrum: The Art of Doing Twice the Work in Half the Time, UK: Random House Business Books, 2014.

[2] W. Cunningham and others, "Manifest für Agile Softwareentwicklung," 2001. [Online]. Available: http://agilemanifesto.org/iso/de/manifesto.html.

[3] B. Gloger, Scrum: Produkte zuverlässig und schnell entwickeln, 5. Auflage Hrsg., München: Carl Hanser Verlag, 2016.

[4] G. Kim, J. Humble, P. Debois, J. Willis and T. Demmig, Das DevOps-Handbuch, Heidelberg: O'Reilly / dpunkt.verlag GmbH, 2017.

[B. Collins-Sussman, B. W. Fitzpatrick and C. M. Pilato, "Version Controll

5] with Subversion," 2017. [Online]. Available: http://svnbook.red-bean.com/index.de.html.

[6] S. Chacon and B. Straub, Pro Git book, US: Apress, 2009.

[7] M. Doar, Practical JIRA Administration, UK: O'Reilly, 2011.

[8] "Jenkins Handbook," [Online]. Available: https://jenkins.io/doc/book/. [Accessed 11 10 2017].

[9] „GoCD: Open Source Continuous Delivery and Automation Server," [Online]. Available: https://www.gocd.org/. [Zugriff am 11 10 2017].

[10] „Polyspace. Making Critical Code Safe and Secure," MathWorks, [Online]. Available: https://de.mathworks.com/products/polyspace.html. [Zugriff am 11 10 2017].

[11] „Klocwork: Source Code Analysis Tools for Security & Reliability," [Online]. Available: https://www.klocwork.com/. [Zugriff am 11 10 2017].

[12] „BullseyeCoverage Code Coverage Analyzer," [Online]. Available: http://www.bullseye.com/. [Zugriff am 11 10 2017].

[13] P. Hodgson, „Feature Branching vs. Feature Flags: What's the Right Tool for the Job?," 23 05 2017. [Online]. Available: https://devops.com/feature-branching-vs-feature-flags-whats-right-tool-job/. [Zugriff am 11 10 2017].

[14] W. Buchwalter, „A Git Workflow for Continuous Delivery," 21 06 2016. [Online]. Available: https://blogs.technet.microsoft.com/devops/2016/06/21/a-git-workflow-for-continuous-delivery/. [Zugriff am 11 10 2017].

[15] "Docker - Build, Ship, and Run Any App, Anywhere," [Online]. Available: https://www.docker.com/. [Accessed 11 10 2017].

[16] J. Schlosser, „Frühe Verifikation von Regelungssystemen mit Model-Based Design," in *Embedded Software Engineering Konress*, Sindelfingen, 2011.

[17] J. Schlosser und G. Sandmann, „Keine Angst vor Autosar – Leitfaden und nutzenbetrachtung für Funktionsentwickler," *ATZelektronik,* Bd. 4, Nr. 2, pp. 66-72, 2009.

[18] J. Schlosser, Architektursimulation von verteilten Steuergerätesystemen, Berlin: Logos Verlag, 2006.

[19] T. Schneider, „Achieving Cloud Scalability with Microservices and DevOps in the Connected Car Domain," in *Software Engineering (Workshops)*, Wien, 2016.

[20] M. Hillbrand, „Agile Methoden skalieren, aber bitte nicht dogmatisch! Erfolgreich skalieren!," *OBJEKTspektrum,* Nr. 2, 2017.

[21] W. Cunningham, „Manifest für Agile Softwareentwicklung," 2001. [Online]. Available: http://agilemanifesto.org/iso/de/manifesto.html.

Autoren

Martin Hillbrand ist seit über 10 Jahren in Software Engineering und Prozessoptimierung in verschiedenen Industrien unterwegs, und unterstützt bei Elektrobit Kundenorganisationen in der Einführung agiler Methoden. Er studierte Software Engineering und Informations-Ingenieurwesen und –Management an der FH Oberösterreich. Martin Hillbrand ist zertifizierter Scrum Master, Product Owner und Scrum Professional.

Dr. Joachim Schlosser führt bei Elektrobit Automotive in München Informatiker und Ingenieure, die Automobilhersteller und Zulieferer zu Agilen Entwicklungsmethoden, Funktionaler Sicherheit und Software Architektur beraten. Nach seiner Promotion in Informatik an der TU München war er unter anderem über 10 Jahre für MathWorks, dem Hersteller von MATLAB & Simulink tätig. Er bloggt auf www.schlosser.info