Erfolgsfaktoren für Scrum in der Produktentwicklung

Praktische Tipps für erfolgreiche Embedded-Software-Projekte

Jens Kinzel, Dr. Jörg-Volker Müller, Systemum GmbH & Co. KG

Agile Verfahren sind darauf ausgelegt, mit unklaren oder wechselnden Kundenanforderungen umzugehen. In der Produktentwicklung gilt es dagegen, ein vorher (relativ gut) festgelegtes Produkt in einer vorgegebenen Zeit zu realisieren. Wie passt dies zusammen? Von den bekannten agilen Verfahren hat sich vor allem Scrum als gleichzeitig flexibles, aber auch verbindliches Gerüst für einen erfolgreichen iterativen Entwicklungsprozess bewährt. Praktische Beispiele aus verschiedenen Projekten zeigen, welche Vorteile Scrum als Entwicklungsprozess bietet, welche Fallstricke immer wieder bestehen und welche Erfolgsfaktoren für die Nutzung von Scrum in Embedded Projekten in der Produktentwicklung gelten.

Embedded Projekte sind doch nicht agil!

Entwicklungsabteilungen mit etablierten Prozessen, in denen die Anforderungen etwa von den Produktverantwortlichen mit einem vorgegebenen Zeitrahmen in die Entwicklung gegeben werden, tun sich häufig schwer, agile Methoden zu adaptieren und umzusetzen. Auf den ersten Blick scheinen die im agilen Manifest (1) genannten Prinzipien nicht auf Projekte und Produkte, bei denen der Fokus weniger auf den Geschäftswert als auf die technische Funktionalität liegt, anwendbar zu sein:

Individuals and interactions over processes and tools: Qualitätsanforderungen einer Embedded Software, gerade im Safety Bereich, verlangen einen definierten Entwicklungsprozess wie das V-Modell. Die Kreativität der Entwickler und die Kommunikation innerhalb des Projektes stehen im Hintergrund.

Working software over comprehensive documentation: Bei der Produktentwicklung stellen technische Dokumentationen einen Teil des Produktes dar. Qualitätsanforderungen verlangen eine umfassende Dokumentation – nicht nur der Anforderungen sondern auch der technischen Konzepte und Lösungswege. Safety-Normen verlangen eine lückenlose Nachverfolgbarkeit von der Anforderung bis hin zum Programmcode. Dass die Software dann auch funktioniert, wird als selbstverständliches Ergebnis angesehen.

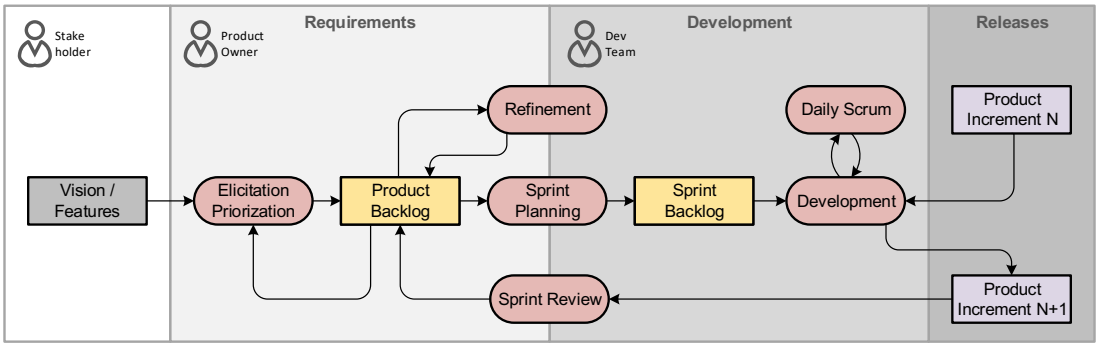
Customer collaboration over contract negotiation: Bei Anwendungen, deren Fokus auf dem Geschäftswert liegt, stehen die Anforderungen des Kunden im Mittelpunkt. Bei Anwendungen mit einem technischen Fokus stehen jedoch vor allem die vertraglich zugesicherten oder vorab geplanten technischen Anforderungen im Vordergrund.

Responding to change over following a plan: Technische Software unterliegt wenigen Änderungen. Sobald sie etwa auf der zugehörigen Hardware installiert und das Gerät ausgeliefert wurde, wird sie über Jahre nicht mehr geändert. Anforderungen ergeben sich durch vertraglich zugesicherte Spezifikationen. In einem Projekt mit gleichzeitiger Entwicklung von Elektronik und Mechanik müssen die Termine der Softwareentwicklung mit den Meilensteinen der anderen Teilprojekte synchronisiert werden. Der Plan zur Realisierung des Gesamtsystems stellt einen wesentlichen Erfolgsfaktor dar.

Obwohl die Forderungen des agilen Manifestes auf den ersten Blick nicht für Embedded Software zu passen scheinen, ergeben sich – insbesondere bei der Anwendung des Scrum-Vorgehensmodells – eine Reihe von Vorteilen. Wird akzeptiert, dass sich Anforderungen und deren Änderungen weniger aus der Fachlichkeit als aus der Technik und der gewählten Architektur ergeben, lassen sich die Methoden sehr gut auf technisch getriebene Software anwenden.

Scrum Basics

Scrum definiert Aktivitäten, Artefakte und Rollen, die hier kurz im Zusammenhang vorgestellt werden:



Vorgehensmodell nach Scrum

Verantwortlich für das Produkt ist der Product Owner (PO). Er verantwortet den Inhalt des Projektes. Als Requirements Engineer erhebt und priorisiert er die Anforderungen der Stakeholder. Diese sind zunächst nur vage in Form von Epics beschrieben. Er verfeinert sie mit Unterstützung des Entwicklungsteams zu User Stories, so dass sie einen für die Entwicklung geeigneten Reifegrad erreichen – überprüft durch die Definition of Ready (DoR) und abgeschätzt durch das Team. Epics und UserStories werden als Product Backlog Items im Product Backlog (PBL) verwaltet.

Die Umsetzung erfolgt durch das Entwicklungsteam in Iterationen mit festgelegter Dauer, den Sprints. Zu Beginn eines Sprints stellt der PO dem Team die zur Umsetzung geplanten PBL-Items vor. Product Owner und Team einigen sich auf die Menge der in der Iteration umzusetzenden Items, diese wird dann nicht mehr verändert. Das Team plant daraufhin den Sprint, in dem es die für die Umsetzung nötigen Aufgaben identifiziert und im Sprint Backlog (SBL) verwaltet.

Während der Entwicklung trifft sich das Team täglich im Daily Scrum, um Feedback über die erledigten Aufgaben zu geben und die nächsten Aufgaben zu koordinieren. Gleichzeitig werden aktuelle Hindernisse, die Impediments, identifiziert, so dass sie vom Scrum Master (SM) gemanagt und nach Möglichkeit aus dem Weg geräumt werden können. Aufgaben gelten als erledigt, wenn die Definition of Done (DoD) erfüllt ist.

Durch die Entwicklung wird aus dem bisherigen Produktinkrement das nachfolgende erzeugt. Das Team arbeitet selbstorganisiert ohne Steuerung von außen. Es committet sich, den vereinbarten Umfang in der vorgesehen Zeit und in der vorgesehenen Qualität zu liefern.

Am Ende eines Sprints wird das neue Inkrement im Sprint Review zusammen mit dem PO begutachtet. Typischerweise werden dabei neue Anforderungen oder Änderungen identifiziert, die ins PBL zurückfließen.

Der gesamte Prozess wird in regelmäßig stattfindenden Retrospektiven gereviewt und ggf. angepasst. Der SM unterstützt als Prozessverantwortlicher bei der Einhaltung der getroffenen Maßnahmen.

Wichtige Merkmale

Das selbst organisierte Team benötigt kein (feinteiliges) Management von außen. Das Team plant die eigenen Aufgaben und kann flexibel auf auftretende Probleme reagieren.

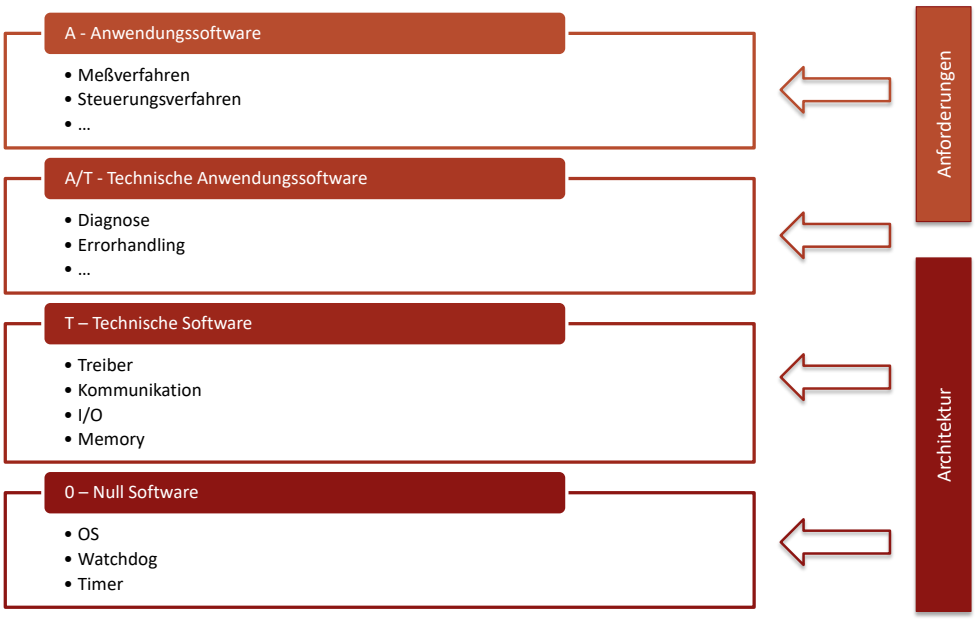
Durch die Organisation der Arbeit in regelmäßige und überschaubare Arbeitsabschnitte erhält der Prozess die nötige Flexibilität, um Feedback einzuholen und auf Änderungen rechtzeitig reagieren zu können. Da der Umfang eines Sprints nicht geändert werden kann (Closed Window Rule), kann das Team zielgerichtet und ohne Störungen von außen an der Lösung arbeiten.

Die vereinbarte Definition of Ready stellt sicher, dass Anforderungen rechtzeitig, vollständig und präzise bereitgestellt werden und dass Missverständnisse vermieden werden. Die Definition of Done stellt sicher, dass Aufgaben wirklich abgeschlossen und dass die geforderten Qualitätsmerkmale eingehalten werden. Nur wenn Aufgaben wirklich abgeschlossen sind, kann ein verlässliches Controlling stattfinden.

Besonderheiten von Embedded Projekten

Wie in der Einleitung bereits motiviert, ist Embedded Software weniger durch Anwendungsfälle und fachliche Anforderungen als vielmehr durch technische und architektonische Anforderungen geprägt.

- Der Funktionsumfang ist im Wesentlichen von vornherein definiert und zum Teil nicht verhandelbar: Die Anforderungen sind wenig agil.
- Der Zieltermin und die Meilensteine stehen fest: Die Anzahl der Sprints ist vorgegeben.
- Anforderungen sind technisch geprägt: Der Business Value spielt bei der Priorisierung keine Rolle, sondern eher die technischen Abhängigkeiten.
- Die Architektur ist der zentrale Treiber für die Umsetzung: Inhalt und Priorisierung des Backlogs ergeben sich maßgeblich aus der Architektur.



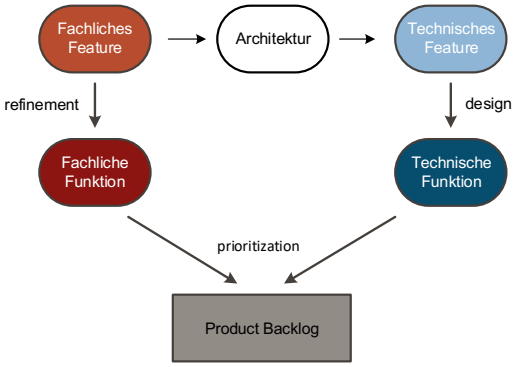
Softwareblutgruppen nach Sindersleben

Die technische Prägung wird deutlich, wenn die Bestandteile der Software den von Siedersleben (2) vorgeschlagenen „Softwareblutgruppen“ zugeordnet werden:

Nur die Anwendungssoftware wie z.B. Mess- und Steuerungsverfahren und zum Teil die technische Anwendungssoftware wie z.B. Diagnose und Fehlerbehandlung wird durch fachliche Anforderungen bestimmt. Der weitaus größere Teil der technischen Software wie z.B. Treiber, Kommunikation, I/O- und Memoryhandling und die sogenannte Null-Software mit den Komponenten des Betriebssystems werden durch die Architektur und die technischen Anforderungen bestimmt.

Dies hat Auswirkungen auf die Aufgaben des Product Owners, der sich mehr um das Design der technischen Funktionen kümmert, als um die Verfeinerung der fachlichen Anforderungen.

Fachliche und technische Anforderungen



Anforderungen der Stakeholder werden zunächst als fachliche Features erfasst (wir verwenden die im technischen Kontext passenderen Begriffe „Feature“ und „Funktion“ statt „Epic“ bzw. „User-Story“). Aus den fachlichen Features und den Qualitäts- und Entwurfsanforderungen wird zunächst die Architektur entworfen. Dazu werden Komponenten und deren Abhängigkeiten auf abstrakter Ebene identifiziert. Aus dieser werden dann die technischen Features abgeleitet. So kann z.B. aus dem fachlichen Feature „sichere Kommunikation“ das technische Feature „zweikanalige Auslegung der Kommunikationskanäle“ abgeleitet werden, wenn zwischen zwei Komponenten Daten sicher ausgetauscht werden sollen.

Der PO wird nun die fachlichen Features weiter verfeinern bis die fachliche Funktionalität soweit konkretisiert ist, dass das Team in der Lage ist, die Funktionen zu implementieren. Typischerweise bezieht der PO dabei Fachexperten und das Team ein. Im Idealfall haben Team und PO eine DoR vereinbart, anhand derer geprüft werden kann, ob eine Anforderung den entsprechenden Reifegrad hat. Wichtig ist hier die Angabe von Test- und Abnahmekriterien.

Wesentlicher sind jedoch die Spezifikation der technischen Features und das daraus abgeleitete Design der technischen Funktionen, da im Embedded Umfeld wesentlich mehr Anforderungen technisch geprägt sind. Hier werden die eigentlichen technischen Entscheidungen getroffen, die über die Eigenschaften und Qualitäten des Produktes entscheiden. Randbedingungen, Normen und Gesetzte wie auch Safety-Anforderungen müssen eingehalten werden.

Refinement und Design erfolgen parallel zur Entwicklung. Dies hat den Vorteil, dass sich technische Schwierigkeiten oder Fehler im Design, die erst während der Entwicklung erkannt werden, korrigiert und in den nächsten Sprints behoben werden können.

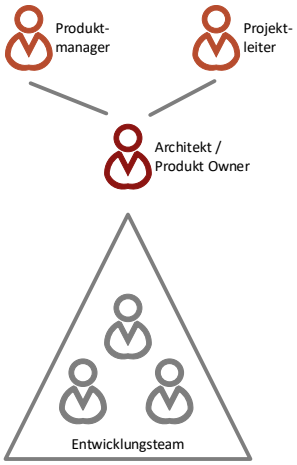
Changes ergeben sich hier nicht auf Grund von Änderungen der fachlichen sondern der technischen Funktionen.

Fachliche als auch technische Anforderungen werden im Product Backlog gemanagt. Die Priorisierung und damit die Reihenfolge der Implementierung erfolgt im Wesent-

lichen aufgrund von technischen Abhängigkeiten, da sich der Geschäftswert der fachlichen Anforderungen nicht ändert – der Funktionsumfang des Produktes liegt im Vornherein fest.

Der Architekt als Product Owner

Da die Hauptaufgabe des POs das Treffen der technischen Entscheidungen und das technische Design ist, entspricht dies dem Aufgabenfeld des Architekten. Dieser hat die Verantwortung für die korrekte Funktionsweise des Produktes und kann Abhängigkeiten und Auswirkungen von Entscheidungen beurteilen.

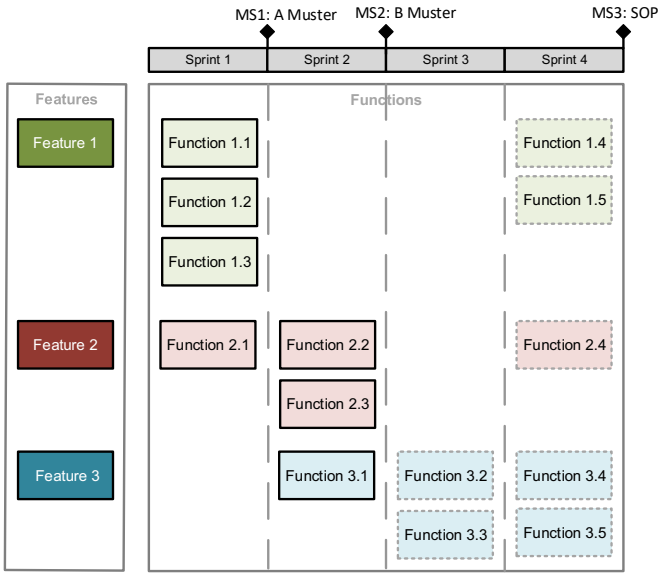


Da der Architekt den organisatorischen und fachlichen Aspekt der Product Owner Rolle typischerweise nicht ausfüllen kann, muss er sich mit dem Projektleiter und dem Produktmanager abstimmen. Diese verantworten i.d.R. die Elektronik- und Mechanikentwicklung und tragen die budgetmäßige Verantwortung.

In den von uns betreuten Projekten, in denen die Rolle des inhaltlich steuernden PO nicht durch den Architekten übernommen wurde, kam es zu projektgefährdenden Problemen: Entweder versuchte der Projektleiter die Aufgaben nach dem Wasserfallprinzip durchzusetzen, ohne technische Entscheidungen und Abhängigkeiten zu berücksichtigen, oder der Fachexperte war mit der Durchdringung der Komplexität der technischen Zusammenhänge (der Architektur) überfordert. In beiden Fällen wurden die nötigen Konzepte, Entscheidungen oder Priorisierungen usw. nicht getroffen, geschweige denn nachhaltig etabliert.

Verzahnung per Story Map

Ziel der Projektplanung ist, die Zwischenschritte für die Erreichung des endgültigen Ziels festzulegen. Für die im Entwicklungszeitraum vorgesehenen Iterationen werden Ziele definiert, die die im Gesamtprojektplan definierten Meilensteine berücksichtigen und mit denen die Software-, Elektronik- und Mechanikentwicklung koordiniert werden. Die Ziele sind so gewählt, dass am Ende einer jeden Iteration eine Teilfunktionalität des Systems fertiggestellt werden kann.

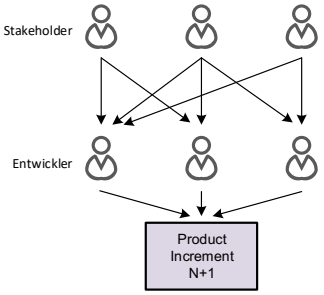


Für die Planung hat sich der Einsatz einer Story Map bewährt. Die geplanten Iterationen werden den aus dem Gesamtplan übernommenen Meilensteinen zugeordnet. Die zu den einzelnen Features zugehörigen Funktionen werden Sprints zugeordnet, so dass die zu den Meilensteinen benötigte Funktionalität fertig ist und die technischen Abhängigkeiten eingehalten werden. Der Aufwand zur Umsetzung der Funktionen muss dabei so weit wie möglich abgeschätzt sein. Bereits identifizierte Funktionen, die bei Planungsbeginn noch nicht genügend detailliert und geschätzt sind, werden späteren Sprints zugeordnet. Vor jedem Meilenstein werden die Sprints nicht voll verplant, so dass die nötigen Puffer entstehen.

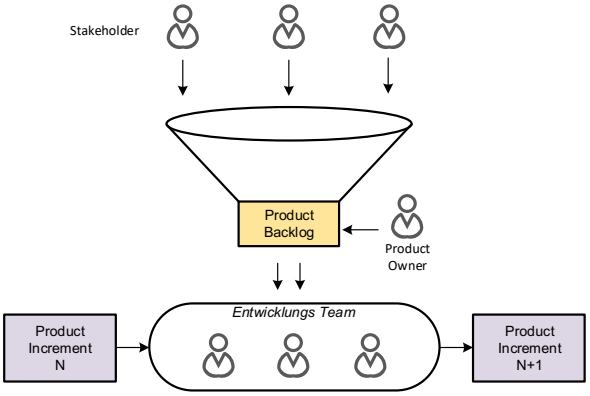
Erfolgsfaktoren von Scrum in der Entwicklung von Embedded-Software

Die folgenden Erfolgsfaktoren haben sich in verschiedenen Projekten als entscheidend für den Erfolg von Scrum in der Softwareentwicklung von Embedded Software im Produkt herausgestellt.

Konsolidierung der Anforderungen – Durch ein einziges Product Backlog mit priorisierten Einträgen können die Anforderungen der Produktmanager koordiniert, nach geschäfts- oder technischem Wert priorisiert, geordnet der Entwicklung zugeführt werden. Das Product Backlog dient hier quasi als „Trichter", um die Koordination der Stakeholder-Anforderungen zu erzwingen. Der PO steht in der Verantwortung Zielkonflikte zu lösen. Nur dadurch ist es möglich, eine konsistente Architektur zu etablieren und die Ressourcen effizient zu planen.



Keine Koordination der Anforderungen



Koordination der Anforderungen durch den Product Owner

Closed Window Rule – Das Team hat sich im Sprint Planning mit dem PO auf die Umsetzung eines bestimmten Funktionsumfangs im Sprint Backlog committet und trägt die Verantwortung dafür, dass die Aufgaben mit der geplanten Priorität zum vereinbarten Zeitpunkt umgesetzt werden. Änderungen am Umfang des Sprint Backlogs und insbesondere der Zugriff von außen auf einzelne Entwickler sind nicht erlaubt, da sie das Sprintziel gefährden würden. Der Scrum Master sorgt für die Einhaltung dieser Regel. Das Team kann sich ungestört auf das Sprintziel fokussieren und so die vereinbarte Funktionalität pünktlich liefern.

Definition of Done – Um den Abschluss einer Aufgabe festzustellen, werden die Kriterien der Definition of Done (DoD) – quasi eine Checkliste mit Fertigstellungskriterien – überprüft. Eine Aufgabe ist erst dann fertig, wenn die DoD erfüllt ist *und* keinerlei Aufwand für diese Aufgabe mehr erforderlich ist. Sind die Aufgaben wirklich fertig, werden potentielle Probleme und Risiken („Um das kleine Problem kümmere ich mich später") früh erkannt und gelöst.

Kurze Iterationen – Durch kurze Iterationen, in denen jeweils ein funktionierendes Inkrement fertiggestellt wird, erhalten das Team und die Stakeholder frühes Feedback, ob die geplante Funktionalität umsetzbar ist und den Anforderungen entspricht. Risiken wie nicht passende Anforderungen, zu hohe Laufzeiten oder Ressourcenverbrauch oder organisatorische Risiken wie fehlende Entwicklungskapazitäten können so früh erkannt und vermieden werden.

Gleichzeitig zwingt das Vorgehen dazu, regelmäßige Integrationstests durchzuführen. Durch geeignete Priorisierung des PBL und Auswahl der umzusetzenden Funktionen

entstehen zunächst vertikale Prototypen, die dann horizontal ausgebaut werden. Das Risiko, dass die Teile am Ende nicht zusammenpassen, wird verringert.

Offene und effiziente Kommunikation – Die Prinzipien des agilen Manifestes (1) zielen auf eine hohe Produktivität des gesamten Teams ab. Die offene, persönliche Kommunikation – unter Einhaltung der Spielregeln – zwischen den Projektbeteiligten schafft Transparenz und Vertrauen. Verschriftlichte Barrieren, wie sie etwa durch Eskalations-E-Mails entstehen, werden überwunden.

Endlose, nur zur Pflicht gewordene Statusmeetings, werden durch kurze tägliche Treffen (Daily Scrum) ersetzt und machen eine effiziente Kommunikation zur Routine. Impediments werden hier transparent gemacht und können strukturiert behoben werden. Regelmäßiges Feedback über Ergebnisse und die Erreichung von Zielen verbunden mit einer verlässlichen Transparenz steigern die Motivation.

Dies alles ist allerdings nicht selbstverständlich und erfordert eine hohe Kooperations- und Kommunikationsfähigkeit der Teammitglieder, die im Zweifelsfall explizit eingefordert werden muss.

Transparenter Projektstatus – Das Team plant seine Aufgaben klar und kleinteilig, sodass diese eine überschaubare Dauer von 1-2 Tagen haben. Durch die kleinen Aufgabenpakete zusammen mit der Definition of Done kann der reale Projektfortschritt mit einer Auflösung von wenigen Tagen gemessen werden, in dem die Anzahl der erledigten Aufgaben bestimmt und in einem Burndown Chart im zeitlichen Verlauf dargestellt wird.

Werden die Aufgaben entsprechend ihrem Status etwa nach „offen“, „in Arbeit“, „fertig“ und „geblockt“ (für Aufgaben, in denen Impediments aufgetreten sind) auf einem Kanban Board visualisiert, kann der Stand des Projektes täglich mit einem Blick erfasst werden, ohne dass ein aufwändiger Projektstatusreport erzeugt werden muss.

Best Practices

Aus der Arbeit in zahlreichen Projekten haben sich eine Reihe von Best Practices im Umgang mit Scrum bewährt:

Up-front-Architektur – Die Architektur eines Softwaresystems leitet sich im Wesentlichen aus den Entwurfsanforderungen (vgl. Nicht-funktionale Anforderungen) ab, in dem sie z.B. langfristig wartbar, sicher oder effizient ist. Es hat sich bewährt, die Entwurfsanforderungen zu Beginn detailliert zu analysieren und daraus die Architektur abzuleiten. Hierfür ist der Architekt des Teams verantwortlich. Die Architektur kann durch ein den Iterationen vorgelagertes Basiskonzept, in einem inkrementellen Vorgehen (mit Iterationen als Architekturprototypen) oder durch eine umfangreiche Envisioning-Phase erfolgen.

Dokumentation von Anforderungen und Architektur – Die Qualitätsanforderungen erfordern in der Produktentwicklung eine durchgängige Dokumentation von der Anforderung bis zum Quellcode. Damit Anforderungen und Architekturkonzepte für Abnahmetest, Changes, Wartung oder eine spätere Migration weiterhin verfügbar sind, müssen die Anforderungen parallel zu den Backlog Items in der Fachspezifikation und der Architekturdokumentation festgehalten werden. Dies erfolgt je nach Technik, in dem die entsprechenden Dokumente angepasst oder in einem Requirements-Engineering-Tool nachgeführt und die Backlog Items verlinkt werden.

Typischerweise erfolgt die Anpassung der Anforderungsdokumentation noch im jeweiligen Sprint durch entsprechende Aufgaben mit entsprechend formulierter DoD.

Gleiches gilt für die Erstellung von Benutzer oder Systemdokumentation. Die Dokumentation des Codes und die Referenzierung der umgesetzten Anforderungen erfolgt als Teil der Entwicklungsaufgabe.

Kompetenzen im Team – Ein gut funktionierendes Team ist die unabdingbare Voraussetzung für ein erfolgreiches agiles Projekt. Neben den schon erwähnten kommunikativen Skills müssen alle Fähigkeiten zur Erstellung eines funktionierenden Inkrements vorhanden sein. Dazu können auch Softwaretester oder Dokumentatoren gehören. Wichtig ist, Entscheidungskompetenzen festzulegen und zu kommunizieren. Rollen und Kompetenzen können zwischen den Iterationen ggf. wechseln.

Umsichtiger Einsatz von Kommunikationsmitteln – Werkzeuge wie Chat-Tools sollten erst eingesetzt werden, wenn die persönliche Kommunikation funktioniert. Menschen, die sich nichts mehr zu sagen haben, werden dies auch nicht mit einem Tool tun, in dem die Nachrichten schriftlich dokumentiert auf Dauer nachvollziehbar sind.

Planung auf Basis kleinteiliger Aufgaben – Entwickler müssen in der Lage sein, ihre Arbeit feingranular zu strukturieren, abzuschätzen und zu planen. Aufgaben, die länger als 1-2 Tage dauern, können entweder in kleinere aufgeteilt werden oder bergen zu viel Risiko, da deren Inhalt zu unkonkret ist und deren Dauer nicht angegeben werden kann. Ein kleinteiliger und meist viel zu detaillierter Projektplan ist nicht nötig – die darin im Vornherein getroffenen Annahmen wären schon nach kurzer Zeit obsolet.

Fertig ist nicht gleich Fertig – Bekannt ist die Weisheit, dass die letzten 20% einer Aufgabe 80% des Aufwands bedürfen. Oder anders ausgedrückt: Eine Aufgabe hat aus Sicht eines Entwicklers entweder den Fertigstellungsgrad 0%, 50% oder 80%. 100% sind aus seiner Sicht erst erreicht, wenn die Software ausgeliefert und in Betrieb ist. Auf dieser Grundlage ist keine solide Planung möglich: a) Es ist nicht bekannt, wann abhängige Aufgaben gestartet werden können und b) der aktuelle Stand des Projektes kann nicht belastbar beurteilt werden. Erst wenn eine Aufgabe keinerlei Aufwand mehr erfordert, ist sie als fertig zu betrachten und zählt als Projektfortschritt.

Elektronik und Mechanik separat – Die Elektronik- und Mechanikentwicklung sollte in eigenen Teams erfolgen, da hier die Zyklen, in denen sinnvolle Inkremente (z.B. Hardwaremuster) erstellt werden können, wesentlich länger sind. Die Synchronisation der Teams erfolgt über die Definition von Meilensteinen, die mit der Sprintplanung zusammenpassen.

Scrum Meetings – Das standardmäßige Vorgehen nach Scrum sieht die Durchführung der Sprint Planning Meetings am Anfang und den Sprint Review und die Retrospektive am Ende vor. Da Sprintende und -anfang direkt aufeinander folgen, führt dies häufig zu wahren Meeting-Marathons. Daher sollten die Meetings nach Möglichkeit entzerrt werden: Die Retrospektive kann separat erfolgen (wobei es häufig günstiger ist, Prozessänderungen am Beginn einer Iteration vorzunehmen). Refinement und Schätzungen des PBL können während des laufenden Sprints erfolgen. Dabei muss jeweils darauf geachtet werden, dass das Timeboxing der Meetings eingehalten wird.

Code Freeze – Es kann sinnvoll sein, kurz vor Sprintende ein Code Freeze einzuführen, um Zeit für die Abschlussarbeiten wie Reviews, Tests und Dokumentation sowie etwaiges Bugfixing zu haben. Somit wird sichergestellt, dass die Arbeiten zum Sprintende auch tatsächlich abgeschlossen werden.

Regelmäßige Integrationstests – Um am Ende jeder Iteration ein fertiges und funktionsfähiges Inkrement zu erhalten, ist es nötig, die fertiggestellte Software zu testen. Entwicklungs- und Modultests können hardwareunabhängig von der Softwareentwicklung durchgeführt werden. Es ist jedoch nötig, auch die Integrationstests am Ende

jeder Iteration vorzunehmen. Damit können potentielle Probleme frühzeitig erkannt und Maßnahmen zu deren Behebung eingeplant werden. Das Risiko des Scheiterns am Ende des Projektes wird so wirkungsvoll vermindert.

Da häufige Integrationstests sehr aufwändig sind, sollten sie so weit wie möglich automatisiert werden. Steht die Hardware zu Projektbeginn noch nicht zur Verfügung, kann mit Simulationen gearbeitet werden.

Umgang mit „Altlasten“ – Ist es nicht möglich, Mitarbeiter in Vollzeit in einem neuen Scrum-Projekt einzusetzen, weil etwa existierende Produkte gewartet oder supportet werden müssen, besteht die Möglichkeit, regelmäßige, feste Zeiträume (Wochentage, Kalenderwochen) zu vereinbaren, in denen alle Mitarbeiter im neuen Projekt arbeiten.

Zusammenfassung

Die fachlichen Anforderungen sind in der Embedded Entwicklung wenig agil – die Entwicklung wird eher durch technische Anforderungen getrieben. Die Architektur und das technische Design dienen als „Fachkonzept“, so dass es sinnvoll ist, dass der Architekt die Rolle des Product Owners übernimmt. Anforderungen werden durch das Product Backlog konsolidiert und der Entwicklung zugeführt, die sich auf die Fertigstellung der Aufgaben zum Sprintende fokussiert. Die Kriterien der Definition Of Done dienen der Qualitätssicherung und sorgen dafür, dass Aufgaben wirklich fertig sind. Die Inhalte der Sprints sind mittels Story Map grob geplant und folgen dem Gesamtmeilensteinplan, so dass die Softwareentwicklung mit der Hardware- und Mechanikentwicklung synchronisiert ist. Durch kurze Iterationen werden Risiken früh erkannt und können vermieden werden. Dem gesamten Prozess ist im Gegensatz zu traditionellen Projektmanagementmethoden eine Transparenz über den aktuellen Status inhärent.

Werden die Besonderheiten der Embedded-Softwareentwicklung bei der Anwendung von Scrum berücksichtigt, ergibt sich eine höhere Effizienz und Transparenz bei beherrschbarem Risiko als bei klassisch durchgeführten Projekten.

Verweise

1. **Beck, Kent und al., et.** Manifesto for Agile Software Development. *Manifesto for Agile Software Development.* [Online] August 2018. http://agilemanifesto.org/.

2. **Siedersleben, Johannes.** *Moderne Software-Architektur: Umsichtig planen, robust bauen mit Quasar.* s.l. : dpunkt Verlag, 2004. 978-3898642927.

3. **Gloger, Boris.** *Scrum - Produkte zuverlässig und schnell entwickeln.* 5. Auflage. München : Hanser Verlag, 2016.

Über die Autoren

Dr. Jörg-Volker Müller ist Gründer und Geschäftsführer der Systemum GmbH & Co. KG. Er ist seit mehr als 20 Jahren in leitender Funktion in der Softwareentwicklung tätig und hat erfolgreich IT-Systeme, Frameworks und Softwareprodukte mit unterschiedlichen Teams realisiert. Seine fachlichen Schwerpunkte sind vor allem Architekturen und Prozesse in der Entwicklung von Embedded Software und IT-Lösungen. Seit 2012 hat er einen Lehrauftrag für Softwaretechnik an der Hochschule Harz.

Jens Kinzel ist Mitarbeiter der Systemum GmbH & Co. KG. Er ist mehr als 20 Jahre als Softwarearchitekt, Requirements Engineer und Teamleiter tätig und hat erfolgreich Softwareframeworks, Individualsoftware und Produkte in unterschiedlichen Branchen konzipiert und in den Markt gebracht.