Viel Scrum herum

Thomas Kestler

Agile SW-Entwicklung setzt sich zusehends durch und mit Scrum wurde eine Projektorganisationsform entwickelt, die agile Entwicklung unterstützen soll. Entsprechend hoch sind die Erwartungen. Ob Scrum diese Erwartungen erfüllen kann zeigt der folgende Artikel.

Der Begriff Scrum bedeutet eigentlich Gedrängel und ist vom Rugby abgeleitet in Anlehnung an die im Daily Scrum Meeting zusammen stehenden Entwickler. Scrum ist eine Methode zur Projektorganisation, kein Vorgehensmodell wie das V-Modell oder RUP. Scrum wird mittlerweilen in vielen Projekten eingesetzt, jedoch in unterschiedlichen Ausprägungen. Scrum eignet sich vom Ansatz her eher für kleinere Teams (7 +/-2 Mitglieder). Das Team soll sich selbst organisieren, ein Scrum-Master ist für die Organisation zuständig und soll das Team unterstützen und Hindernisse beseitigen. Idealerweise ist der Scrum-Master ein Mitarbeiter außerhalb des Entwicklerteams, ggfs. ein externer Berater. Scrum organisiert das Projekt in Sprints mit einer Dauer von 2-4 Wochen. Zu Beginn eines Sprints erfolgt das Planning Meeting, in dem die Anforderungen in Form von User Stories analysiert, in Tasks zerlegt und geschätzt werden (Stories in groben Einheiten, Story-Points oder T-Shirt-Sizes, Tasks in Tagen oder Stunden, max. 2 Tage/Task). Für jeden Sprint wird festgelegt, welche Stories in diesem Sprint umgesetzt werden sollen. Der Projektfortschritt wird täglich im Burndown-Chart visualisiert, welches anzeigt wie viele der Tasks dieses Sprints bereits erledigt sind. Tasks und Stories, die nicht abgearbeitet wurden, gelangen in das Sprint-Backlog und ggfs. in das globale Product-Backlog. Am Ende des Sprints werden die Ergebnisse den Stakeholdern im Review-Meeting vorgestellt. In der anschließenden Retrospektive diskutiert das Team die Ergebnisse des Sprints und leitet daraus ggfs. Konsequenzen für den nächsten Sprint ab.

Ziel der agilen Entwicklung ist es ja, möglichst frühzeitig anwendbare Ergebnisse zu erzielen, also ein lauffähiges Programm, das der Auftraggeber oder Kunde bewerten kann. So fließt frühzeitiges Feedback in die Entwicklung ein. Ob ein Sprint erfolgreich ist oder als gescheitert angesehen werden muss, entscheidet der Product-Owner. Wurden die User-Stories nicht oder unzureichend umgesetzt, erklärt er den Sprint für gescheitert. Dies kann theoretisch sogar während des Sprints passieren, wenn klar wird, dass das Team die Ziele nicht erreichen wird/kann (in der Praxis eher selten).

Das Team organisiert sich selbst, d. h. jeder ist für den Gesamterfolg zuständig und arbeitet an allen Aufgaben mit (keine Spezialisierung, im Idealfall). Das Team legt die internen Regeln und Arbeitsweisen selbst fest, z. B. Verwendung von Tools, Abläufe, etc. Dazu trifft das Team anfangs Vereinbarungen (Commitments) – in der Praxis mangelt es aber leider gerade hier oftmals. Während des Sprints sollten von außen keine Anforderungsänderungen an das Team gestellt werden (Bunkereffekt). Das es zu Scrum genügend Information im Internet gibt, erspare ich dem Leser hier weitere Details.

Scrum ist eine selbstlernende Form der Projektorganisation, d. h. man geht davon aus, dass man mit den Schätzung Anfangs sicher daneben liegt und im Laufe des Projektes durch das Gelernte immer zutreffender schätzen lernt. Bei Einführung von Scrum in ein Projekt sollte man davon ausgehen, dass die ersten beiden Sprints nicht ideal laufen, aber genau dieser Lerneffekt ist ja wertvoll.

Vorteile

Scrum bietet eine Reihe von Vorteilen zur konventionellen, kommandoartigen Projektorganisation. Der wichtigste Vorteil ist die Transparenz über den Projektfortschritt und

die (zunehmende) Genauigkeit der Aufwandschätzungen. Durch die tägliche Visualisierung des Fortschritts und das Herunterbrechen auf einzelne Tasks wird schnell ersichtlich, wenn sich Verzögerungen ergeben (Plan, Do, Check, Act). Daraus können unmittelbar Konsequenzen abgeleitet werden, so können Tasks oder ganze User-Stories frühzeitig in das Sprint-Backlog verschoben werden, um einerseits die Planung realistischer zu gestalten und andererseits Druck unnötigen vom Team zu nehmen.

Die Aufwände einer Anforderung (User-Story) werden durch die Zerlegung in Tasks bewusster und besser ersichtlich. So werden Aufwände sichtbar, die sonst erst während der Entwicklung offenbar werden und die Fertigstellung unvorhersehbar verzögern. Für Querschnittsaufgaben muss ggfs. eine eigene Story angelegt werden und es ist oft erstaunlich, wie viele Aufwände dann dort in Form von Tasks aufschlagen – aber genau das ist der große Vorteil: jetzt wird es publik.

Dadurch, dass das Team die Aufwände selbst schätzt, werden die Schätzungen realistischer als im Fall, dass der Projektleiter von oben herab schätzt (eigene Einschätzung mal zwei). Oftmals führt das anfangs zu Diskussionen mit dem Management, aber wenn dieses lernfähig ist, wird es erkennen, dass das Team durchaus realistisch schätzt (nach anfänglichen Lerneffekten). Auch das ist hilfreich, weil das Management nun realistisch die Kosten der geforderten Features erfährt und ggfs. mit Blick auf das Budget die einst so wichtigen Features nun als Nice-to-have als Ende der Prioritätenliste sortiert.

Durch die täglichen StandUp-Meetings (Daily Scrum) wird die Kommunikation im Team verbessert und jeder erfährt, woran die Kollegen gerade arbeiten und wo Hemmnisse bestehen. Der Scrum-Master sollte allerdings darauf achten, dass diese Meetings kurz und zackig durchgeführt werden, andernfalls steigt der Overhead. Zeichnet sich im Daily-Scrum-Meeting zusätzlicher Diskussionsbedarf ab, wird dazu ein eigenes Meeting anberaumt.

Neben der reinen Kommunikation wird durch die Daily-Scrum-Meetings auch die Wissensvermittlung gefördert, ähnlich dem Teeküchenеffekt. Oftmals reicht hier schon ein kurzer Hinweis aus, wie ein Framwork oder Toll effektiver zu nutzen ist oder einfach die Bekanntgabe der Verfügbarkeit von Tools. Pair-Programming ist eigentlich kein direkter Aspekt von Scrum, kann aber gezielt gefördert werden, in dem bei der Annahmen von Tasks Paris gebildet werden.

Nachteile und Defizite

Scrum hat offensichtliche Vorteile, aber auch einige Nachteile. Zudem entstehen in der Praxis oft auch Defizite bei der Umsetzung.

Ein Nachteil von Scrum ist die Fokussierung auf einzelne Tasks innerhalb eines Sprints und der fehlende Gesamtüberblick auf die gesamte Projektstrecke. Sprint-Backlog und Product-Backlog sind hierfür zuwenig transparent. Das dürfte auch ein Grund sein, warum sich Scrum in anderen Projektarten (Architektur, Fahrzeugentwicklung, etc.) nicht durchsetzt. Oftmals führt die Task-Fokussierung zu Scheuklappendenken bei den Entwicklern, die dann „nur“ Ihre Tasks mit Blick auf das Burndown-Chart abarbeiten und so der Gesamt überblick verloren geht.

Ein Defizit, das in der Praxis daraus entsteht ist die fehlende oder späte Integration oder Akzeptanztests. Mit Blick auf die Tasks entwickelt das Team bis zum Ende des Sprints, um dann am Ende festzustellen, dass das Endergebnis nicht den Erwartungen entspricht.

Auch die ungeregelten Zuständigkeiten führen häufig zu Defiziten, weil sich niemand für Querschnittsthemen wie Architektur, Continous Integration oder Systemtests zuständig fühlt. Hier ist der Scrum-Master entsprechend gefordert. Oftmals ist das aber auch bedingt dorch fehlende oder schwammige Commitments. Hier ein Beispsiel: wenn die nächtlichen

Integrationstests (oder Systemtests) fehlschlagen, muss das neben dem Burndown-Chart sofort visualisiert werden (rote Ampel) und die Ursache muss umgehend behoben werden. Wenn das verschlampt wird, dann kümmert das bald niemanden mehr. Der Scrum-Master sollte hier sofort eingreifen. Ist das Problem bis zum nächsten Tag nicht behoben, sollte der die Entwicklung anhalten, bis das Problem behoben ist. Erfolgt bis zum nächsten Tag keine Behebung, sollte er den Sprint abbrechen.

Die fehlende Führungsposition kann zu gruppendynamischen Effekten führen, die u. U. kontraproduktiv sein können. Der Scrum-Master kann die Führungsfunktion nicht übernehmen und sollte ggfs. den organisatorischen Gruppenleiter einbeziehen. Scrum fördert tendenziell das Aufschieben von Entscheidungen (Prokrastination).

Der Overhead durch Scrum kann erheblich sein, wenn nicht straff organisiert wird. Sieben Entwickler mal 4 Stunden Planning Meeting ist eine Menge Holz. Plus Review Meeting und Retrospektive und das alle 2-4 Wochen. Das Ziel muss daher sein, das Planning Meeting möglichst effektiv vorzubereiten (Tasks vorab schätzen lassen, Meetings kurz halten). Wenn die Entwicklung komplett steht, bis das Planning Meeting erfolgt ist, geht zuviel Zeit verloren. Stattdessen wollten während dieser Vorbereitungsphase Tasks aus dem Sprint-Backlog umgesetzt werden. Weiterhin geht oft Zeit durch Build-Breaks verloren. Hier sollte über eine Entwicklung in Form von Feature-Branches nachgedacht werden (in großen Projekten a la Microsoft ohnehin bereits lange Standard).

Das Sprint-Backlog und das Product-Backlog werden oft nicht ausreichend visualisiert. Meist hat das Team eine sehr sprint-orientierte Sichtweise. Längere Sprints (4 Wochen und länger) bergen die Gefahr, dass zu große Stories aufgenommen werden und damit wieder Eisberge auftauchen wie im Wasserfallmodell.

In der Theorie sollen die Entwickler während des Sprints vor Änderungen von außen geschützt sein. Je besser das erreicht werden kann, desto besser wird der Sprint laufen. In der Praxis lässt sich das aber oft nicht vollends durchhalten.

Erkannte Mängel und Probleme aus Review-Meeting und Restrospektive werde oftmals nicht konsequent beseitigt. Hier ist wiederum der Scrum-Master gefragt. Häufig sind ungenügende Requirements die Ursache für Verzögerungen oder Fehlentwicklungen, auch bei Scrum. Gerade hier besteht die Gefahr von stark verkürzten User-Stories, die als Grundlage für Entwickler nicht ausreichen. Aufgabe des Scrum-Masters ist es, die nötigen Spezifikationen zu besorgen, was aber zu Verzögerungen führen kann. Auch werden die Akzeptanzkriterien seitens der Auftraggeber oft nur unscharf definiert („es soll funktionieren“).

Praxiserfahrungen

In der Praxis werden die ersten Sprints sicher noch mehr oder weniger holpern. Je besser die Vorbereitung und das Commitment von Teammitgliedern und Stakeholdern ist, deso besser wird es laufen. Ein externer Coach kann hilfreich sein, insbesondere wenn noch keine Erfahrung mit Scrum besteht (manchmal auch gerade wenn dem so ist). Was tun, wenn es schlecht lief? Wenn am Ende der ersten Iteration kein lauffähiges Programm steht, am Ende der zweiten Iteration nur ein wackeliger Prototyp, nach der dritten eine Anwendung, die immer noch nicht überzeugt? Plan, Do, Check, Act. Plan und Do sind getan, ob gut oder weniger gut, das sollte Check liefern. Dazu müsste man aber erst einmal objektive Kriterien haben, gegen die man messen kann. Und wenn man dann ein Ergebnis ermittelt hat, wie damit umgehen? Welche Konsequenzen ziehen? Die Konsequenzen können vielfältig sein: Sprintlaufzeiten ändern, Anforderungen genauer dokumentieren, Teammitglieder austauschen, etc.

Gerade wenn man große Stories in einen Sprint gepackt hat, kann es passieren, dass es am ende ein echtes Gedrängel gibt: keiner wird fertig, der CodeFreeze wird mehrfach

geschoben und zum Review ist das Produkt immer noch nicht in der geforderten Qualität verfügbar.

Um eine gleichbleibende Qualität sicherzustellen, sind automatisierte Tests unverzichtbar. Diese decken Regressionen auf, wenn vormals funktionierender Code gebrochen wurde. Agile Methoden wie TDD oder XP stellen Tests in den Vordergrund. Refactoring ist ein zentraler Bestandteil agiler Entwicklung, aber Refactoring ohne Tests in ein riskantes Spiel. Gerade im Zusammenhang mir der Selbstorganisation der Teams ist eine hohe Testabdeckung und ständige Ausführung der Tests ein Muss. Für die Abnahme durch die Stakeholder müssen zudem Akzeptanzkriterien definiert werden, die durch Akzeptanztests überprüft werden (möglichst ebenfalls automatisiert). Es besteht aber die Gefahr des Scheuklappendenkens: jeder Entwickler sieht nur noch auf seine Tests und verliert die Gesamtperspektive aus dem Blick (alle Tests grün, Produkt trotzdem unbenutzbar).

Wie bei allen agilen Methoden besteht die Gefahr, dass die Dokumentation zu kurz kommt. Daher sollte jeder Sprint auch eine Doku-Task am Ende des Sprints vorsehen und die Dokumentation sollte Teil der Akzeptanzkriterien sein.

Fazit

Scrum bietet klare Vorteile, insbesondere in puncto Transparenz und Kommunikation. Aufwände werden auch für das Management wesentlich besser ersichtlich und der Projektfortschritt wird zeitnah dokumentiert und Planung und voraussichtliche Fertigstellung können besser verglichen werden.

Dem gegenüber stehen aber zahlreiche Nachteile und Defizite. Ein hoher Overhead, fehlende Zuständigkeiten, mangelnde Gesamtsicht und weitere, oben genannte, Defizite können den Projekterfolg gefährden.

Kurze, straff organisierte Sprints mit möglichst wenig Overhead sind anzustreben – erreichen lässt sich das aber erst im Laufe der Zeit. Ein erfahrener, externer Scrum-Master kann von Vorteil sein. Ziel könnte eine Art Light-Weight-Scrum sein. Meetings sollten gut vorbereitet sein und sich strikt an Zeitvorgaben halten. Sprint- und Product-Backlog müssen aktuell und für alle Beteiligten einsehbar sein.

Scrum macht keine Vorgaben über Entwicklungsstrategie (z. B. TDD), Vorgehensmodell oder Arbeitsweise (Disziplin). In einem professionellen Team kann Scrum gut funktionieren, in anderen, heterogenen Teams evtl. weniger gut. Ist die Entwicklung auf zwei oder mehr Teams aufgeteilt, fördert Scrum eher die Abschottung anstelle eines integrativen Arbeitsstils.

Das Team muss klare Commitments treffen und diese auch einhalten. Die Einhaltung muss kommuniziert und kontrolliert werden (zumindest anfangs).

Ziel eines jeden Sprints sollte es sein, neuen Geschäftswert zu schaffen („Value Creation“) und sei er noch so klein. Sieben Entwickler mal zwei bis vier Wochen, das ist eine Investition, dafür möchte das Management auch Gegenwerte sehen und es wird das weitere Budget eher freigeben, wenn es greifbare Fortschritte sieht. Andererseits kann es auch nötig werden, die technologische Zeit nach einigen Sprints per Refactoring zu bereinigen. Hier sollten ggfs. Refactoring-Sprints eingelegt werden, evtl. werden daran aber nicht alle Mitarbeiter arbeiten und evtl. ein bis zwei Wochen Urlaub in diesem Sprint nehmen.

Wichtig ist vor allem die Erkenntnis, dass man auch mit Scrum immer wieder in Sackgassen gerät, dies aber schneller erkennt und hoffentlich Konsequenzen daraus ableitet: Plan, Do, Check, Act!