Praktische Tipps zu Ihrer Projektarbeit

Erfahrungsbericht:

Projektmanagement-Erfahrungen bei agilen Projekten im Bereich Operations Support Systems (OSS)

Veränderungen – Herausforderungen – Lösungsansätze

von Thomas Lauffer, PMP

Teil 1

Viele Firmen haben die Notwendigkeit erkannt, in ihrem Geschäft „agiler“ zu werden und sich entschieden sich diesbezüglich zu verändern, häufig unterstützt und ergänzt mit der Einführung von „lean“ Konzepten.

Die agile Transformation tangiert verschiedene etablierte Projektmanagement-Methoden und bedarf einiger neuer Ansätze und Sichtweisen insbesondere im Umfeld von Planungsprozessen. Der individuelle Grad der Veränderung hängt natürlich vom Ausgangspunkt der Transformation ab. Aber die neue „agile Sichtweise“, Interpretation und Erwartungshaltung an den agilen Prozess und insbesondere die Corporate Herangehensweise und der Status Quo der agilen Transformation sind entscheidend für das Arbeiten in agilen Projekten.

Die Gründe, warum man Methoden adaptieren muss, sind daher mannigfaltig. Da eine Transformation sich eher in Jahren als in Monaten bewegt, gilt es zudem in Übergangszeiten beim Aufeinandertreffen von Theorie und Wirklichkeit, Brücken zu schlagen und Lücken zu schließen und das kontinuierlich.

Wenngleich die hier geschilderten Erfahrungen individuell und Domänenspezifisch sind, sind die Problemstellungen typisch für Projektleiter in agilen Projekten und können zu einer Sensibilisierung für mögliche oder ähnliche Probleme und Fallstricke im anderen Umfeld beitragen.

Der Bericht beruht auf konkreten Erfahrungen in agilen SW-Entwicklungsprojekten und ist zweigeteilt. Teil 1 skizziert zum besseren Verständnis die Kernaussagen der lean und agile Konzepte sowie des Scrum-Modells. Dies hilft im späteren Verlauf Problemursachen im Projekt besser einzuordnen, zu verstehen und zu lösen. Teil 1 schließt ab mit der notwendigen Beschreibung der Rahmenbedingungen des Beispiel-Projektes. Teil 2 nimmt sich dann ganz konkreter Problem-Fälle, deren Ursachen und Lösungsansätze an.

1. Motivation für die Veränderung?

Der Anstoß zur Veränderung der Vorgehensweise liegt in unserem Fall in Forderungen des Senior Managements. Sie enthielten folgende Kernpunkte:

- Verkürzung der “**Cycle Time**” von Identifizierung der Kundenanforderung bis hin zur Auslieferung der Funktionalität

Max L. J. Wolf
Dipl. Volkswirt

Buchenstraße 4 A
85716 Unterschleißheim
Tel. 089 / 3107992
Fax 089 / 3104852
E-Mail max.wolf@wolf-pmt.de

- **Höhere Effizienz** in der Produktentwicklung
- **Höhere Flexibilität** in der Produktentwicklung, schnellere Reaktion auf sich ändernde Kundenanforderungen
- Klare **Kundenausrichtung**
- Klare **Ausrichtung nach dem Wert** der Entwicklung
- **Höhere Transparenz** des Projektstandes
- **Bessere Motivation** durch bevollmächtigte Teams

Die oben aufgeführten Ziele sind typisch für Geschäftsbereiche, die aufgrund veränderter Markt- und Wettbewerbs-Situation ihre Prozesse adaptieren müssen.

Hinweise zu möglichen lean & agilen Konzepten sind darin bereits enthalten. Schauen wir daher in einem kurzen Exkurs auf diese beiden Modelle.

2. Exkurs Lean

Interessant ist die Tatsache, dass lean Konzepte anfänglich gar nichts mit SW Entwicklung zu tun hatten.

„Lean“ kommt originär aus einer Benchmark-Studie zwischen europäischen/ amerikanischen und japanischen Automobilherstellern (1990). Einige japanische Hersteller zeigten einen deutlichen Produktivitäts- und Qualitäts-Vorsprung.

Lean fokussiert auf den Produktions-Stil, insbesondere auf den von Toyota (Toyota Production System).

Folgende elementare lean Konzepte aus der Produktion fanden Eingang in die Lean und Agile SW Entwicklung:

- Fokus liegt auf der **Wertschöpfung** für den Kunden
- Verringerung der **Cycle-Time**
- **Einbeziehung aller** in die Prozessdefinition
- Regelmäßige **Arbeitstaktung** (Zyklen), Continous Flow
- Limitierung des „work in progress“ (**Minimierung WIP**)
- Limitierung der Arbeit durch die **Kapazität**

2.1 Lean SW Entwicklung

Für die lean SW Entwicklung lassen sich 7 Lean Prinzipien identifizieren:

1. Eliminierung von **überflüssigen Arbeiten** (z.B. Übergaben, Warten, unnötige Prozesse ➔ Werteflussanalyse)
2. Aufbau nutzbaren Wissens, Weitergabe, **Intensivierung von Lernen**
3. **Qualität in-build** (Fehler erkennen und verhindern, Design der SW)

Praktische Tipps zu Ihrer Projektarbeit

4. **Keine frühen Entscheidungen** (nicht unnötig früh, Informationen zur Entscheidung zusammentragen)
5. **Schnelle Lieferung** (Kundenfeature an den Kunden ➔ Kunden-Feedback)
6. Respektiere die Arbeiter und **bevollmächtige die Teams** (sinnvolle und positive Arbeitsgestaltung, Vertrauen auf die Teams)
7. Betrachte und **verbessere das Gesamtsystem**

Diesen Prinzipien können 5 zentrale lean Konzepte gegenübergestellt werden:

1. Kundenorientierung
2. Wert-getrieben
3. Inkremente
4. Kontinuierliches Hinterfragen und Verbessern
5. Selbstorganisation

3. Exkurs agile

Das Fundament der agilen Entwicklung ist mit Sicherheit das agile Manifest, das 2001 u.a. von Ken Schwaber (➔ Scrum), Kent Beck (➔ XP), Ward Cunningham (➔ Wiki) und Martin Fowler verfasst wurde und bibelgleich 4 elementare Grundsätze festlegt.

1. **Individuen und Interaktionen mehr als Prozesse und Werkzeuge**. - Zwar sind wohldefinierte Entwicklungsprozesse und Entwicklungswerkzeuge wichtig, wesentlicher sind jedoch die Qualifikation der Mitarbeitenden und eine effiziente Kommunikation zwischen ihnen.
2. **Funktionierende Software mehr als umfassende Dokumentation**. - Gut geschriebene und ausführliche Dokumentation kann zwar hilfreich sein, das eigentliche Ziel der Entwicklung ist jedoch die fertige Software.
3. **Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung**. - Statt sich an ursprünglich formulierten und mittlerweile veralteten Leistungsbeschreibungen in Verträgen festzuhalten, steht vielmehr die fortwährende konstruktive und vertrauensvolle Abstimmung mit dem Kunden im Mittelpunkt.
4. **Reagieren auf Veränderung mehr als das Befolgen eines Plans**. - Im Verlauf eines Entwicklungsprojektes ändern sich viele Anforderungen und Randbedingungen ebenso wie das Verständnis des Problemfeldes. Das Team muss darauf schnell reagieren können.

M.E. sollte die Interpretation der Grundsätze derart sein, dass im Konfliktfall die linke Seite einen höheren Stellenwert hat als die rechte. Allerdings begegnet man im Projektalltag nicht selten auch der Sichtweise, dass die rechte Seite vernachlässigt werden kann oder darf bzw. soll. Eine fatale Fehleinschätzung, die Ursache vieler Diskussionen und Probleme in Projekten ist.

3.1 Agile SW Entwicklung

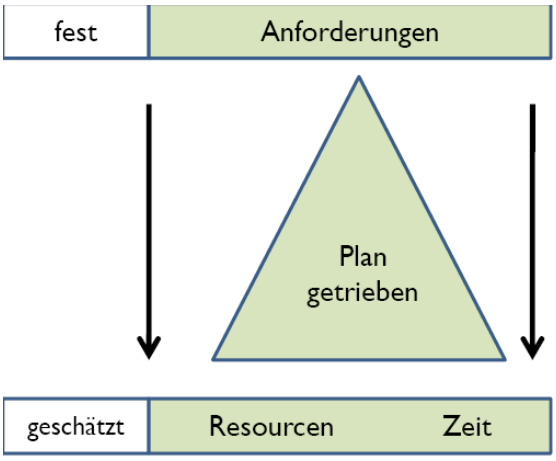
Ähnlich wie beim Lean Ansatz sind 12 Prinzipien für die agile SW Entwicklung identifiziert:

1. Unsere höchste Priorität ist es, den Kunden durch **frühe und kontinuierliche Auslieferung wertvoller Software** zufrieden zu stellen.
2. **Anforderungsänderungen** selbst spät in der Entwicklung **sind willkommen**. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.
3. Liefere **funktionierende Software regelmäßig** innerhalb weniger Wochen oder Monate und bevorzuge dabei die kürzere Zeitspanne.
4. Fachexperten und Entwickler müssen während des Projektes **täglich zusammenarbeiten**.
5. Errichte Projekte rund um **motivierte Individuen**. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie die Aufgabe erledigen.
6. Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungsteam zu übermitteln, ist im Gespräch von **Angesicht zu Angesicht**.
7. Funktionierende Software ist das **wichtigste Fortschrittsmaß**.
8. Agile Prozesse fördern **nachhaltige Entwicklung**. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.
9. Ständiges Augenmerk auf **technische Exzellenz** und **gutes Design** fördert Agilität.
10. **Einfachheit der Realisierung** ist essentiell, keine potentiellen Vorgriffe
11. Die besten Architekturen, Anforderungen und Entwürfe entstehen durch **selbstorganisierte Teams**.
12. In regelmäßigen Abständen **reflektiert das Team**, wie es effektiver werden kann und passt sein Verhalten entsprechend an.

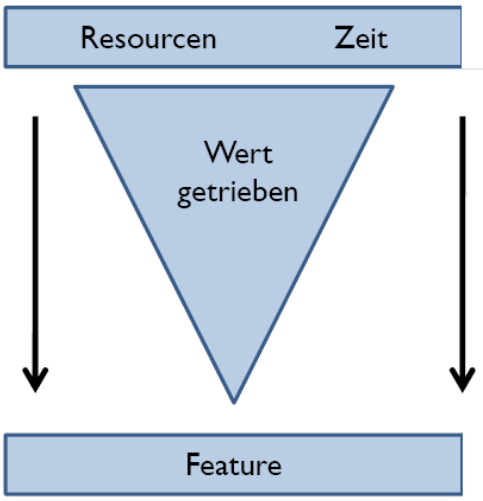
Um das agile Vorgehensmodell zu verstehen, muss man sich noch mal den eigentlichen, zentralen Paradigmenwechsel vergegenwärtigen, der im folgenden Abschnitt erklärt wird.

4. Der Paradigmenwechsel

Das Verständnis der grundsätzlich verschiedenen Vorgehensweisen ist elementar für das Verständnis des agilen Projektmanagements.
Das tradierte Vorgehen in Wasserfallmodellen basiert auf fixierten Anforderungen auf deren Basis Projekte Pläne erarbeitet werden, die die benötigten Ressourcen (Kosten) und die benötigte Zeit (Projektplan) abschätzen. Die Pläne determinieren Ressourcen und Zeit.



Der agile Weg stellt diesen Ansatz gleichsam auf den Kopf. Fest sind in diesem Fall die Ressourcen (feste Teams) und die Zeit (timebox). Feature-Prioritäten zusammen mit Team-Effizienz und konkreter Timebox determinieren die Feature.



5. Exkurs Scrum

Unter den verschiedenen agilen Modellen ist Scrum wohl das populärste Modell. Auch innerhalb meines Programms war Scrum die erste Wahl.
Was ist also Scrum?

- Es ist ein Projekt-Vorgehensmodel.
- Es bringt sowohl lean & agile Konzepte als auch Elemente der “time-boxed” iterativen und inkrementellen Entwicklung zusammen.
- Es erlaubt die schnelle inhaltliche Adaption.
- Es bringt Transparenz zum Projekt Status Quo.
- Es ermöglicht die regelmäßige Bereitstellung von Funktionalität.

5.1 Rollen in Scrum

Scrum kennt genau 3 Rollen, die im Folgenden skizziert werden sollen. Die zuvor beschriebenen lean und agile Konzepte finden Eingang in die Rollendefinition.

Praktische Tipps zu Ihrer Projektarbeit

Der "**Produkt-Owner**" (PO). Er

- ist verantwortlich für die Produktvision, kennt Requirements, definiert Features des Produktes, legt Release Datum fest
- priorisiert die Feature entsprechend ihres "values"
- akzeptiert oder verwirft funktionale Lieferungen aus den Iterationen (Sprints)

Das "**Team**". Es

- ist cross-funktional besetzt (aus Entwicklung, Test, Kudo, ..); ca. 8-10 Mitglieder
- organisiert sich und seine Arbeit selbst
- legt den Sprintinhalt fest; lädt ein zu Demos mit dem PO

Der "**Scrum-Master**" (SM). Er

- sichert ab, dass das agile Modell und Prinzipien befolgt werden.
- lädt ein zu "daily scrum", Planungsmeetings, Retroperspektiven
- unterstützt die Linie dabei, das Team arbeitsfähig und produktiv zu machen bzw. zu erhalten
- vermittelt im Problemfall und beseitigt Probleme ("Impediments") und schützt das Team von jeglichen Einflüssen auf ihre Arbeit

Es sei hier bemerkt, dass die Rolle des Projektmanagers nicht Teil von Scrum ist. Dazu später mehr.
In jedem Fall möchte ich an dieser Stelle mit einem häufig kommunizierten Missverständnis aufräumen: Der SM ist mit Absicht ein Coach und Facilitator ohne jegliche Direktive, er ist mitnichten ein Projektmanager.
Neben der Rollencharakteristik gibt es noch einen weiteren trivialen Grund, dass der SM nicht der PM sein kann. Im Best-Fall gibt es ein 1:1 Mapping zwischen Teams und SMs. Vielleicht übernimmt ein SM in Ausnahmefällen auch die SM-Rolle für 2 oder 3 Teams. Bedenkt man, dass Projekte, wie man später in unserem Beispiel-Projekt sehen wird, leicht aus 2-3 Dutzend Teams bestehen können, ist es nur logisch, dass sich die SM und PM Rolle ausschließen.
Es toben sogar interessante Diskussionen im Netz, ob ein Projektmanager überhaupt ein guter SM sein kann.

5.2 Scrum-Charakteristika

Im Folgenden sind ein paar zentrale Eigenschaften aufgelistet.

- Der PO priorisiert die Anforderungen in seinem "Produkt backlog"
- Die Produktrealisierung erfolgt in Iterationen fester Länge, den sogenannten Sprints (typischerweise 3-4 Wochen)
- Während der Sprint-Laufzeit wird der Sprint-Inhalt nicht verändert.

Praktische Tipps zu Ihrer Projektarbeit

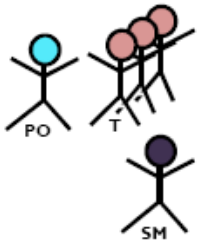
- Es gibt ein Sprint-Planungsmeeting (2-teilig)
  - Klärung der Anforderungen mit PO
  - Einzig das Team plant und committet den Sprintinhalt
- Teams arbeiten “self-organized”
- tägliche kurze scrum meetings
- Sprints enden mit einer Demo der lauffähigen SW (“shipable increment”) und einer Retroperspektive
- Pflege des Produkt Backlogs in “Grooming”-Meetings

5.3 Scrum-Zyklen, Meetings und Artefakte

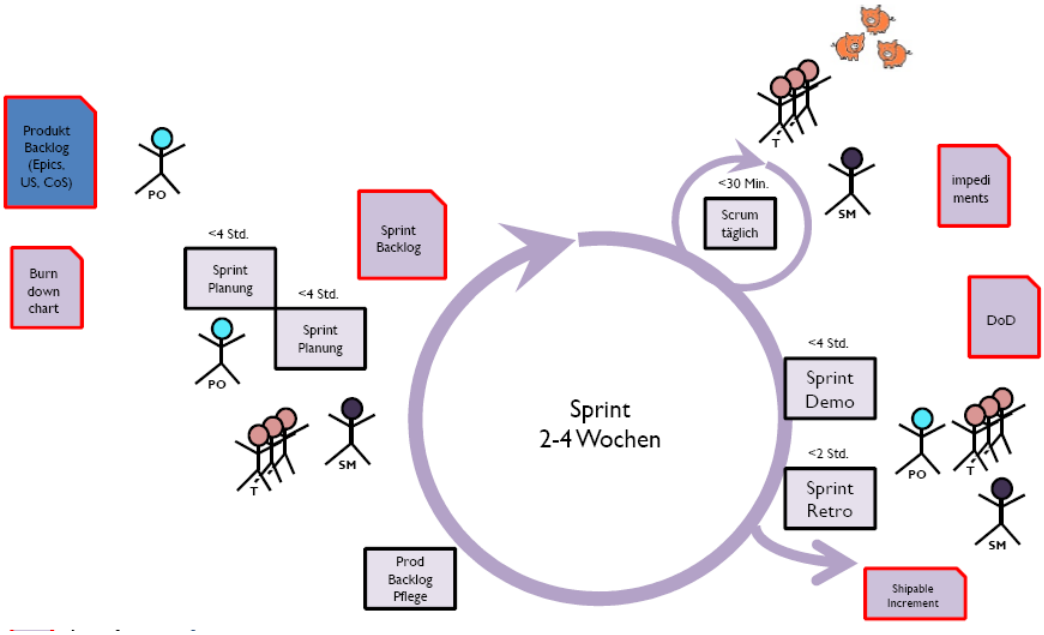
Während eines Sprints gibt es wohldefinierte Aktivitäten mit wohldefinierten Teilnehmern und Artefakten, die in folgender Übersicht erklärt werden sollen:

Die Nomenklatur:

Die Protagonisten: PO, Team, SM



Der Sprintzyklus:



Praktische Tipps zu Ihrer Projektarbeit

Am Anfang steht der PO mit seinem Produkt Backlog. Dieser enthält entsprechend der Produktvision und nach Kundenwert priorisierte Funktionalitäten. Aufgrund von rollierender Planung haben diese Beschreibungen eine unterschiedliche Größe und Reifegrad. Typischerweise werden die Funktionalitäten in sogenannte User-Stories (US) beschrieben. Große Funktionalitäten, die erst noch in weitere US zerlegt werden müssen, werden in sogn. Epics beschrieben. Die Conditions of satisfaction (CoS), die es zu US geben kann und Akzeptanzkriterien beschreiben, komplettieren die wesentlichen Backlog-Elemente.

Neben dem Backlog ist das burn-down-chart ein wichtiges Artefakt. Es zeigt den Fortschritt der fertiggestellten und akzeptierten US und liefert eine Prognose auf Basis der „not-done US“ und der „done-Rate“. Wir werden in Teil 2 noch mal auf diese Metrik zurückkommen.

Ein Sprint hat typischerweise eine Länge von 2-4 Wochen. Zu Sprint Beginn gibt es das zweigeteilte Sprintplanungsmeeting. Im ersten Teil stellt der PO dem Team die gewünschten priorisierten US vor. Alle Fragen des Teams und entsprechende Klärungen durch den PO sollten erfolgt sein, wenn der 2. Teil beginnt, an dem das Team und der SM teilnehmen.

Das Team schätzt Aufwände (häufig relativ mittels Planning-Poker-Methode basierend auf Fibonacci-Zahlen) und identifiziert tasks. Es ist einzig das Team, das sich zu einem Sprintinhalt committet. Die Dokumentation der Tasks erfolgt im Sprint Backlog.

Das Team trifft sich zu einem täglichen, kurzen stand-up Meeting, zu dem der SM einlädt. Probleme, Fortschritte werden angesprochen und neue Task-Zuweisungen erfolgen hier. Vom Team identifizierte Team-Performance-Probleme werden vom SM aufgenommen und im Impediment-Backlog geführt. Es ist zentral SM Aufgabe geeignete Maßnahmen zu treffen oder zu veranlassen um Team-Impediments zu eliminieren.

Am Ende des Zyklus steht die Sprint Demo, bei der das Team den PO einlädt, um ihm die fertiggestellten, lauffähigen US zu zeigen. Es werden nur „done“ US gezeigt, die den CoS genügen und vor allen der „Definition of Done“ (DoD). Letzteres ist ein Dokument, das zwischen PO und Team vereinbart worden ist und das beschreibt, wann eine US „done“ ist. Hier geht es z.B. um Kriterien zur Testabdeckung, Regression oder Dokumentation.

Ebenfalls am Ende des Sprints findet die Retroperspektive statt, in der die Dinge identifiziert werden, die gut gelaufen und auch schlecht gelaufen sind. Korrekturen für den nächsten Sprint i.S. einer kontinuierlichen Verbesserung werden hier initialisiert.

Am Ende des Sprints gibt es eine Software, die akzeptierte US enthält und die im best case an den Kunden gegeben werden kann, um sein Feedback zu bekommen.

Die Pflege des Produkt Backlogs, z.B. Analysen oder Feature-Zerlegungen finden in einem backlog-Pflegemeeting statt, zu dem der PO das Team einlädt - dem sogenannten. Feature-Grooming Meeting.

6. Das Programm

Das konkrete Programm, das agil durchgeführt wurde, sei im Folgenden mit seinen wesentlichen Charakteristika beschrieben.

- SW-Funktionalität
  - Domänen-spezifische Applikationsentwicklung (Managementsystem mit E2E Funktionalität für Transport Netzwerkelemente (multi-vendor))
  - Architekturwechsel auf neue Plattform
  - Basisfunktionalität durch Plattform (FCAPS, Mediation)

- Major Dependencies
  - Plattform-Funktionalität wird parallel entwickelt
  - Netzwerkelemente, Simulatoren (NE BU)

Es sei an dieser Stelle bemerkt, dass die Herangehensweise und der Status der agilen Transformation in den involvierten Bereichen sehr unterschiedlich sein können. Dies führt leicht zu Problemen in der Release-Planung. Dazu mehr in Teil 2.

- Programm-Größe
  - ~ 25 Teams in der Domäne, weltweit verteilt, tlw. virtuelle Teams
  - >>60 Teams in der Plattform, weltweit verteilt, tlw. virtuelle Teams

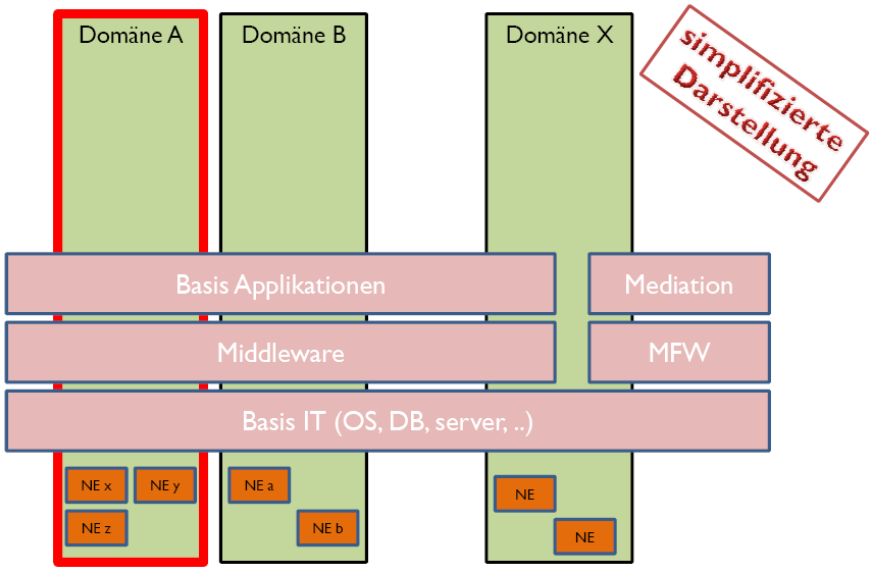
Virtuelle Teams sind verständlich nicht im Fokus des agilen Manifestes und der agilen SW Entwicklungsprinzipien, aber sehr häufig Realität und eine starke Einschränkung.

- Set Up
  - Realisation in mehreren Projekten (continuous)
  - Konkurrierend zu parallelen weiteren Domänen-spezifischen Applikationsentwicklung und der Plattformentwicklung
  - Vorgehensmodel: Scrum
  - Teams <= 10 Mitglieder
  - Sprintlänge 3 Wochen

6.1 Programmabgrenzung

Die folgende Grafik zeigt die Abgrenzung des Programms zur Plattform und die konkurrierende Situation zu weiteren Domänenentwicklungen.

Praktische Tipps zu Ihrer Projektarbeit



Die grünen Rechtecke stellen die parallelen Applikations-Entwicklungen da. Die rot umrandete Box sei unser Beispiel-Programm, das auf bestimmte zu unterstützende Netzwerkelemente fokussiert.

Die rosa Rechtecke stellen Funktionsblöcke der Plattform dar, die von den Applikations-Entwicklungen genutzt werden. Die Funktionsblöcke der Plattform im Detail sind nicht relevant für das weitere Verständnis.

Diese Übersicht schließt den Teil 1 des Erfahrungsberichts und ermöglicht es in Teil 2, sich auf konkrete Projektmanagement-Problemfälle zu konzentrieren.

Teil 2

Der Anfang Juni 2012 erschienene Teil 1 dieses Berichts konzentrierte sich auf die Kernkonzepte der lean und agilen SW Entwicklung, den Paradigmenwechsel, das agile Vorgehensmodell Scrum sowie Rahmenbedingungen des Beispielprojektes. Er stellt gleichsam das Rüstzeug für das Verständnis von Problemursachen als von auch Lösungsansätzen dar.

Dieser Teil 2 fokussiert auf konkrete Situationen im Projekt und referenziert auf Informationen aus Teil 1. Das erste Problem kommt aus der Initialisierungsphase, zwei weitere aus der Planungsphase. Gerade die Planungsphase birgt aufgrund des Paradigmenwechsels eine Reihe von Problemen, die es zu lösen gilt. Ein letztes Beispiel kommt aus dem Monitoring & Controlling. Den Abschluss dieses Artikels bildet ein kurze Zusammenfassung und persönliche Bewertung.

Bevor es in „medias res“ geht, sei daran erinnert, dass es darum geht, mögliche Probleme und Lösungsansätze aufzuzeigen. Vielleicht hat der eine oder andere Leser bereits Erfahrungen in agilen Projekten und ähnliche Situationen und Verhaltensweisen erlebt und zieht einen Mehrwert aus den Lösungsansätzen. Für wieder andere ist das Thema Neuland und die Fälle dienen für eine Sensibilisierung. Mitnichten geht es darum, den agilen Ansatz zu verteufeln. Im Gegenteil, - ohne meiner abschließenden Conclusio vorzugreifen - ich kann meine Affinität zu dem agilen Ansatz nicht verleugnen.

Praktische Tipps zu Ihrer Projektarbeit

1. Beispiel I: „Die Verantwortung und Rolle des Projektmanagers (PM)“

Dieses Beispiel stammt aus der Initialisierungsphase des Projektes. Eine klassische Verantwortungsübertragung i.S. einer Projektcharta findet nicht statt. Die Rolle des Projektmanagers scheint unklar im Dunstkreis eines von der Vorgehensweise unabhängigen Product Lifecycle Prozesses und dem agilen Ansatz.

Was ist verändert?

- Im Kick-Off präsentiert der PO seine Produktvision, key-Features, den Terminrahmen und Ziel-Kunden (grober Projekt-Scope).
- Entsprechend dem agile Konzept/ Scrum, werden keine Planungsvorgaben (Meilensteine) fixiert und das Thema PM-Verantwortung ist ausgeschlossen.

Was ist das Problem?

- Der PM Projektverantwortungsbereich ist unklar.
- Die Abgrenzung der Rolle des PO und PM ist unklar.

Weitere Hintergründe

- Der zentrale Produktentstehungsprozess definiert: Der PM koordiniert und leitet die Produktentstehung über alle cross-Funktionen im Programm. Der PM ist verantwortlich für das Erreichen der Programmziele in Bezug auf Zeit, Budget, Funktionalität, Kosten und Qualität. Das ist alles (theoretisch) unabhängig zum Vorgehensmodell.
- Das Vorgehensmodell Scrum kennt die Rolle des PM überhaupt nicht.

Eine agile Transformation (der Wechsel von klassischer zur agilen SW-Entwicklung) bemisst sich in Jahren und erfolgt nicht über Nacht. Der „Mode of Operation“ (MoO) während der Transformation ist nicht (vollständig) definiert.

Erschwerende Randbedingungen

- Der PO ist überschüttet mit neuen Managementaufgaben (viele ehemalige Themen des PMs).
- Der PO muss häufig erst lernen, mit seiner erweiterten Rolle umzugehen. Er muss verstehen, wie zentral und mächtig diese Rolle ist und wie er Gestaltungsoptionen nutzen kann.
- Das Produkt-Management ist häufig nicht entsprechend aufgestellt. Es fehlen Personen in Ko-Lokation zu den Teams und es fehlt häufig agiles Know-How.
- Agile-Prozess-Hardliner blockieren den Projektfortschritt.

Praktische Tipps zu Ihrer Projektarbeit

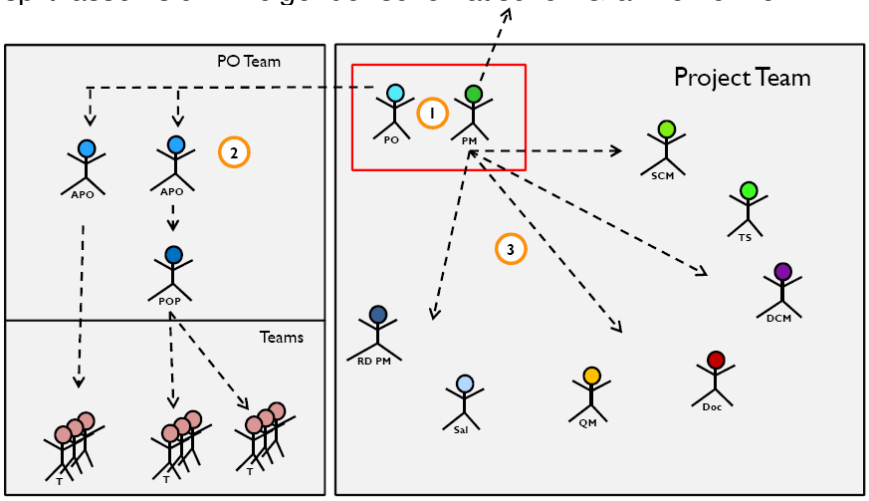
Was war der Lösungsansatz?
Die gewählte Lösung ist simpel und pragmatisch und versucht nicht, eine generelle Antwort auf die Kompatibilität von PLM-Prozess und agiler Vorgehensweise zu geben. Ziel ist es, das zu klären, was unmittelbar notwendig ist zu klären. Es entsprechend zu dokumentieren und in der Projektorganisation zu berücksichtigen. Dazu gehören:

- Definition des worksplits zwischen PM und PO. Dies erfolgte schriftlich für alle wesentlichen Teil-Prozesse und Themen.
- Klärung der Verantwortlichkeiten. Wer treibt was (PM, PO, Teams, etc.)?
- Update des worksplits wann immer notwendig. Diese Updates folgen dem Fortschritt der agilen Transformation.

Allerdings ist zu bedenken:

- Diese worksplit-Definition kann nie vollständig sein (spezifisch, insbesondere fehlende Erfahrung) und es bleiben daher Fragen offen. Z.B. im Beispielprojekt die Realisierung des Risikomanagements für die Gesamt-Release oberhalb des Scrum-Levels. Dies ist dem PM zugewiesen, aber unklar in der Umsetzung. Etwa, wie organisiert der PM diese Aufgabe, wenn die Teams keinen PM kennen?
- Eine gute Kooperation (-Bereitschaft) und hoher Grad an Pragmatismus zwischen PO und PM ist erforderlich.

Der im Projekt gewählte Projekt-Management Team Set-Up und der grobe Rollensplit lassen sich in folgender schematischen Grafik erkennen.



1. Es gibt eine duale Führung des Projektes durch den PO und PM.
2. Das Interface zu den Scrum-Teams bilden der PO und sein „PO Team“. Dies besteht aus einer Hierarchie von „Area Product Ownern“ (APOs) und „Product Owner Proxies“ (POPs). Das Priorisieren der Scrum-Teamaktivitäten (Wert-getrieben) und alle anderen PO-Aufgaben in Scrum erfolgen über dieses Team. In unserem Beispielprojekt war dies aufgrund der vielen Scrum Teams und des weltweiten set ups ein vergleichsweise sehr großes Team.
3. Der PM übernimmt die cross-funktionale Koordination der Aktivitäten und Prozesse mit Service, technischen Support, Produktion/ Delivery, Sales, Dokumentation etc. Darüber hinaus übernimmt er das externe Projekt-Reporting.

2. Prozess Hardliner

Einem zu erwartenden Phänomen, das Ursachen einiger Probleme in vielen Aufgabenbereichen und Grund für intensive Diskussionen war, sei hier ein eigener Abschnitt gewidmet.

Es gibt eine Gruppe von Stakeholdern, die die Ursache für Probleme im Projekt darin sehen, dass der aktuelle Grad der Agilität unzureichend ist und man daher sein Heil darin suchen sollte, noch strikter agil (entsprechend der Theorie) zu werden. Dass es eine derartige Haltung gibt ist ohne Vorwurf und eher natürlich. Wichtig ist es, sich darauf einzustellen.

Diese Leute kommen typischerweise aus dem R&D-Umfeld und nehmen eine sehr polarisierende Haltung im Projekt ein.

Die Ansichten sind teilweise

- richtig i. S. von “Scrum nach der Theorie”, aber aktuell nicht hilfreich für die laufenden Projekte.
- falsch aufgrund von Miss-Interpretationen agiler Konzepte.

Der Fehler, der hier leicht gemacht ist, ist den momentanen Status der agilen Transformation und Möglichkeiten der Organisation zu ignorieren. Dies führt leicht zu einer Gefährdung der Projektziele und steht der Aufgabenstellung des PM völlig entgegen.

Ein paar Beispiele aus dem Projekt:

- “Der PM spricht nicht mit den Teams.”
- “Die „Definition of Done“ (DoD) wird einzig zwischen den Teams und dem PO vereinbart.”
- “Ende mit „Comand und Control.“
- “Keine Metriken, die die Teams beeinflussen.”
- “Was nicht priorisiert mit Wert-Nachweis im Backlog steht, wird nicht vom Team gemacht.”
- “Wir planen nur den nächsten Sprint.”
- “Release-Planung ist nur eine Illusion. Immer schon gewesen. “
- “Was wir nicht können, können wir nicht. Priorisiere!”

Die Beispiele zeigen eine fatale Best-Effort Sichtweise und vor allem ein sich Nicht-verantwortlich-Fühlen. Aber genau diese Verantwortungsübernahme durch die Teams ist eines der zentralen Elemente der agilen SW Entwicklung.

Die Konsequenzen:

- Der Fokus liegt primär auf Umsetzung des agilen Prozesses, weniger auf dem operativen Geschäft.
- Eine pragmatische Lösungsfindung für den PM ist grundsätzlich erschwert.

Der PM ist daher gut beraten, i.d.R. seiner inhärenten Kommunikationsaufgaben diese Stakeholder zu identifizieren und sich ein geeignetes Stakeholder Management zu überlegen.

Eine Guideline, die im Beispielprojekt sehr hilfreich war, war der Gedanke des „controlled agile“. Versuche das an Agilität zuzulassen, zu dem die Organisation (insbes. die Teams) tatsächlich aktuell fähig und willens sind und was dem Projekt förderlich ist. Vermeide dadurch eine Gefährdung der operativen Projektziele. Beachte sorgfältig den Stand der agilen Transformation.

3. Beispiel II: Release-Planung („water scrum“)

Dieses Beispiel aus der Planungsphase zeigt ein Problem auf, das sich daraus ergibt, dass man einerseits die angenehmen und gewünschten Vorteile des agilen Ansatzes (etwa die leichte Einbringung von Scope-Changes) nutzt, aber andererseits noch eine Erwartungshaltung aus wasserfallbasierten Vorgehen pflegt (etwa die Qualität der Release-Planung)

Was ist verändert?

- Tradierte planerische Aktivitäten entfallen (Scope Baseline, WBS, Ressource Planung, Schedule Baseline, etc.)
- Planungen sind auf einen wesentlich kürzeren Horizont ausgerichtet und erfolgen kontinuierlich. (just in time)

Was ist das Problem?

- Was macht der PM in bezug auf den von ihm erwarteten „Projekt-Contract“ und die Release-Planung?
  - Der „Projekt-Contract“ spiegelt die Balance wider zwischen dem, was Produkt-Ziel ist und dem, was das Projekt (bzw. die Organisation) leisten kann und wird früh im Projekt erwartet.
- Wie kann man den „Contract“ und die damit verbundene Release-Planung sinnvoll gestalten?

Weitere Hintergründe
Der PO und Sponsor brauchen i.d.R. eine frühe, stabile Aussage zu Lieferterminen und -Umfang. Verträge (insbes. die Fixierung von Pönalen) basieren auf Kunden-Roadmaps mit klarer Funktionalität und festen Release-Terminen in einem mittelfristigen Zeithorizont (häufig 1.5 - 2 Jahre oder sogar mehr). Das schließt die Aussagen zu internen NE/ Solution Roadmaps mit ein.

Man stellt jedoch erschwerend fest:

- Die Agile Transformation ist nicht umfassend. Sie schließt insbesondere nicht alle Endkunden mit ein. „Agile Argumentation“ ist hier nicht zielführend.
- Planungsaussagen gegenüber internen (nicht agilen) Kunden sind nicht ausreichend.

Die Frage bleibt: Release-Planung – ja, aber wie? Dabei erschweren die agilen Charakteristika und die eigene Erfahrung mit einer agilen Release-Planung das PM-Leben:

- keine stabile Basis, rollierende Planung (sukzessive Analyse, Feature-Grooming).

Praktische Tipps zu Ihrer Projektarbeit

- kein generelles Konzept und historische Erfahrung für die agile Release-Planung.
- Umgang mit „Uncertainties“: Aufwandsabschätzungen sind unterschiedlich genau, die „Velocity“ ist Team-spezifisch und nicht konstant.
- Berücksichtigung (externer) Abhängigkeiten.
- Zulieferungen (aus z.B. Plattform) erfolgen teilweise selbst agil und sind einem Neu- bzw. Um-Priorisieren unterworfen.

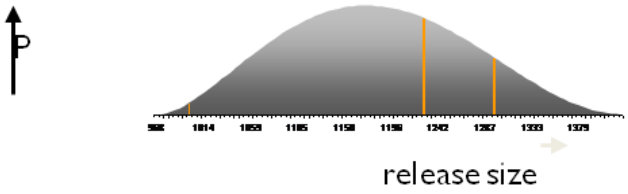
Wie kann man das Gesamt-System aus Plattform & Applikation betrachten und konsistent ausrichten bzw. priorisieren?

Was war der Lösungsansatz?
Der erste wichtige Schritt war zu erkennen, dass wir es mit vielen unterschiedlich ungenauen Ereignissen und Parametern zu tun haben. Aussagen zu Aufwänden besitzen eine individuelle Verteilung/ Konfidenz. Dies gilt dito für die Velocity. Beide Elemente sind Team-spezifisch.

Der weitere Gedankengang war: Akzeptiere die Charakteristika einer agilen Vorgehensweise sowie deren Konsequenzen (das ist ohne negative Wertung gemeint) und spiegele das in der Release-Planung und dem „Contract“-Diskussionen mit dem PO wider. Wenn das gewählte Vorgehen (aus vielen guten Gründen) agil ist, dann kann man andererseits keine Wasserfall-ähnlichen Aussagen vom PM erwarten.

Ein wahrscheinlichkeitstheoretischer Zugang erschien adäquat (der sicherlich auch auf andere Vorgehensmodelle übertragbar ist).

In einem ersten Schritt wurde kaskadiert für Teams eine Normal-Verteilung für Aufwand und Velocity berechnet.



Anschließend wurden die Verteilungen in Relation gesetzt. Dies konnte in 2 Varianten erfolgen. Die wichtigere war sicherlich:

- Errechne Konfidenz-Intervalle für den Inhalt bei fixem Zieltermin (z.B. 1. August). Wenn also n Sprints vorgegeben sind, weise aus, mit welcher Konfidenz welche Backlog-Elemente am Zieltermin verfügbar sind:

- Variante 2 berechnet umgekehrt die Konfidenzen für die Anzahl von Sprints bei gegebener, fixen Menge von Must-Feature.

Der Ansatz hat natürlich einen „Snapshot"-Charakter. D.h. er basiert auf dem aktuellen Backlog-Stand (Prioritäten, Erkenntnissen zu Aufwänden) und aktuellem Stand und derzeitiger Prognose zu den Teams. (Anzahl Teams, Velocity) Er kann als vernünftige Basis für die „Contract-Diskussion" mit PO und Sponsor dienen, um eine Release-Planung inkl. einer gemeinsam akzeptierten Konfidenz zu fixieren.

Letzteres bedeutet einen Vorteil gegenüber simpleren Ansätzen, bei denen z.B. der Release-Backlog nur zu einem bestimmten Prozentsatz gefüllt wird.

Der Lösungsansatz zeigte jedoch auch Schwächen im konkreten Projekt, da es schwierig ist, komplexe Plattform-Abhängigkeiten zu modellieren und in die Release-Aussagen zu integrieren.

4. Beispiel III: Release-Planung (Skalierung)

Dieses Beispiel stammt ebenfalls aus der Planungsphase und zeigt ein Problem auf, das sich in großen, komplexen Projekten ergibt.

Das agile Vorgehen erscheint ohne große Probleme bei Projekten mit 2-3 Teams. Es ist aber ungleich komplexer, wenn - wie im Beispielprojekt - mehr als 25 Teams involviert sind und es eine starke Dynamik aufgrund der parallelen Plattformaktivitäten (mehr als 60 weitere Teams) und sich verändernder Produkt-Prioritäten ergibt.

Was ist das Problem?

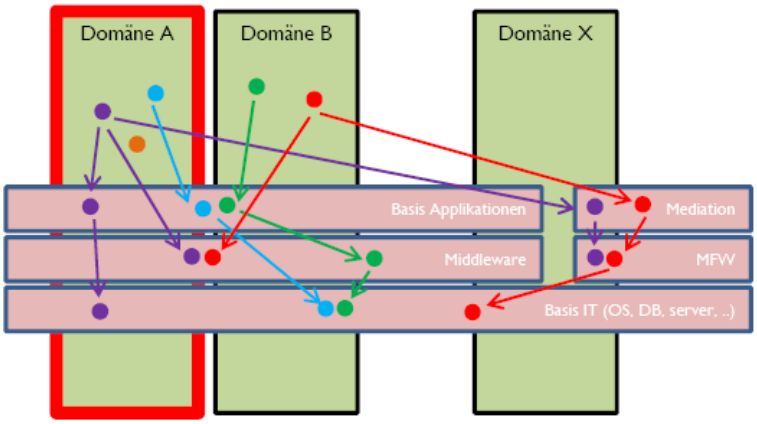
- Wie kann eine stabile Basis für die Planung mit den Teams geschaffen werden?
- Wie kann der agile Ansatz skalieren (sehr viele Teams, viele POs aus unterschiedlichen Domänen)?
- Wie kann das Domänen-übergreifende Priorisieren erfolgen?

Weitere Hintergründe

- Die Plattform ist selbst in Entwicklung.
- Die Plattformanforderungen aus den Domänen sind sehr unterschiedlich.

Die Problemstellung sei anhand der folgenden Skizze ausgewiesen. Das Beispielprojekt (etwa Domäne A) hat viele Abhängigkeiten zu Entwicklungen in verschiedenen Bereichen der Plattform. Da eine Plattform per Definition mehrere Nutzer hat, stehen diese in Konkurrenz zu Anforderungen von weiteren Domänenentwicklungen.

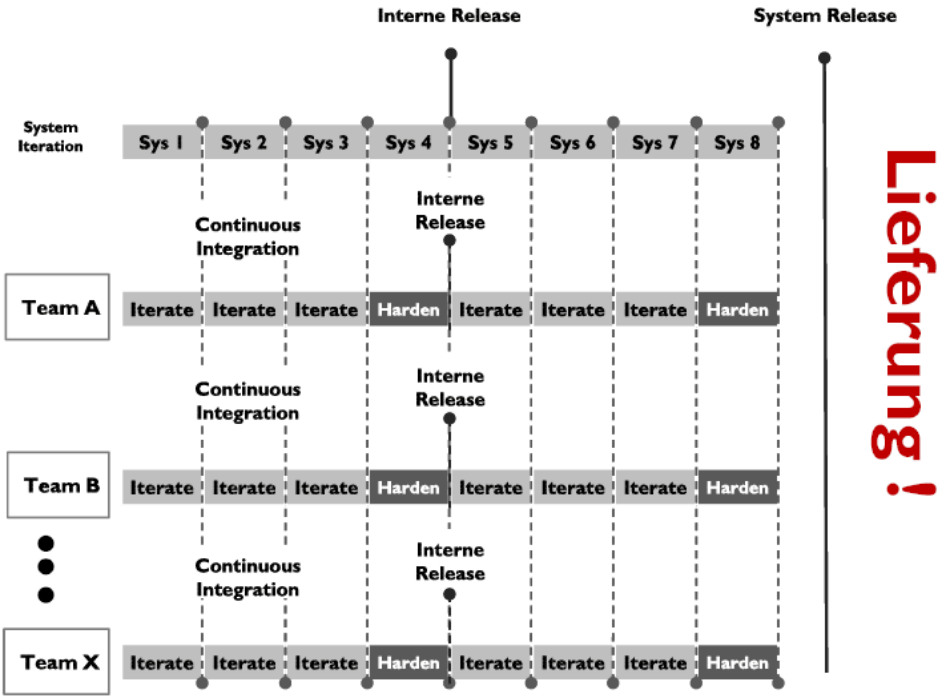
Praktische Tipps zu Ihrer Projektarbeit



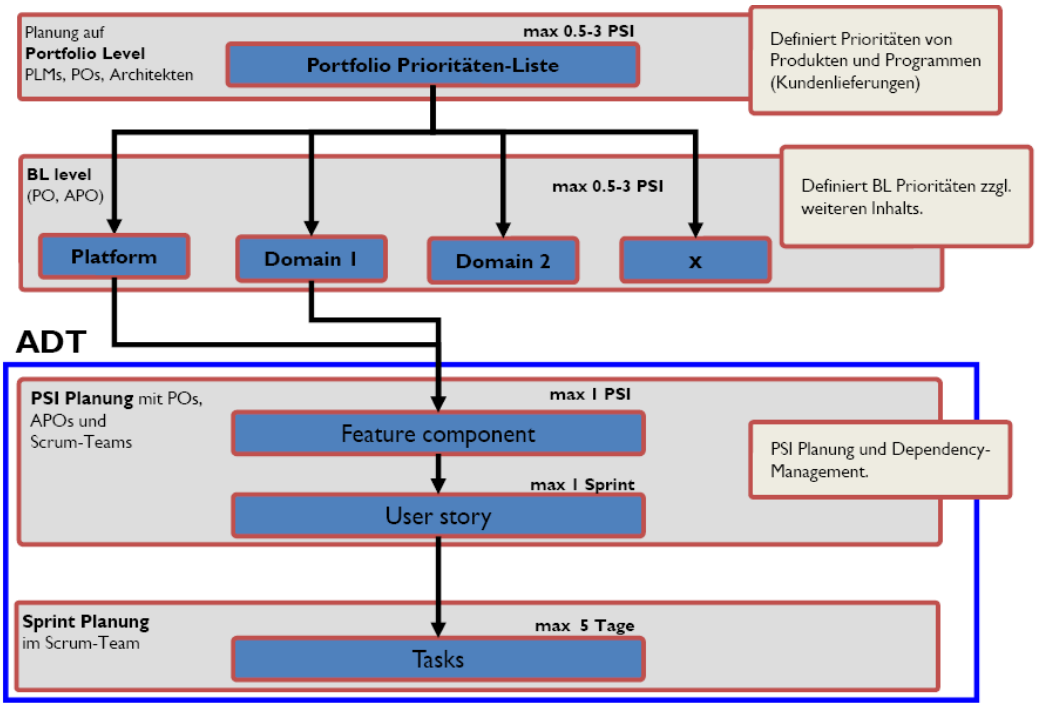
Was war der Lösungsansatz?
Der Domänen-übergreifende Ansatz war der sogenannte „Agile Development Train“ (ADT), der auf dem „Agile Release Train“ (ART) von Dean Leffingwell basiert.

Die Grundelemente sind:

- Time-boxed mega-Sprint (2-3 Monate).
- bestehend aus Entwicklungs- und Stabilisierungs-Sprints.
- fester interner Rhythmus von Sprints (Kadenz).
- Teams sind cross-funktional besetzt.
- alle involvierten Teams planen und comitten sich zur ADT-Planung.
- ADT produziert ein Potential Shippable Increment (PSI).
- keine oder minimale inhaltliche Änderungen.
- der work in progress (WIP) ist limitiert auf den ADT Inhalt.
- Variable ist der Scope, nicht die Zeit.



Das Domänen-übergreifende Priorisieren aller Aktivitäten inklusive der Plattform erfolgte durch folgenden top-down Ansatz:



Erfahrungen mit diesem Ansatz
Es zeigten sich einige problematische Aspekte in der Arbeit mit dem ADT:

- Breite Hierarchie im Produktmanagement (PO-APOs-POPs)
- Komplettierung des PO-Teams durch RD; Alignment schwierig
- Skalierung in den Teams durch „Scrum of Scrums“ (SoS) oder „Scrum of SoS“ (SoSoS) tlw. suboptimal und nicht effektiv
- Sehr komplexe ADT-Planungsmeetings
- Planung und ADT-Inhalt waren letztlich nicht stabil (over-Commitment, Dependency-Mgmt. schwierig)
- Kein Komplett-SW-Build, stattdessen: schrittweise Integration von Plattform und Domäne (mehrere Integrations-Punkte)
- Regressionsprobleme in der Plattform, insbes. in Bezug auf Domänen-spez. Funktionalität.
- Plattform ist selbst in der Entwicklung.
- Klare Architekturvorgaben für Plattform und Applikationen.
- Integration von nicht klar Wert-zuweisbaren User Stories kritisch (z.B. Architektur-Feature, Performance-Feature).
- Änderungen des Portfolios bzw. Portfolio-Prioritäten zwischen den ADTs.

Der ADT wurde final nach mehreren ADT-Zyklen als vorerst nicht adäquat für das Umfeld identifiziert und durch einen weiteren Ansatz („virtual feature teams“) ersetzt.

5. Beispiel IV: Release-Controlling (Metriken)

Das letzte Beispiel kommt aus dem Monitoring & Controlling und fokussiert auf die Frage, welche Metriken sinnvoll sind.

Praktische Tipps zu Ihrer Projektarbeit

Was ist verändert?

- Tradierte Metriken, die in irgendeiner Art und Weise die Konformität zu einem Plan bewerten, sind nicht passend, da im agilen Ansatz die Veränderung (Change) im Vordergrund steht.
- Metriken, die Wasserfall-basierte Annahmen haben, sind nicht passend.
- Beispiele: MTA, Abweichungen zu Aufwandsschätzungen, Fehlerfindungsprofile, Anforderungs-Änderungsrate, ….

Was ist das Problem?

- Viele „tradierte“ Metriken sind nicht anwendbar.
- Welche Metriken können im Projekt sinnhaft benutzt werden?
- Wie kann ich frühzeitig verstehen, was ich zum Release-Datum erwarten kann?
- Wie berücksichtige ich die kontinuierliche Veränderung?

Weitere erschwerende Randbedingungen

- Veränderung ist eine zentrales unterstütztes, gewolltes Konzept (➔ agiles Manifest)
- Planungen in die weitere Zukunft können nicht als Referenz benutzt werden ➔ „Cone of uncertainty“
- Messungen um Teams und Zielsetzungen können sehr leicht zu selbsterfüllenden Prophezeiungen führen. Z.B. eine höhere Burn-Down Rate (d.h. mehr Story-Points pro Sprint) von einem Team zu verlangen, ist für das Team trivial zu erreichen (das Team schätzt). So hat dies keinen positiven Effekt für das Projekt.

Was war der Lösungsansatz?

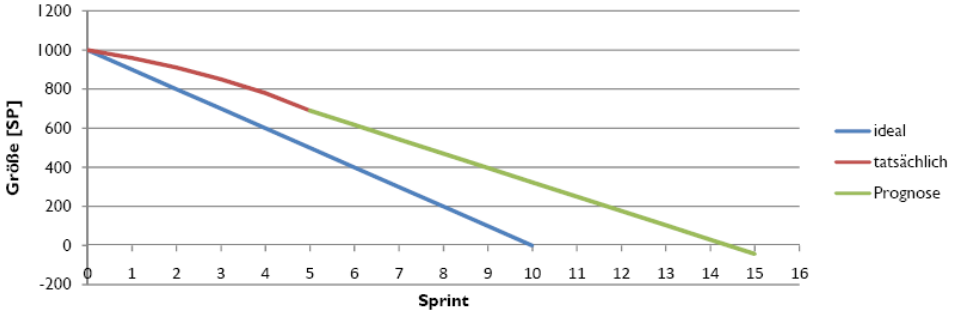
- Nutzung simpler Metriken
- Messung direkter Ergebnisse
- Vorsicht bei Extrapolationen

Im Folgenden seien einige Metriken mit ihren Charakteristika und Vor- und Nachteilen vorgestellt.

A. Das Release burn down chart

In dieser Form ist es wohl der Klassiker, der auch in vielen Büchern referenziert wird.

- Basis ist der Product-Backlog
- zeigt “done” und “not done” Story-Points
- Prognose auf Basis der durchschnittlichen Velocity



Problematisch:

- Welche durchschnittliche Velocity wird für die Prognose genommen?
- Wie wird man der Tatsache gerecht, dass es (häufig) Entwicklungs- und Stabilisierungs-Sprints gibt?

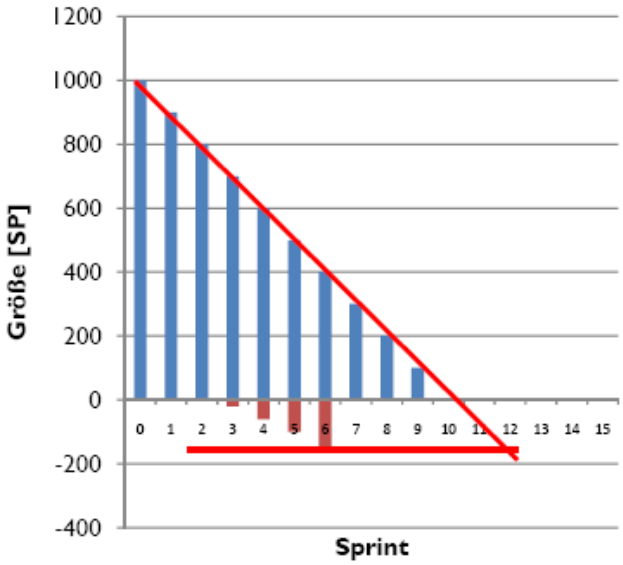
Eine Erfahrung aus dem Beispielprojekt:

Der Trend ist wichtiger als absolute Werte. Dies gilt für alle vorgestellten Metriken.

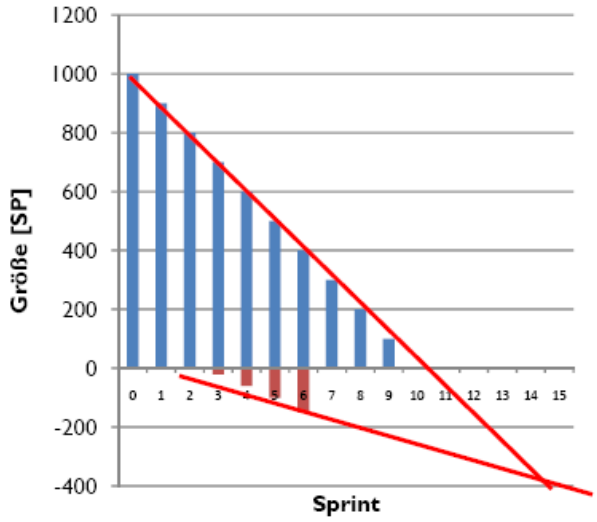
B. Das Release burn down chart (2 Varianten)

Die folgenden Varianten erlauben die Ausweisung und Berücksichtigung von zusätzlichen Anforderungen.

Hier ohne Berücksichtigung der Änderungsrate:



Hier mit Berücksichtigung der Änderungsrate:



Problematisch:

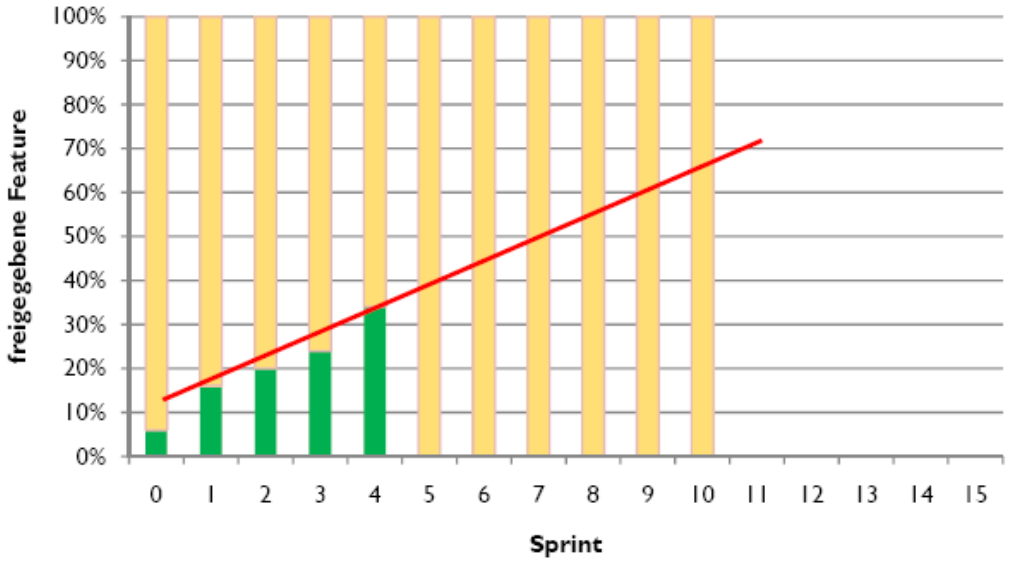
- Die Aussagekraft bzgl. dem Trend wird immer ungenauer.

Praktische Tipps zu Ihrer Projektarbeit

C. Anzahl freigegebener Feature

Diese Metrik verfolgt die Entwicklung von freigegebenen Feature:

- Feature in Prozent, die Freigabestand erreicht haben (inkl. evtl. nachgeschalteter Stabilisierungs-/ Test-Phase)
- mit Prognose



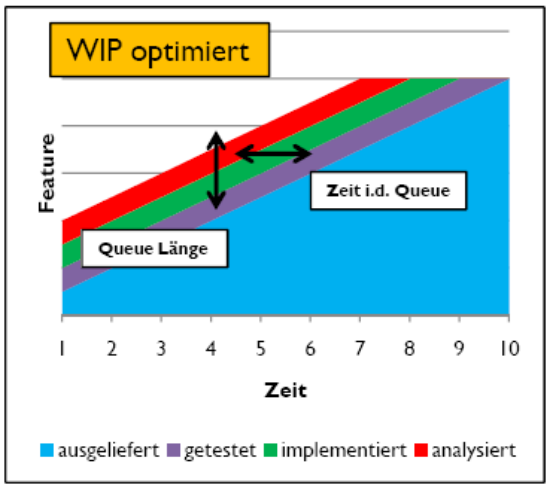
Problematisch:

- unterschiedliche Feature-Größen und Komplexitäten sind nicht berücksichtigt.
- Der Wert der Feature ist nicht berücksichtigt.

D. Kontrolle der Queue-Längen

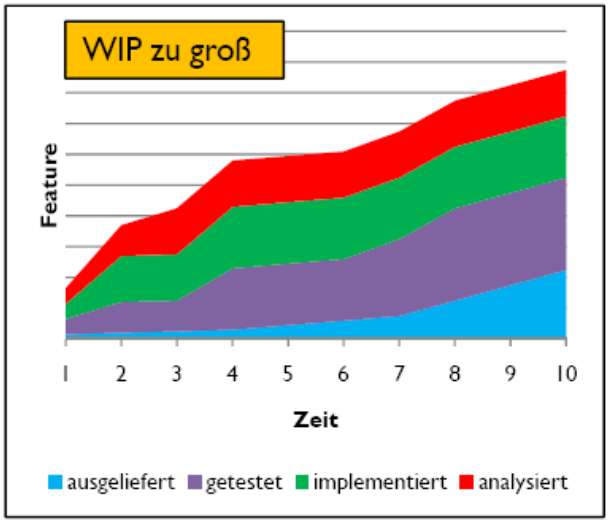
Diese Metrik ist mein persönlicher Favorit, da sie ein elementares Kernkonzept der agilen SW Entwicklung misst: In einer guten Entwicklung ist der „Work in Progress“ (WIP) limitiert. Daraus ergeben sich unmittelbar Vorteile für die Gesamteffizienz.

- Eine WIP Minimierung optimiert den Output.
- Lange Queues erhöhen das Risiko, den Overhead, erzeugen Qualitätsprobleme, erschweren Innovationen und insbesondere die Cycle Time.



Eine Kontrolle des WIP ist Voraussetzung für ein besseres Erreichen der Timelines.

Die vertikale Dimension zeigt die Länge der Queues, die horizontale Dimension die Verweildauer in der Queue. Zur Verdeutlichung im Folgenden ein Chart mit optimiertem WIP und ein zweites mit Problemen im WIP.



6. Summary

Die agile SW Entwicklung zeigt sich als ein hoch effizientes Vorgehen, wenn die Organisation in der Lage ist, die Konzepte umzusetzen und für aktuelle Probleme ein sinnhafter Pragmatismus angewandt werden kann. Die agile Transformation gibt es zweifelsfrei nicht umsonst und die Beispiele haben viele schwierige Themen aufgezeigt, die es zu lösen gilt.

Der agile Ansatz ist nicht grundsätzlich die erste Wahl. Es gibt Bereiche, für die ein anderes Vorgehen vorteilhafter erscheinen mag, etwa bei sehr kritischen Security und Compliance Anforderungen.

Meiner Meinung nach gibt es folgende top-Probleme im Projektmanagement agiler Projekte in den folgenden Bereichen:

A. Kontext Agile Transformation

- Die Übergangszeit bis zur „vollständigen“ Transformation ist zu beplanen (MoO).
- Es darf keine Gefährdung der laufenden Projekte geben. (“controlled agile move”).
- Die PM Rollen Anpassung muss entsprechend des Transformationsfortschritts erfolgen.
- Die Transformation muss gleichmäßig und unternehmensweit sein (Gesamt System-Betrachtung).
- Es gibt keine „Musterlösung“ nach dem Buch. Die Implementierung des agilen Ansatzes muss Flexibilität erlauben.
- Aufwand und Zeit für den Transformations-Prozesses werden unterschätzt.

Die PM-Rolle wird zunehmend zu einer Coaching-Rolle.

Praktische Tipps zu Ihrer Projektarbeit

B. Kontext Scrum

- Es existiert die große Gefahr der Über-Simplifizierung im Projekt aufgrund des vermeintlich einfachen Scrum Ansatzes.
- Die Aufwände, Zeit und Probleme werden massiv unterschätzt.
- Eine Lösungsfindung für Skalierungsprobleme ist erforderlich.

C. Kontext agile

- "Hard practicalities"- z.B. Production Pipe, Testautomation, Radiatoren, etc. sind elementare Herausforderungen. Kritisch, aber eher lösbar, da technischer Natur.
- Deutlich schwieriger und damit der größte Problembereich sind die "Soft practicalities z.B. CI, TDD, Scrum-Team Verantwortlichkeiten etc. und die damit verbundenen Mindset-Changes.

Die Kernessenz des agilen PM ist für mich:

Der PM soll und muss sich auf die Über- und Annahme von Projekt-Verantwortung (Planung, Qualität, Risiko etc.) durch die Teams verlassen können. Dieser Verantwortungsübergang ist entscheidend und einer der agilen Grundpfeiler.

Es gibt Teams, die das verstanden haben, aber auch andere, die aus verschiedensten Gründen nicht in der Lage sind, dem zu folgen, und wieder andere, die dies gar nicht erst wollen. Ein intensives, kontinuierliches und koordiniertes Coaching auf allen Ebenen ist hier für den Projekterfolg notwendig und von dem PM einzufordern.

Trotz der vielen potentiellen Probleme im PM bleibt mir zu sagen: Hätte ich die Wahl in meinem nächsten Projekt und wäre der Projektinhalt geeignet, ich würde es wieder „agil" tun, um die Power und Effektivität dieses Modells zu erleben.