📖

Skript

Vorgehensmodelle der Softwareentwicklung

In der Anfangszeit der Informatik waren Computerprogramme sehr überschaubar und daher die Softwareentwicklung ohne viel Planung durchführbar. Dies änderte sich jedoch Mitte des 20. Jahrhunderts rapide, als starke technologische Fortschritte dazu führten, dass auch sehr komplexe Anwendungen plötzlich möglich und erforderlich waren. Die Entwicklung komplexer Software wurde in dieser Zeit mehr als Kunst angesehen als ein planbarer Prozess. Dies führte in den 1960ern zur sogenannten „Softwarekrise“, eine Zeit, in der Softwareentwicklung selten termingerecht erfolgte und der entstandene Programmcode meist komplett unübersichtlich war. Es wurde also notwendig, dass der Entwicklungsprozess selbst in den Mittelpunkt rückte, wodurch schnell klar wurde, dass die Zeit von „einfach drauf los programmieren“ vorbei war und standardisierte Verfahren zur Softwareentwicklung benötigt werden.

Wichtige Grundsätze, welche in dieser Zeit beschlossen wurden, waren die Aufteilung der Entwicklung der „Gesamtsoftware“ in logisch abgeschlossene Teilaufgaben und das Denken in Projektphasen. Beide Grundsätze wurden aus den schon länger existierenden Ingenieurdisziplinen übernommen (daher auch der Begriff Software Engineering). Es wurden standardisierte Vorgehensmodelle entwickelt, welche den Softwareentwicklungsprozess in unterschiedliche Phasen einteilen und den Entwicklern bei der Planung helfen. Ziel war hierbei immer, eine funktionsfähige Software bei planbaren und überschaubaren Kosten zu bekommen.

Definition Vorgehensmodell:

Vorgehensmodelle unterstützen den Gesamtprozess der Software-Entwicklung. Sie beschreiben das Idealbild eines Entwicklungsprojektes in abstrakter Weise und definieren, in welcher Reihenfolge die Entwicklungsaktivitäten erledigt werden sollen. Oft wird jede Entwicklungsaktivität als extra Phase gesehen. Jede Phase hat die Erstellung eines definierten Zwischenproduktes (z.B. dokumentierte Kundenanforderungen oder erster Codeentwurf) als Ziel.

Es gibt zwei Hauptkategorien von Vorgehensmodellen

1. **Sequenzielle Vorgehensmodelle**

Sequenzielle Modelle nehmen an, dass die Projektphasen der Softwareentwicklung streng nacheinander angeordnet sind. Das Ergebnis einer Phase dient als Input für die nächste Phase. Jede Phase wird genau ein Mal durchgeführt und abgeschlossen, bevor die nächste beginnt. Streng sequenzielle Modelle sind die ältesten Vorgehensmodelle. Sie gelten für viele Entwickler jedoch als aus der Zeit gefallen, da sie nicht berücksichtigen, dass sich Projekt- oder Kundenanforderungen in unseren schnelllebigen Zeiten schnell ändern können.

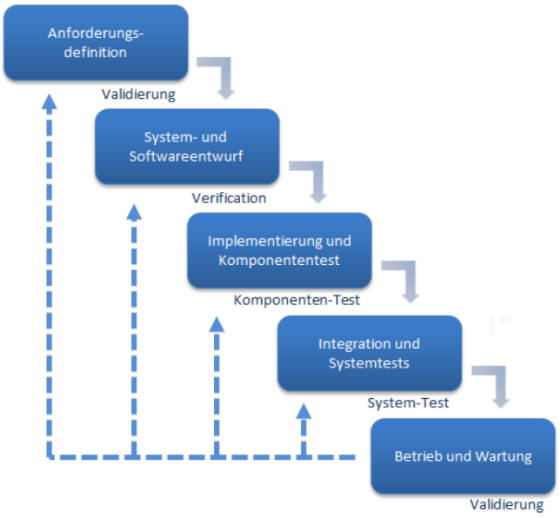
1. **Iterative Vorgehensmodelle**

Bei iterativen Modellen werden die einzelnen Phasen in der Regel mehrfach durchlaufen. Es wird angenommen, dass die Softwareentwicklung evolutionär stattfindet (d.h. schnelles Fertigstellen eines ersten Softwareentwurfs, welcher dann Schritt-für-Schritt verbessert wird). Dadurch werden im iterativen Modell unterschiedliche Entwicklungszyklen durchlaufen, bei dem jeder Zyklus das Ziel hat, die bereits implementierte Software weiter zu verbessern. Der große Vorteil iterativer Modelle ist, dass die konkreten Anforderungen sich im Laufe des Projektes ändern können bzw. noch nicht vollständig zu Projektbeginn bekannt sein müssen. Daher sind iterative Modelle heutzutage deutlich weiterverbreitet als sequenzielle Modelle.

Im Folgenden werden drei der bekanntesten Vorgehensmodelle dargestellt, von denen zwei einen sequenziellen (Wasserfall- und V-Modell) und eines einen iterativen (Spiralmodell) Charakter hat.

Wasserfallmodell

Das älteste Vorgehensmodell in der Software-Entwicklung sieht den Software-Erstellungsprozess als Wasserfall. Die einzelnen Projektschritte sollen nach dem Modell nacheinander (von oben nach unten, wie ein Wasserfall) abgearbeitet werden. Die Ursprünge des Modells reichen zurück bis in die 60er, wobei der Name „Wasserfall-Modell“ erst in den 70ern von Barry Boehm eingeführt wurde. Entgegen einer weitverbreiteten Annahme ist das Wasserfallmodell kein rein sequenzieller Ansatz, sondern erlaubt auch in Sonderfällen Rückkopplungen von nachgelagerten Phasen in vorherige Phasen (sogenannte Feedback-Schleifen). Diese Rückkopplungen sind jedoch als absolute Ausnahmen zu sehen, wodurch die sequenzielle Natur erhalten bleibt. Zudem gibt es auch streng sequenzielle Versionen des Wasserfall-Modells komplett ohne Feedback-Schleifen. Es werden in der Regel 5 Phasen unterschieden, welche jeweils durch einen Qualitätscheck (Meilenstein) abgeschlossen werden:



1. **Anforderungsdefinition:**

Eine vollständige Dokumentation aller Anforderungen, Funktionalitäten, und Einsatzgebiete der Software mit hohem Detaillierungsgrad. Der dazugehörige Meilenstein heißt Validierung.

1. **System- und Softwareentwurf**

Die Anforderungen werden analysiert und in die technische (Software-)Ebene übersetzt. Ein abstrakter Entwurf der gesamten Software (bzw. eine Softwarearchitektur) sowie ihrer Teilprogramme wird entwickelt, welcher alle Anforderungen aus Schritt 1 erfüllt. Der dazugehörige Meilenstein heißt Verifizierung.

1. **Implementierung**

Die Software mit all ihren Teilprogrammen wird programmiert bzw. implementiert. Es wird durch den Meilenstein geprüft, ob alle Teilprogramme (bzw. Komponenten) ihre Funktion erfüllen.

1. **Integration**

Die einzelnen Teilprogramme werden zu dem in Schritt 2 definierten Softwareentwurf vereinigt (integriert) und die Software als Ganzes getestet. Ist der Meilenstein erreicht, kann die Software ausgeliefert werden.

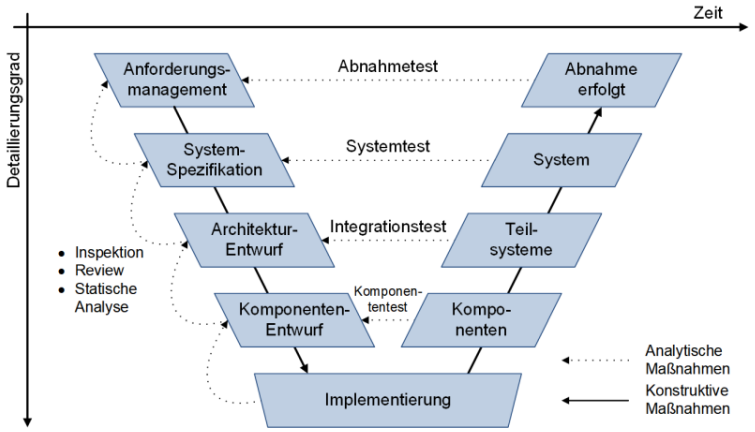
1. **Betrieb und Wartung**

Nachdem die Software installiert wurde, sollten regelmäßige Patches und Bugfixes durchgeführt werden, um die Softwarequalität weiter zu erhöhen.

Die Einsatzbereiche des klassischen Wasserfallmodells findet man heutzutage meist in Projekten, in denen eine Änderung der Anforderungen ausgeschlossen werden kann (z.B. bei Kleinprojekten). Vorteile des Modells sind eine klare Strukturierung sowie einfache Verständlichkeit. Zudem wird die Erstellung von ausführlichen Dokumentationen gefördert. Problematisch ist die Inflexibilität bezüglich der Anforderungen sowie die Tatsache, dass der Kunde die Software erst nach ihrer kompletten Fertigstellung das erste Mal sieht. Zudem kann es dazu kommen, dass die Dokumentation einen sehr hohen Zeitraum einnimmt, welcher den Nutzen nicht mehr rechtfertigt.

Das V-Modell

Das V-Modell wird in der Regel als eine Weiterentwicklung des Wasserfallmodells gesehen, bei dem die Qualitätssicherung in den Mittelpunkt gerückt wird. Der Softwareentwicklungsprozess wird dabei in der Regel als ein großer „V“ visualisiert. Es handelt sich ebenfalls um ein sequenzielles Vorgehensmodell, bei dem Rückkopplungen die Ausnahme darstellen. Die Idee wurde Ende der 70er veröffentlicht und zunächst hauptsächlich in militärischen Bereichen verwendet, bevor es auch in die Industrie Einzug hielt. Es gibt unterschiedliche Versionen des V-Modells, welche jedoch dasselbe Grundkonzept teilen. Es wird jeder Entwicklungsphase vor der Implementierung eine dazugehörige Integrationsphase nach der Implementierung zugeordnet. Die beiden Phasen sind dabei jeweils durch einen Qualitätssicherungsabgleich verbunden. Dabei steigt der Detaillierungsgrad einer Phase mit zunehmender Nähe zur Implementierung an:

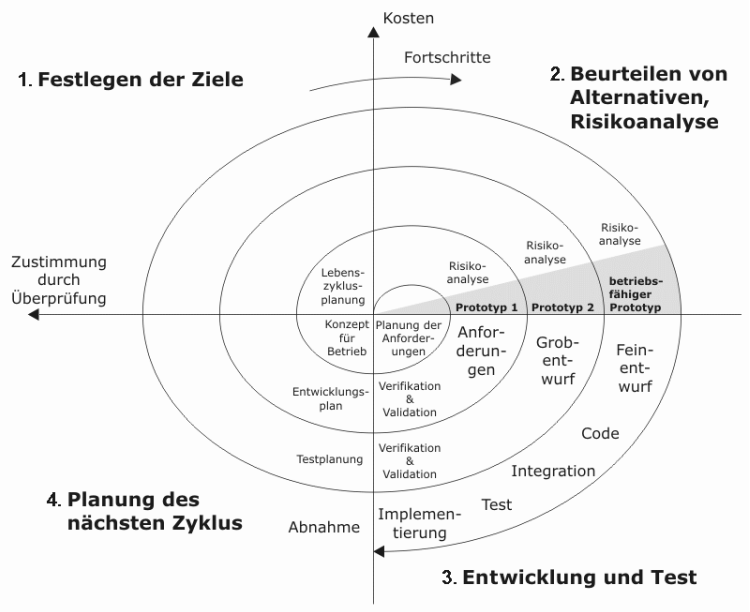


1. Der Anforderungsdefinition wird der Abnahmetest zugeordnet, welcher bei der Abnahme überprüft, ob die erstellte Software die gestellten Anforderungen vollständig erfüllt (Wurden alle Anforderungen richtig dokumentiert und in die Software eingearbeitet?)
2. Der Systemspezifikation wird der Systemtest zugeordnet, welcher testet, ob die implementierte Software (bzw. das System) das macht, was sie laut technischer Spezifikation machen soll (Wurde die beauftragte Software richtig und fehlerfrei entwickelt?)
3. Der Integrationstest vergleicht Architekturentwurf und implementierte Software-Teilsysteme
4. Der Komponententest vergleicht Komponentenentwurf und implementierte Komponenten.

Der Vorteil des V-Modells ist der Fokus auf Qualitätssicherung, welcher vor allem bei großen Softwareprojekten voll zu Tragen kommt. Es wird jedoch oft als schwerfällig bezeichnet und benötigt viele Dokumente, was nicht immer kosteneffizient ist.

Spiralmodell

Das Spiralmodell hat einen iterativen Charakter, bei dem Softwareentwicklung als ein evolutionärer Prozess angesehen wird. Vier Schritte werden dabei in einer festgelegten Reihenfolge mehrmals wiederholt. Ziel ist es dem Kunden möglichst früh eine Vorstellung von der fertigen Software zu geben, sodass dieser direktes Feedback geben kann, was in den darauffolgenden Entwicklungszyklen mit einfließen kann. Die Entwicklung des Spiralmodells wurde in den 80ers notwendig, als klar wurde, dass sequenzielle Modelle vor allem bei nicht vollständig bekannten Anforderungen oder Projekten in dynamischer Umgebung nicht zielführend sind. Es wurde notwendig, ein Vorgehensmodell zu designen, welches die zu implementierende Software Stück-für-Stück (evolutionär) weiterentwickelt und verbessert, bis die Software betriebsfähig ist. Eine Hauptannahme ist, dass der Entwicklungsprozess sowohl aus der Erstentwicklung der Software als auch aus dessen Weiterentwicklung besteht, was als Spirale visualisiert wird. Jeder Zyklus der Spirale beinhaltet vier Schritte:



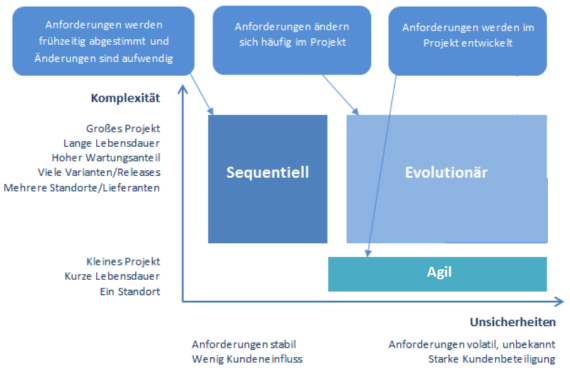
1. Festlegen der zyklusspezifischen Ziele und Softwareanforderungen
2. Beurteilung von Alternativen, Risikoanalyse und Maßnahmen zur Risikominimierung:
3. Entwicklung, Integration und Test der Software anhand einer definierten Vorgehensweise
4. Planung des nächsten Zyklus (falls notwendig)

Ganz zentral für das Spiralmodell ist der Fokus auf Risiken. Risiken sollen früh im Projekt sowohl identifiziert, bewertet, als auch eliminiert oder minimiert werden, um den Projekterfolg zu erhöhen.

Die Vorteile des Modells sind vor allem seine Flexibilität und Offenheit; auch dynamische Softwareprojekt voller Unsicherheit zu Beginn können mit dem Modell verwirklicht werden. Der Kunde bekommt früh die Möglichkeit, die Software abzunehmen und kann dadurch mit neu-erworbenem Wissen bessere Anforderungen für die folgenden Zyklen definieren. Zudem sorgt der starke Fokus auf Risikoerkennung dafür, dass es (zumindest laut Theorie) selten vorkommt, dass die Entwicklung nach hohem Zeit- und Finanzinvest erfolglos abgebrochen werden muss. Nachteile sind die hohe Unsicherheit bei Budget- und Zeitplanung, sowie der hohe Managementaufwand.

Agile Softwareentwicklung

Agile Softwareentwicklung bezeichnet eine Art der Entwicklung, bei der die Reaktion auf geänderte bzw. neue Anforderungen im Mittelpunkt steht. Sie haben eher einen leichtgewichtigen Charakter, was bedeutet, dass versucht wird Dokumentationen und Bürokratie möglichst klein zu halten. Zudem sehen Selbstorganisation und eine spezielle Art der Zusammenarbeit im Mittelpunkt. Agile Methoden der Softwareentwicklung werden zumeist bei etwas kleineren Projekten verwendet, welche von hoher Dynamik, Unsicherheit und Kundenbeteiligung charakterisiert sind.



Den Ursprung hat die Agile Philosophie im Jahre 2001, in dem als Antwort auf die bürokratischen und oft starren klassischen Vorgehensmodelle mit dem agilen Manifest ein Gegenentwurf präsentiert wurde. In dem Agilen Manifest wurden vier Leitsätze festgehalten:

- **Personen und Interaktionen** haben Vorrang vor Prozessen und Tools.
- **Funktionsfähige Software** hat Vorrang vor umfassender Dokumentation.
- **Zusammenarbeit mit dem Kunden** hat Vorrang vor Vertragsverhandlungen.
- **Eingehen auf Veränderungen** hat Vorrang vor dem Befolgen eines Plans.

Dabei gilt nicht, dass die nicht-fettgedruckten Punkte bedeutungslos sind, sondern nur dass die fettgedruckten Punkte von höherer Bedeutung im Software-Entwicklungsprozess sind. Basierend auf diesen vier Leitsätzen wurde 12 Prinzipien formuliert, welche die agile Philosophie detaillierter beschreiben:

- Unsere höchste Priorität ist es, den Kunden durch frühe und kontinuierliche Auslieferung

  wertvoller Software zufrieden zu stellen.
- Heiße Anforderungsänderungen selbst spät in der Entwicklung willkommen. Agile Prozesse nutzen Veränderungen zum Wettbewerbsvorteil des Kunden.
- Liefere funktionierende Software regelmäßig innerhalb weniger Wochen oder Monate und

  bevorzuge dabei die kürzere Zeitspanne.
- Fachexperten und Entwickler müssen während des Projektes täglich zusammenarbeiten.
- Errichte Projekte rund um motivierte Individuen. Gib ihnen das Umfeld und die Unterstützung, die sie benötigen und vertraue darauf, dass sie die Aufgabe erledigen.
- Die effizienteste und effektivste Methode, Informationen an und innerhalb eines Entwicklungs-teams zu übermitteln, ist im Gespräch von Angesicht zu Angesicht.
- Funktionierende Software ist das wichtigste Fortschrittsmaß.
- Agile Prozesse fördern nachhaltige Entwicklung. Die Auftraggeber, Entwickler und Benutzer sollten ein gleichmäßiges Tempo auf unbegrenzte Zeit halten können.
- Ständiges Augenmerk auf technische Exzellenz und gutes Design fördert Agilität.
- Einfachheit -- die Kunst, die Menge nicht getaner Arbeit zu maximieren -- ist essenziell.
- Die besten Architekturen, Anforderungen und Entwürfe entstehen durch selbstorganisierte Teams.

- In regelmäßigen Abständen reflektiert das Team, wie es effektiver werden kann und passt sein Verhalten entsprechend an.

Im Folgenden schauen wir uns zwei der beliebtesten Agilen Entwicklungsmodelle, nämlich Kanban und Scrum genauer an:

Kanban

Kanban ist eine Entwicklungsmethode, welche sich darauf fokussiert einen kontinuierlichen Fluss an Teilergebnissen mit stetig steigendem Kundennutzen zu produzieren. Kanban kommt ursprünglich aus der Industrie, genauer gesagt wurde es von Toyota nach dem 2.Weltkrieg entwickelt mit dem Fokus Effizienz- und Qualitätssteigerungen zu erzielen. Später wurde die Methodik auch vermehrt bei der Software-Entwicklung angewandt. Kanban hilft, den Fluss der laufenden Entwicklungs-Aktivitäten in Abhängigkeit von der vorhandenen Entwicklungskapazität zu optimieren. Ein Kanban-System basiert auf dem „pull"-System, das heißt das Aufgaben in einen aktiven „in Bearbeitung"-Status gezogen werden, sobald die vorausgelagerte Aktivität abgeschlossen ist. Die „pull"-Philosophie soll dafür sorgen, dass nur an notwendigen, tatsächlich nachgefragten Arbeitsschritten gearbeitet wird, wodurch ein effizienterer Projektablauf ermöglicht werden soll. Kanban besitzt 3 Kernprinzipien:

1. **Visualisiere den Projektablauf mit Hilfe eines Kanban-Boards**

Ein Kanbanboard soll den Fluss an Entwicklungsaktivitäten übersichtlich visuell darstellen. Je nach Anwendungsgebiet kann ein Kanbanboard unterschiedliche Spalten haben, in der Softwareentwicklung sind zumeist backlog (in Zukunft zu erledigende Aufgaben), To-do-Liste (als nächstes zu erledigende Aufgaben), in Bearbeitung (Aufgaben, welche aktuell von Entwicklern bearbeitet werden), Test (fertig Bearbeitete Aufgaben, deren Ergebnisse überprüft werden), und Erledigt (erledigte Aufgaben mit nachgewiesener Qualität). Das Pull-Prinzip bedeutet, dass die einzelnen Aufgaben miteinander verknüpft sind, und das Wechseln einer Aufgabe in eine Spalte nach rechts bedeutet, dass eine andere, nachgelagert Aktivität auch eine Spalte nach rechts gezogen wird.

1. **Versuche den Anteil an Tätigkeit, welche gleichzeitig „Work in Progress" sind, zu limitieren**

Eine Spalte sollte zu keinem Moment eine definierte Anzahl an Aufgaben überschreiten (außer Backlog); dies soll über ein Pull-System sichergestellt werden. Der Fokus sollte immer die vom Kunden aktuell meiste nachgefragten Programm-Features sein.

1. **Sorge für einen kontinuierlichen Output mit geringen Durchlaufzeiten**

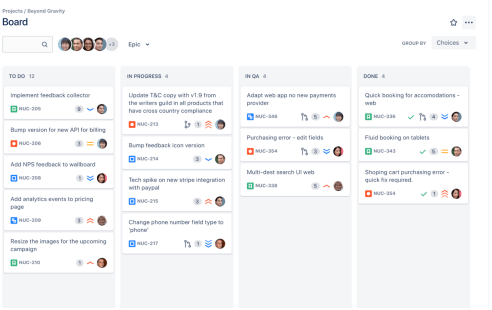
Die Durchlaufzeit einer Aktivität durch den gesamten Wertschöpfungsprozess im Kanbanboard sollte gemessen und optimiert werden. Durch eine gute Vorausplanung und optimierte Durchlaufzeiten soll ein relativ kontinuierlicher Output an Softwareupdates sichergestellt werden. Ständige Verbesserung ist eine der wichtigsten Philosophien im Kanban.

Kanban kann sehr flexibel und agil gestaltet werden, neue Anforderungen (bzw. Aktivitäten) können jederzeit in das Kanbanboard eingefügt werden. Während eine Aktivität in der Regel einem sequenziellen Fluss folgt, ist es die Aufteilung der Software in viele kleine Teilaktivitäten, welche zu unterschiedlichem Zeitpunkt die Kanban-Spalten durchlaufen, was der Methodik den agilen Charakter gibt. Es gibt zudem im Vergleich zu anderen Methoden wie Scrum wenige Vorgaben was Teamrollen und Zusammensetzung angeht. In der Regel werden Feedbackschleifen im Kanban integriert, bei dem sich das Team Feedback gibt und die Fluss-Koordination verbessert werden soll.

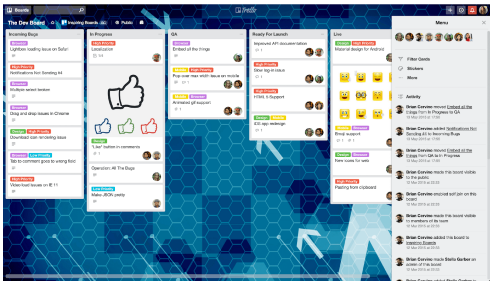
Während früher eine Kanbanboard in der Regel eine physische Tafel voller Papier-Karten war, ist man heute natürlich weiter. Es gibt heutzutage zahlreiche Software, welche Unternehmen hilft, die Kanban-Methodik digital anzuwenden. Hier sind vor allem digital Ticketmanagementsysteme wie Jira und Trello zu nennen, welche Workflow- und Aufgabenmanagement unterstützen:

Ticketmanagementsysteme

Jira ist eine Software von Atlessian, welche vor allem als Unterstützungstool für Projektmanagement und Anfragenverwaltung (z.B. IT-Helpdesk-Ticketsystem) verwendet wird. Es können sowohl Workflows abgebildet und organisiert werden, als auch Anfragen priorisiert und zentral verwaltet werden. Jira ist generell sehr flexibel und kann vielen verschiedenen Problem eingesetzt werden. Jira enthält eine Vorlage eines Kanban-Boards, welches Unternehmen bei Software-Entwicklungsprojekten helfen kann, die Kanban Methodik effizient einzusetzen. Die einzelnen Aktivitäten in dem Jira-Board können dabei priorisiert und Personen zugeteilt werden, sowie Zusammenhänge zwischen den Aktivitäten abgebildet werden.



Trello ist ein reiner Aufgabenverwaltungsdienst, welcher 2011 entwickelt wurde und mittlerweile von Atlessian aufgekauft wurde. Es basiert auf dem Kanban-Prinzip und bietet eine bietet einen Überblick über alle Aufgaben eines Projektes. Ein Trello-Board besteht aus unterschiedlichen Listen (Spalten), welche mit Karten (Aufgaben) gefüllt werden können. Die Karten können Personen zugeordnet, mit Deadlines versehen und mit Zusatzinformationen bestückt werden.



DevOps und Agile Methoden

DevOps und agile Software Entwicklung werden in der Praxis oft nicht sauber auseinandergehalten, obwohl die beiden Begriffe unterschiedliche Konzepte und Ursprünge beinhalten. Bei DevOps steht das Überwinden der Trennung von Software Entwicklung (Development) und IT-Betrieb (Operations, Administration) im Mittelpunkt. DevOps beschreibt das Konzept, dass diese Bereiche in allen Stadien eines Software-Lebenszykus zusammenarbeiten und dadurch ein kontinuierlicher Output gesichert ist. Die Gemeinsamkeiten und Unterschiede zu agilen Methoden sind in der folgenden Tabelle dargestellt:

Aa Agile Methoden	DevOps	Files
Beides sind iterative Methoden bzw. Leitfäden zum Entwickeln von SoftwareBeide zielen auf die Erhöhung der Geschwindigkeit und Qualität der SoftwareentwicklungBeides sind neuere Konzepte im Vergleich zu den klassischen VorgehensmodellenDie Grundbausteine eines Softwareentwicklungsprojektes sind bei beiden identisch (Planung, Code-Entwicklung, Test, Deployment)		
Sicherstellen eines flexiblen Softwareentwicklungsprozesses durch iteratives und inkrementelles Entwickeln, Deployen und Testen ohne viel bürokratischen AufwandDie agile Softwareentwicklung fokussiert sich auf den reinen Softwareentwicklungsprozess, welcher auch irgendwann abgeschlossen ist. Der operationale Betrieb der Software wird in der Regel nicht betrachtet.Kleinere Team mit fortgeschrittenen Entwicklerskills, welche selbstständig arbeiten	Fokus auf verbesserte Zusammenarbeit und Produktivität durch Aufhebung der Trennung von Entwicklung und Betrieb/Administration DevOps fokussiert sich neben der Entwicklung auch auf den operativen Betrieb der Software und stellt eine kontinuierliche Überwachung und Softwareentwicklung sicher. Der	https://lh5.googleusercontent.com/TY4Q4TzHV4b3ePdOKXMEMRnnkP7n3bTORzgl7szbBAdNnjirhVp7rkQ5moB

Aa Agile Methoden	DevOps	Files
könnenWenig DokumentationenFeedback kommt meist von intern	Entwicklungs-prozess gilt nie als vollständig abgeschlossen. Größeres Team mit unterschiedlichen Skillsets (Entwickeln, Administration, Qualitätssicherung) Ausführliche Dokumentationen Feedback kommt direkt vom Kunden	

Scrum

Scrum ist ein Leitfaden zur Softwareentwicklung mit dem Prinzip „Die wenigen Regeln, die wir haben, werden strikt befolgt.“ Scrum baut auf den Werten und Prinzipien des agilen Manifests auf und gilt als weit verbreitetste Methode der agilen Softwareentwicklung. Eine Gesamtplanung erfolgt in Scrum zunächst nur grob, bevor diese grobe Planung Schritt für Schritt, oder „Sprint-by-Sprint“ verfeinert wird (evolutionäre Vorgehensweise). Es kann damit ohne Probleme auf geänderte Anforderungen oder Rahmenbedingungen reagiert werden.

Der Name Scrum kommt aus dem Rugby und kann am besten mit „Gedränge“ übersetzt werden. Scrum ist eine sehr enge kreisförmige Aufstellung von zwei Rugby Mannschaften, bei dem gemeinschaftlich versucht wird, dem Gegner keinen Raum zu geben. Die Analogie entsteht hauptsächlich durch die zeitmäßig begrenzte sehr enge Zusammenarbeit der Spieler bzw. Mitarbeiter.

Der Ursprung von Scrum ist in Japan zu finden, wo es 1986 als neue Produktentwicklungsmethodik im Konsumgüterbereich vorgestellt wurde und 10 Jahre später von Schwaber in die Software-Domain adaptiert wurde. Der Grundgedanke resultierte aus den Ergebnissen von Studien, welche zeigten dass kleine, selbstorganisierte und auch selbstverantwortliche Teams eine höhere Produktivität im Vergleich zu größeren Teams haben.

Zentrale Konzepte von Scrum

Die Scrum Methodik unterscheidet zwischen drei Rollen, welche zwingend ausgefüllt werden müssen:

1. **Product-Owner**

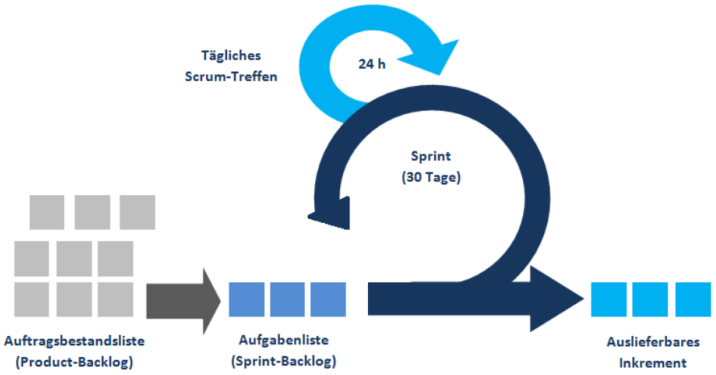
Der Product Owner hat die Gesamtverantwortung über das zu entwickelnde Produkt (bzw. Software). Er bestimmt und priorisiert die Software-Eigenschaften und muss sich am wirtschaftlichen Erfolg der Software direkt messen lassen. Er ist in ständiger Zusammenarbeit mit den Entwicklern und steuert den Gesamtprozess der Softwareentwicklung. Er sollte versuchen die Sichtweise des Kunden einzunehmen, um den Kundennutzen des finalen Produktes zu maximieren. Er ist letztendlicher der Entscheider in den kritischen Fragen bezüglich der Ausgestaltung der Software und der Entwicklungsreihenfolge ihrer Features. Daher verwaltet er auch den Product Backlog, welches alle Anforderungen des zu entwickelnden Produktes (und die daraus abgeleiteten Entwicklungsschritte) beinhaltet. Zudem ist er derjenige, welche regelmäßig Kontakt mit den Stakeholdern (Kunden, Management, …) aufnimmt und diese updatet.

1. **Entwicklerteam**

Das Entwicklerteam ist verantwortlich die definierten Software Features zu programmieren, integrieren, und zu verbessern. Es bekommt vom Product-Owner vorgegeben, welche Features in welcher Reihenfolge und Qualität umzusetzten sind. Bei der Frage, wie die Umsetzung konkret aussehen soll, sind die Teams jedoch komplett frei. Ein Team sollte so zusammengestellt sein, dass es die jeweilige Aufgabe in einem Sprint ohne größere Abhängigkeiten von außen selbstständig lösen kann. Die Idee von Scrum ist es, dass jedes Team selbsorganisiert und selbstverantworltlich arbeitet. Teams sollten nicht zu groß (max 10 Personen) und interdisziplinär zusammengesetzt sein, wichtige Expertise sollte innerhalb eines Teams z.B. in den Bereichen Dokumentation, Datenbanken, Coding, Software-Architektur und Code-Testing vorhanden sein.

1. **Scrum Master**

Der Scrum Master stellt sicher, dass die Regeln des Scrum-Leitfadens eingehalten werden. Er ist nicht direkt an der technischen Entwicklung beteiligt, arbeitet aber eng mit den Entwicklerteams und dem Product Owner zusammen. Er sorgt für eine produktive Art der Zusammenarbeit in einem Team, sowie zwischen Product Owner und Entwicklungsteams. Bei Konflikten nimmt er in der Regel eine Moderatoren bzw. Vermittler-Rolle ein. Er kennt sich detailliert mit dem Scrum Regelwerk und der Scrum Philosophie aus und vermittelt diese an alle Teilnehmer im Projekt. Seine Rolle wird oft als „dienende Führungskraft“ beschrieben ohne direkt Anweisungsbefugnis. Vor allem zu Beginn eines Projektes, wenn die Abläufe bzw. das Scrum Regelwerk noch unbekannt sind, hat der Scrum Master eine sehr wichtige Rolle.



Ein Zyklus in der Softwareentwicklung besteht laut Scrum-Leitfaden aus drei Etappen:

1. Spring-Planning oder Pre-Sprint-Etappe:

Diese Etappe fokussiert sich auf die Gesamt-Projektplanung. Das Product-Backlog ist eine dynamische Liste von Produktanforderungen, welche vom Product Owner gepflegt und priorisiert werden. Die Anforderungen sollten nicht zu technisch sein, sondern eher aus Anwendersicht formuliert sein. Im Sprint-Planning wird dann eine oder mehrere Anforderungen aus dem Backlog entnommen, um diese im folgenden Sprint als Software-Features zu implementieren. Es wird detailliert definiert, welche Aufgaben erledigt werden müssen, um die ausgewählte Produktanforderungen im Sprint technisch umzusetzen (Sprint-Backlog).

2. Sprint-Etappe:

Eine in einem Sprint zu implementierende Produktanforderung (genannt Inkrement) sollte während des Sprints nicht verändert werden, da dies die Fertigstellung am Ende des Sprints erschweren könnte. Ein Sprint dauert normalerweise 30 Tage, in denen alle Arbeitspakete im Sprint-Backlog vollständig vom Entwicklerteam abgearbeitet werden sollen. Ein fertiges Inkrement muss immer ein neuer nutzbarer Zustand der Software sein. Nicht im Sprint-Backlog enthaltende Anforderungen können in dieser Zeit verändert, ergänzt oder geupdatet werden. Es gibt ein tägliches kurzes Meeting (Daily Scrum), an dem in der Regel das Entwicklerteam und optional Product Owner und Scrum Master teilnehmen. Das Meeting sollte kurz sein und rein dem Austausch von Informationen/Updates dienen.

3. Sprint Review bzw. Post-Sprint-Etappe:

Im Sprint Review präsentiert das Entwicklerteam seine Ergebnisse und es wird kontrolliert, ob alle für den Sprint gesteckte Ziele erreicht wurden. Basierend auf den Ergebnissen wird das Product Backlog angepasst und schon die nächsten Schritte definiert. In der Regel sind im Sprint Review sowohl alle Projekt Teilnehmer als auch Stakeholder (Kunde, Anwender, …) anwesend, um vollständiges Feedback zu gewährleisten. Falls Probleme in dem Software-Inkrement entdeckt werden, werden diese als Bugfix ins Product Backlog eingetragen, priorisiert, und später gefixt. Nach endlich vielen Sprints sollte der Softwareerstellungsprozess abgeschlossen sein und die Software an den Kunden ausgeliefert werden können.

Wie alle agilen Methoden hat Scrum da seine Stärken, wo Projektanforderungen zu Beginn entweder unbekannt oder sehr unsicher sind. Zudem wurde Scrum in der Vergangenheit oft dann erfolgreich eingesetzt, wenn Projekte zuvor außer Kontrolle gewachsen waren und dann ein strukturierterer inkrementeller Ansatz notwendig wurde. Voraussetzungen sind das Vorhandensein von Expertise und hohes Vertrauen in Entwicklerteams, sodass Softwareentwicklung durch Selbstorganisation und Eigenverantwortung auch erfolgreich sein kann. Zudem darf das Projekt bzw. der Workload eines Sprints nicht zu groß sein, Entwicklerteams sollten die Größe von 10 Mitgliedern nicht überschreiten. Als großer Vorteil von Scrum werden neben der Agilität und Flexibilität durch die kurzen Sprints zumeist genannt, dass die Selbstorganisation von Entwicklerteams Motivation und Teamgeist steigert. Zudem sorgt der tägliche Kurzaustausch dafür, dass jedes Teammitglied weiß, was die aktuellen Hauptthemen bzw. -probleme im Projekt sind, ohne dass zu viel Zeit dadurch verloren geht. Die Selbstorganisation der Teams birgt jedoch natürlich auch Risiken und Konflikte, wenn nicht alle mitziehen. Zudem birgt eine die konstant gehaltene Sprintdauer die Gefahr, dass man bei falscher Planung Zeit verliert oder ein Inkrement nicht fertiggestellt bekommt. Auch der hohe Kommunikations- und Abstimmungsaufwand ist zu nennen.