# Agile Softwareentwicklung

Sie kennen den Aufbau und die wichtigsten Phasen des Wasserfallmodells.

Aufbau:

Das Wasserfallmodell ist ein sequenzielles Modell der Softwareentwicklung, das aus einer linearen Abfolge von Phasen besteht, die nacheinander durchlaufen werden.

Wichtigste Phasen:

System Anforderungen --> Softwareanforderungen --> Analyse --> Programm-Design --> Codierung --> Testen ---> Betrieb

Sie kennen die Probleme bei der Arbeit mit dem Wasserfallmodell.

Es ist unflexibel und starr --> Wenn Änderungen, in Späteren Phasen vorgenommen werden kann es passieren, dass alle.

Da jede Phase des Wasserfallmodells in sich geschlossen ist, ist es schwierig Feedback von Kunden oder Benutzern während der Entwicklung zu integrieren.

Da das Testing erst in einer späten Phase des Wasserfallmodell stattfindet, ist es möglich das Fehler zu spät erkannt werden.

Sie kennen die vier Werte vom Manifest für Agile Softwareentwicklung.

1. Individuen und Interaktionen mehr als Prozess und Werkzeug
2. Funktionierende Software mehr als umfassende Dokumentation
3. Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlung
4. Reagieren auf Veränderung mehr als Befolgen eines Plans

Sie kennen den Zielkonflikt “gut, schnell, günstig: wähle zwei” und können diesen visuell darstellen.

Ein Bild, das Diagramm enthält.

Automatisch generierte Beschreibung

Es können meistens nur zwei Punkte zutreffen und vom dritten Punkt trifft dann das Gegenteil ein.

Sie kennen das Konzept der Velocity und wissen, warum diese steigen oder sinken kann.

Velocity zeigt die Menge der Punkte auf, die ein Entwicklungsteam innerhalb einer Iteration (eines Sprints) abarbeitet.

Je nachdem wie das Team arbeitet oder es nicht vorhersehbare Komplikationen gibt kann die Velocity sinken und steigen.

Sie wissen, was ein Burn Down Chart aufzeigt.

Das Burn-Down Chart zeigt die verbleibenden Punkte bis zur Erreichung des nächsten Meilensteins auf.   
Der Burn-Down-Chart zeigt den Fortschritt beim Abarbeiten des Backlogs über die Zeit an.

Sie kennen die vier Interventionen, um auf das Erreichen der vier Projektziele Einfluss nehmen zu können (schnell: Planung, günstig: Personal, gut: Qualität, fertig: Umfang), und können Vor- und Nachteile jeder Intervention benennen.  
Ein Bild, das Diagramm enthält.

Automatisch generierte Beschreibung

Interventionen: Planung, Qualität, Umfang, Personal

Vor- und Nachteile:

Planung:

* Ein agiles Planungssystem ermöglicht eine schnellere Anpassung an Änderungen und eine hohe Flexibilität.
* Der Fokus liegt auf der Entwicklung funktionsfähiger Software und nicht auf der Erstellung umfangreicher Dokumentationen.
* Ein Mangel an umfassender Dokumentation kann die Wartung und Unterstützung der Software erschweren.
* Unklare oder sich ändernde Anforderungen können zu Problemen bei der Umsetzung führen.

Qualität:

* Agile Softwareentwicklung legt einen starken Fokus auf Qualität, was zu einer hohen Kundenzufriedenheit führt.
* Ein iterativer Ansatz ermöglicht kontinuierliches Feedback, um die Qualität zu verbessern.
* Eine zu starke Fokussierung auf Qualität kann zu Verzögerungen bei der Entwicklung und höheren Kosten führen.
* Die Kompromisse zwischen Kosten, Zeit und Qualität können schwierig zu navigieren sein.

Umfang:

* Agile Entwicklung ermöglicht es, den Umfang der Software zu ändern, wenn dich die Anforderungen ändern oder wenn neue Funktionen benötigt werden.
* Ein iterativer Ansatz kann dazu beitragen, den Umfang der Software in kleinere handhabbare Stücke zu zerlegen.
* Unklare Anforderungen können zu Schwierigkeiten bei er Bestimmung des Umfangs und zu Verzögerungen bei der Entwicklung führen.
* Ein zu häufiger Wechsel des Umfangs kann zu Problemen bei der Schätzung von Kosten und Zeit führen. C

Personal:

* Agile Entwicklung fördert die Zusammenarbeit zwischen Entwicklern und Kunden.
* Ein selbstorganisiertes Team kann schneller auf Veränderungen reagieren.
* Ein unerfahrenes Team kann Schwierigkeiten haben, ein agiles Vorgehen zu verstehen und umzusetzen.
* Ein zu häufiger Wechsel von Teammitglieder kann die Zusammenarbeit beeinträchtigen.

Sie kennen den Aufbau einer User Story.

Eine User Story wird aus Perspektive des Benutzers geschrieben und beschreibt ein Feature des Systems, das zu entwickeln ist. Details werden zuerst weggelassen und erst geklärt, wenn die Entwickler die Story für die Umsetzung aufnehmen.

Beispiel:  
"Als Benutzer möchte ich gefragt werden, ob ich mein Dokument speichern möchte, wenn ich die Applikation schliesse, ohne vorher gespeichert zu haben."

Sie kennen den Zielkonflikt genau/zutreffend einer Schätzung.

Grob kann man sagen, je genauer eine Schätzung ist desto weniger sind die Chancen, dass die Schätzung zutrifft.

Zum Beispiel ist eine Zeitspanne von 5 – 15 Tagen (ungenau) einfacher einzuhalten als eine Zeitspanne von 5 – 7 Tagen (genau).

Sie wissen, was eine trivariate Schätzung ist, und welche Vorteile sie gegenüber einer exakten Schätzung hat.

Eine trivariate Schätzung gibt für eine Aufgabe einen Idealfall, einen Normalfall und einen ungünstigen Fall, sodass diese mit einer Wahrscheinlichkeit von 5%, 50% und 95% innerhalb der geschätzten Zeit umgesetzt.

Sie können mit den vier Quadranten basierend auf den Achsen “Kosten” und “Wert” User Stories priorisieren.

|  |  |
| --- | --- |
| Ein Bild, das Tisch enthält.  Automatisch generierte Beschreibung | 1. Wertvoll, aber günstig: diese Stories sollten gleich umgesetzt werden. 2. Wertvoll, aber teuer: diese Stories sollten erst später umgesetzt werden 3. Nicht wertvoll und teuer: diese Stories sollten gleich verworfen werden. 4. Nicht wertvoll, aber günstig: diese Stories sollten, wenn überhaupt, erst später umgesetzt werden. |

Sie kennen die sechs INVEST-Kriterien für eine gute User Story und deren Bedeutung.

**I**ndependent (unabhängig):

User Stories müssen nicht in einer bestimmten Reihenfolge umgesetzt werden, weil sie voneinander unabhängig sind. --> kann manchmal trotzdem Abhängigkeiten geben, sollte aber auf ein Minimum reduziert wird.

**N**egotiable (verhandelbar):

User Stories sollten Raum für Verhandlungen zwischen dem Geschäft und der Entwicklung bieten. --> Kostensparen, indem man dich auf einfache Features und einfache Implementierungen einigt.

**V**aluable (wertvoll):

User Stories müssen einen klaren und messbaren wirtschaftlichen Nutzen schaffen.

**E**stimable (schätzbar):

User Stories müssen konkret genug sein, damit sie von den Entwicklern geschätzt werden können.

**S**mall (kein):

User Stories sollen maximal so gross sein, dass sie von einem oder zwei Entwicklern innerhalb eines Sprints umgesetzt werden können.

**T**estable (testbar):

User Stories sollen jeweils mit Tests einhergehen, welche von der Geschäftsseite definiert werden. Eine Story ist dann abgeschlossen, wenn all ihre Tests durchlaufen.

Sie kennen eine Möglichkeit, die Anzahl der Punkte für eine User Story mit einem Team zu schätzen.

Fliegende Finger:

Nach dem Lesen und Diskutieren der Story halten alle Entwickler hinter ihrem Rücken die Anzahl Finger hoch, die sie der Story geben. Auf drei zeigen alle Entwickler ihre Finger.

Planungspoker:

Karten mit Zahlen 1, 2, 3, 5, 8 und manchmal mit einem unendlichen Zeichen für Stories die zum Schätzen zu gross sind und einem Frageziechen falls nicht genügend Informationen vorhanden sind und 0, wenn eine Story zu trivial zum Schätzen ist.

Sie kennen Form, Frequenz, Inhalt und Ablauf eines Standup Meetings und können begründen, wer daran teilnehmen sollte

Form: Das Meeting findet im Stehen statt, um sicherzustellen, dass es kurz und fokussiert bleibt.

Frequent: Das Meeting findet jeden Tag zur gleichen Zeit statt, normalerweise am Anfang des Arbeitstages - Das Meeting muss nicht jeden Tag stattfinden, es ist aber der Normalfall

Inhalt: Jedes Teammitglied berichtet kurz darüber, was es am Vortag erreicht hat, welche Herausforderungen es hatte und was es heute erreichen wird.

Ablauf: Das Meeting dauert in der Regel nicht länger als 15 Minuten um sicherzustellen, dass es kurz und fokussiert bleibt.

Sie kennen die Zusammensetzung, die sinnvolle Grösse und die wichtigsten Eigenschaften eines Scrum-Teams.

Zusammensetzung:

Kleines Team von Menschen bestehend aus:

* Scrum Master
* Product Owner
* Developer

Innerhalb des Scrum Team gibt es keine Teilteams oder Hierarchien.

Grösse:

Üblicherweise 10 Personen

Wichtige Eigenschaften:

* Scrum Team ist umsetzungsverantwortlich
* Scrum Team ist ergebnisverantwortlich

Sie kennen die Rolle des Developers (Entwicklers) und dessen Verantwortlichkeiten.

Developer sind die Personen, die sich der Aufgabe verschrieben haben, jeden Sprint jeden Aspekt eines nutzbaren Inkrements zu schaffen.

Verantwortungen:

* Einen Plan für den Sprint zu erstellen, das Sprint Backlog;
* Qualität durch die Einhaltung einer Definition of Done einzubringen;
* Täglich ihren Plan zur Erreichung des Sprint-Ziel anzupassen;
* Sich wechselseitig als Experten zur Verantwortung zu ziehen

Sie kennen die Rolle des Product Owners und dessen Verantwortlichkeiten.

Der Product Owner ist ergebnisverantwortlich für die Maximierung des Wertes des Produkts, der sich aus der Arbeit des Scrum Teams ergibt.

Verantwortung:

* Das Produkt-Ziel zu entwickeln und explizit zu kommunizieren;
* Die Produkt-Backlog-Einträge zu erstellen und klar zu kommunizieren;
* Die Reihenfolge der Produkt-Backlog-Einträge festzulegen;
* Sicherzustellen, dass das Produkt Backlog transparent, sichtbar und verstanden ist;

Sie kennen die Rolle des Scrum Masters und dessen Verantwortlichkeiten.

Der Scrum Master ist ergebnisverantwortlich für die Einführung von Scrum, wie es im Scrum Guide definiert ist. Er tut dies, indem er allen dabei hilft, die Scrum-Theorie und -Praxis zu verstehen, sowohl innerhalb des Scrum Teams als auch in der Organisation.

Verantwortung:

* Die Teammitglieder in Selbstmanagement und interdisziplinärer Zusammenarbeit zu coachen;
* Das Scrum Team bei der Fokussierung auf die Schaffung von hochwertigem Inkrement zu unterstützen, die der Definition of Done entsprechen.
* Die Beseitigung von Hindernissen (impediments) für den Fortschritt des Scrum Teams zu bewirken
* Sicherzustellen, dass alle Events von Scrum stattfinden, positiv und produktiv sind und innerhalb der Timebox bleiben.

Sie kennen die Scrum-Ereignisse (Events) Sprint, Sprint Planning, Daily Scrum, Sprint Review, Sprint Retrospektive und deren Dauer, Frequenz, Ziele und Ablauf.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Event** | **Dauer** | **Frequenz** | **Ziele** | **Ablauf** |
| Sprint |  |  |  |  |
| Sprint Planning | 4 – 8 Stunden | Einmal pro Sprint | Definition des Sprint-Ziels – Erstellung des Sprint-Backlogs | Product Owner stellt das Ziel und die wichtigsten PBIs vor. Das Team diskutiert und schätzt die PBIs. Das Team plant, wie es die PBIs im Sprint umsetzt. |
| Daily Scrum | Ca. 15 Minuten | Täglich | Abstimmung des Teams – Verbesserung des Arbeitsprozesses | Jedes Teammitglied beantwortet,   1. Was habe ich gestern gemacht 2. Was werde ich heute machen 3. Gibt es 3. Hindernisse |
| Sprint Review | 4 Stunden pro Monat des Sprints | Einmal pro Sprint | Feedback zum Inkrement – Überprüfung des Product Backlogs | Das Team präsentiert das Inkrement und erläutert, welche PBIs erledigt wurden – Product Owner prüft den Product Backlog und aktualisiert ihn ggf. – Stakeholder geben Feedback |
| Sprint Retrospektive | 2-3 Stunden | Einmal pro Sprint | Reflexion und Verbesserung des Arbeitsprozesses – Identifikation von Optimierungsmöglichkeiten | Das Team reflektiert den vergangenen Sprint – Das Team identifiziert was gut und was weniger gut funktioniert hat – Das Team plant Verbesserungen für den nächsten Sprint |

\*PBI – Product Backlog Item

Sie kennen die drei Hauptartefakte von Scrum und wissen, wie diese entstehen, und welchen Zweck diese erfüllen.

Sie wissen, welche Aspekte der agilen Softwareentwicklung Scrum vorgibt und welche es offenlässt