Informationssammlung

# Agile Softwareentwicklung

## Das kleine Agile-Buch – Sander Hoogendoorn (2013)

1. **Wasserfall (S.15)**
   * Ursprung
   * Probleme
     + Wissensverlust nach jeder Phase
     + Wissen aus momentanen Phasen kann nicht mehr in die abgeschlossenen Dokumente einfließen
     + Anforderungen ändern sich
     + Vollständigkeit
     + Schätzung
2. **Agiles Manifest (S.21)**
   * 4 Prinzipien:
     + Individuen und Interaktion mehr als Prozesse und Werkzeuge
     + Funktionierende Software mehr als umfassende Dokumentation
     + Zusammenarbeit mit dem Kunden mehr als Vertragsverhandlungen
     + Reagieren auf Veränderung mehr als das Befolgen eines Plans
   * Linke Aussage wird stärker betont, jedoch wird die rechte Aussage nicht ausgeschlossen
   * Zusammenarbeit zwischen den Rollen
   * Rollen arbeiten miteinander/gleichzeitig 🡪 optimale Dokumentationsweise finden
   * Zusammenarbeit mit Kunden 🡪 Anforderungsanpassungen
   * Merkmale:
     + Kurze Iterationen
     + Zusammenarbeit der Teams
     + Kleine Arbeitseinheiten (Use Cases)
3. **Iterationen (S.33)**
   * Kurz 🡪 aber Projekt abhängig
   * Ablauf: Kickoff 🡪 Arbeit 🡪 Evaluation
   * Fester Zeitraum (Timebox)
   * Vorteile:
     + Risiken werden früher deutlich/entdeckt
     + Schnelle Veränderungen möglich
     + Rollen profitieren voneinander
     + Fortschritt ist gut messbar
4. **Kickoff**
   * Wahl der Arbeitseinheiten:
     + Wert
     + Zusammenhang mit anderen
     + Risiken
     + Balance: Rollen auslasten
   * 🡪 Kunde such aus NICHT Projektleiter (in Zusammenarbeit mit Team)
   * ! nicht zu viele und nicht zu wenige Use Cases einplanen
5. **Evaluation (S. 41)**
   * Verbesserungen können in den folgenden Iterationen sofort einfließen
   * Teilnehmer: Auftraggeber, Endanwender, Team
   * Umsetzung der Work Items präsentieren und Fortschritt klarstellen
   * Arbeitsweise
   * Tipp: Kickoff für nächste Iteration gleich im Anschluss
   * Go/no-go Diskussion 🡪 Abbruch des Projektes?
6. **Teams/Zusammenarbeit (S.50)**
   * Blickwinkel jeder Rolle 🡪 multidisziplinär
   * Kleine/kompakte Teams 🡪 Bildung mehrerer Teams bei zu großer Anzahl
   * Multidisziplinäre Mitglieder 🡪 Team deckt alle Aufgabenbereiche ab
   * Never change a winning Team 🡪 Teams auf Basis des Zusammenspiels bilden 🡪 Projekte den Teams zuordnen (nicht umgekehrt)
7. **Meetings/Techniken (S.56)**
   * Kickoff (zu Beginn jeder Iteration)
   * Standup (täglich)
     + Transparenz: Fortschritt, Hilfe, Probleme
     + Im Stehen 🡪 10-15 min
   * Evaluation (Ende)
   * Workshops
     + Multidisziplinär
     + Alle Ansichten treffen zusammen
     + Schnelle Entscheidungsfindung
     + Keine Übergaben
   * Pair Programming (an einem PC)
     + Driver + Navigator
     + Wissen teilen und voneinander lernen
     + Schneller zur Lösung
     + Code ist weniger komplex und weniger Code
     + Cleaner Code
   * Side-by-Side (jeder an einem PC)
     + Entwickler arbeiten intensive miteinander
     + Work Item in mehrere Teile zerlegen, die parallel entwickelt warden können
     + Hohe Produktivität
   * Pair Testing (mit Endanwender)
     + Tester in Kombination mit anderen Rollen einsetzen (spezielles
     + Tester als Navigator und Endanwender als Driver
8. **Rollen (S.72)**
   * Rollendefinition ist Projekt abhängig
   * Für jedes Projekt gilt:
     + Alles benötigte Fachwissen muss vertreten sein (Analyse, Entwicklung, Tests)
     + Generalisierende Spezialisten 🡪 bringen weitere Erfahrungen und Fachkenntnisse mit
     + Team und Rollenbesetzung nicht ändern
     + Automatische ausgewogene Beschäftigung
     + Externe Spezialisten für bestimmte Zeit
9. **Stakeholderanalyse (S.74)**
   * Ermitteln aller Personen, die an dem Projekt beteiligt oder davon beeinflusst werden
   * Rollen: Team, Betreiber, Externe, Personalabteilung, Vertrieb, Kunde
   * Stakeholder werden regelmäßig und direkt einbezogen
   * Kunde:
     + Ist für das Projekt verantwortlich und wird deshalb regelmäßig einbezogen
     + Projektbeginn: Scope und Anforderungen bestimmen, Backlog füllen und Work Items schätzen gemeinsam mit Team
     + Iterationsbeginn: Work Items auswählen (Team berät), definition of done festlegen
     + Täglich: Ausarbeitung der Work Items
     + Iterationsende: Abnahme der Work Items
     + Rollen: Sponsor, Product Owner, Anwender, Fachexperte
   * Team:
     + Multidisziplinäre Zusammenarbeit der Rollen
     + Setzen die Software um
     + Allgemeine Rollen: Fachexperte, Entwickler, Tester, Agile Coach
     + Projektleiter
       - Selbstorganisiertes Team
       - Organisierende und unterstützende Rolle
       - Transparenz: Teil des Teams
       - Projektplan aufstellen
       - Risiken managen
       - Status und Fortschritt kommunizieren
       - Kickoff und Evaluation organisieren
       - Iterationsplan aufstellen
       - Fortschritt überwachen
       - Infrastruktur betreuen
       - Team vertreten
       - Kümmert sich um die „Umgebung“ des Teams
     + Agile Coach
       - Kümmert sich um die Arbeitsweise und Zusammenarbeit innerhalb des Teams
       - Unterstützung bei der Umsetzung der agilen Best Practices
       - Fachwissen über agile Ansätze und Techniken
       - Flexibel 🡪 frühere gute Techniken des Unternehmens trotzdem verwenden
     + Allround-Entwickler
       - So früh wie möglich einbeziehen 🡪 Aussagen über Machbarkeit, fachliches Wissen über Projekt aufbauen
       - Schnittstelle zwischen Kunde und Anwender sowie Fachexperten und Testern
       - Zusammenarbeit mit anderen Rollen
       - Architektur 🡪 Softwarearchitektur, Sprache, Frameworks, Services
       - Design 🡪 Benutzeroberfläche, Design Patterns, Objektorientierung, Modellierung
       - Testen
     + Tester
       - Testen in Agile viel effektiver
       - Work Item in einer Iteration entwickeln und testen
10. **Techniken (S. 88)**
    * Automagische Architektur
      + Softwarearchitektur während des Projektes natürlich entstehen lassen
      + Benötigt trotzdem Auseinandersetzung und am besten eine baseline
    * Test Driven Development
      + Tests für den geplanten Code schreiben
      + Code schreiben und Tests laufen lassen
      + Vorteile:
        - Entwickler denkt über das nach außen hin wahrnehmbare Verhalten des Codes nach
        - Cleanerer Code
        - Bessere Qualität (ständige Prüfung)
      + Erfordert Disziplin
      + Testet nur das korrekte Funktionieren des Codes und nicht die gewünschte Funktionalität 🡪 dafür sind Acceptance Tests
      + Mehr Aufwand bei der Pflege 🡪 mehr Code
      + Code Coverage 🡪 wie viel Code ist durch die Tests abgedeckt
    * Refactoring
      + Struktur und Qualität des Codes verbessern
      + Verschieben von Code, Extrahieren neuer Methoden, Umbenennung
      + Verringert Komplexität und steigert die Lesbarkeit
      + Nötig weil Code durch weitere Work Items immer komplexer wird
      + Automatismus
    * Continuous Integration
      + Automatischer Build-Prozess
    * Specification by Example
      + Kunde und Team stellen eine gemeinsame eindeutige Definition für Anforderungen von Work Items auf
      + Beispiele suchen, an denen die Definitionen automatisch getestet werden können
      + Acceptance Test Driven Developement
      + Abnahme von Software teilweise automatisieren
11. **Agile Anforderungen (S. 101)**
    * Prinzip: You ain’t gonna need it 🡪 alle Arbeit an Anforderungen erst erledigen, wenn sie wirklich gebraucht werden
    * Wissenszuwachs: alles was man bis zur Ausarbeitung dazugelernt hat, fließt mit ein (keine Change Requests)
    * Wissen des Kunden und des Teams wächst 🡪 können später im Projekt die Anforderungen schneller ausarbeiten
    * Nur die Anforderungen werden ausgearbeitet, die auch tatsächlich umgesetzt werden
12. Arbeitseinheiten (S.103)
    * Work Items immer wieder neu priorisieren und ordnen
    * Kleine Work Items
    * Bessere Verteilung
    * Bessere Reaktion auf Verzögerung 🡪 Ausweichen
    * Arbeitseinheiten definieren:
      + Ableitbar aus den Anforderungen des Kunden
      + Niedrige Abhängigkeit zu anderen 🡪 Granularität
      + Frühzeitig
      + Umfang und Komplexität müssen schätzbar sein
      + Als Arbeitspacket umsetzbar
      + Testbar und abnehmbar
      + Wiederverwendbar
      + Ausreichend Dokumentation möglich
    * Lebenszyklus
      + Anzeige an einem Dashboard
      + Beispiele:
        - Scrum: Geplant, in Arbeit, Fertig
        - Smart: Neu, in Iteration, in Arbeit, Test, Überarbeitung, Abnahme
    * Use Cases
      + Beschreiben funktionelle Anforderungen
      + Definiert die Interaktion zwischen einem Akteur und dem System
      + Bereits vor Agile eingesetzt
      + Smart Use Cases (kleine Einheiten eines Use Cases) hilfreich für agiles Vorgehen
      + Bestandteile der Definition:
        - Name
        - Akteure
        - Vorbedingungen
        - Nachbedingungen
        - Standardablauf
        - Alternative Abläufe
      + Ablauf
        - Scope definieren
        - Gruppieren
        - Use Cases
        - Use Cases durch Smart Use Cases erweitern
        - Schätzen
        - Umsetzen
      + Smart Use Cases dienen als Arbeitseinheiten
13. **Stolpersteine (S.179)**