Inhalt

User-Stories

- Nutzergeschichten
- Grundlagen für SCRUM-Projekt

Anforderungen

- Aus Sicht und in der Sprache des Kunden formuliert
- Beschreibt was Software für den Kunden machen muss
- Genau eine Sache beschreiben

Muster

Als <*Benutzerrolle*> will ich <*das Ziel*>*[, so dass <Grund für das Ziel>]*

- Benutzerrolle kennzeichnet Urheber
- Ziel entspricht der Anforderung
- Optionaler Grund ist die Motivation

INVEST-Merkmale

I ndependant: unabhängg

N egotiable: verhandelbar

V aluable: wertvoll

E stimable: schätzbar

S mall: klein

T estable: testbar

Planning Poker (SCRUM Poker)

- Agiles Schätzverfahren
- Spielerisch Aufwände für Elemente im Backlog (Tasks, Epics, User Stories) schätzen
- **Story-Points**:
    - Ergebnis der Schätzung -> Aufwand des Elements
    - Relative Metrik
- Basis für eine Runde: Eine User-Story
- Unterschiedliches Know-How -> gleicht fehlerhafte Bewertungen aus
- Durch Diskussionen über Aufwand wird ein einheitliches Verständnis hergestellt
- Durch Einigung wird eine geteilte Verantwortung hergestellt

Ablauf

- Idealerweise nicht mehr als 10 Teilnehmer

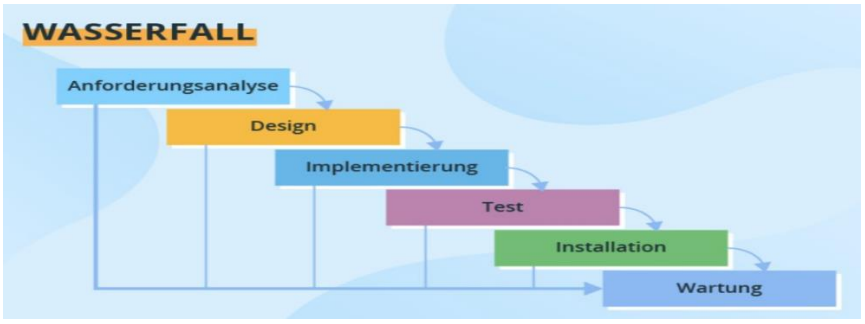
1. Jeder Mitspieler hat 13 Karten (0; 0,5; 1; 2; 3; 5; 8; 13; 20; 40; 100; ?; Pause)
2. Scrum-Master liest Backlog-Item vor
3. Scrum-Master fordert zum Schätzen auf
4. Jeder Teilnehmer legt eine seiner Karten verdeckt ab
5. Nach einem Zeichen des Scrum-Masters werden alle abgelegten Karten umgedreht
    a. Bei gleichen Ergebnissen steht das Ergebnis fest

   b. Bei starken Abweichungen begründen die Spieler mit dem höchsten und niedrigsten Schätzwert (Redeverbot für alle anderen)
- Schätzung wird solange fortgeführt bis ein Konsens entstanden ist (Normal: 3 Runden)

Vorgehensmodelle

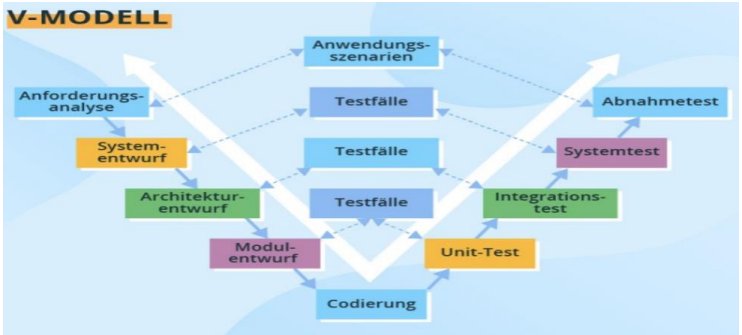
Klassische Modelle

Wasserfallmodell



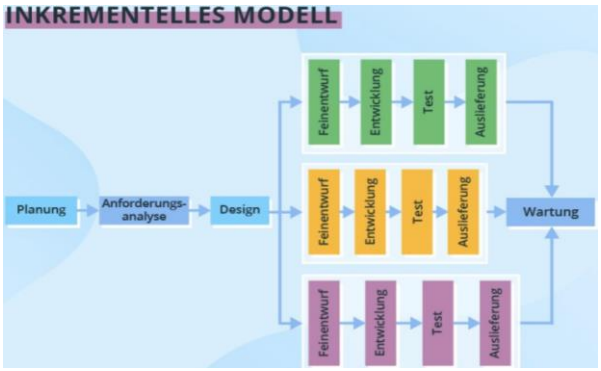
- Aufeinanderfolgende Stufen
- Stark dokumentiert
- Jede Stufe basiert auf Ergebnissen von der vorherigen Stufe
- Fehler fallen erst am Ende des Projekts auf
- Test erst nach Ende der Entwicklung
- **Einsatz**:
  - Kleine oder mittelgroße Softwareprojekte
  - Projekte bei denen eine starke Kontrolle notwendig ist
  - Projekte bei denen ein bekanntes Tech-Stack und Tools zum Einsatz kommen

V-Modell



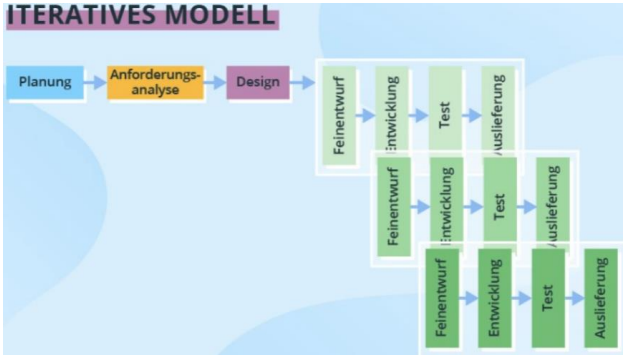
- Linearer Aufbau: Alle Phasen nacheinander
- Jede Stufe hat eigene Tests
- Alle Anforderungen zu Beginn erfasst -> Keine Änderung möglich
- **Einsatz**:
  - Projekte bei denen Störungen und Ausfallzeiten inakzeptabel sind (Medizin)

Inkrementelles Modell



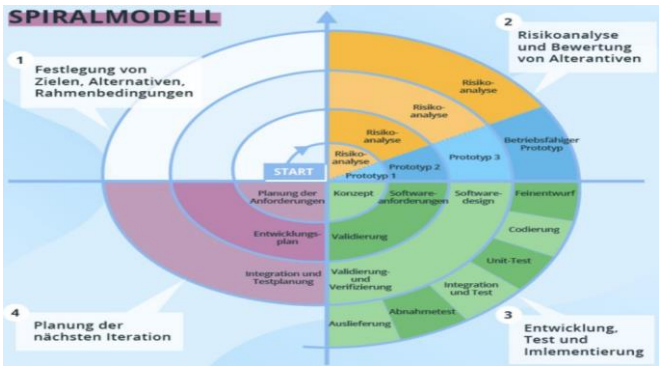
- Entwicklungsmodell in mehreren Iterationen -> Modularer Ansatz
- Zuvor hinzugefügte Teile bleiben unverändert
- Entwicklung kann entweder sequentiell(langsamer) oder parallel (schneller) verlaufen

Iteratives Modell



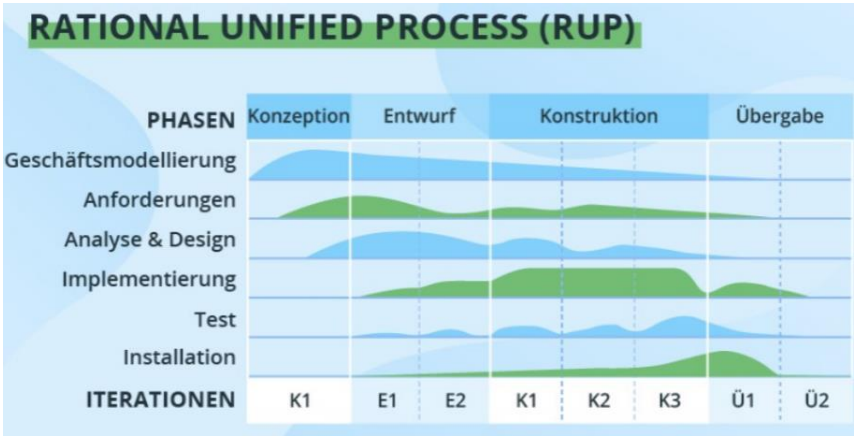
- Durchläuft einzelne Phasen mehrmals
- Jedes Mal wird die Software weiterentwickelt
- Keine Notwendigkeit der vollständigen Spezifikation zu Beginn
- Nur die wichtigsten Anforderungen zu Beginn definiert
- Flexibel
- Basiert auf Kundenfeedback
- **Einsatz**:
    - Große Projekte

Spiralmodell



- Fokussiert auf gründliche Risikobewertung
- Iteration von 6 Monaten in je 4 Teilen:
    - Planung
    - Risikoanalyse
    - Erstellung von Prototypen
    - Bewertung des zuvor ausgelieferten Teils
- Kunden werden schon in frühen Phasen eingebunden
- Während der Entwicklung sind Ergänzungen inakzeptabel
- **Einsatz**:
    - Projekte mit anspruchsvollen Anforderungen

RUP: Rational Unified Process



- Kombination aus linear und iterativ
- Vier Phasen:
    - Konzeption
    - Entwurf
    - Konstruktion
    - Übergabe
- Jede Phase außer Konzeption in mehreren Iterationen
- Aktivitäten werden parallel über die Phasen durchgeführt
- **Einsatz**:
    - Große und risikoreiche Projekte

Agile Modelle

Agile Grundbedingungen

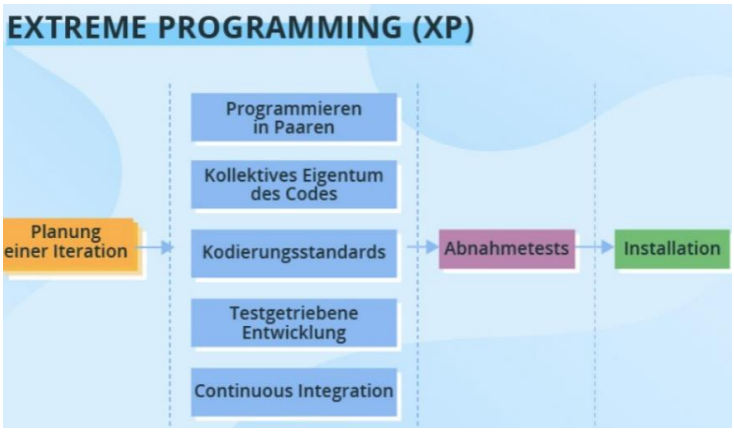
- Iterative Entwicklung
- Intensive Kommunikation
- Frühzeitiges Kundenfeedback
- Jede Iteration (mehrere Wochen)
    - Vollständige funktionierende Softwareversion
    - Schnelle Entwicklung von Zwischenprodukten
- Weniger Wert auf Dokumentation
- Mehr Aufmerksamkeit auf Testaktivitäten

Agiles Manifest

- Enge Zusammenarbeit im Team und mit Kunden
- Kundenbedürfnisse im Mittelpunkt
- Von Iteration zu Iteration Qualität verbessern
- Effektiver Entwicklungsprozess

Modelle

XP: Extreme Programming



- Kurze Iterationszyklen (1-2 Wochen) und Kundenbindung im Mittelpunkt
- Änderungen in den Entwicklungsprozess integrieren
- Flexibilität erschwert die Auslieferung
- Bewährte Praktiken für XP
    - Programmieren in Paaren
    - Testgetriebene Entwicklung
    - Testautomation
    - Continuous Integration
    - Kleine Releases

SCRUM

- Siehe Cheat-Sheet

Best Practice

Verständlichen Code schreiben

Gründe

- **Teamwork**, Wartung
- **Fehlerrate**: Nachvollziehen was passiert
- **Debugging**: Leichter verstehen
- **Änderbarkeit**
- **Wiederverwendbarkeit**
- **Produktqualität**
- **Bessere Entwicklungszeit**

Lösungen

Entwicklungsrichtlinien

- Auf Ebene der Modellierung (Architekturentscheidungen)
- Gestaltungsrichtlinien für graphische Oberflächen
- Auf Prozessebene (Pull Requests, Conventional Commits)
- Können teilweise automatisch erzwungen werden (Linting)

Namenskonventionen

- So spezifisch wie möglich
- Mehrdeutigkeit vermeiden
- Erhöhte Lesbarkeit z.B. CamelCase
- Direkte Unterscheidung von Klassen, Methoden, Variablen, Konstanten…
- Konventionen abhängig vom Ökosystem
- An etablierten Bibliotheken orientieren

Beispiel JAVA

- Klassennamen
    - Einfache, beschreibende Namen
    - Ganze Wörter
    - Erster Buchstabe groß
- Methodennamen
    - Verb oder Verbalphrase in „camelCase“
    - Schwache Verben vermeiden
    - Prädikate mit „is“ beginnen (isEmtpy())

Magische Zahlen

- Konstante Zahlenwerte im Code durch Variablen ersetzen

Selbstdokumentierender Code

- Guter Code benötigt wenig oder keine Kommentare
- Dokumentation durch
    - Gute Variablen- & Methodennamen
    - Geringe Komplexität

UML

- Siehe Cheat-Sheet