Übung 09

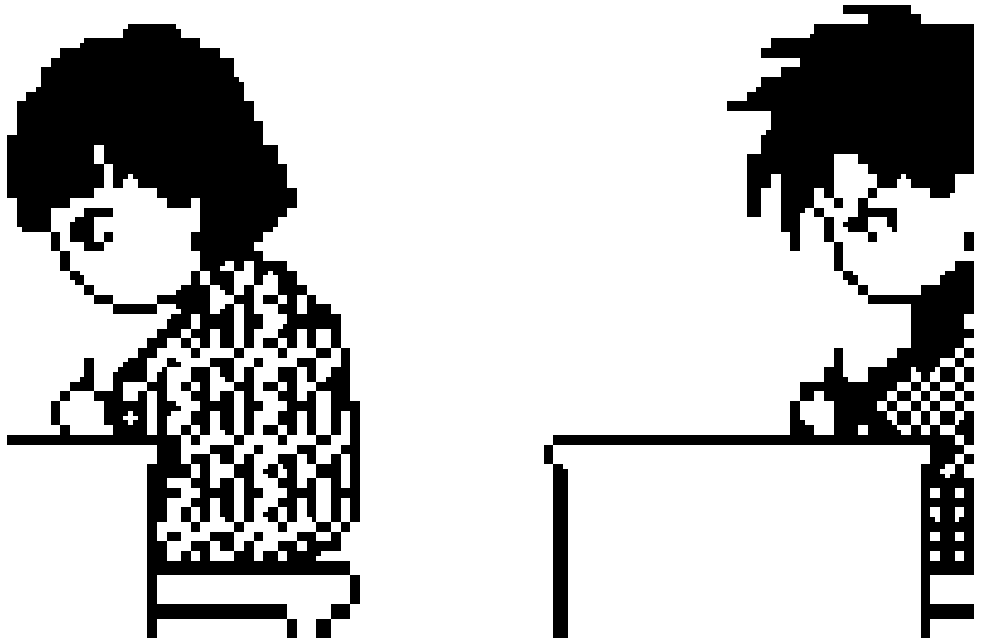
Softwareentwicklungsprozesse

0. Organisatorisches

- Nur noch 1x Übung
  - 27.01.2017

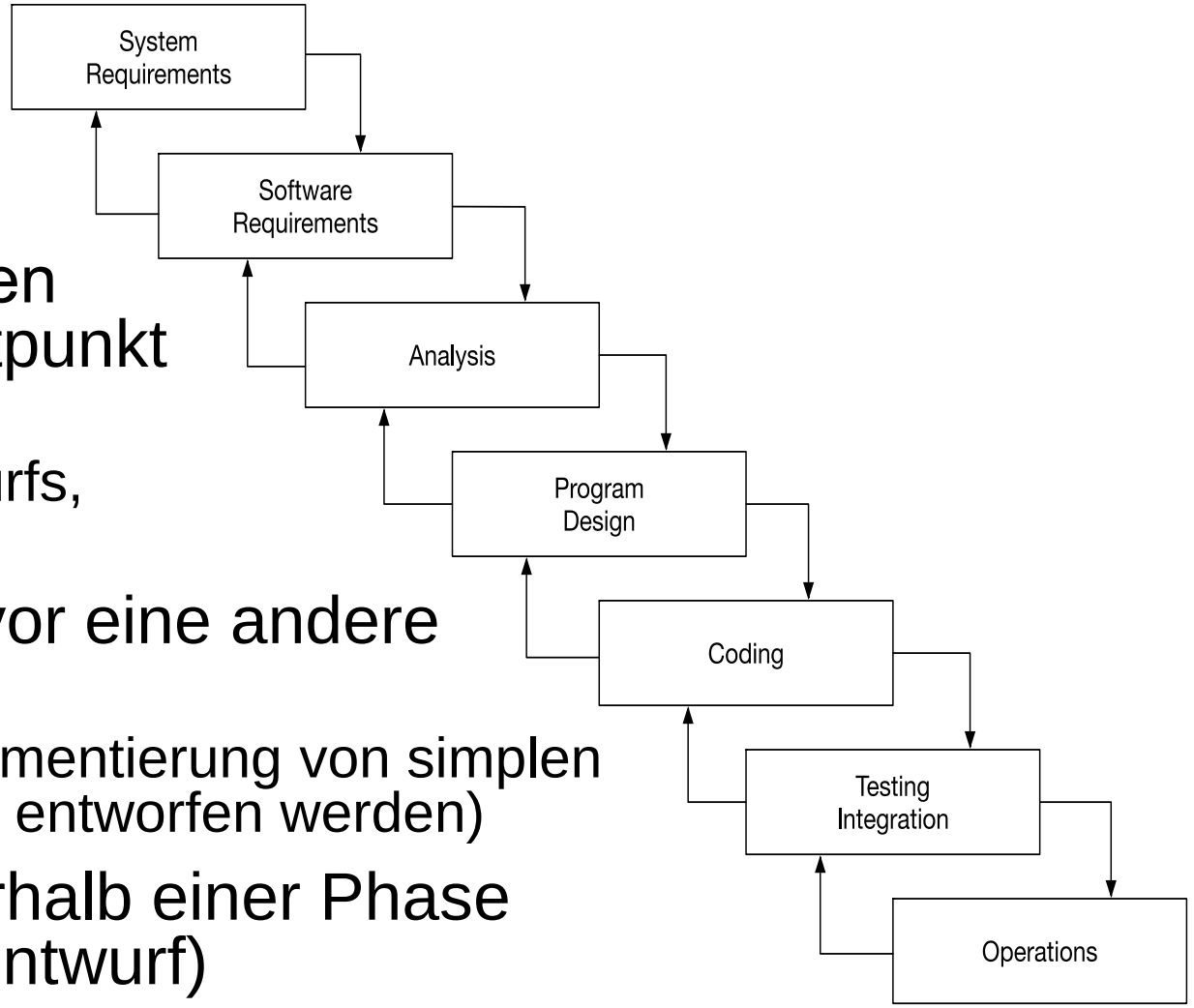
    → Übung 10: Projektmanagement
  - 03.01.2017

    → **Klausurvorbereitung?**
- **Gastvortrag**: 26.01.2017
- **Klausur**: Do. 09.02.2017, 13.00 Uhr, SR 015, B11



01. Prozesse: Wasserfall-Modell vs. moderne Entwicklung

- Wasserfallmodell verlangt klar abgegrenzte und statische Phasen, was ein Software-Projekt nicht besitzt
- “Roll-Back“ bei neuen oder vergessenen Anforderungen zu einem späteren Zeitpunkt gefordert
  - Ggf. Überarbeitung des kompletten Entwurfs, der Implementierung und der Test-Suite
- Eine Phase muss beendet werden bevor eine andere Phase begonnen werden kann
  - Somit wird kein Prototyping erlaubt (Implementierung von simplen Komponenten, während komplexere noch entworfen werden)
- Berücksichtigt keine Änderungen innerhalb einer Phase (unrealistisch für Anforderungen und Entwurf)

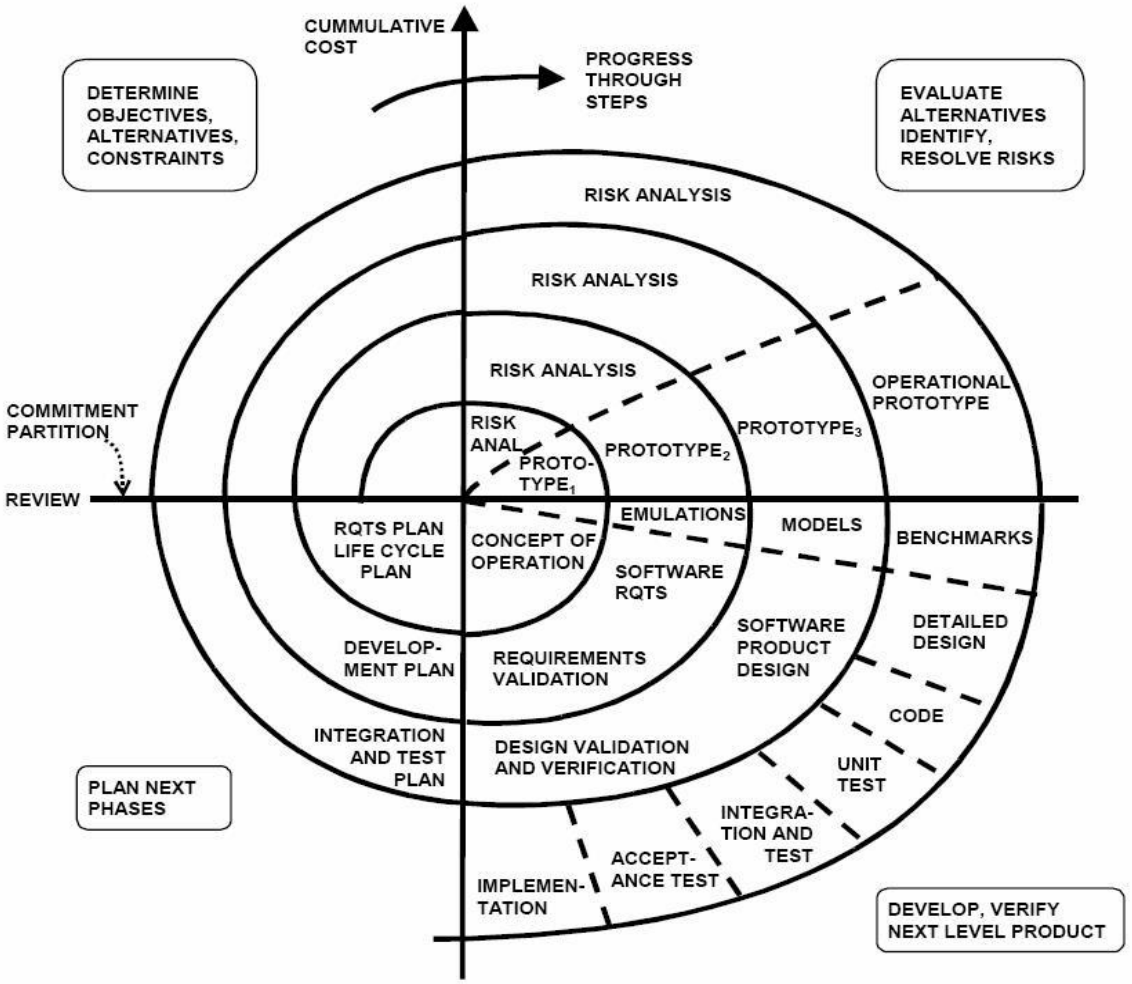


01. Prozesse: Erfolg des Wasserfall-Modells

- **Einfach und intuitiv**, da es die Prozessaktivitäten (z.B. Spezifikation, Entwicklung, Validierung) direkt auf entsprechende, voneinander getrennte Phasen (Design, Implementierung, Testen) abbildet
- Erlaubt **vermeintlich gute Projektplanung**
- Wird in **anderen Bereichen verwendet**
  - Gut verstanden
  - Wird in Projekten eingesetzt, in denen die Entwicklung von Software nur Teil eines größeren Projektes ist
- Kleine Projekte und Teilprojekte mit jeweils **konkret definierten Anforderungen** können auch innerhalb eines inkrementellen Software Prozess-Modells mit dem Wasserfall-Modell umgesetzt werden, sofern dies **zweckmäßig** erscheint

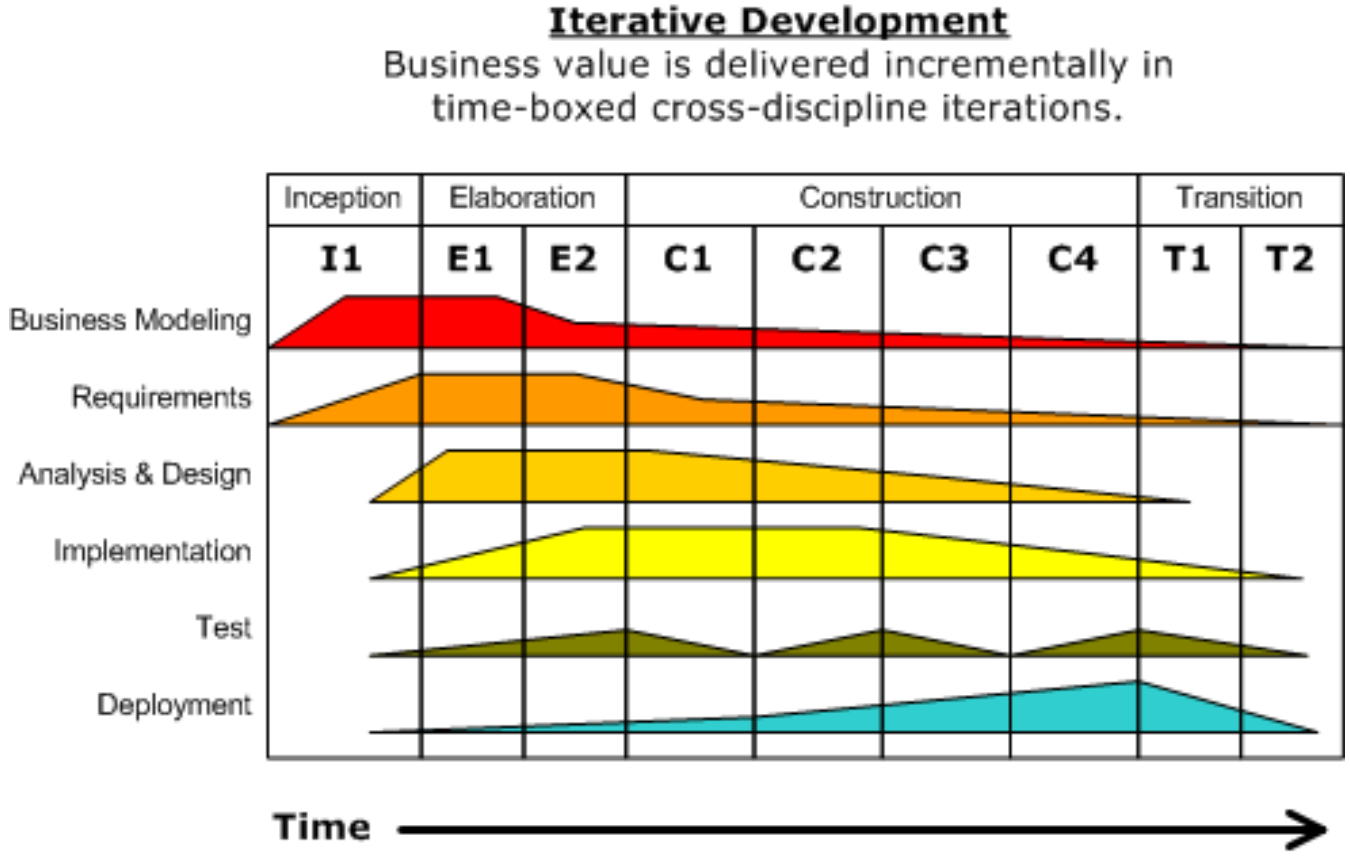
01. Prozesse: neue Prozess-Modelle: Spiralmodell

- Frühes Erkennen von Mängeln durch iteratives durchlaufen von verschiedenen Phasen
- Prototyp je Iteration ermöglicht zeitnahe Präsentation einer vorläufigen Version für den Kunden
  - Aktive Korrektur mit Hilfe des Kunden von falscher Anforderungsumsetzung



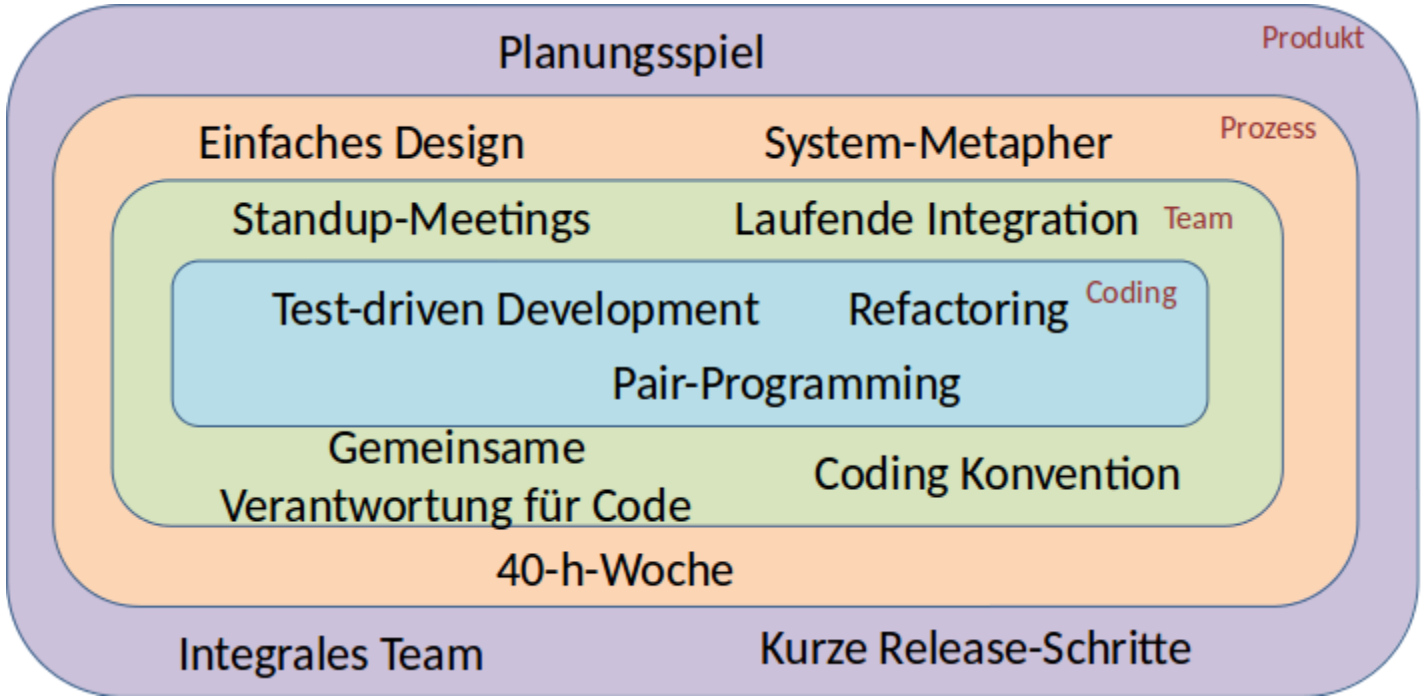
01. Prozesse: neue Prozess-Modelle: Unified Process

- Auftrennung des Prozesses in
  - **statische Workflows:** Anforderungen, Analyse, Entwurf, Implementierung, Test
  - **dynamische, iterative Phasen:** Konzeption, Entwurf, Konstruktion, Übergang
- Bewirkt Entkopplung, was die Parallelisierung von Aufgaben ermöglicht
- Kurze Iterationen (Wochen)
- Jede Iteration endet mit vollständig laufendem System



01. Prozesse: neue Prozess-Modelle: Extreme Programming

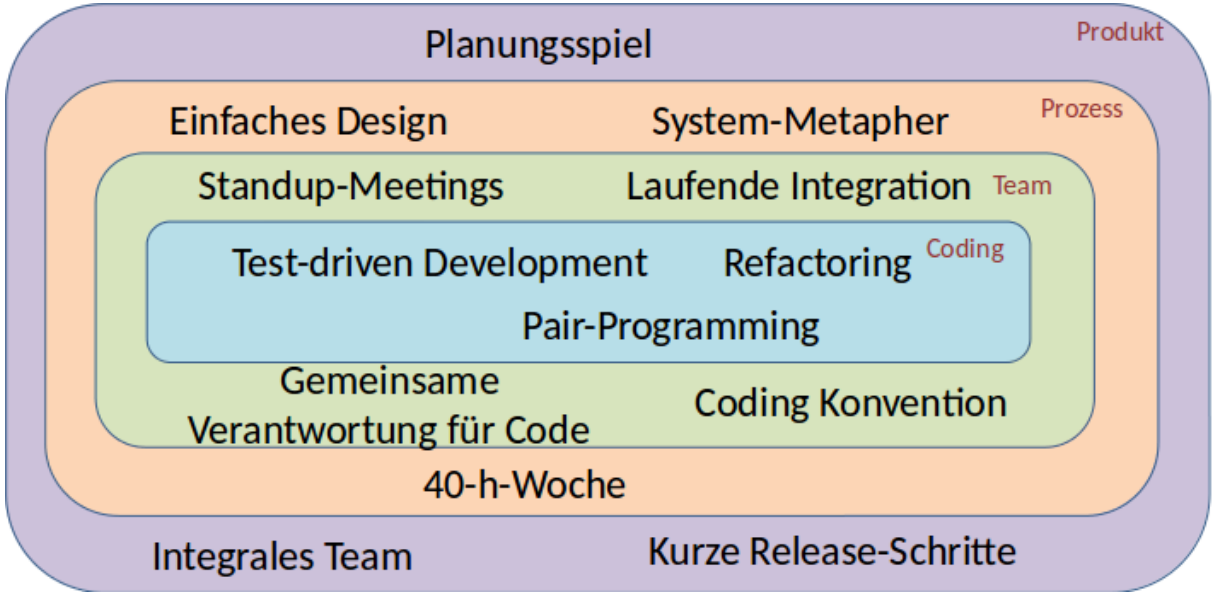
- Extreme Version der iterativen Entwicklung
- Inkrementelle Planung → fortlaufende Integration und testgetriebene Entwicklung
- Kunde aktiv beteiligt
- Identifizierung von Schwachstellen früh und leicht behebbar
- Aufteilung in Menge von verschiedenen Methoden bzw. Praktiken



01. Prozesse: neue Prozess-Modelle: Extreme Programming

- **Test-driven-Development**
  - Mittels unit-Testing
  - Tests werden vor der Implementierung erstellt
- **Refactoring**
  - Nur was aktuell benötigt ist wird implementiert
  - Refactoring wird früher oder später ein muss
- **Pair-Programming**
  - Zwei Entwickler an einem Rechner
  - Zwangloser Review-Prozess: jede Codezeile wird von mindestens 2 Personen begutachtet
  - Impliziter Wissensaustausch

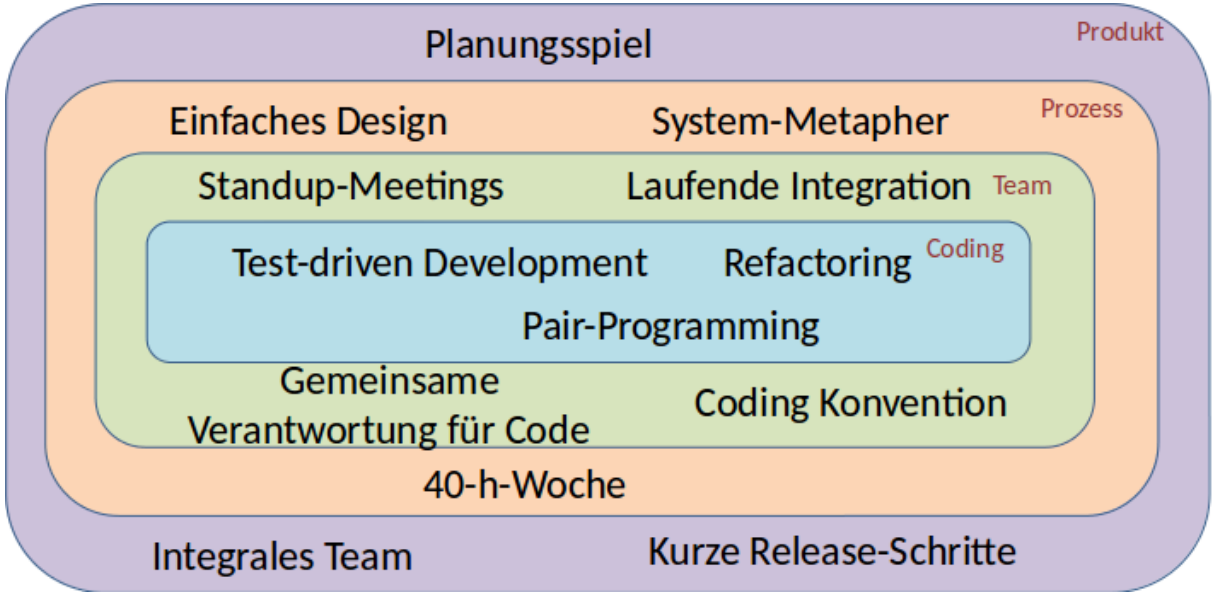
{Coding}



01. Prozesse: neue Prozess-Modelle: Extreme Programming

- **Standup-Meetings**
  - Tägliche kurze Besprechung
  - Was wurde am Vortag geleistet? Wo gab es Probleme? Was will man heute schaffen?
- **Laufende Integration**
  - Kontinuierliche Integration einzelner Komponenten zu einem lauffähigen Gesamtsystem in kurzen Zeitabständen
- **Gemeinsame Codeverantwortung**
  - Jeder trägt Verantwortung für alles
  - Jeder kann gesamte Codebasis ändern
  - Fehler können schnell korrigiert werden
- **Coding Konvention**
  - Team hält sich bei der Programmierarbeit an Standards
  - Erhöht Lesbarkeit und verbessert Zusammenarbeit

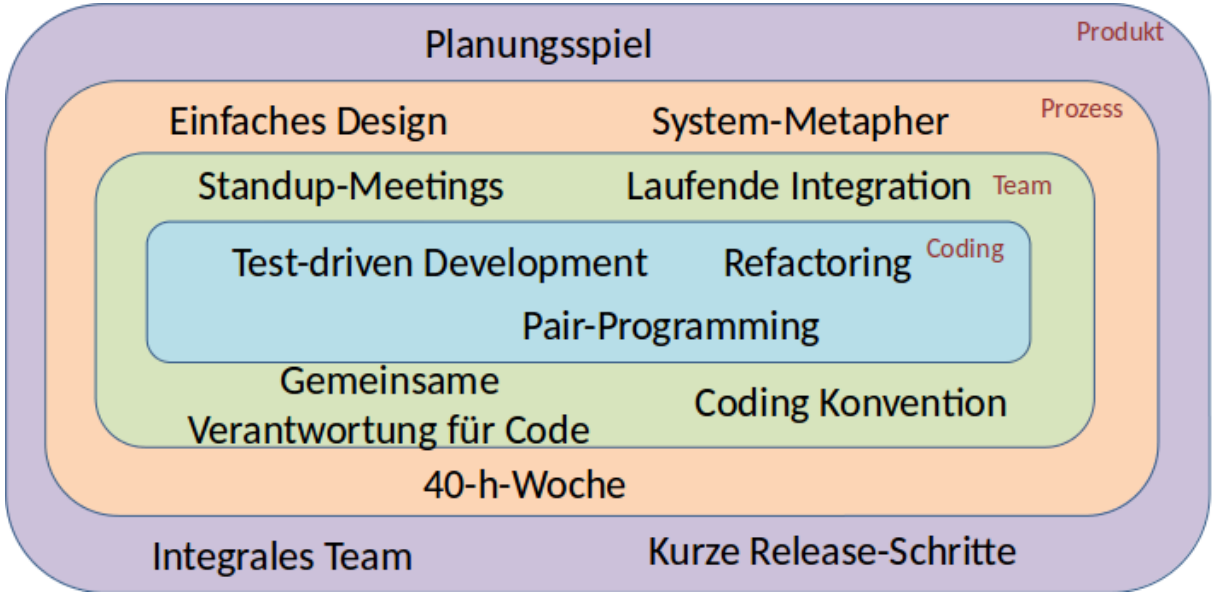
{Team}



01. Prozesse: neue Prozess-Modelle: Extreme Programming

- **Einfaches Design**
  - Die einfachste Umsetzung bzw. Lösung für eine Funktionalität oder ein Problem soll bevorzugt werden
- **System-Metaphor**
  - Anforderungen werden im fachlichen Vokabular des Kunden beschrieben, damit keine Missverständnisse zwischen Entwickler und Kunde auftreten
- **40-Stunden-Woche**
  - Überstunden sind zu vermeiden
  - Produktivität eines Entwicklers sinkt mit Überstunden
  - Überstunden sind Anzeichen für falsche Planung

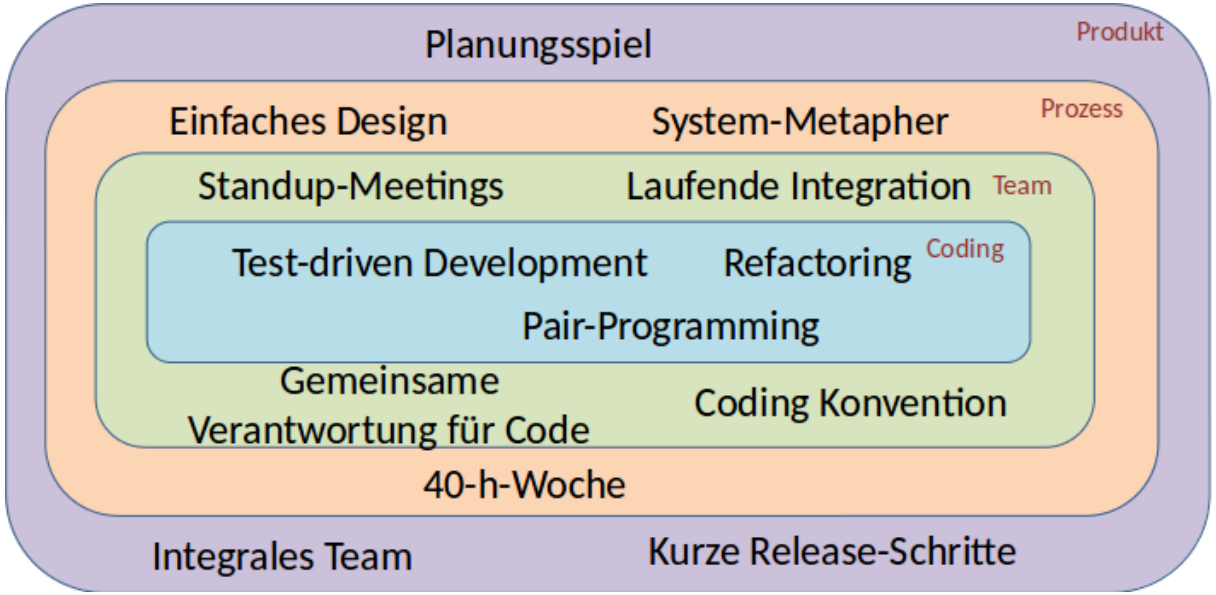
{Prozess}



01. Prozesse: neue Prozess-Modelle: Extreme Programming

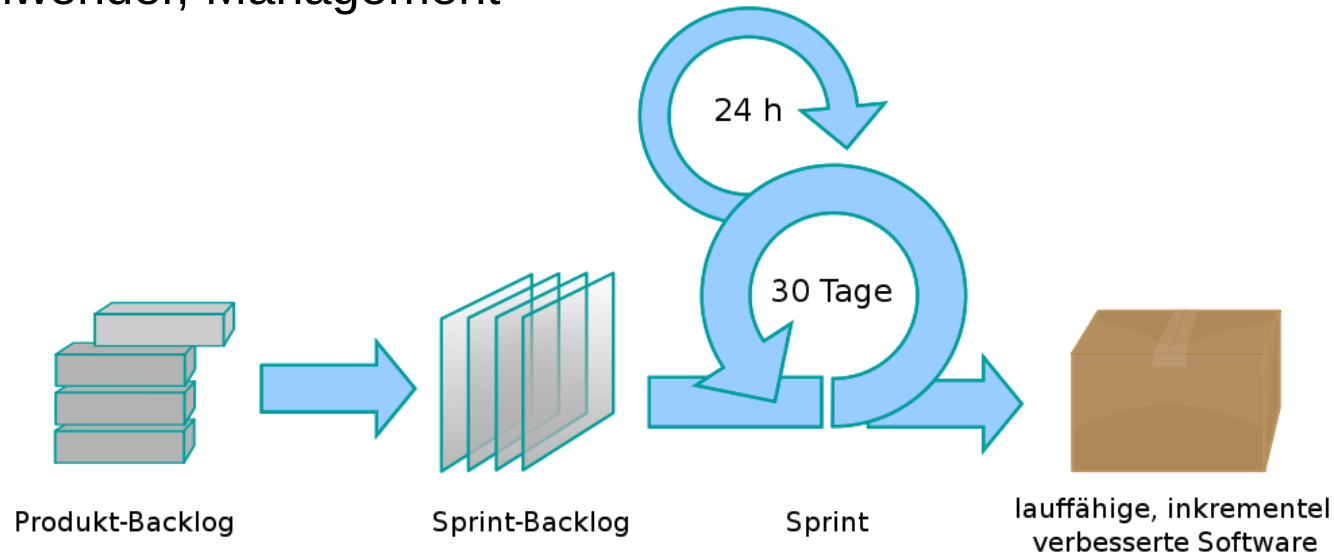
- **Planungsspiel**
  - Treffen ein mal pro Iteration
  - Release-Planning: mit Kunde; welche Requirements sollen enthalten sein und wann sollen sie ausgeliefert werden
  - Iteration-Planning, nur Entwickler; Aktivitäten und Aufgaben der Entwickler
  - Ziel: Das Produkt zu seiner Auslieferung verhelfen
- **Integrales Team**
  - Kundenvertretung ist am Entwicklungsprozess aktiv beteiligt
  - Verkürzt Reaktionszeit bei Fragen
  - Erhöht das Verständnis auf beiden Seiten
- **Kurze Release-Schritte**
  - Prototypische Implementierungen werden nach und nach vervollständigt
  - Kunde kann früh Anpassungen und Änderungen beauftragen

{Produkt}



02. Scrum vs. Extreme Programming

- **Scrum**
  - Vorgehensmodell der agilen Softwareentwicklung
  - Planung nach dem Prinzip der schrittweisen Verfeinerung
  - Produkteigenschaften (Features) in Product Backlog in priorisierter Reihenfolge festgelegt
  - Entwicklung in sogenannten Sprints
  - Rollen:
    - Intern: Product Owner, Entwicklungsteam, Scrum Master
    - Extern: Kunde, Anwender, Management



02. Scrum vs. Extreme Programming

Scrum

- Fokussiert auf Managementaspekten
- 3 Phasen: Outline, Sprint Cycles, Project Closure
- Iterationen: 1-4 Wochen
- Keine Änderungen während eines laufenden Sprints
- Keine spezifischen Praktiken – Teams können ihre Engineering-Methoden zum Projekt selbst wählen
- Nur Product Owner hat direkten Kontakt zu Kunden

Extreme Programming

- Fokussiert auf Entwicklungsaspekten
- Sammlung von Praktiken
- Iterationen: 1-2 Wochen
- Solange ein Feature noch nicht implementiert wurde, kann es durch ein gleichgroßes Feature ausgetauscht werden
- Förderung von test-driven-development, pair programming, ...

03: Agile Methods: User Stories

03: Agile Methods: User Stories

- Kurzer, 1-2 Sätze langer Text
- Beschreibt ein bestimmtes Bedürfnis eines Benutzers
- Aus Anwendersicht: „Als <Benutzerrolle>, möchte ich <Funktionalität>, damit ich <Ziel> erreichen kann.“
- Einfache Sprache, keine technischen Fachbegriffe
- Längere User Story wird auch Epic genannt
  - Implementierung enthält häufig Bugs
  - Oft falsch verstandene Implementierungsversuche

03: Agile Methods: User Stories

- **Story:**
  - Jeder angemeldete Anwender muss vor der Verwendung personenbezogener Daten bestätigen, dass er die Datenschutzerklärung gemäß den rechtlichen Anforderungen beachtet.
- **Tasks:**
  - Textinformation wird bei jeder Systemanmeldung angezeigt und muss bestätigt werden.
  - Bestätigt der Anwender die Datenschutzerklärung nicht, wird er nicht am System angemeldet.
  - Die Datenschutzerklärung muss durch beauftragte Personen (Rollen) inhaltlich gepflegt werden können.

http://www.hwk-chemnitz.de/fileadmin/user_upload/ausbildung/Pruefungen/Gesellenpruefungen/Fragezeichen.JPG