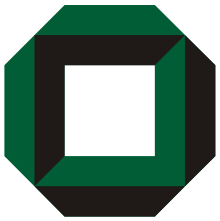


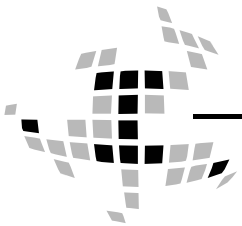
Universität Karlsruhe (TH)

Forschungsuniversität · gegründet 1825

Softwaretechnik 1 Tutorium

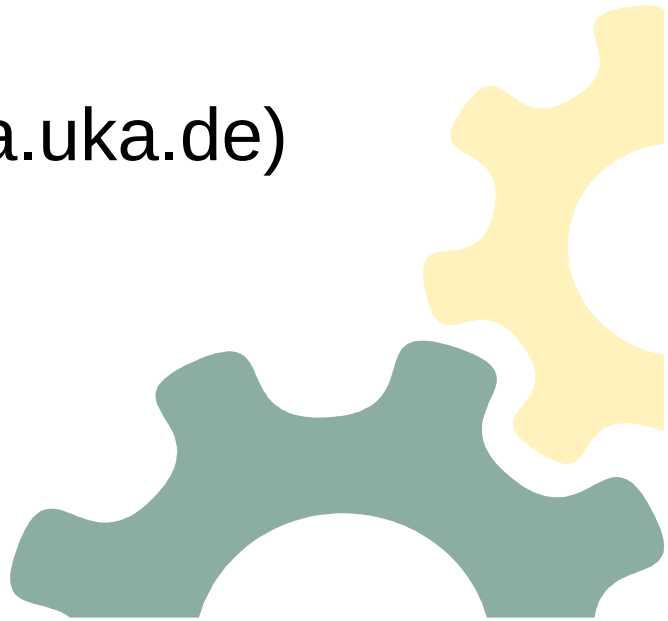
06. Juli 2009

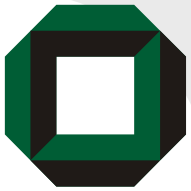
Matthias Thoma (s_thoma@ira.uka.de)



Fakultät für **Informatik**

Lehrstuhl für Programmiersysteme

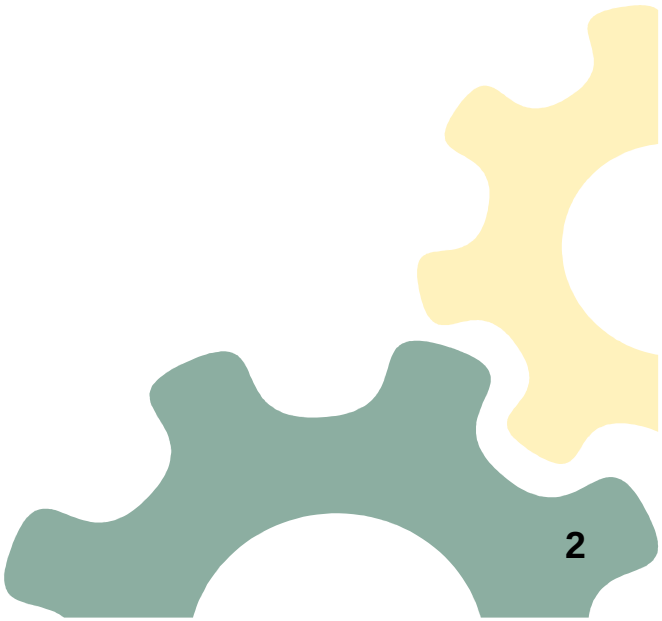


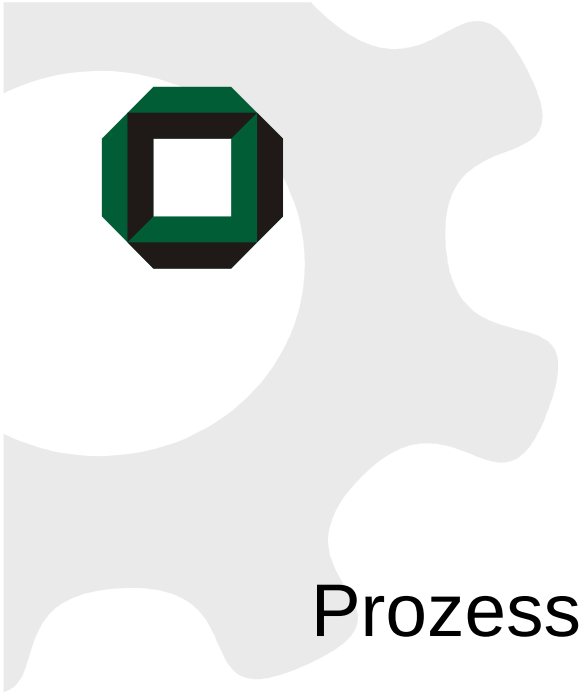


Heute

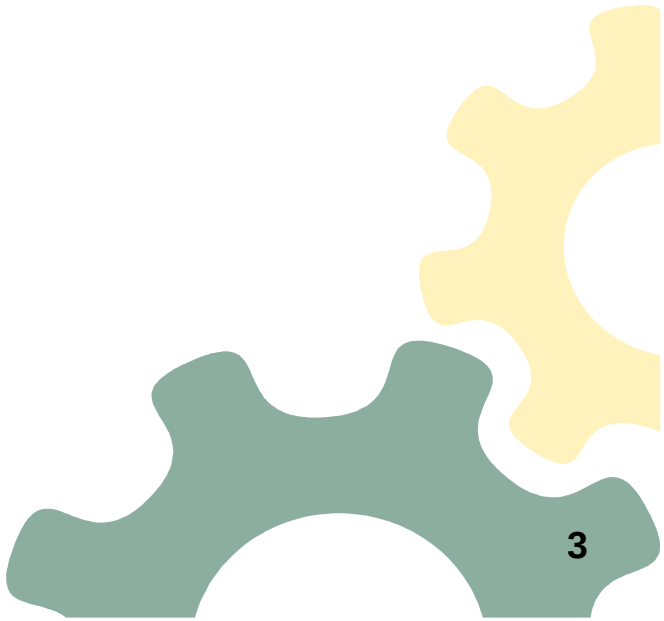
- Prozessmodelle

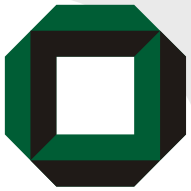
- Parallelverarbeitung
- Planungsphase
- Definitionsphase
- Entwurfsphase
- Implementierungsphase
- Testen & Abnahme
- Einsatz und Wartung





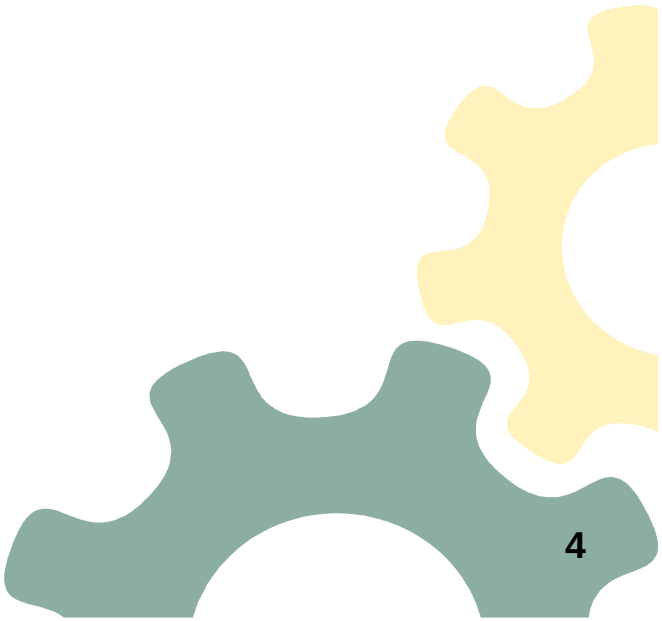
Prozessmodelle



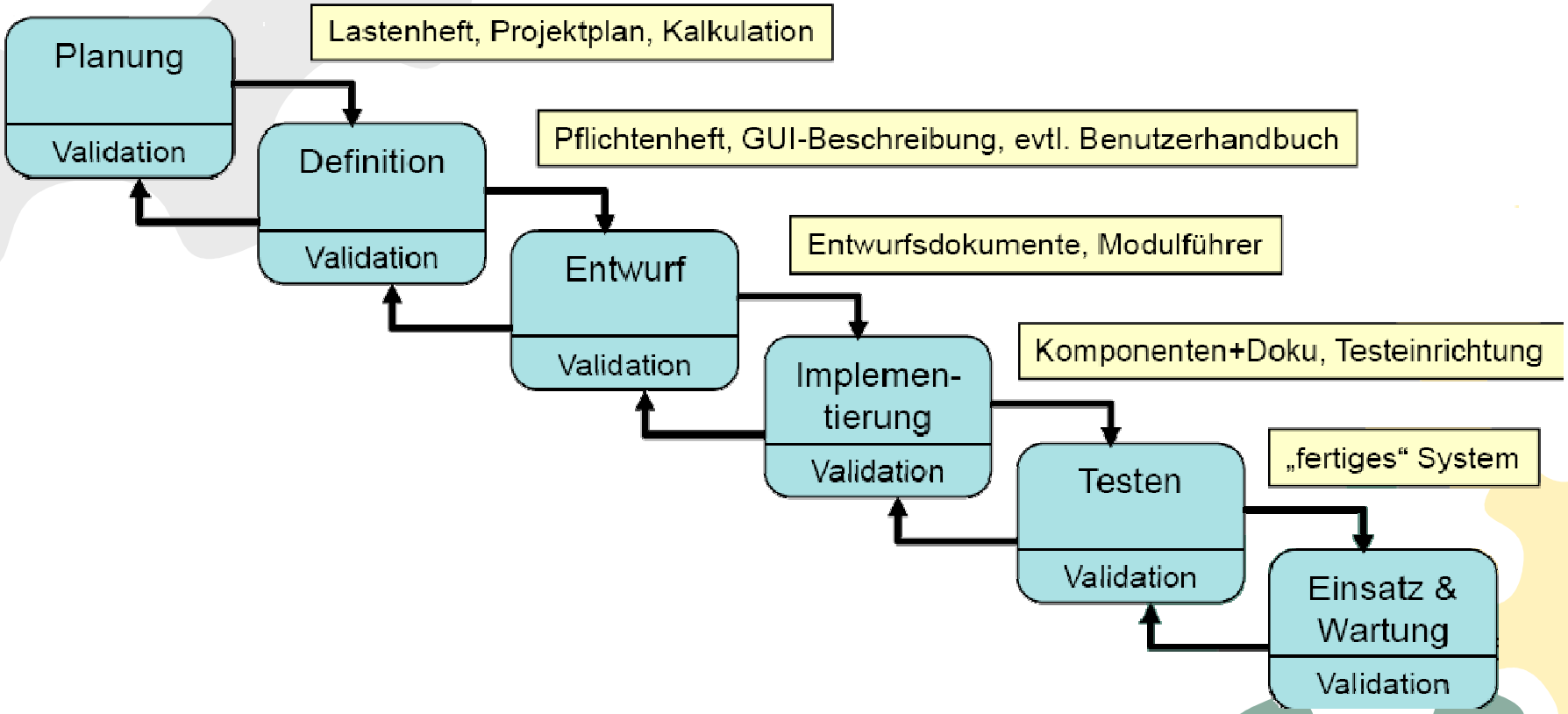


Prozessmodelle - kurzgefasst

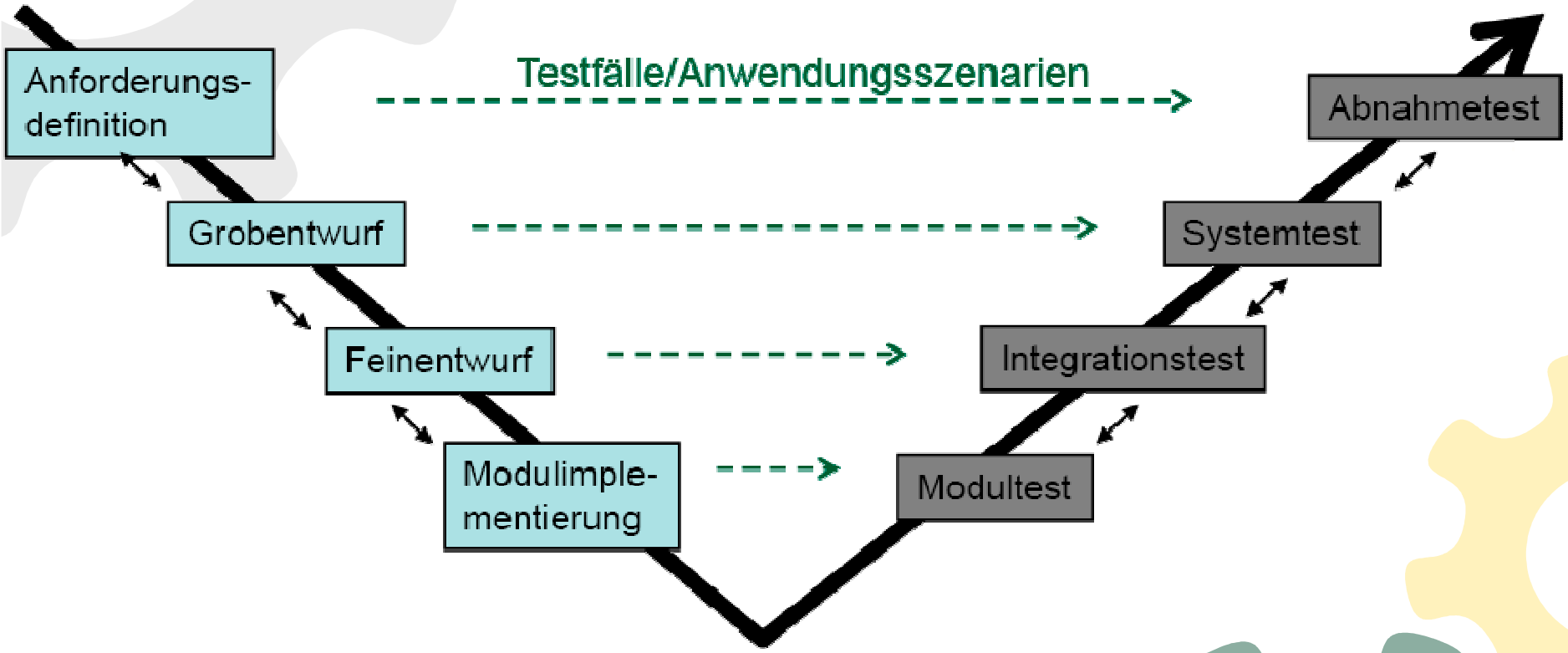
- Klassisches Wasserfallmodell
- V-Modell
- Iteratives Modell
- Synchronize and Stabilize
- Extreme Programming

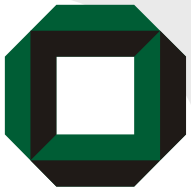


Wasserfallmodell



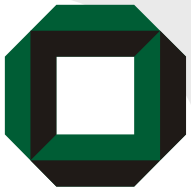
V-Modell


Anforderungs-
definition
Testfälle/Anwendungsszenarien
Abnahmetest
Grobentwurf
Systemtest
Feinentwurf
Integrationstest
Modulimple-
mentierung
Modultest



V-Modell XT

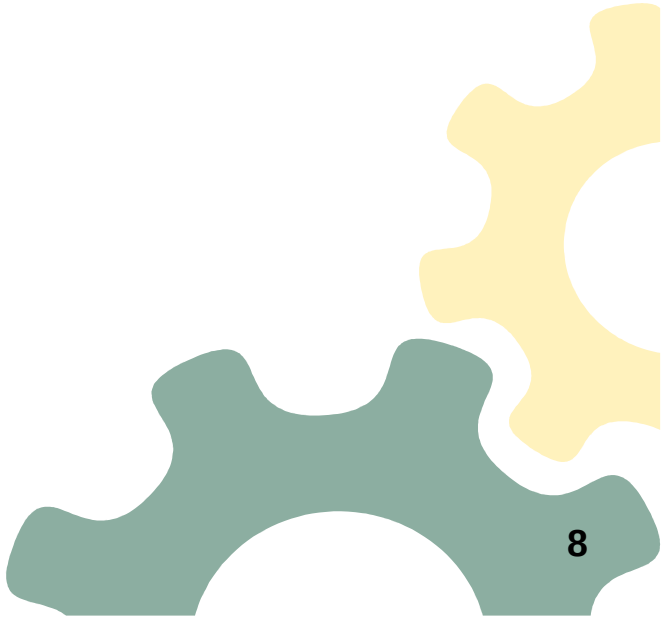
- Produkt wird aus vielen (>30) verschiedenen Perspektiven (Rollen) betrachtet.
- V-Modell XT gliedert die Submodule (Projektmanagment, Qualitätssicherung, Konfigurationsmanagment, Systemerstellung) in Vorgehensbausteine.
- Jedes definierte Produkt durchläuft vier Zustände:
  - Geplant, in Bearbeitung, vorgelegt, akzeptiert

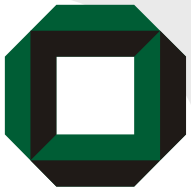


Iteratives Modell

Idee: Auch wenn sich nicht immer ein komplettes System definieren lässt, so lassen sich zumindest Teile der Funktionalität klar definieren und realisieren.

=> Funktionalität wird Schritt für Schritt erstellt und dem Produkt „hinzugefügt“

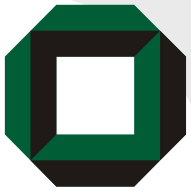




Iteratives Modell

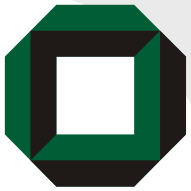
- 2 Ansätze:
  - Evolutionär: Plane und analysiere nur den Teil, der als nächstes hinzugefügt wird (x-faches Wasserfall-Modell)
  - Inkrementell: Plane und analysiere alles und iteriere dann n-mal über Entwurfs-, Implementierungs-und Testphase.

=> Oft Mischformen!



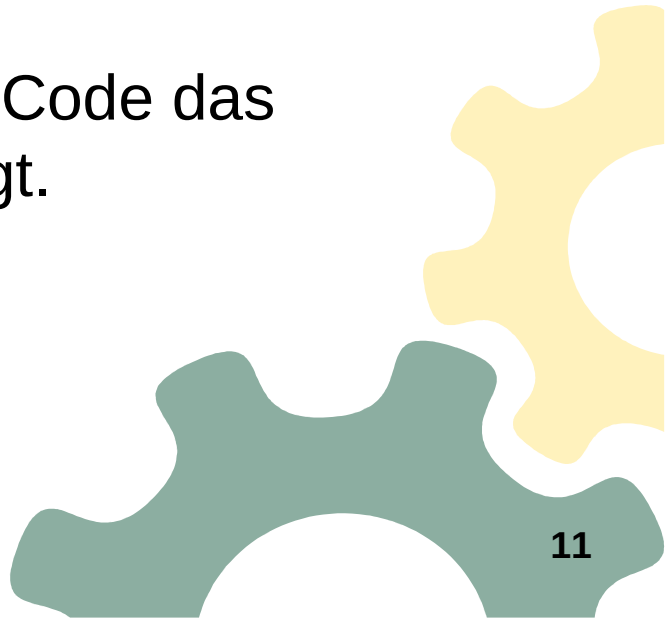
Synchronize and Stabilize

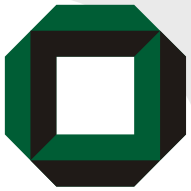
- Ansatz:
  - Organisiere die 200 Programmierer eines Projektes (z.B. Windows 95) in „kleinen Hacker-Teams“→ Freiheit für eigene Ideen/Entwürfe
  - Aber: Synchronisiere regelmäßig (nächtlich)
  - Und Stabilisiere regelmäßig (Meilensteine, 3 Mon.)
- Phasen:
  - Planungsphase
  - Entwicklungsphase
  - Stabilisierungsphase



Synchronize and Stabilize

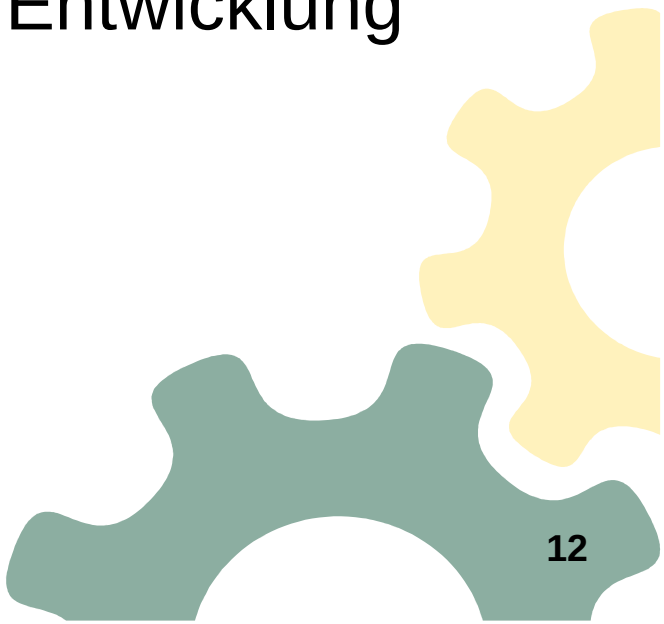
- Entwicklungsphase:
  - Aufteilung in drei Teilprojekte (erstes Drittel: notwendige Funktionen, drittes Drittel: eher unwichtige Funktionen)
  - Nächtliches Neuübersetzen aller Quellen.
  - Automatische Regressionstests
  - Oft Sanktionen für den, der dessen Code das nächtliche Build zum scheitern bringt.

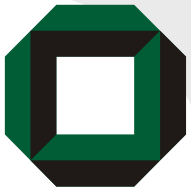




Agile Prozesse

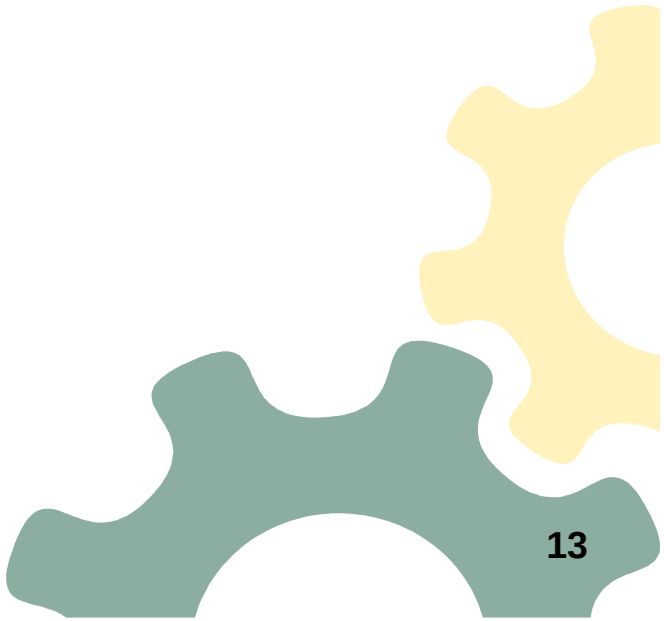
- Minimum an Vorausplanung
- Planung erfolgt inkrementell
- Vermeiden unterstützender Dokumente
- Schnelle Reaktion auf Änderungen
- Einbeziehung des Kunden in die Entwicklung

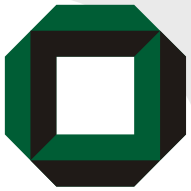




Extreme Programming (XP)

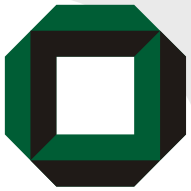
- Praktiken:
  - Paarprogrammierung
  - Testgetriebenes Entwickeln (Test-First)
  - Inkrementelles Design durch Umstrukturierungen (Refactoring)
  - Iterative Planung in kurzen Zyklen.
  - Beteiligung eines echten Kunden.





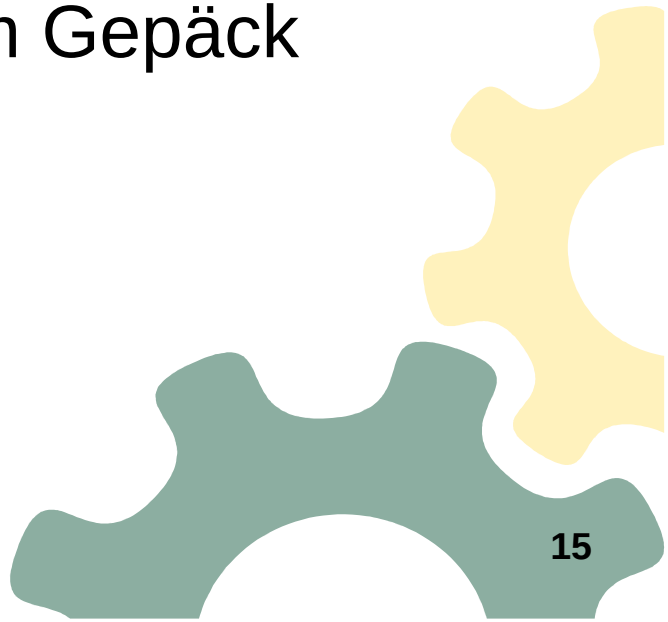
Extreme Programming (XP)

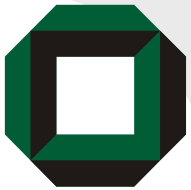
- Kritik:
  - Asymptotische Kostenkurve beruht auf der Erfahrung einzelner
  - Fehlende Produktdokumentation
  - Nicht reproduzierbarer ad-hoc Prozess
  - Paarprogrammierung kostenintensiv, unklar ob dies durch den Zuwachs an Qualität kompensiert wird



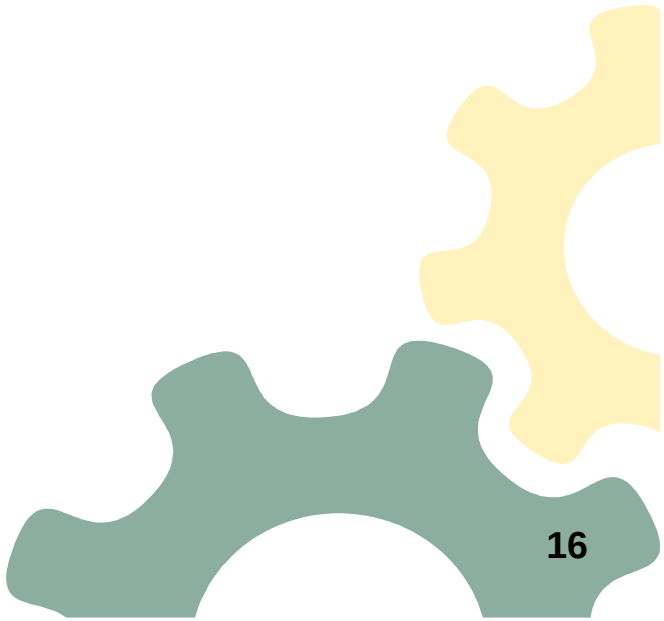
Extreme Programming (XP)

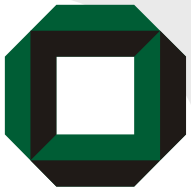
- XP ist Softwareprozess
  - Geeignet für vage und sich schnell ändernde Anforderungen
  - Für kleines Entwicklerteam
  - Ohne zeitraubenden Verwaltungsaufwand
- Softwareentwicklung mit leichtem Gepäck





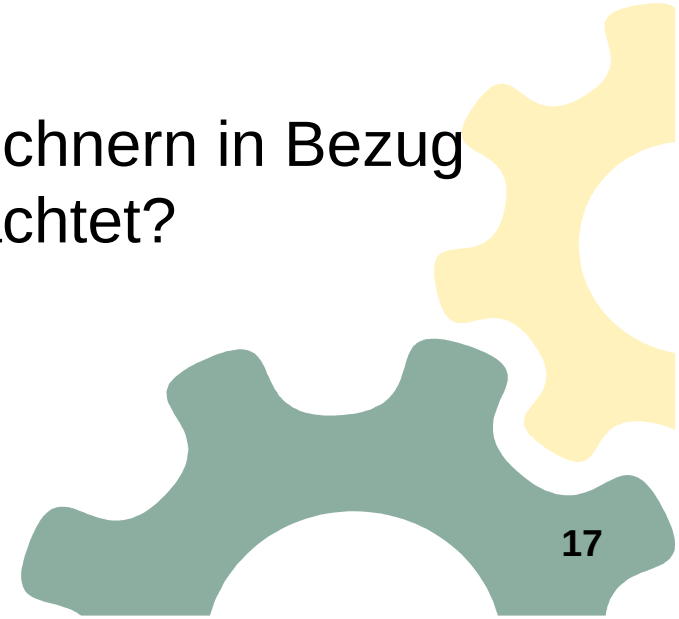
Wiederholung

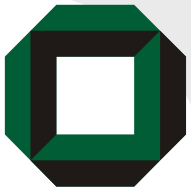




Parallelverarbeitung

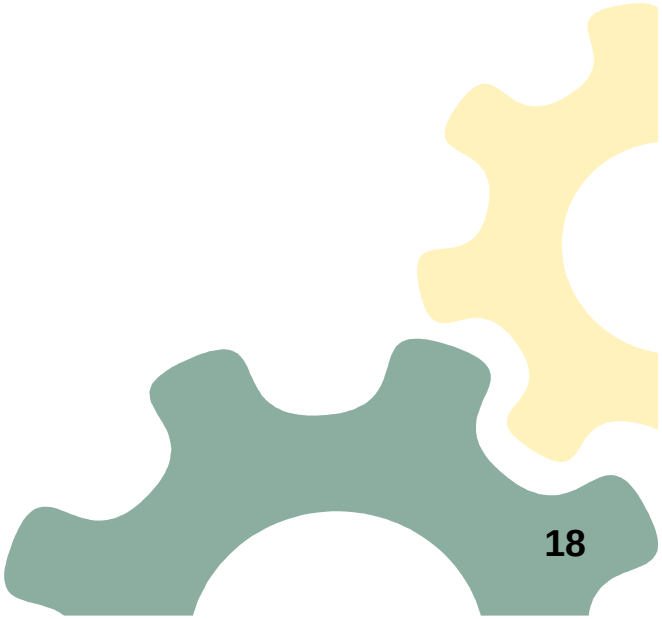
- Warum soll/wird Parallelverarbeitung betrieben?
  - Welche Ziele werden verfolgt?
  - Wie heißen die drei „Walls“
- Wie funktioniert eine Barriere? Wo liegt der Unterschied zum Count-Down-Latch?
- Warum soll ein Wait() in Java immer in einer Schleife aufgerufen werden?
- Welche beiden Modelle von Parallelrechnern in Bezug auf die „Datenkopplung“ wurden betrachtet?

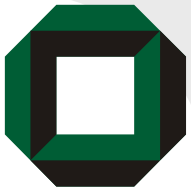




Parallelverarbeitung

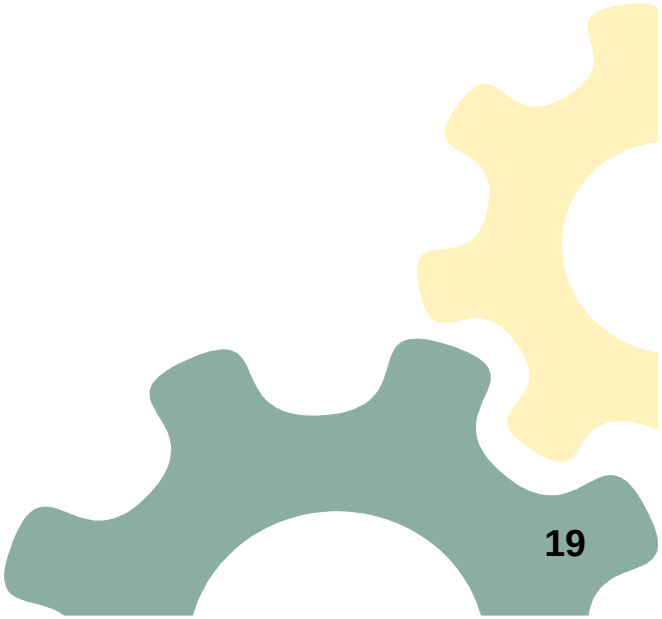
- Wie lautet Amdahls Gesetz? Was ist die Aussage?
- Was sind Speedup und Effizienz?
- Wie kann superlinearer Speedup Zustande kommen?

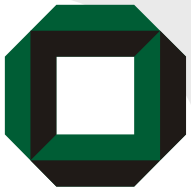




Parallelverarbeitung

- Ziele:
  - Größere Probleme
  - Schneller
- 3 Walls:
  - ILP Wall
  - Power Wall
  - Memory Wall
- Wait: S. Wakeups
- Gemeinsamer Speicher vs. Nachrichtenaustausch





Parallelverarbeitung

- Effizienz: Anteil der Ausführungszeit, die der Prozessor mit nützlicher Arbeit verbringt.

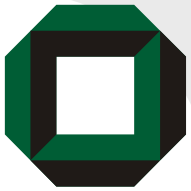
$$E(p) = \frac{T(1)}{p * T(p)} = \frac{S(p)}{p}$$

- Speedup: Der Speedup gibt an, um wie viel schneller der Algorithmus mit p Prozessoren im Vergleich zur besten sequentiellen Ausf

$$S(p) = \frac{T(1)}{T(p)}$$

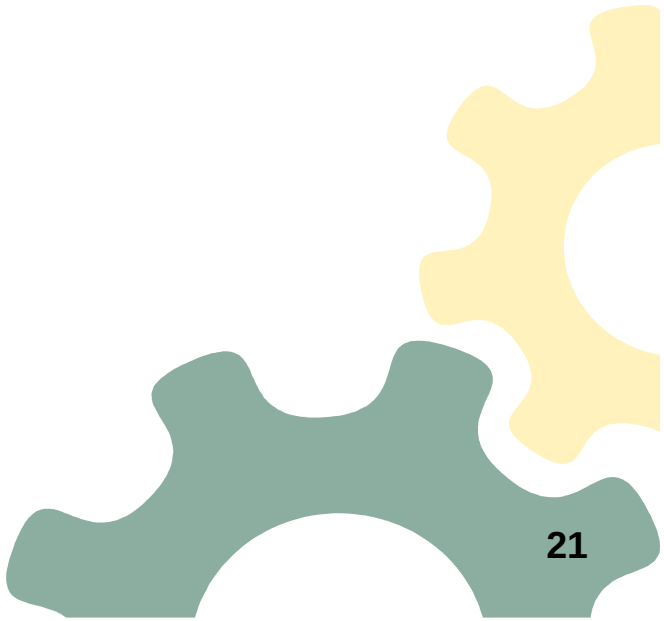
- Amdahls Gesetz:

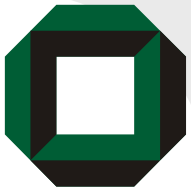
$$S(p) \leq \frac{1}{f}$$



Planungs- und Definitionsphase

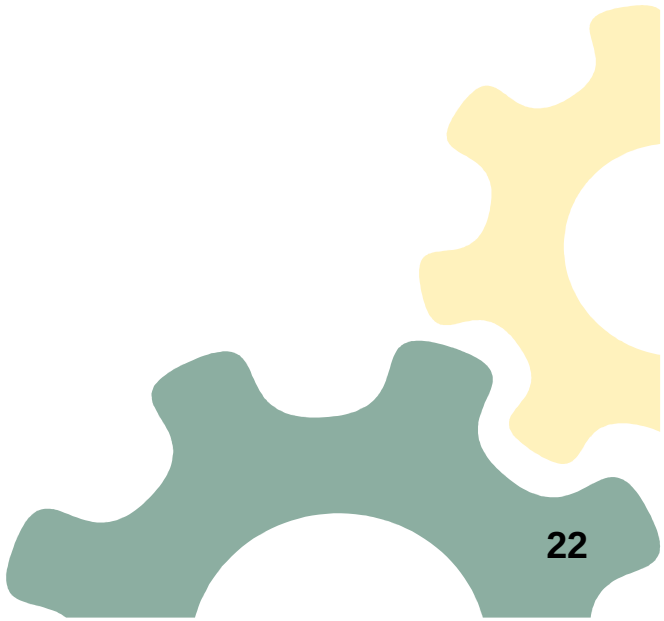
- Anforderungen erheben
- Lastenheft erstellen
- Lastenheft in Pflichtenheft überführen

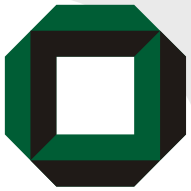




Planungs- und Definitionsphase

- Was ist ein Lastenheft? Was ist ein Pflichtenheft?
- Was sind nicht-funktionale Anforderungen?
- Welche Möglichkeiten gibt es Systeme zu modellieren (in der UML)?

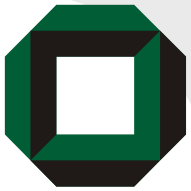




UML Modelle

- Ihr solltest mit folgenden Diagrammen umgehen können:

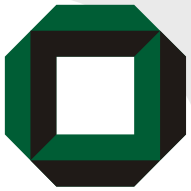
- UML Anwendungsfalldiagramm
- UML Klassendiagramme
- UML Sequenzendiagramm
- UML Zustandsdiagramme
- UML Aktivätsdiagramm



Planungs- und Definitionsphase
Wichtige Begriffe

- Funktionale Anforderungen: Beschreiben Benutzer-Aufgaben, die das System unterstützen muß.
- Nichtfunktionale Anforderungen: Beschreiben Eigenschaften des Systems oder der Domäne.
- Lastenheft: Aus Sicht des Kunden
- Pflichtenheft: „Was“ – aus Sicht des Entwicklers

Hilfsmittel: Szenarien und UML Anwendungsfälle



Anwendungsfalldiagramme

- Ein Anwendungsfall ist eine typische, gewollte Interaktion eines oder mehrerer Akteure mit einem (geschäftlichen oder technischen) System.
- Ein Anwendungsfall wird stets durch einen Akteur initiiert und führt i.d.R. zu einem durch einen Akteur wahrnehmbaren Ergebnis.
- Ein Anwendungsfall beschreibt was ein System leisten muss, nicht *wie* es das leisten muss – er kann insbesondere mehrere verschiedene Ablaufvarianten umfassen

UML Zustandsdiagramme

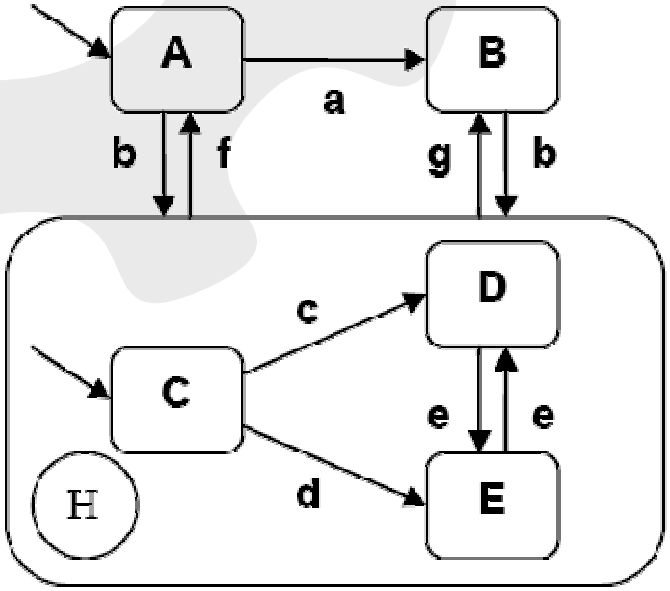
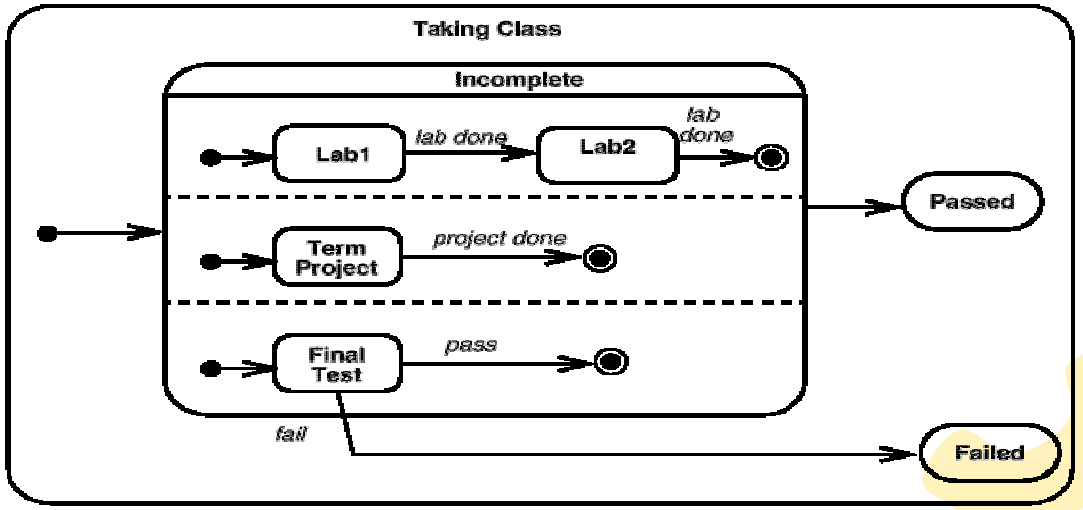


Figure 44. Concurrent substates

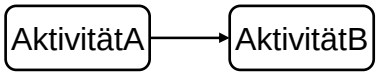


UML Aktivitätsdiagramme

Aktivität

Kontrollfluß

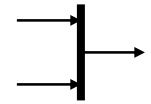
Kontrollfluß, der unter der angegebenen Bedingung gewählt wird.



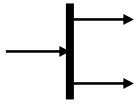
AktivitätB wird im Abschluß an AktivitätA gestartet.

Verzweigunsaktivität

Objektknoten



Synchronisation der Kontrolle (AND) mit Synchronisationsbedingung

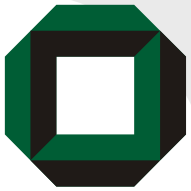


Aufsplitten der Kontrolle (Parallelität)

Startknoten (beliebig viele) möglich

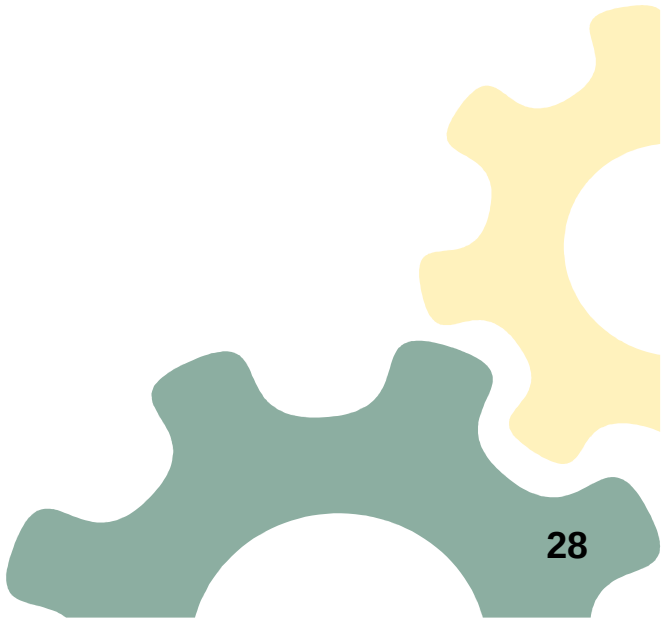
Aktivitätsende (beendet die komplette Aktivität)

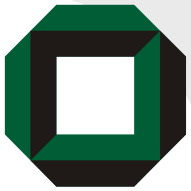
Kontrollflußende (beendet nur den Kontrollfluß)



Entwurfsphase

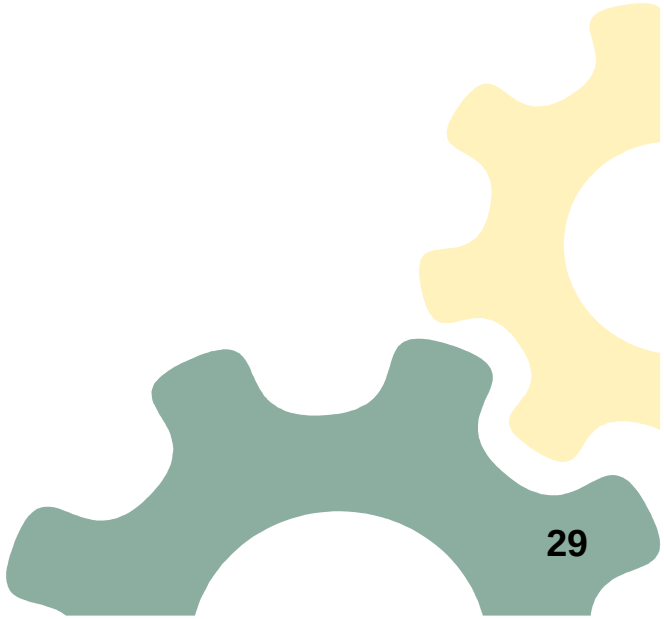
- Grundfrage: Wie Strukturieren wir das Softwaresystem?

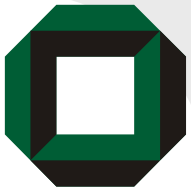




Fragen zur Entwurfsphase

- Was ist eine Software-Architektur?
- Welche beiden Arten des Entwurfs wurden betrachtet?
- Wie lautet die Benutzt-Relation?



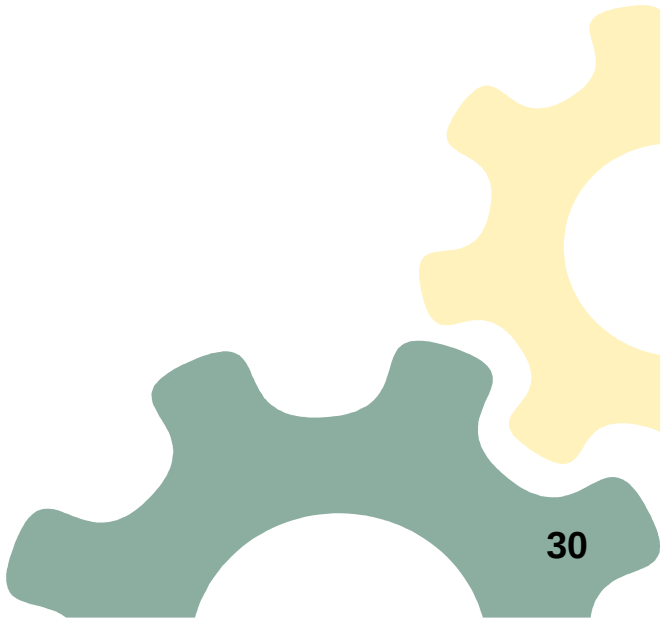


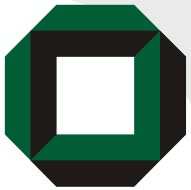
Entwurfsphase

- Software-Architektur:

  Gliederung eines Softwaresystems in Komponenten (Module oder Klassen) und Subsysteme (Pakete, Bibliotheken). Dabei entsteht eine Bestandshierarchie. Spezifikation der Komponenten und Subsysteme, sowie Aufstellung der Benutztrelationzwischen Komponenten und Subsystemen.

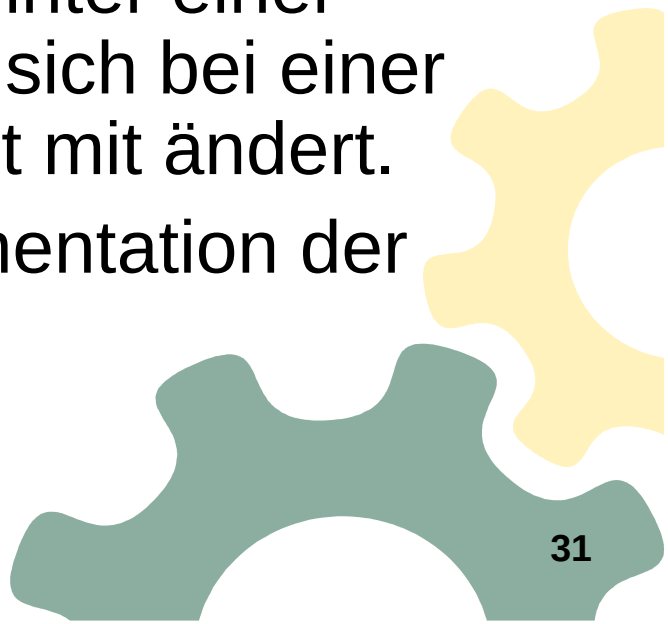
- Entwurfsmethoden:
  - Modularer Entwurf
  - Objektorientierter Entwurf

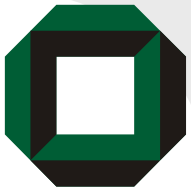




Modularer Entwurf: Begriffe

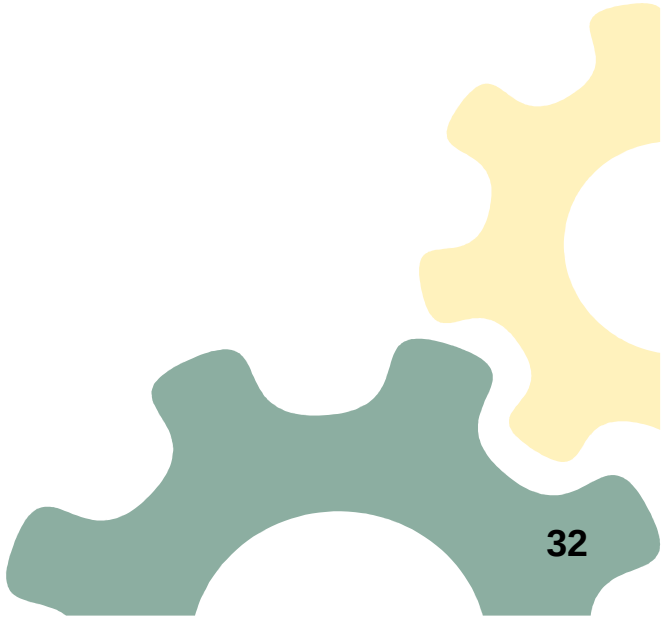
- Modul: Ein Modul ist eine Menge von Programmelementen, die nach dem *Geheimnisprinzip* gemeinsam entworfen und geändert werden.
- Geheimnisprinzip: Jedes Modul verbirgt eine wichtige Entwurfsentscheidung hinter einer wohldefinierten Schnittstelle, die sich bei einer Änderung der Entscheidung nicht mit ändert.
- Modulführer: Entwurf und Dokumentation der Modulstruktur.

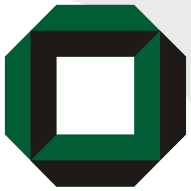




Benutz-Relation

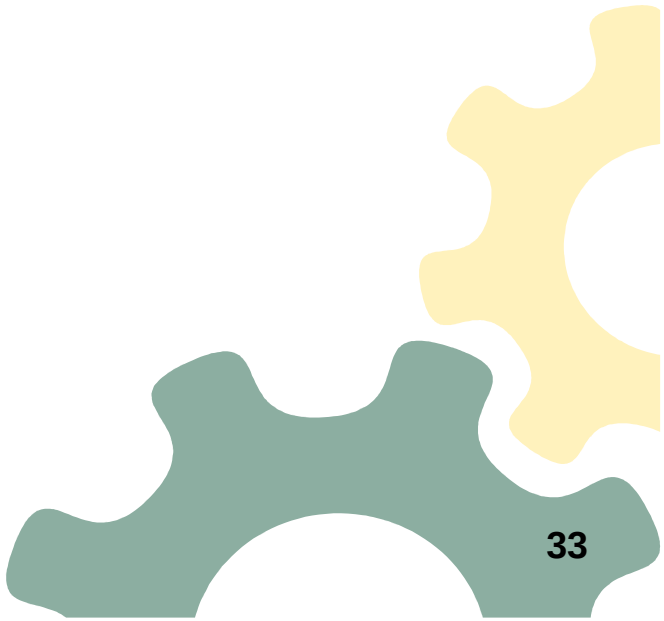
- Eine Programmkomponente A benutzt eine Programmkomponente B genau dann, wenn A für den korrekten Ablauf die ***Verfügbarkeit* einer korrekten Implementierung** von B erfordert.

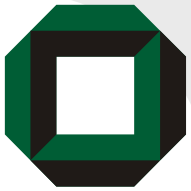




Entwurfsmuster

- Sind enorm wichtig! Es wird garantiert eine Aufgabe zu Entwurfsmustern kommen.
- Nur weil zwei Entwurfsmuster gleich aussehen, sind die noch lange nicht dasselbe => Ziele und Randbedingungen beachten

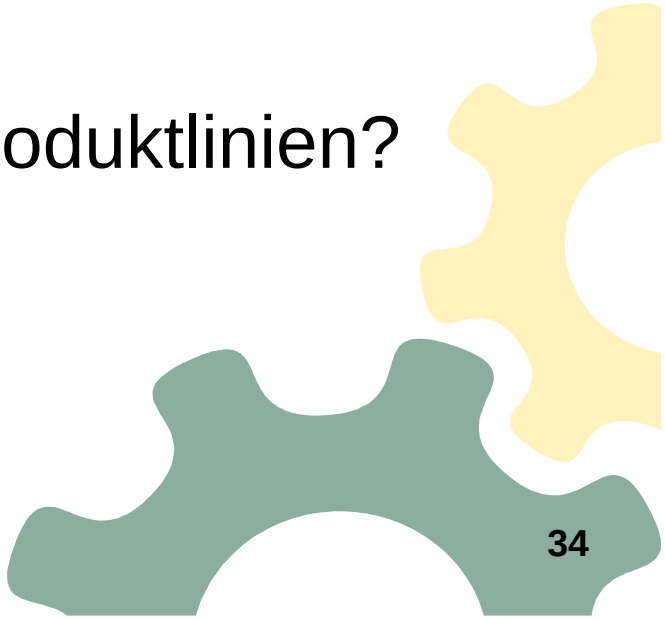


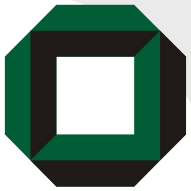


Software-Produktlinien

- Eine **Programmfamilie** oder **Software-Produktline** ist eine Menge von Programmen, die erhebliche Anteile von Anforderungen, Entwurfsbestandteilen oder Softwarekomponenten gemeinsam haben.

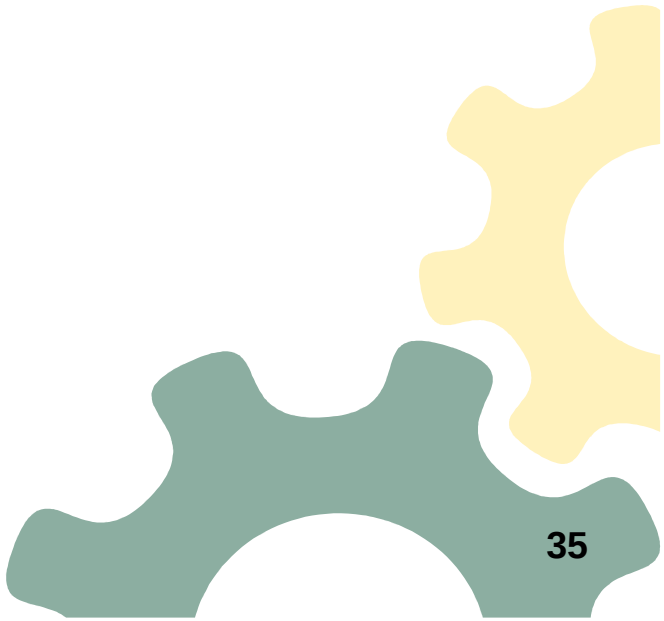
- Frage: Wozu dienen Software-Produktlinien?

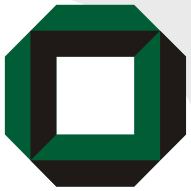




Architekturstile

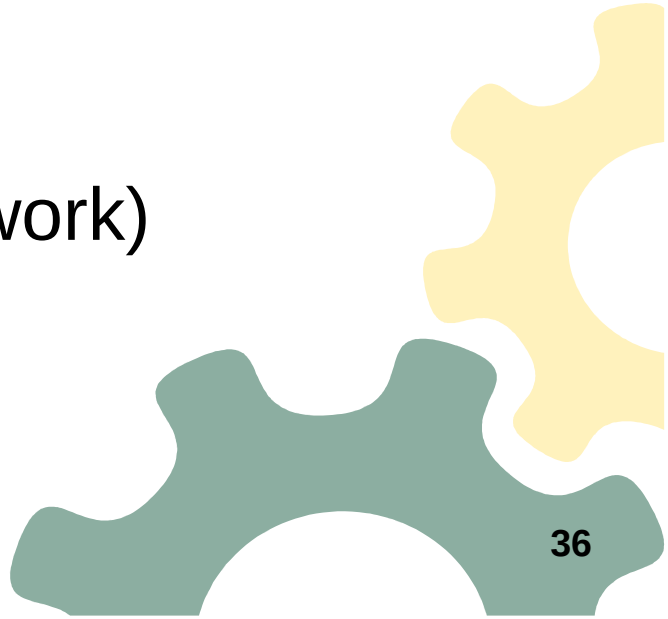
- Welche Architekturstile wurden behandelt?
- Wie sehen die Architekturstile aus?

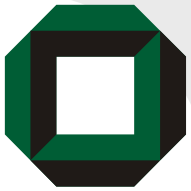




Architekturstile

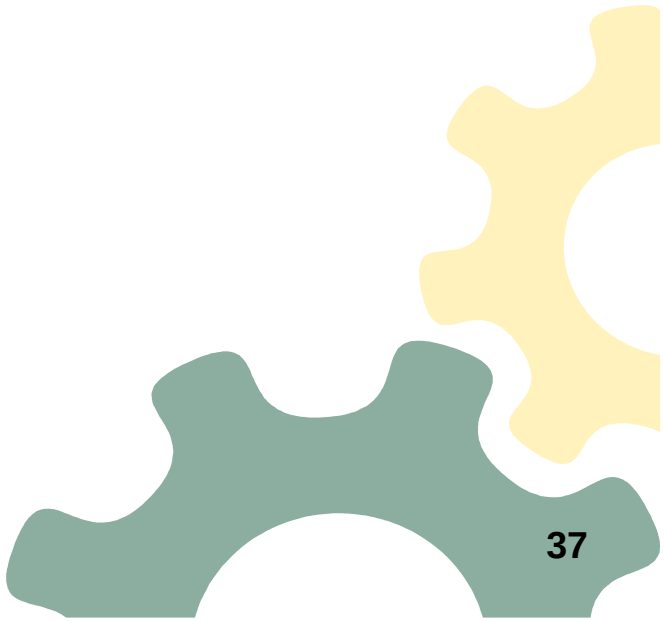
- Schichtenarchitektur
- Klient/Dienstgeber (engl. client/server)
- Partnernetze (engl. peer-to-peer)
- Datenablage (engl. repository)
- Model/View/Controller
- Fließband (engl. pipeline)
- Rahmenarchitektur (engl. framework)

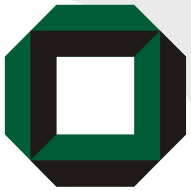




Implementierungsphase

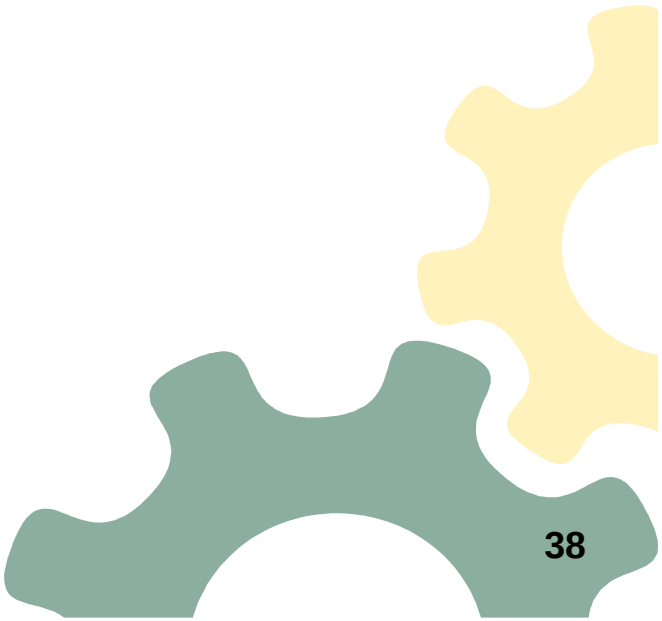
- Programmierung, Dokumentation und Testen
- Abbildung von UML auf Code
- Programmierrichtlinien
- Selbstkontrolliertes Programmieren

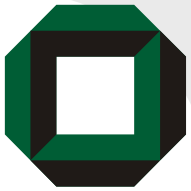




Implementierungsphase

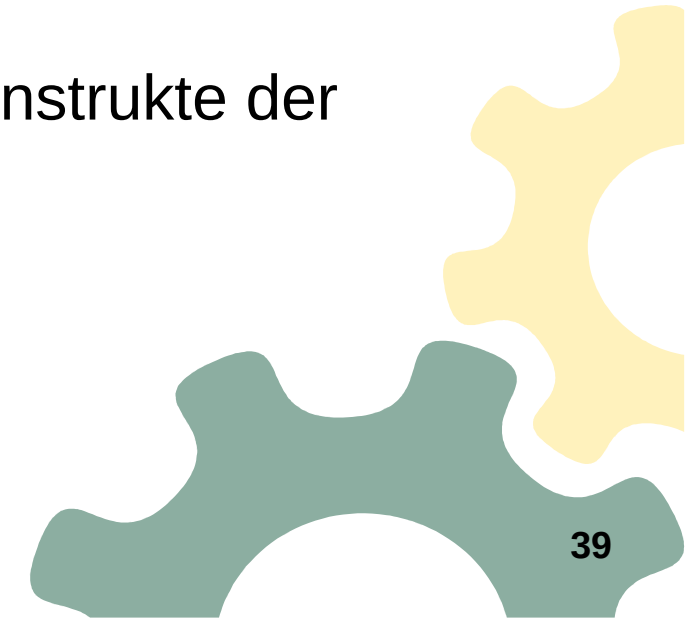
- Welche Aktivitäten werden in der Implementierungsphase durchgeführt?
- Warum sind Programmierrichtlinien notwendig?

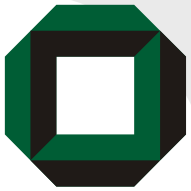




Aktivitäten der Implementierungsphase

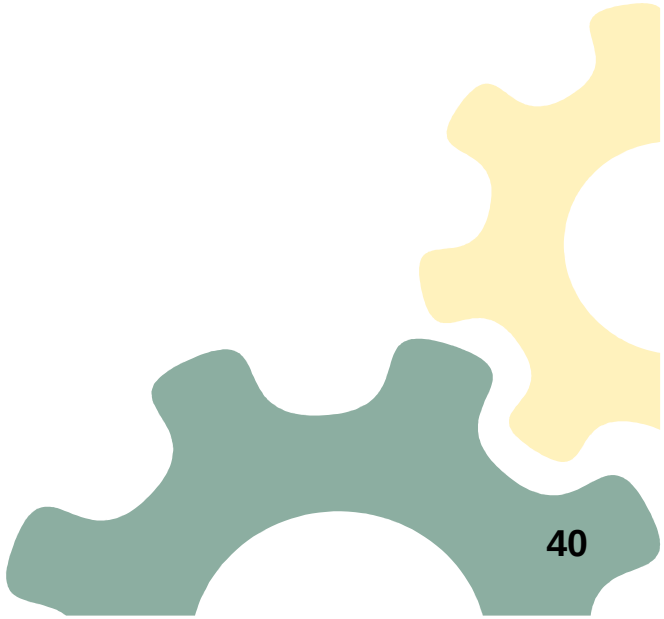
- Aktivitäten
  - Konzeption von Datenstrukturen und Algorithmen
  - Strukturierung des Programms durch geeignete Verfeinerungsebenen
  - Dokumentation der Problemlösung und der Implementierungsentscheidungen
  - Umsetzung der Konzepte in die Konstrukte der verwendeten Programmiersprache

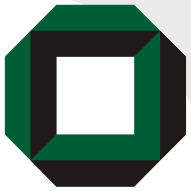




Programmierrichtlinien

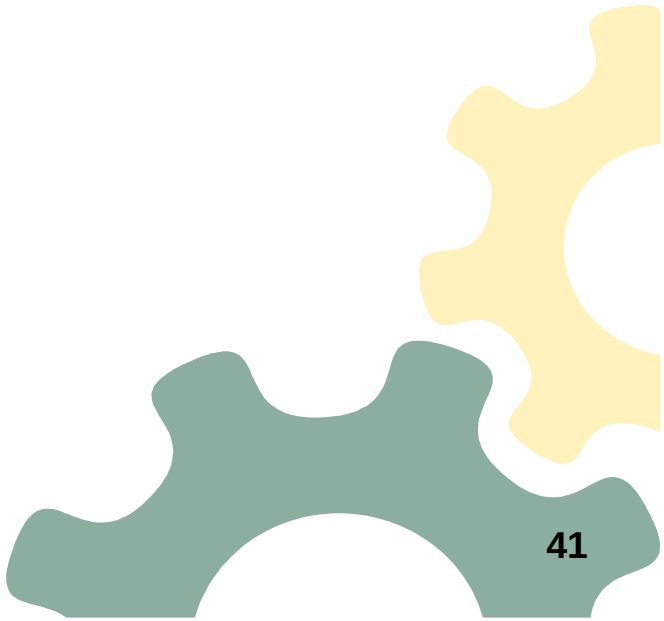
- Konsistenter Stil erleichtert die Lesbarkeit,
- Beschleunigt Einarbeitung bei Personalwechsel und Wiedereinarbeitung,
- Zeitersparnis bei Fehlerfindung, Erweiterung und Pflege des Programms.

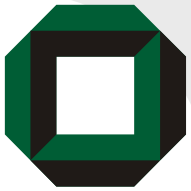




Fragen

- Was ist der Unterschied zwischen
  - Testenden Verfahren
  - Verifizierenden Verfahren
  - Analysierenden Verfahren?
- Was versteht man unter:
  - Defekt
  - Fehler (Failure)
  - Versagen
- Was ist
  - Stummel
  - Attrape
  - Nachahmung

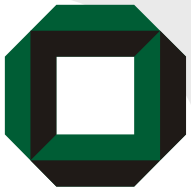




Verfahren

- Testende Verfahren: Fehler erkennen
- Verifizierende Verfahren: Korrektheit beweisen
- Analysierende Verfahren: Eigenschaften einer Systemkomponente bestimmen.

- Defekt: Ein Entwurfs- oder Programmierfehler, welcher zu einem unnormalen Verhalten führen kann, aber nicht muß.
- Fehler: Sichtbare Folge eines Defekts.
- Versagen: Abweichung des aktuellen Verhaltens des Systems von der Spezifikation.



Testhelfern

- Ein **Stummel**(engl. stub) ist ein nur rudimentär implementierter Teil der Software und dient als Platzhalter für noch nicht umgesetzte Funktionalität.
- Eine **Attrappe**(engl. dummy) ersetzt eine echte Implementierung zu Testzwecken.
- Eine **Nachahmung**(engl. mock) ist eine Attrappe mit zusätzlicher Funktionalität, wie bspw. das Einstellen der Reaktion der Nachahmung auf bestimmte Eingaben oder das Überprüfen des Verhaltens des „Klienten“.

Teststrategien