Softwaretechnik C - Softwaremanagement

LE 03: Vorgehens- und Prozessmodelle

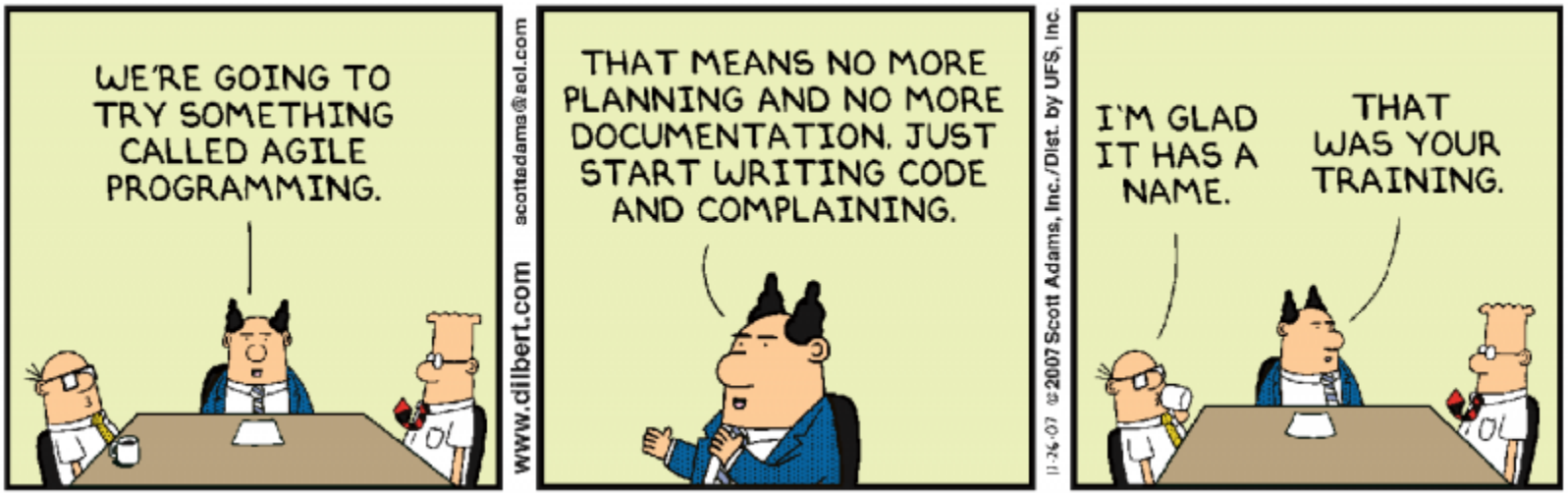
Fachhochschule Dortmund
University of Applied Sciences

Organisatorisches

Agenda

- Klassische Vorgehensmodelle
  - Wasserfallmodell
  - Nebenläufiges Modell
  - Spiralmodell
  - Prototyping
- Monumentale Vorgehensmodelle
  - V-Modell XT
  - Rational Unified Process (RUP)
- Agile Prozesse
  - Scrum
  - Kanban (IT-Kanban)
  - Extreme Programming (XP)
  - Feature Driven Development
  - Adaptive Software Development
  - Crystal
  - Lean Software Development
  - SAFe

- Klassische Vorgehensmodelle
  - Wasserfallmodell
  - Nebenläufiges Modell
  - Spiralmodell
  - Prototyping
- Monumentale Vorgehensmodelle
  - V-Modell XT
  - Rational Unified Process (RUP)
- **Agile Prozesse**
  - Scrum
  - Kanban (IT-Kanban)
  - Extreme Programming (XP)
  - Feature Driven Development
  - Adaptive Software Development
  - Crystal
  - Lean Software Development
  - SAFe



Quelle: 40 Key Computer Science Concepts Explained In Layman's Terms | Programming humor, Work humor, Tech humor (pinterest.de)

Agile Modelle
Einführung und Überblick

- Agile Prozessmodelle
  - unternehmen den Versuch, in der Softwareentwicklung mit **minimalem bürokratischen Aufwand** und **wenigen Regeln** auszukommen
  - als Gegenbewegung zu immer mächtiger werdenden Vorgehens- und Prozessmodellen der Softwareentwicklung

- Der Begriff „agil“ wird mittlerweile nicht nur für Prozess- und Qualitätsmodelle, sondern auch in anderen Bereichen verwendet

Agile Modelle
Einführung und Überblick

- Idee: Agile Prozessmodelle setzen das Prinzip der Agilität (Beweglichkeit, Lebendigkeit) im Rahmen der Software-Entwicklung ein
- Annahmen:
  1. Softwareentwicklung ist ein Prozess, der ein **hohes Maß an Kreativität** erfordert
  2. Anwendung eines **stark strukturierten, deterministischen** Fertigungsprozesses (in Analogie zur Autoproduktion oder zum Haus- und Brückenbau) ist daher **ungeeignet**
- Ausgangspunkt für die agilen Prozessmodelle waren die Ideen von Kent Beck
  - Veröffentlicht im Jahr 2000 unter dem Titel „eXtreme Programming Explained“ (XP)

Agile Modelle
Einführung und Überblick

- Bekannteste agile Modelle
  - Scrum
  - Kanban (IT-Kanban)
  - Extreme Programming (XP)
  - Feature Driven Development
  - Adaptive Software Development
  - Crystal
  - Lean Software Development

Agile Modelle
Ursprung des Agilen Manifest

- Manifesto for Agile Software Development (agilemanifesto.org)
- Wurde im Februar 2001 formuliert
- 17 Erstunterzeichner
  - Begründer von Extreme Programming (Kent Beck, Ward Cunningham, Ron Jeffries)
  - Begründer von Scrum (Ken Schwaber, Jeff Sutherland)
  - Weitere Vertreter von Feature Driven Development, Adaptive Software Development, Crystal, Dynamic Systems Development Method
- Enthält **4 Werte und 12 Prinzipien**

Agile Modelle
Ursprung des Agilen Manifest

Fachhochschule Dortmund
University of Applied Sciences

- **Gegensätzlich** zu klassischen Vorgehensmodellen
  - Anforderungen zu Projektbeginn nicht vollumfänglich bekannt oder bestimmt
  - Prioritäten oder Anforderungen können sich im Projektverlauf ändern
  - Development-Team verfügt nicht über alle Informationen, um eine verlässliche Aufwandsschätzung abzugeben
  - Damit nicht zu Beginn über die gesamte Dauer planbar
- Autoren suchten nach **Alternativen zu dokumentationsintensiven, schwerfälligen Entwicklungsprozessen**
- Ken Schwaber: Projektplan ist bei komplexer, kreativer Arbeit **kontraproduktiv**

Agile Modelle
Agiles Manifest

4 Werte des Agilen Manifest:

1. **Einzelpersonen und Interaktionen** sind wichtiger als Prozesse und Werkzeuge.
   - Wohldefinierte Entwicklungsprozesse und Entwicklungswerkzeuge sind grundsätzlich wichtig
   - Wesentlicher sind jedoch die Qualifikation der Mitarbeitenden und eine effiziente Kommunikation zwischen ihnen.
2. **Funktionierende Softwaresysteme** sind wichtiger als eine umfassende Dokumentation.
   - Gut geschriebene und ausführliche Dokumentation kann hilfreich sein
   - Das eigentliche Ziel der Entwicklung ist jedoch die fertige Software

Agile Modelle
Agiles Manifest

Fachhochschule Dortmund
University of Applied Sciences

4 Werte des Agilen Manifest:

3. **Die Zusammenarbeit mit dem Kunden** ist wichtiger als die Vertragsverhandlung
   - Statt sich an ursprünglich formulierten und ggf. veralteten Leistungsbeschreibungen in Verträgen festzuhalten, steht vielmehr die fortwährende konstruktive und vertrauensvolle Abstimmung mit dem Kunden im Mittelpunkt.
4. **Das Reagieren auf Veränderungen** ist wichtiger als das Befolgen eines Plans
   - Im Verlauf eines Entwicklungsprojektes ändern sich viele Anforderungen und Randbedingungen ebenso wie das Verständnis des Problemfeldes.
   - Das Team muss zu jedem Zeitpunkt schnell darauf reagieren können.

Agile Modelle
Agiles Manifest

- **12 Prinzipien des Agilen Manifest:**

  - **Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.**
  - Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
  - **Deliver working software frequently**, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
  - Business people and developers must work together daily throughout the project.
  - Build projects around motivated individuals. Give them the environment and support their need, and trust them to get the job done.
  - The most efficient and effective method of conveying information to and within a development team is **face-to-face conversation**.

Agile Modelle
Agiles Manifest

- **12 Prinzipien des Agilen Manifest:**

  - **Working software is the primary measure of progress**.
  - Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
  - Continuous attention to technical excellence and good design enhances agility.
  - **Simplicity-**--the art of maximizing the amount of work not done--is essential.
  - The best architectures, requirements, and designs emerge from **self-organizing teams**.
  - At **regular intervals**, the team **reflects on how to become more effective**, then tunes and adjusts its behavior accordingly.

Agile Modelle
Charakteristika

- Es soll gerade **so viel Prozess wie nötig** vorhanden sein, sodass sich der entsprechende Aufwand lohnt
- Es werden **möglichst wenig Dokumente** gefordert
  - Im Extremfall ist der Code das Dokument
- **Iterative Entwicklung** mit **häufig ausgelieferten lauffähigen Versionen** des zu entwickelnden Softwaresystems
- **Lauffähige Versionen** des Softwaresystems setzen jeweils eine **Teilmenge** der geforderten Anforderungen um

Agile Modelle
Charakteristika

- Die lauffähigen Versionen:
  - besitzen zu Beginn oftmals nur „abgespeckte“ oder **Teilfunktionalität**
  - sollen beim Kunden bzw. Auftraggeber **Vertrauen** in das endgültige Softwaresystem schaffen
  - sollen voll **integriert** werden
  - sollen **schnelles Feedback** seitens des Kunden ermöglichen
  - werden so **sorgfältig getestet** wie das endgültige Softwaresystem
- Stabile Pläne gelten jeweils nur für kurze Zeiträume
- Pläne werden jeweils **nur für eine Iteration** im Voraus gemacht

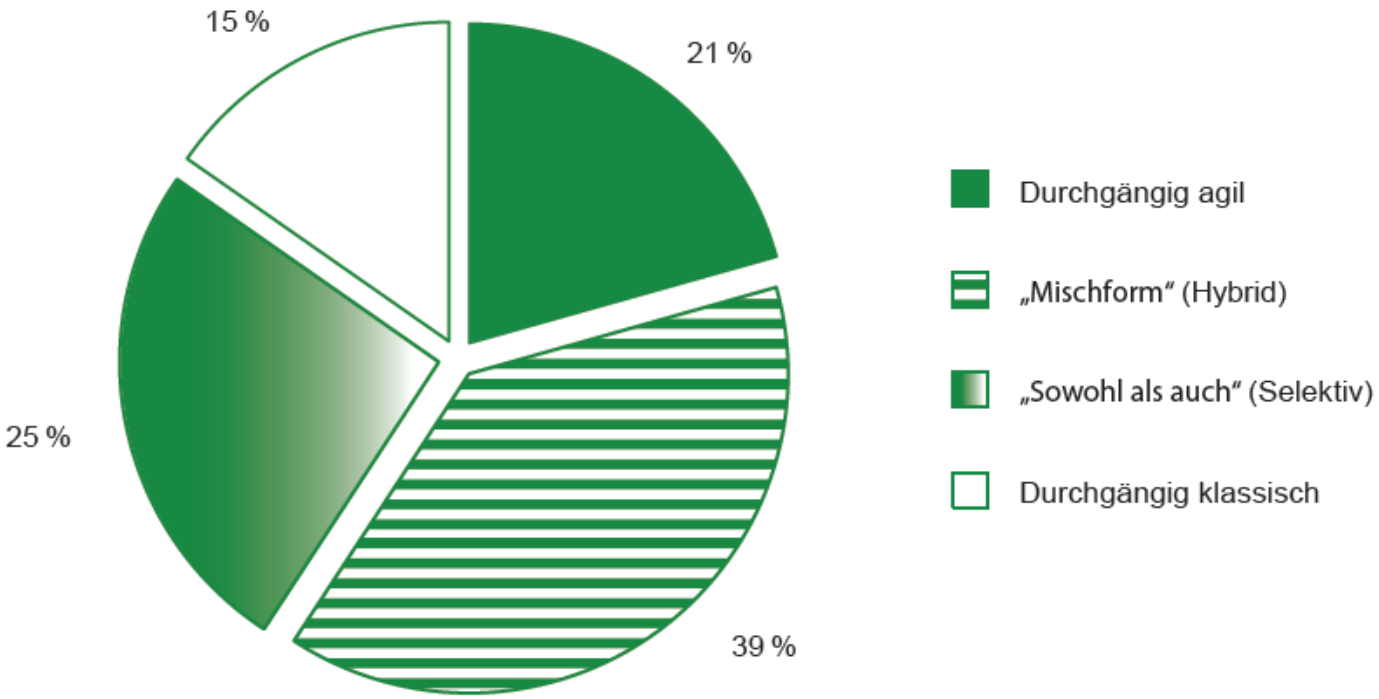
Agile Modelle
Bedeutung agiler Vorgehensmodelle

- Studie aus 2015:
  - **Status Quo Agile**
    Studie zu Verbreitung und Nutzen agiler Methoden
  - Prof. Dr. Ayelt Komus, Moritz Kuberg, Prof. Dr. Yvonne Schoper
  - GPM Deutsche Gesellschaft für Projektmanagement e.V., Hochschule Koblenz
  - Koblenz, Oktober 2015
- Ziele und Durchführung:
  - Untersuchung von Zufriedenheit, Erfolg und Anwendungsformen agiler Methoden in der Praxis
  - Online-Fragebogen
  - 612 Teilnehmende aus über 30 Ländern (61% aus Deutschland)
  - Unternehmensgrößen etwa gleichverteilt von < 25 Personen bis über 50.000 Personen
  - > 90% Einsatz für Softwareentwicklung, immerhin 27% Einsatz ohne IT-Bezug

Agile Modelle
Bedeutung agiler Vorgehensmodelle

Fachhochschule Dortmund
University of Applied Sciences

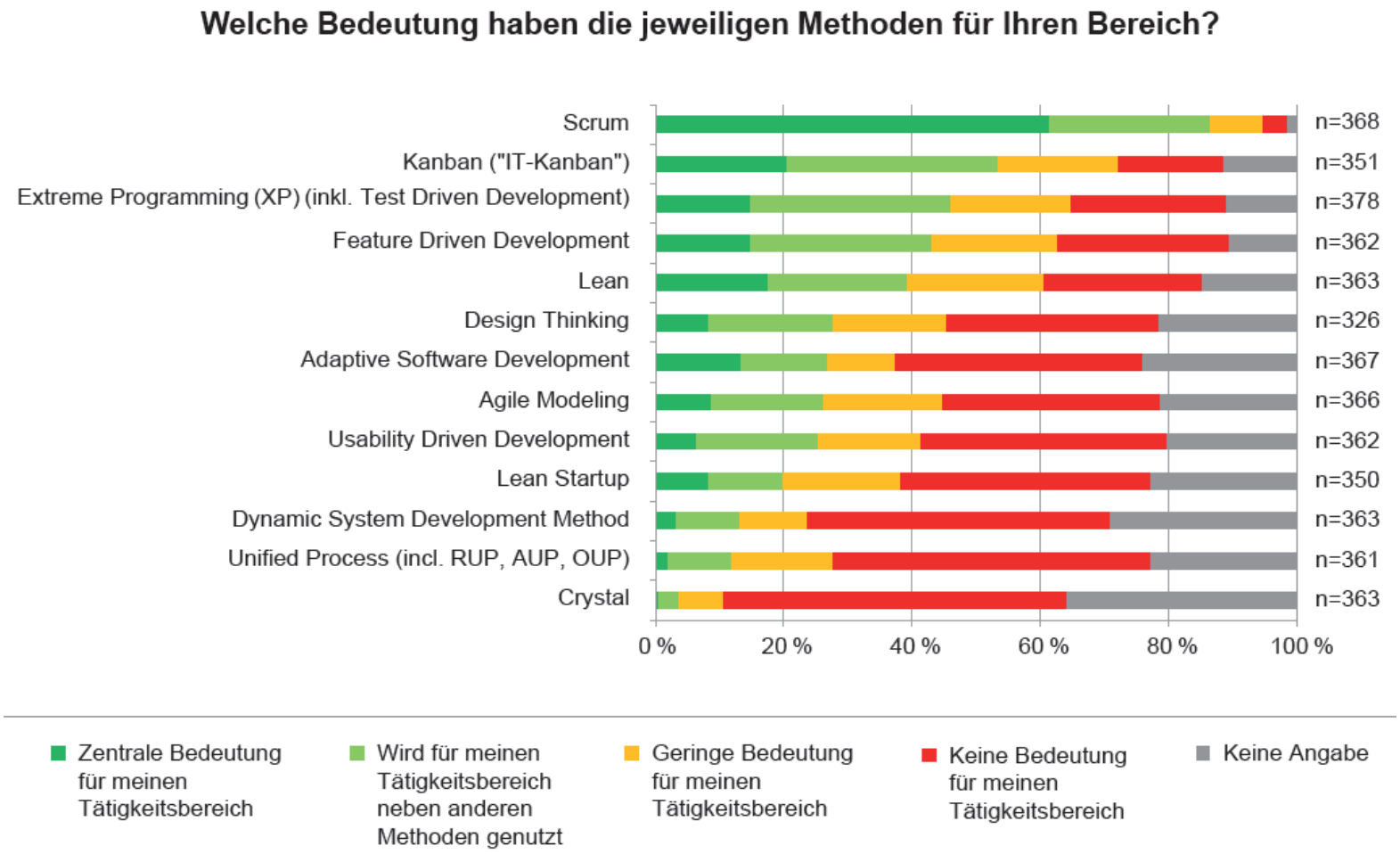
Projekte / Entwicklungsprozesse werden im Tätigkeitsbereich geplant / durchgeführt …



n=601 (Nur eine Antwort möglich, Pflichtangabe)

Agile Modelle
Bedeutung agiler Vorgehensmodelle

Fachhochschule Dortmund
University of Applied Sciences



- Klassische Vorgehensmodelle
  - Wasserfallmodell
  - Nebenläufiges Modell
  - Spiralmodell
  - Prototyping
- Monumentale Vorgehensmodelle
  - V-Modell XT
  - Rational Unified Process (RUP)
- Agile Prozesse
  - Scrum
  - **Kanban (IT-Kanban)**
  - Extreme Programming (XP)
  - Feature Driven Development
  - Adaptive Software Development
  - Crystal
  - Lean Software Development

Quelle: https://cdn.business2community.com/wp-content/uploads/2014/06/Simple-kanban-board.jpg

Kanban
Überblick

Fachhochschule Dortmund
University of Applied Sciences

- 1947 durch Taiichi Ohno entwickeltes Prozessmodell in der Fertigungsindustrie (Toyota Motor Corporation)
  - „Signalkarte“ (kan „Signal“, ban „Karte“)
  - Ziel: Gleichmäßiger Fluss in der Fertigung zur Reduktion von Lagerbeständen
  - Fokus: optimaler Fluss jedes einzelnen Produkts durch die Fertigung
- Kanban in der IT
  - 2010 durch David J. Anderson beschrieben
  - Basiert auf Grundsätzen von Lean Production und Lean Development
    - Vermeidung von Überflüssigem (Muda)
  - Soll Wertgenerierung entlang der Wertschöpfungskette optimieren

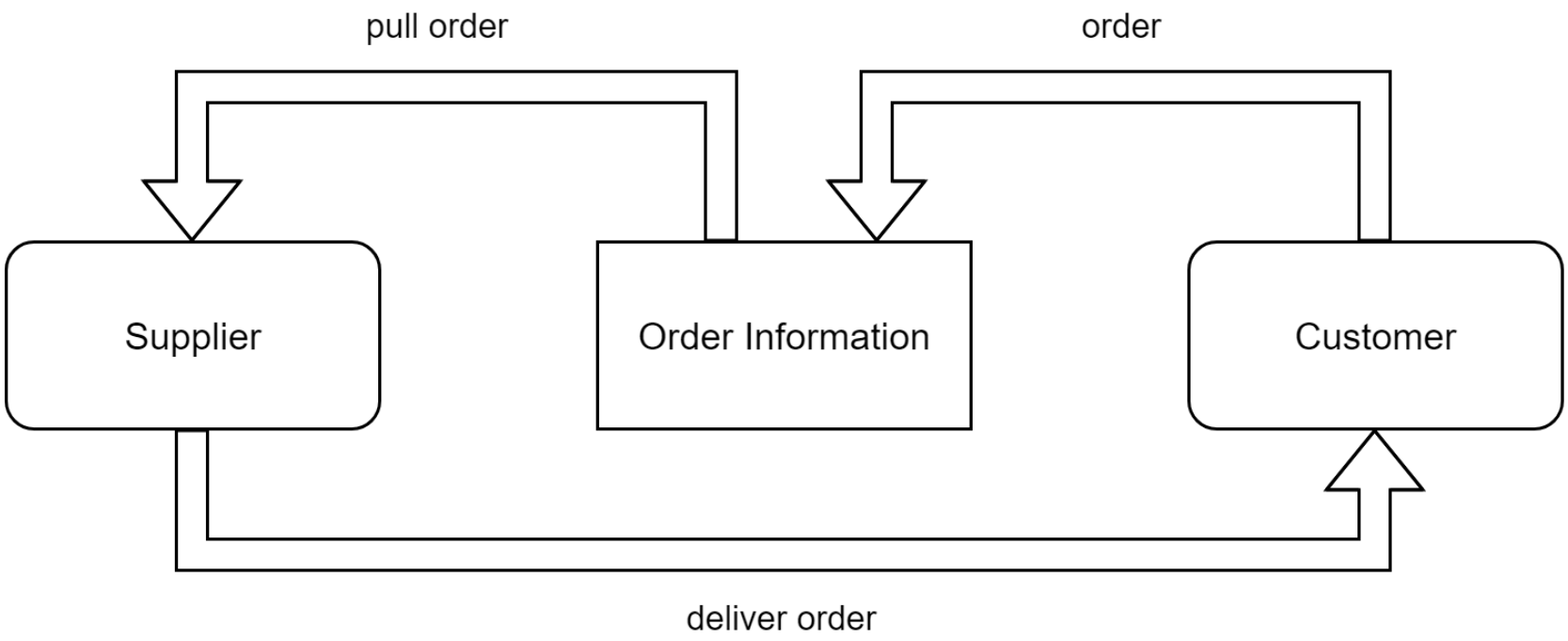
Kanban
Charakteristika

Fachhochschule Dortmund
University of Applied Sciences

- Generisches Modell
  - Stationen im Entwicklungsprozess werden individuell festgelegt und bei Bedarf angepasst
- Softwareentwicklungsphilosophie
  - Im Fokus steht nicht nur Entwicklungsprozess, sondern auch Kollaboration und Prinzipien
- Selbstorganisierte, häufig interdisziplinär besetzte Teams
- Pull System
  - Systematischer Weg für nachhaltige Arbeitsgeschwindigkeit
- Kaizen
  - Kontinuierlicher Verbesserungsprozess
- 4 Grundprinzipien
- 6 Kernpraktiken
- 4 Flight Levels

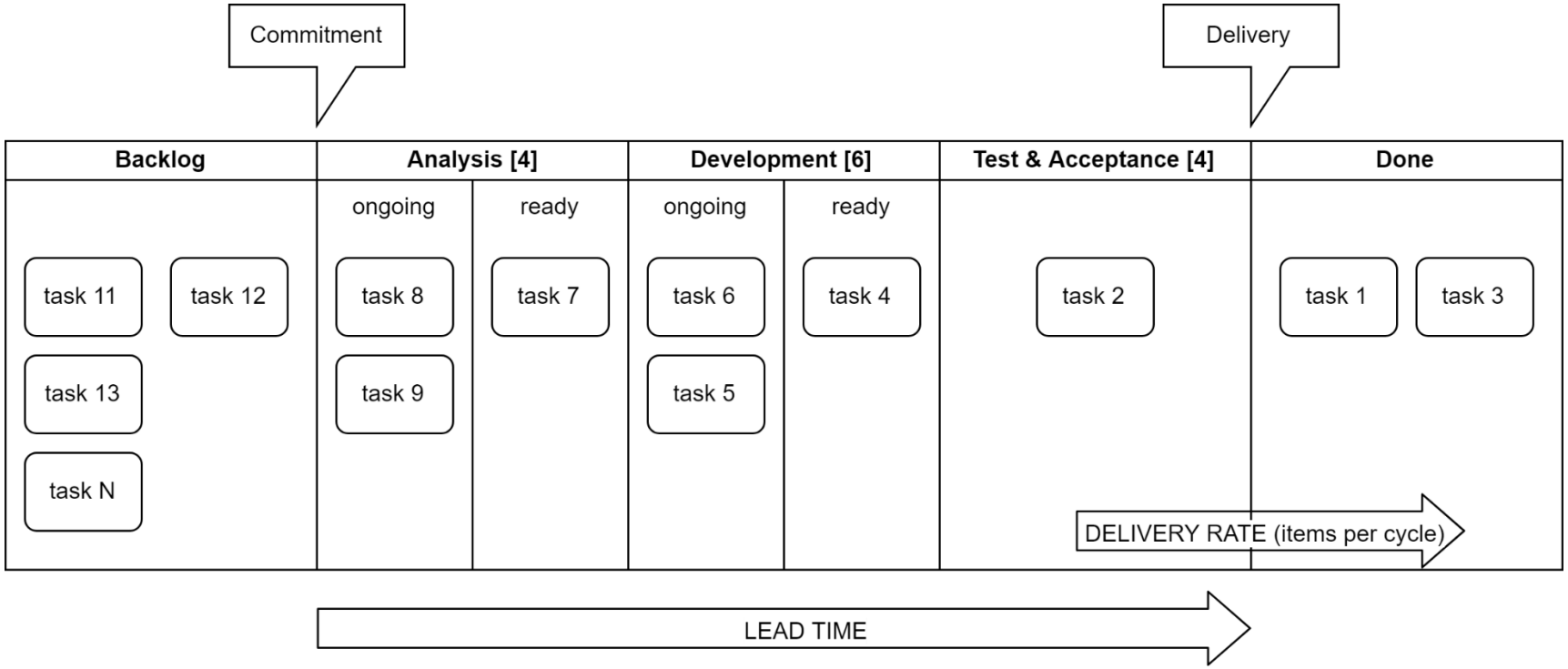
Kanban
Darstellung Grundidee

Fachhochschule Dortmund
University of Applied Sciences



Kanban
Kanban Board

Fachhochschule Dortmund
University of Applied Sciences

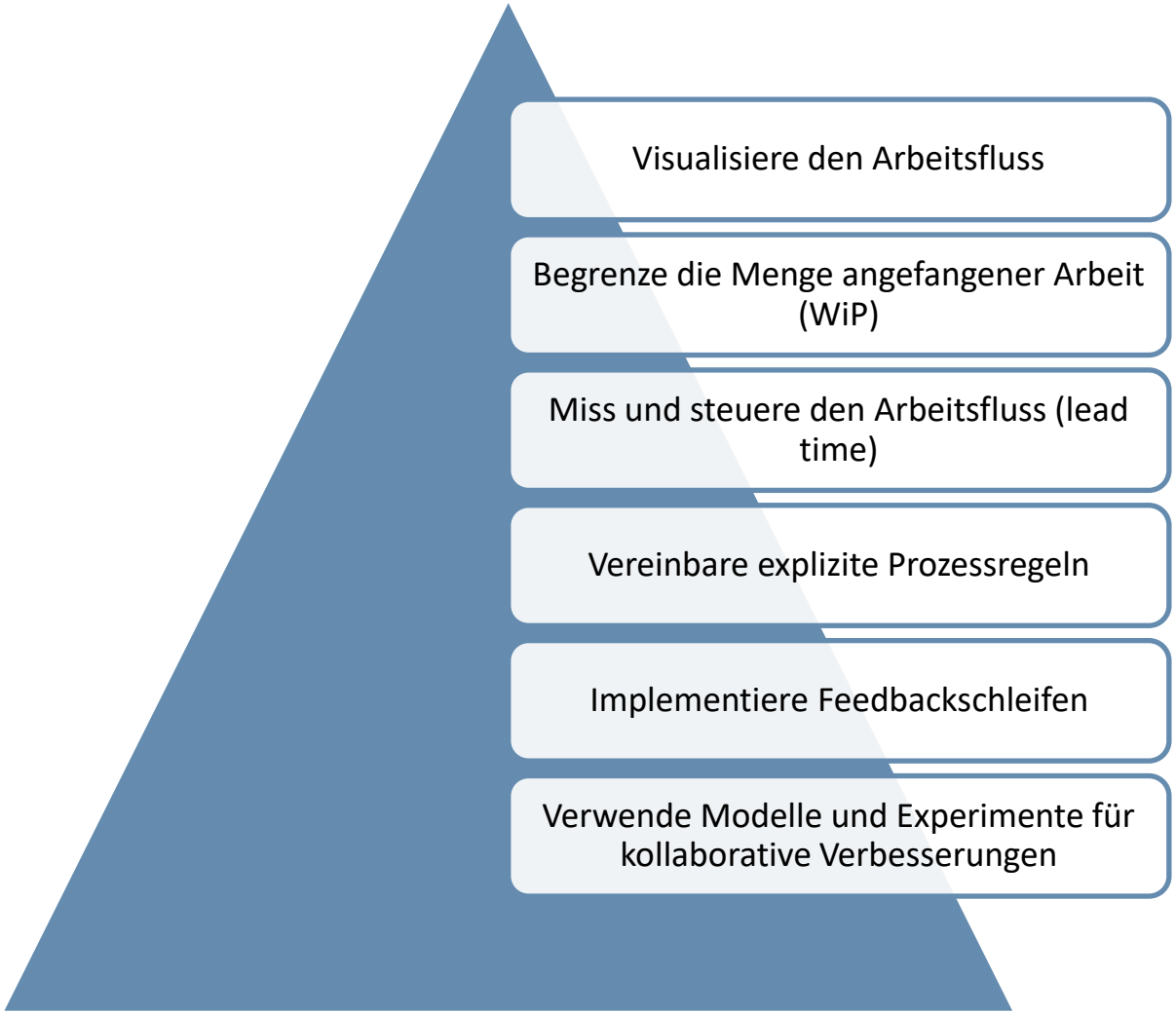


Kanban
Grundprinzipien

Fachhochschule Dortmund
University of Applied Sciences

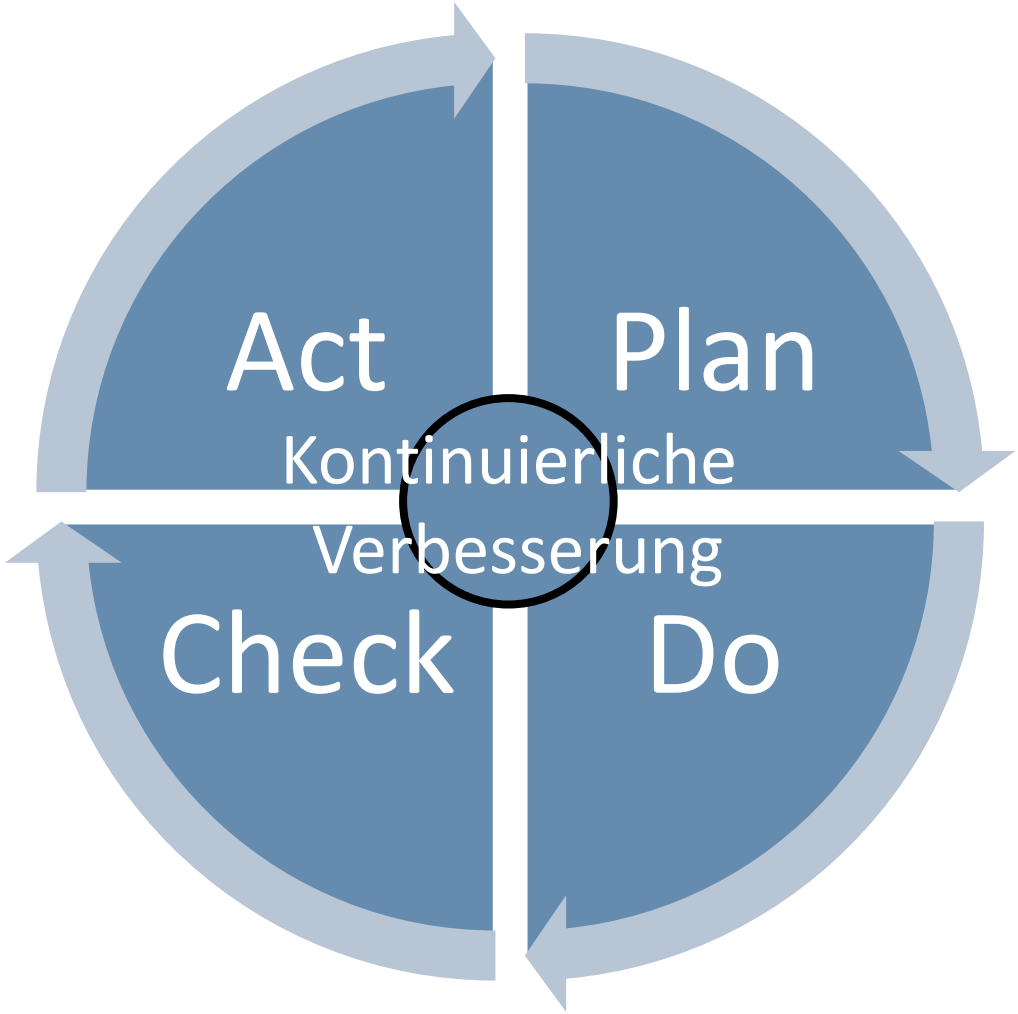
Kanban
Kernpraktiken

Fachhochschule Dortmund
University of Applied Sciences



Kanban
Kaizen

Fachhochschule Dortmund
University of Applied Sciences



Kanban
Kaizen & Feedbackschleifen

Fachhochschule Dortmund
University of Applied Sciences

Kanban
Feedbackschleifen

- Strategy Review:
  - Quartalsweise
  - Um zu diskutieren, wie sich die Außenwelt im Verhältnis zum Prozess oder zu den Prozessen der Organisation verändert hat. Ggf. kann das Produkt- bzw. Dienstleistungsangebot überdacht werden.
- **Operations Review**:
  - Monatlich
  - Um zu besprechen, ob ein Gleichgewicht zwischen den Prozessen herrscht (v.a. Verteilung begrenzter Ressourcen)
- **Risk Review**:
  - Monatlich
  - Es wird thematisiert, was das Team behindert oder die Produktivität des Prozesses vermindert.

Kanban
Feedbackschleifen

- **Service Delivery Review**:
  - 2-Wöchentlich
  - Die Effektivität der letztendlich abgelieferten Arbeit wird beurteilt und verbessert.
- Replenishment Meeting:
  - Wöchentlich
  - Ähnlich zur Sprint-Planung in Scrum.
- **Kanban Meeting**:
  - Täglich
  - Für die tägliche Abstimmung. Dieses Meeting ermöglicht die Selbststeuerung. Es ähnelt dem Daily Scrum, stellt aber nur die Frage, ob jemand in seiner Arbeit durch etwas behindert wird.
- Delivery Planning:
  - Abh. Von Lieferzyklen
  - Das Überwachen und Planen der Fertigstellung bzw. letztendlichen Lieferung.

Kanban
Service Level Agreements

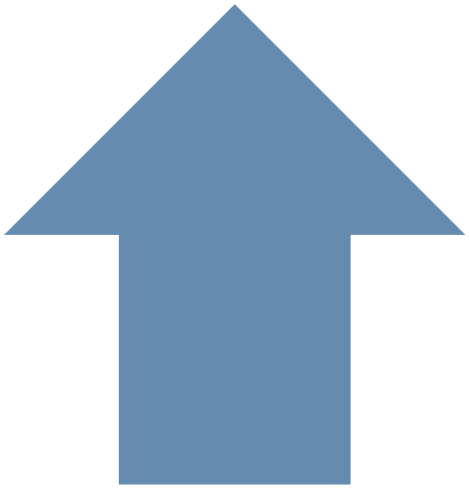
Fachhochschule Dortmund
University of Applied Sciences

- Mittel zur Priorisierung von Tickets
- Häufige SLAs
  - Beschleunigt
    - Hohe Priorität, ggf. muss das ganze Team die laufenden Arbeiten unterbrechen, oder das WiP Limit wird ausgesetzt
  - Fester Termin
    - Priorität richtet sich nach Aufwand und Zeitraum zur Bearbeitung.
  - Standard
    - Normale Priorität, i.d.R. nach FIFO-Prinzip
  - Vage
    - Niedrige Priorität

Kanban
Metriken

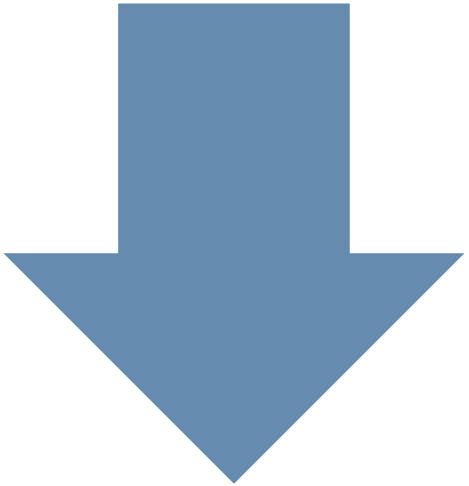
- Cumulative Flow Diagram
  - Zeigt Anzahl der erledigten Tickets pro Zyklus für jede Station
- Work in Progress (WiP)
  - Zeigt die Anzahl der gleichzeitig im System befindlichen Tickets pro Zyklus
- Lead Time
  - Durchlauf- / Lieferzeit pro Ticket als Maß für Arbeitsfluss / Qualität des Arbeitsflusses
- Delivery Rate
  - Anzahl der pro Zyklus erledigten Tickets

Kanban
Vor- und Nachteile



Vorteile

- Transparenz über Projektfortschritt & Probleme
- Fokus auf optimierten Arbeitsfluss und optimale Nutzung der verfügbaren Teamressourcen
- Richtig angewandt können Prozessgeschwindigkeit und Durchlaufzeiten durch kontinuierliche Prozessoptimierung verbessert werden
- Leichtgewichtiges Modell



Nachteile

- Verwertung von Erkenntnissen für Feedback und Optimierung nicht garantiert
- Erfordert zusätzliche Prozesse für Releaseplanung etc.
- Arbeitsfluss hängt von rechtzeitiger Bereitstellung von Tickets und/oder Ressourcen ab

Kanban
Klassifizierung

Fachhochschule Dortmund
University of Applied Sciences

- Agile Softwareentwicklung
- Generisches Modell mit PULL-Prinzip
- Visualisierung linearer Prozesse und Probleme
- Anwendbar für Neuentwicklung oder auch für gesamten Lebenszyklus von Software
- Leichtgewichtig
- Gut geeignet für kleinere Teams
- Inhärente Optimierung des Prozesses (Kaizen)

- Klassische Vorgehensmodelle
  - Wasserfallmodell
  - Nebenläufiges Modell
  - Spiralmodell
  - Prototyping
- Monumentale Vorgehensmodelle
  - V-Modell XT
  - Rational Unified Process (RUP)
- Agile Prozesse
  - Scrum
  - Kanban (IT-Kanban)
  - **Extreme Programming (XP)**
  - Feature Driven Development
  - Adaptive Software Development
  - Crystal
  - Lean Software Development

Extreme Programming
Überblick

- Von Kent Beck im Jahr 1999 erstmalig beschrieben in „Extreme Programming Explained: Embrace Change“
- Kent Beck ist US-amerikanischer Softwareentwickler und Unterzeichner des Agilen Manifest

- Ziele:
  - Auch bei Verzögerungen oder Unstimmigkeiten im Projektverlauf sollen **möglichst viele Anforderungen frühstmöglich verfügbar und nutzbar** sein
  - **Fehler** sollen frühstmöglich durch das Team erkannt werden
  - **Kunden** sollen in den Entwicklungsprozess **einbezogen** werden, um Missverständnisse zu minimieren

Extreme Programming
Values, Principles, Practices

- Values
  - Werte, an denen sich alle Aktivitäten im Team zur Erreichung der Ziele ausrichten
- Principles
  - Bereiche in Softwareprojekten, in denen die Values angewendet werden sollen
  - Brücke zwischen Values und Practices
- Practices
  - Konkrete Umsetzung der Values und Principles
  - Alleinstellungsmerkmal des Extreme Programming
  - In einem Team müssen nicht alle Practices (gleichzeitig) genutzt werden
    - Primary Practices
    - Corollary Practices

Extreme Programming
Values

- Kommunikation
  - U.a. Erarbreitung von Lösungen, Verteilung von Wissen im Team
- Einfachheit (Simplicity)
  - Source Code so einfach wie nötig, um das Problem zu lösen
  - Testbarkeit
  - Ressourcenverschwendung und Kommunikation minimieren
- Feedback
  - Code Quality, Entwicklung von Tests, Durchführung von Tests, Deployment
- Mut (Courage)
  - Offenheit und Vertrauen für Kommunikation und Feedback
- Respekt
  - Grundlage für alle Werte

Extreme Programming Principles

- Humanity
  - Individuen im Zentrum von Entwicklungsprozess und Nutzen
  - Einklang der Bedürfnisse von Individuen, Team & Unternehmen
- Economics
  - Wertschöpfung & Werthaltigkeit von Aktivitäten im Sinn der Unternehmensziele oder Anforderungen
- Mutual Benefit
  - Unmittelbarer Nutzen aufgrund realistischen Bedarfs von Aktivitäten (bsp.: Dokumentation vs. autom. Test)
- Self-Similarity
  - Wiederverwendung von funktionierenden Lösungen und Praktiken
- Improvement
  - Kontinuierliche (Prozess-) verbesserung
- Diversity
  - Vielfältigkeit in Cross-Functional Teams zur konstruktiven Konfliktlösung

Extreme Programming Principles

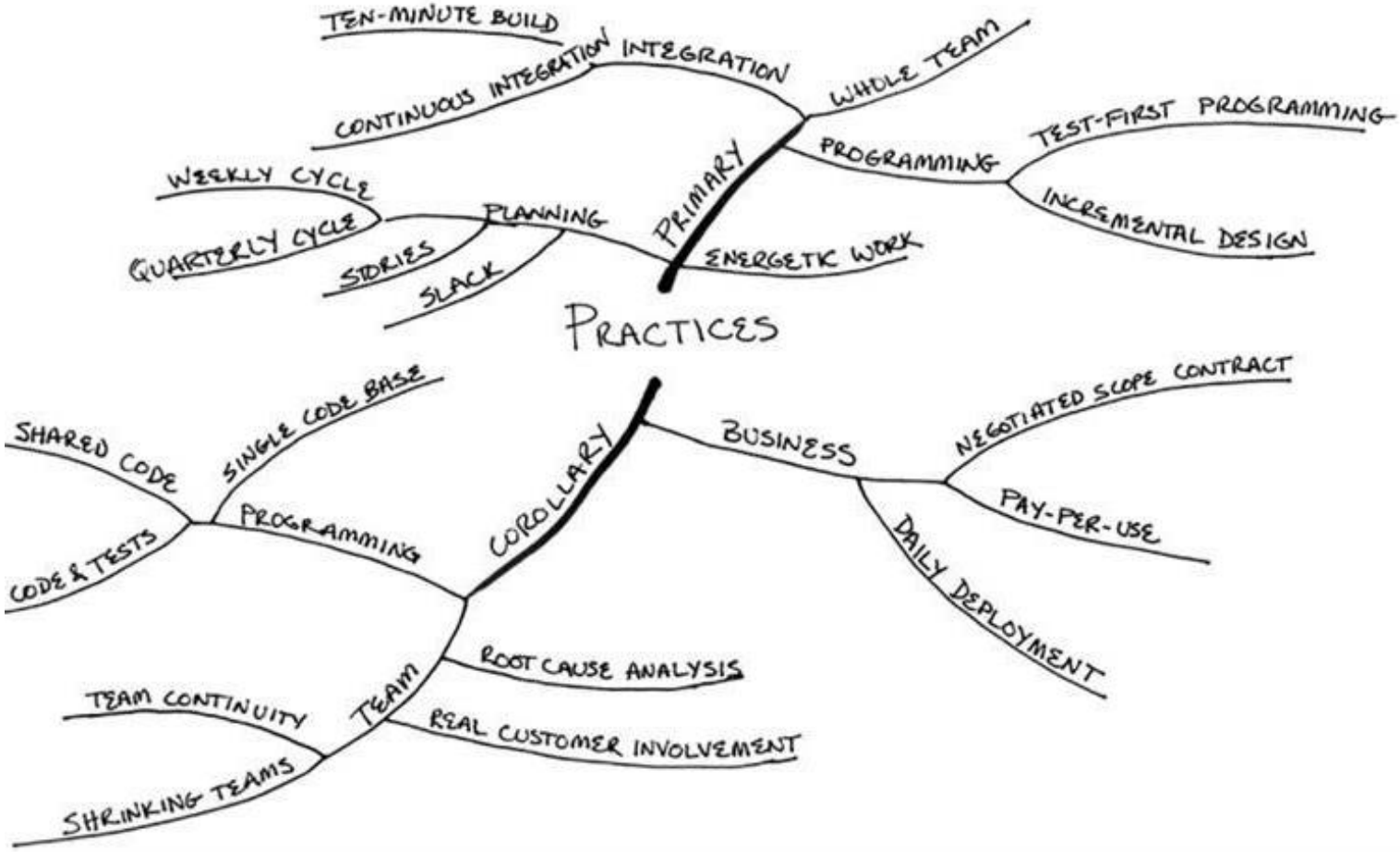
- Reflection
  - Reflektion der Ergebnisse
- Flow
  - Kontinuität in der Erzeugung und Bereitstellung von Ergebnissen
- Opportunity
  - Probleme als Möglichkeiten verstehen
- Redundancy
  - Sicherstellen von Werten durch mehrere, ergänzende Maßnahmen
- Failure
  - Scheitern als Teil der Lösungsfindung
- Quality
  - Gute Softwarequalität für Identifikationspotenzial und Effizienz
- Baby Steps
  - Kleine Inkremente als Ergebnis
- Accepted Responsibility
  - Verantwortung wird nicht festgelegt, sondern akzeptiert

Extreme Programming Practices

- Primary Practices
  - Können direkt vom Team umgesetzt werden
  - Führen zu direkter Prozessverbesserung

- Corollary Practices
  - Bauen auf Primary Practices auf
  - Sollten nur ergänzend zu Primary Practices genutzt und erst eingeführt werden, wenn bereits einige Primary Practices etabliert sind

Extreme Programming Practices



Quelle: Kent Beck, Cynthia Andres: "Extreme Programming Explained: Embrace Change; Second Edition"

Extreme Programming Practices

- Primary Practices (Auszug)
  - Whole Team: Zu jeder Zeit Zugriff auf alle Ressourcen, cross-functional Teams – alle Disziplinen im Team vereint
  - Informative Workspace: Story Board etc. zur Maximierung der Transparenz im Team
  - Pair Programming: Programmierung in Paaren, um Code-Qualität zu erhöhen
  - Stories: Entwicklung werthaltiger Anforderungen aus Benutzersicht
  - Weekly Cycle: Schnelles Feedback, bessere Schätzbarkeit von Anforderungen
  - Quarterly Cycle: Behebung retrospektiver Engpässe, Planung des nächsten Quartals als Gesamtbild

Extreme Programming Practices

- Primary Practices (Auszug)

  - 10-Minute-Build: Test und Build der Software soll < 10 min dauern -> schnelles Feedback
  - Continuous Integration: regelmäßige Integration und Tests mit Gesamtsystem
  - Test-First-Programming: Entwicklung eines automatischen Testfalls vor Entwicklung der Anforderung
  - Incremental Design: Weiterentwicklung des Designs erst mit Notwendigkeit für neue Anforderungen

Extreme Programming Practices

- Corollary Practices (Auszug)
  - Real Customer Involvement
  - Incremental Deployment
  - Team Continuity
  - Shrinking Teams
  - Root-Cause Analysis
  - Shared Code
  - Code and Tests
  - Single Code Base
  - Daily Deployment
  - Negotiated Scope Contract
  - Pay-Per-Use

Extreme Programming
Rollen

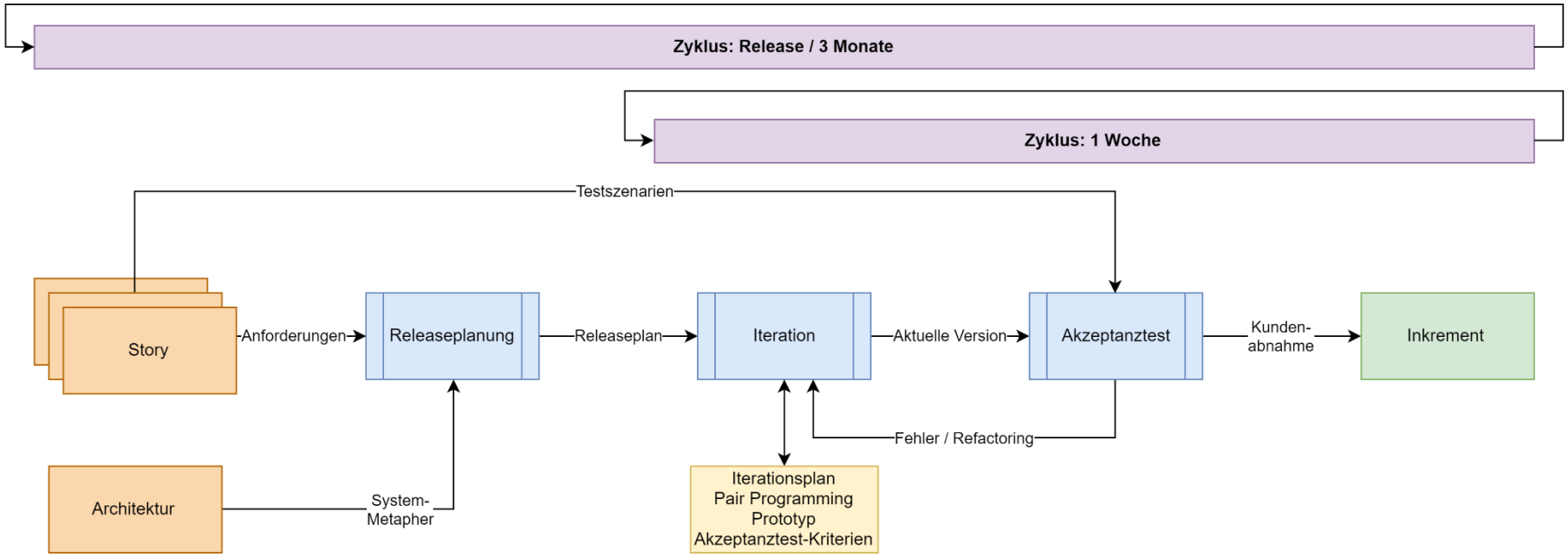
- Tester
  - Beschreibung und Durchführung von Tests, insb. Systemtests
- Interaction Designer
  - Ähnlich Requirements Engineer: Priorisiert neue Anforderungen, bewertet Inkremente hinsichtlich Erfüllung von Anforderungen
- Architect
  - Verwaltung der Architektur
  - Identifikation von Bedarf, Planung und Umsetzung größerer Refactorings
- Project Manager
  - Kommunikation zwischen Stakeholdern und XP-Team
  - Projektmanagementaufgaben

Extreme Programming
Rollen

- Product Manager
  - Analog Product Owner: Erfassung, Verwaltung und Priorisierung von Stories
- Executive
  - Nicht Teil eines XP-Teams
  - Verwaltet i.d.R. mehrere Teams aus Unternehmenssicht
- Technical Writer
  - Erläuterung des Produkts, bspw. durch Betriebsanleitungen, Schulungen, Videoanleitungen
- User
  - Benutzer:innen des Produkts
- Programmer
  - Erstellung von Quellcode

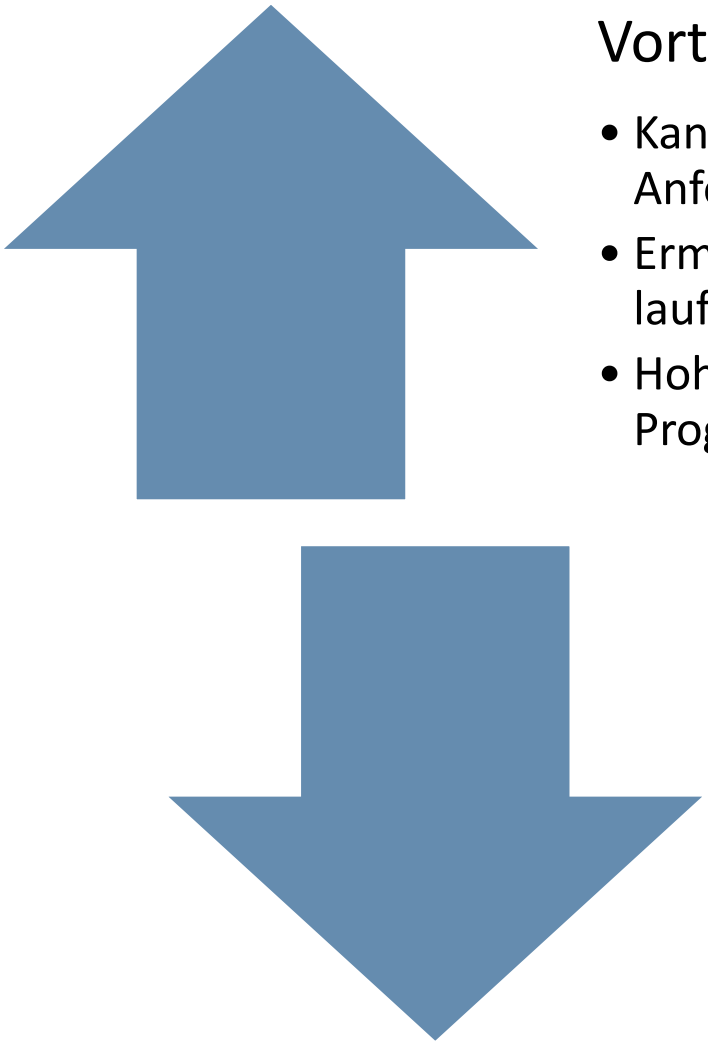
Extreme Programming
Ablauf

Fachhochschule Dortmund
University of Applied Sciences



Extreme Programming
Vor- und Nachteile

Fachhochschule Dortmund
University of Applied Sciences



Vorteile

- Kann flexibel neue oder geänderte Anforderungen berücksichtigen
- Ermöglicht schnelle Bereitstellung lauffähiger Produktinkremente
- Hohe Qualität durch Praktiken wie Pair Programming

Nachteile

- Höhere Entwicklungskosten durch Praktiken wie Pair Programming
- Erfordert hohes Maß an Erfahrung und Selbstorganisation / -disziplin

Extreme Programming
Klassifizierung

- Agiles, iteratives & inkrementelles Vorgehensmodell
- Fokus auf Qualität und einsatzfähigen Produktinkrementen
- Gut geeignet bei
  - Projekten mit hoher Dynamik von Anforderungsdefinitionen oder -änderungen
  - risikoreichen Projekten mit definiertem Endzeitpunkt oder Einsatz neuer Technologien
  - kleinen oder lokal erweiterten Entwicklungsteams
  - Technologien, die automatisierte Tests erlauben

- Klassische Vorgehensmodelle
  - Wasserfallmodell
  - Nebenläufiges Modell
  - Spiralmodell
  - Prototyping
- Monumentale Vorgehensmodelle
  - V-Modell XT
  - Rational Unified Process (RUP)
- Agile Prozesse
  - Scrum
  - Kanban (IT-Kanban)
  - Extreme Programming (XP)
  - **Feature Driven Development**
  - Adaptive Software Development
  - Crystal
  - Lean Software Development

Feature Driven Development
Überblick

Fachhochschule Dortmund
University of Applied Sciences

- 1997 von Jeff DeLuca und Peter Coad entwickelt
  - Zur Durchführung eines zeitkritischen Projekts (15 Monate, 50 Teammitglieder)
  - Erstmals beschrieben 1999 in *Java modelling in Color with UML* von Peter Coad, Eric Lefebvre, Jeff De Luca
  - Seitdem kontinuierlich weiterentwickelt

- Kundenorientiertes und prozesszentriertes Vorgehensmodell
  - Bezieht Entwurfs- und Konstruktionsphasen mit ein
  - Iterative & inkrementelle Vorgehensweise
  - Integration von Kund:innen in die Erstellung von Feature-Listen (statt Stellvertreter:innen wie Product Owner)

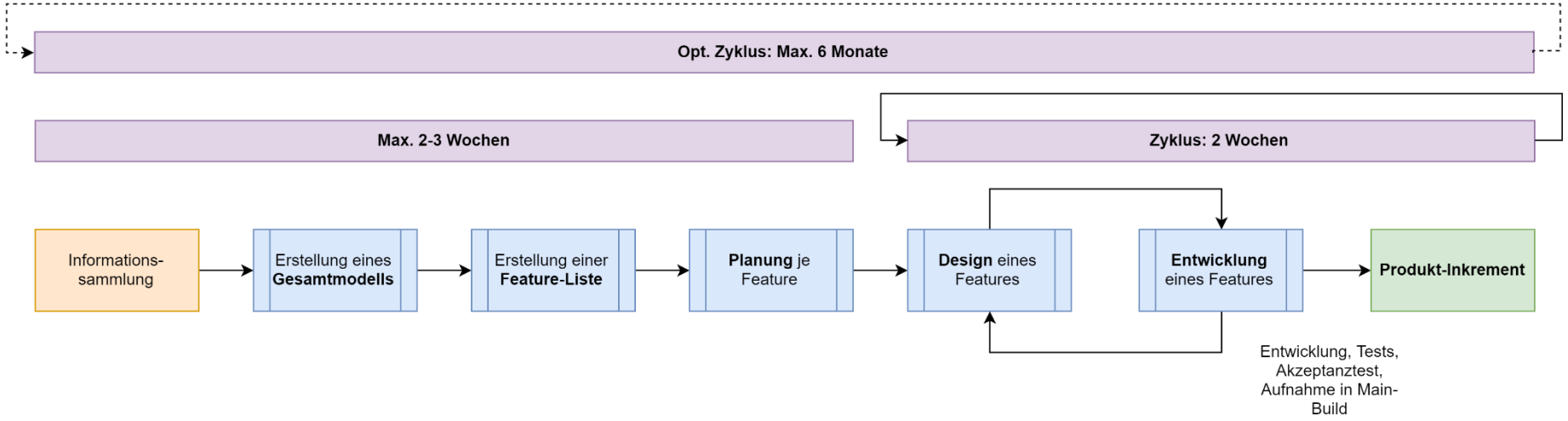
Feature Driven Development
Überblick

Fachhochschule Dortmund
University of Applied Sciences

- Ziel: Bereitstellung lauffähiger Software in Inkrementen
- Fokus auf Features
  - Feature: Konkreter Mehrwert für Benutzer:innen
  - Dauer zur Bereitstellung eines Features: 2 Wochen
  - Hierarchie: Major Feature Set – Feature Set – Feature
  - Feature Team: Chief Programmer + Entwicklungsteam (Class Owner)
- Maximaldauer von 6 Monaten
  - längere Projekte müssten auf kleinere Projekte zerlegt werden

Feature Driven Development
Ablauf

Fachhochschule Dortmund
University of Applied Sciences



Feature Driven Development
Rollen

6 Hauptrollen

- Project Manager
  - Projektleitung für das gesamte Projekt
- Chief Architect
  - Verantwortlich für Design und Modell des Gesamtsystems
- Development Manager
  - Leitet und unterstützt das Entwicklungsteam
  - Überwacht die Aktivitäten des Entwicklungsteams

Feature Driven Development
Rollen

6 Hauptrollen

- Chief Programmer
  - Unterstützt Analyse und Design
  - Kann kleine Entwicklungsteams leiten
- Class Owner
  - Mitglied des Entwicklungsteams eines Chief Programmers
  - Verantwortlich für Design, Quellcode-Erstellung, Testen und Dokumentation eines Features
- Domain Expert
  - Häufig Benutzerinnen und Benutzer oder Kundinnen und Kunden
  - Vermittelt Wissen über die Domäne und das zu lösende Problem sowie Prioritäten

Feature Driven Development
Best practices

- Domain object modeling
  - Teams erstellen Klassendiagramme zur Beschreibung von Objekten und deren Beziehungen einer Domäne
- Developing by feature
  - Wenn eine Funktionalität nicht innerhalb eines Iterationszyklus (2 Wochen) implementiert werden kann, muss sie in kleinere Features zerlegt werden
- Individual class (code) ownership
  - Quellcode (Klassen oder Quellcode-Gruppen) hat genau einen Besitzer / eine Besitzerin
- Feature teams
  - Features umfassen i.d.R. mehrere Klassen mit ggf. mehreren Class Owners, die somit alle zu Design und Implementierung eines Features beitragen

Feature Driven Development
Best practices

Fachhochschule Dortmund
University of Applied Sciences

- Inspections
  - Teams führen Inspektionen zur Identifikation von Fehlern und Maximierung der Qualität durch
- Configuration management
  - Versionsverwaltung und Änderungsdokumentation
- Regular build schedule
  - Fortlaufende Aktualisierung betroffener Systeme
- Progress reports
  - Project Managers und Feature Teams erstellen regelmäßige Fortschrittsberichte

Feature Driven Development
5 Phasen im Detail

1. Entwicklung des Gesamt-Modells

- Ziele:
  - Beschreibung des Geschäftsziels
  - Erstellung von (Domänen-) Modellen zur Definition und Beschreibung des Systems
- Beteiligte Rollen: Domain Expert, Chief Architect, Chief Programmer
- Voraussetzung: Kundinnen und Kunden (Domain Experts) haben bereits eine Vorstellung zu Problemstellung und Software
  - Funktionalitäten und Kernfunktionen

Feature Driven Development
5 Phasen im Detail

2. Entwicklung einer Feature-Liste

- Ziele:
  - Entwicklung von Features auf Basis des Systemmodells
- Beteiligte Rollen: Chief Programmer, Domain Expert
- Vorgehen
  - Definition von Fachgebieten, Geschäftsaktivitäten und Schritten
  - Aus den Schritten werden Features erstellt
  - Schema eines Features: <Aktion> <Ergebnis> <Objekt>
    - Bsp.: Berechne den Gesamtertrag aller Verkäufe
  - Abschließende Überprüfung der Feature-Liste (Selbstkontrolle)

Feature Driven Development
5 Phasen im Detail

3. Planung je Feature

- Ziele:
  - Erstellung eines Entwicklungsplans
  - Definition von Prioritäten und Reihenfolge, in der die Implementierung von Features stattfinden soll
- Beteiligte Rollen: Project Manager, Chief Programmer, Development Manager
- Vorgehen
  - Prioritäten abhängig von Komplexität und Kritikalität/Risiken, Abhängigkeiten zwischen Features und Auslastung der Teams und Individuen
  - Festlegung von Prioritäten und Entwicklungsreihenfolge
  - Zuweisung von Geschäftsaktivitäten zu Chief Programmers
  - Zuweisung von Klassen an Class Owners
    - Klasse: Quellcode-Bereich für Dauer einer Iteration (2 Wochen)
  - Selbstkontrolle des Entwicklungsplans

Feature Driven Development
5 Phasen im Detail

4. Design je Feature

- Ziele:
  - Erstellung eines Entwurfs je Feature
  - Zusammenstellung von Feature Teams für Iteration
- Beteiligte Rollen: Chief Programmer, Class Owners
- Vorgehen
  - Chief Programmer bestimmt, welche Features in der Iteration entworfen und implementiert werden sollen
  - Chief Programmer stellt Feature Team aus betroffenen Class Owners zusammen
  - Softwaretechnischer Entwurf wird erstellt
  - Selbstkontrolle durch Design Review

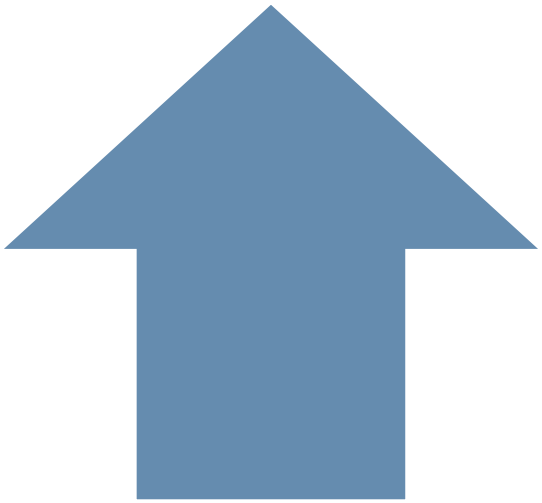
Feature Driven Development
5 Phasen im Detail

5. Entwicklung je Feature

- Ziele:
  - Für die Iteration geplante Features sind implementiert (inkl. User Interface, Feature Prototyp erstellt und getestet)
  - Aufnahme in Main-Build
- Beteiligte Rollen: Chief Programmer, Class Owners
- Vorgehen
  - Implementierung des erstellten Feature Designs
  - Code Inspection, Tests, ggf. Akzeptanztests
  - Freigabe für Main-Build

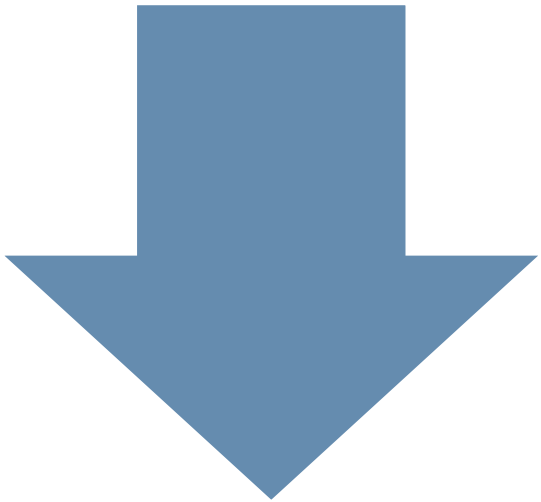
Feature Driven Development
Vor- und Nachteile

Fachhochschule Dortmund
University of Applied Sciences



Vorteile

- Projektfokus und –kontext sind klar umrissen
- Erfordert weniger Meetings
- Benutzerzentriert
- Gut verwendbar für große Teams und komplexe Projekte mit langer Laufzeit
- Feature-orientiert mit regelmäßigen werthaltigen Inkrementen



Nachteile

- Nicht gut geeignet für kleine Projekte oder kleine Teams
- Viel Verantwortung beim Chief-Developer
- Keine (inhärente) schriftliche Dokumentation für Kunden
- Nicht gut geeignet für Wartung oder Ablösung bestehender Systeme
- Nicht so flexibel wie Scrum oder XP

Feature Driven Development
Klassifikation

- Agiles, iteratives & inkrementelles Vorgehensmodell
- Vereint Aspekte der agilen und klassischen Vorgehensmodelle
- Gut verwendbar für
  - Komplexe Projekte mit langer Laufzeit
  - Langzeit- und fortlaufende Entwicklung eines Produktes
  - Projekten mit moderater Dynamik bei Anforderungsänderungen
  - Kontinuierliche Änderung und Ergänzung von Features / Anforderungen in regelmäßigen, vorhersehbaren Iterationen

Herzlichen Dank für Ihre Aufmerksamkeit !