

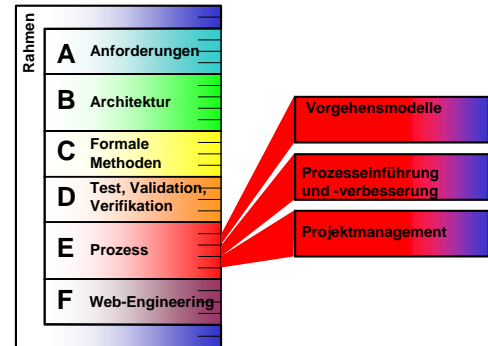
Vorlesung „Methoden des Software Engineering“

Block E „Software-Prozess und Projektmanagement“
Vorgehensmodelle

Martin Wirsing

Einheit E.1, 18.1.2005

Software Engineering (c) 2005, Koch, Störrle, Wirsing LMU München

Gliederung Block E: Software-Prozess

Software Engineering (c) 2005, Koch, Störrle, Wirsing LMU München

Ziele

- **Grundlegende Terminologie und Begriffe (z.B. Prozess, Vorgehensmodell) einführen**
- **Grundlegende Software-Prozessparadigmen und wichtige Vorgehensmodelle kennen lernen: Sequentiell („Wasserfall“), iterativ („spiralförmig“), leichtgewichtig („agil“).**

Software Engineering (c) 2005, Koch, Störrle, Wirsing LMU München

Organisation, Projekt, Prozess, Produkt

- **Jede Gruppe von Menschen, die gemeinsam an einem Ziel arbeiten, hat eine Organisation.**
 - Wenn solch eine Gruppe nur einmalig und für eine begrenzte Zeit zusammenarbeitet, nennt man solch eine Organisation **Projekt**.
- **Es gibt viele verschiedene Organisationsformen.**
 - In der industriellen Produktion und in der Erbringung von Dienstleistungen gibt es häufig eine Aufbauorganisation in Form einer Matrix aus z.B. Funktion und Region.
 - IT-Unternehmen und sonstige höherwertige Dienstleistungen gliedern sich dagegen häufig in eine Aufbauorganisation (z.B. nach Geschäftsbereichen) und quer dazu eine Projektorganisation.
- **In jedem Fall erzeugt die Organisation Wertschöpfung durch Herstellung von Produkten.**
- **Das Schema der Herstellung von Produkten in einer Organisation, aber auch die tatsächliche Herstellung eines konkreten Produkts nennt man Prozess.**
 - Offensichtlich bedingen sich Prozess, Produkt und Projekt(organisation) gegenseitig.

Software Engineering (c) 2005, Koch, Störrle, Wirsing LMU München

Software-Prozess-Lehre (SWP-Lehre)

- **Als SWP-Lehre bezeichnet man den Teilbereich der Softwaretechnik, die sich mit dem Vorgang der Herstellung von Software befasst.**
- **Im Gegensatz dazu steht die Methodenlehre (Methodik), die sich mit dem Lösen eines speziellen Problems in konkreten Einzelschritten befasst.**
- **Ziel der SWP-Lehre ist es, Prozesse besser definieren, ausführen und kontrollieren zu können, insbesondere soll die Sw.-Herstellung dadurch**
 - produktiver (weniger ausgelieferte Fehler bei weniger Aufwand);
 - schneller (Durchlaufzeit, Time-to-market),
 - beweglicher,
 - vorhersagbarer, und
 - nachvollziehbarer werden.
- **Gegenstand der SWP-Lehre sind daher z.B.**
 - SWP-Beschreibungssprachen;
 - Empirische Untersuchungen von Prozessen;
 - Umsetzung, Einführung und Verbesserung von SWP in der Praxis.
- **Die SWP-Lehre hat daher viel mit der Organisationslehre gemeinsam.**

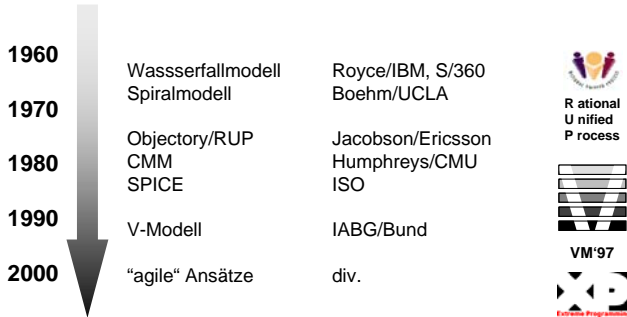
Software Engineering (c) 2005, Koch, Störrle, Wirsing LMU München

Prozess-Paradigma vs. Vorgehensmodell

- **Ein Vorgehensmodell ist eine Sammlung aufeinander abgestimmter Richtlinien, die die Durchführung von Projekten unterstützen.**
 - Dies können sein z.B. Prozess- und Dokumentenschemata („Ergebnistypen“), Rollenbeschreibungen, Muster, Beispiele usw.
 - Vorgehensmodelle sind oft für Länder, Branchen, Firmen oder Projekttypen spezialisiert.
- **Man unterscheidet drei Idealstandards für Vorgehensmodellen, auch SWP-Paradigmen genannt:**
 - sequentiell/phasenorientiert („Wasserfall“),
 - iterativ („spiralförmig“),
 - leichtgewichtig („agil“).

Software Engineering (c) 2005, Koch, Störrle, Wirsing LMU München

SWP-Paradigmen im historischen Überblick

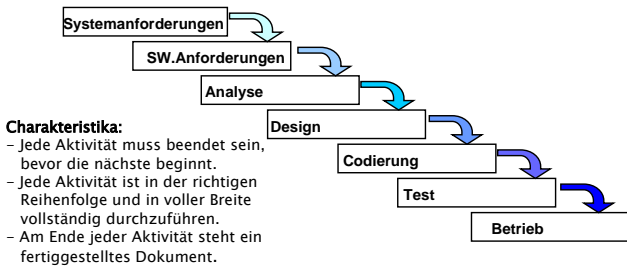


Das sequentielle Paradigma

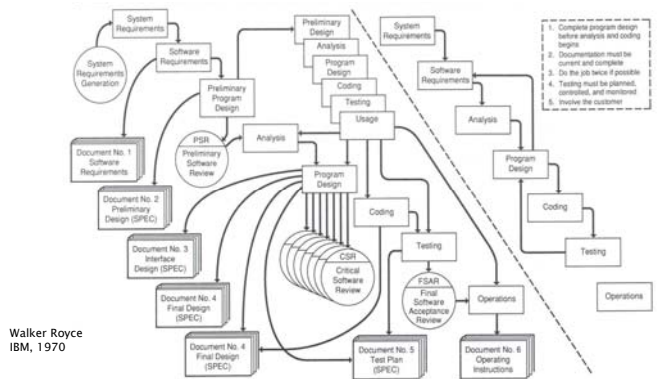
- Das **sequentielle Paradigma** bezeichnet ein sequentielles Vorgehen mit klar definierten Phasen und Ergebnissen
- **Bekannteste Modelle:**
 - Wasserfall-Modell
 - V-Modell

Wasserfallmodell (schematisch)

- **Eingeführt von Walker Royce 1970, Weiterentwicklung der „stufenweisen Entwicklung“ („stagewise development“) von Benington 1956.**
- **Ansatz:**
 - Top-down Stufenmodell mit eingeschränkter Rückkopplung
 - Phasensynchronisation durch (Zwischen-) Produkte
 - Geringer Managementaufwand

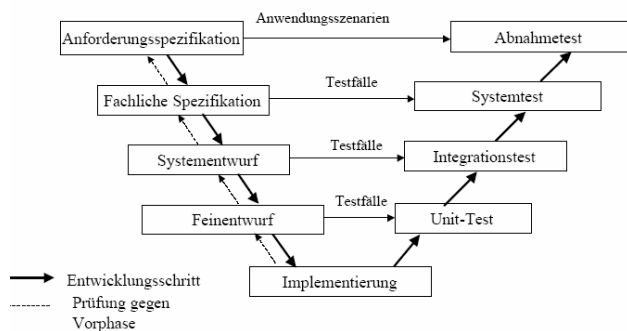


Wasserfallmodell (nach Royce 1956)



V-Modell (Prinzip)

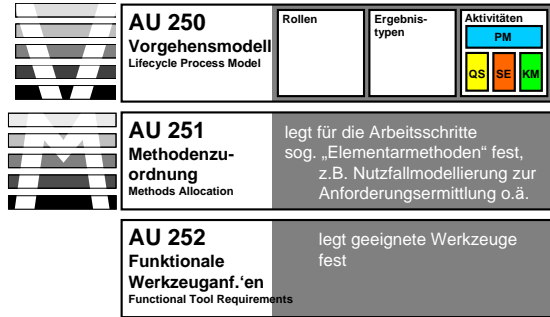
- **Erweiterung des Wasserfallmodells, eingeführt von Barry Boehm 1984**



V-Modell (VM'97)

- **Basis für umfassendes Vorgehensmodell für IT-Systeme der BRD (verbindlich für Wehrbeschaffung und Bundesverwaltung) entwickelt seit 1991 vom Bundesamt für Wehrtechnik und Beschaffung und IABG**
- **Aktuell VM'97, auch für inkrementell/iterative, objekt-orientierte, komponentenbasierte Vorgehen und Reverse-Engineering (federführend: IABG)**
- **Module von VM'97:**
 - Projektmanagement (PM)
 - Systementwicklung (SE)
 - Qualitätssicherung (QS)
 - Konfigurationsmanagement (KM)

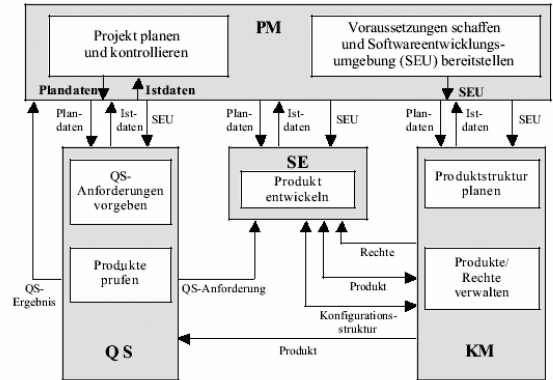
VM'97 Aufbau („3-Ebenen-Modell“)



AU = allgemeiner Umdruck bzw. englisch
GD = general directive

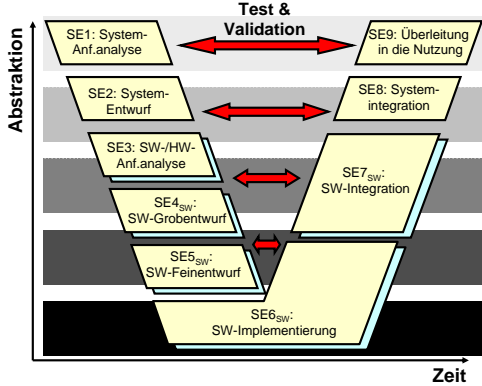
Software Engineering (c) 2005, Koch, Störle, Wirsing LMU München

VM'97 Datenflüsse zwischen den Submodellen



Software Engineering (c) 2005, Koch, Störle, Wirsing LMU München

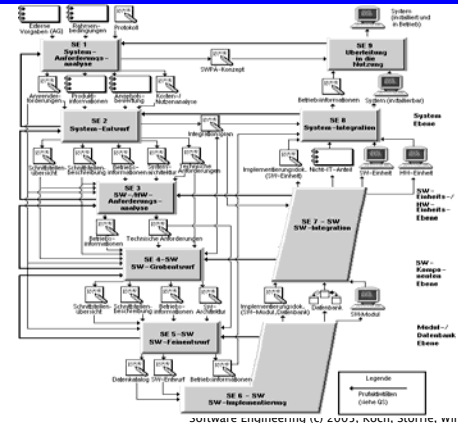
VM'97 Phasen



SW = Software
Anf. = Anforderungen

Software Engineering (c) 2005, Koch, Störle, Wirsing LMU München

VM'97 Datenfluss im Submodell SE



[nach GDP]
SE = Systemerstellung

Software Engineering (c) 2005, Koch, Störle, Wirsing LMU München

VM'97 Bewertung, Verbreitung und Unterstützung

- Das VM ist relativ abstrakt und (dadurch) robust.
 - Insbesondere ist es gut anpassbar, und hat als erstes (großes) Vorgehensmodell die Anpassung („Tailoring“) explizit mit im Vorgehensmodell definiert, und liefert auch Hilfestellungen
- Sehr gute Unterstützung durch Browser, Nutzergruppen, Handbücher, Schulungen usw.
- Sehr große Nutzerbasis, nicht nur in Deutschland

Software Engineering (c) 2005, Koch, Störle, Wirsing LMU München

Das sequentielle Paradigma Vor- und Nachteile

- Vorteile
 - einfach durchzuführen
 - schränkt Freiheitsgrade stark ein, daher auch für sehr große Projekte anwendbar
 - sehr effizient bei bekannten und konstanten Anforderungen
 - ist gut zu verlernen (notwendig z.B. für Prozessverbesserung)
- Nachteile
 - Risiken gesammelt am Schluss („Big Bang“)
 - starr während des Ablaufs

Software Engineering (c) 2005, Koch, Störle, Wirsing LMU München

Das sequentielle Paradigma Anwendungsszenarien

- **Gut anwendbar bei klarer, relativ fixer Funktionalität:**
 - System- und Basissoftware wie OS, DB, Web-Server;
 - Branchen-Software wie SAP R/3.
- **Notwendig bei hohen Qualitäts- und Zuverlässigkeitsanforderungen:**
 - eingebettete Systeme (Automotive, Maschinenbau, Medizinische Geräte, Anlagensteuerung, Aerospace, Defense, ...);
 - Telekommunikations-Systeme;
 - rechtliche Anforderungen wie Revisionsicherheit oder Nachvollziehbarkeit.
- **Kundenwunsch:**
 - VM verpflichtend für viele öffentliche Hände in der BRD.
 - Kunden-Mitarbeiter sollen integriert werden, und können nur VM.

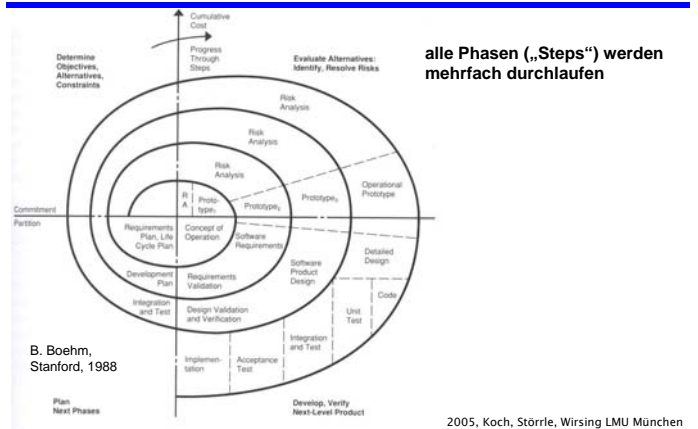
Das iterative Paradigma

- **Iterativ** heißt, dass der Entwicklungsprozess mehrfach iteriert wird: statt den „Wasserfall“ einmal zu durchlaufen, werden kleine Wasserfälle hintereinander gesetzt.
- **Ähnlich aber anders:**
 - **Evolutionär:**
Die Entwicklung folgt einem beliebigen Ansatz, das System wird kontinuierlich weiterentwickelt, z.B. durch kaskadierende Wasserfälle
- **Das iterative Paradigma ist eine Weiterentwicklung des sequentiellen Paradigmas aus der Erkenntnis, dass Software länger lebt als erwartet und auch vom Funktionsumfang her gepflegt werden muss.**
- **Bekanntestes Modell:**
 - **Spiral-Modell (ein Meta-Modell)**
 - **Unified Process**

Inkrementelle Entwicklung

- **Das iterative Paradigma unterstützt inkrementelle Entwicklung, d.h. dass das zu erstellende System nicht in einem Rutsch freigegeben wird, sondern in mehreren Stufen.**
- **Diese Stufen müssen nicht zwangsläufig Endbenutzerfunktionalität enthalten, was aber häufig damit gemeint ist.**
- **Nach Objectory soll ein Inkrement 5-20 Nutzfälle groß sein.**

Das Spiral-Modell

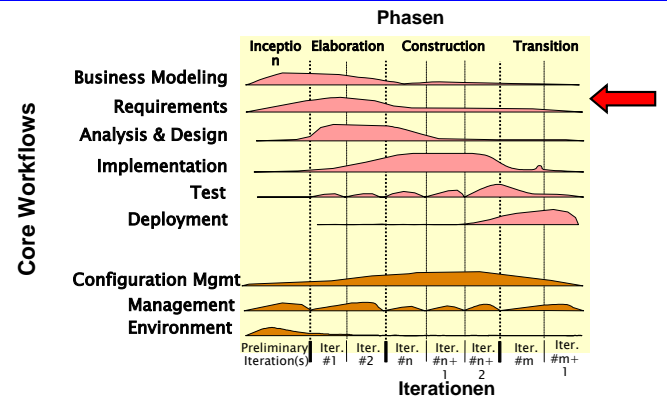


Rational Unified Process (RUP)

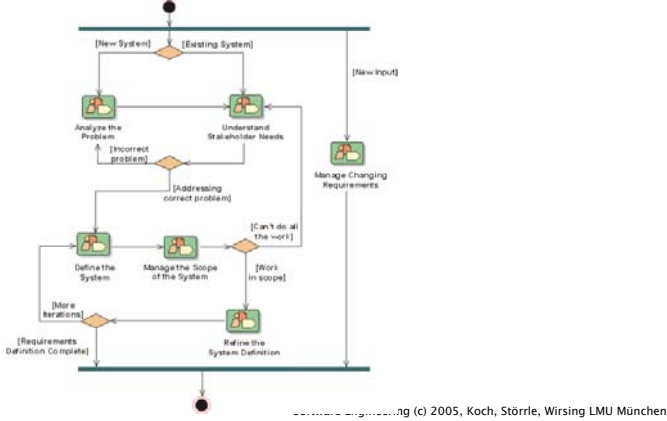
- **Der RUP hieß früher Objectory und wurde dann (zusammen mit, wegen oder trotz Jacobson) von Rational aufgekauft.**
- **Der RUP wird von Rational/IBM bezeichnet als**
 - inkrementell & iterativ,
 - Architektur-zentriert, und
 - Anwendungsfall-getrieben;
- **Der RUP ist sehr feinkörnig**
 - Schlecht: Aufgrund seines Detailgrades und seiner Komplexität benötigt der RUP massive Werkzeugunterstützung.
 - + Gut: ungeübte Nutzer werden sehr detailliert unterstützt.
- **Der RUP hat die UML als Methode (i.S.d. VM'97) „voreingestellt“.**



RUP Phasen, Prozesse und Iterationen



RUP Beispiel: Der Prozess "Requirements"



Software Engineering (c) 2005, Koch, Störrie, Wirsing LMU München

Das iterative Paradigma Vor- und Nachteile

- **Vorteile**
 - Risiken können früher erkannt werden
 - volatile Anforderungen können besser berücksichtigt werden
 - inkrementelle Auslieferung wird erleichtert
- **Nachteile**
 - Mehrarbeit
 - komplexeres Projektmanagement
 - schwerer messbar

Software Engineering (c) 2005, Koch, Störrie, Wirsing LMU München

Das iterative Paradigma Anwendungsszenarien

- **Anforderungen sind:**
 - nicht ganz klar;
 - instabil.
- **Funktionalität und Termin wichtiger als Aufwand.**
- **Nachvollziehbarkeit ist nicht in dem Maße erwünscht wie bei sequentiellem Paradigma**
- **Organisation verfügt über hohe Kompetenz.**

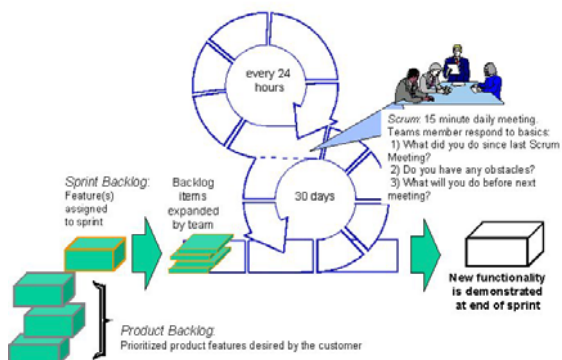
Software Engineering (c) 2005, Koch, Störrie, Wirsing LMU München

Das adaptive (oder agile) Paradigma

- Unter einem **adaptiven SW-Prozess** versteht man eine Weiterentwicklung des iterativen Paradigmas, bei der die Planung der Iterationen dynamisch erfolgt.
- **Charakteristikum:**
kontinuierliche Anpassung an Änderungen
- **Prinzipien:**
 - Individuum und Interaktion (geht) vor Prozess und Werkzeug
 - ausführbare SW vor vollständiger Dokumentation
 - Zusammenarbeit mit Kunden vor Vertragsverhandlung
 - Berücksichtigung von Änderungen vor Beharren auf Plan
- **Bekannte Vorgehensweisen:**
 - XP
 - SCRUM, ...

Software Engineering (c) 2005, Koch, Störrie, Wirsing LMU München

Scrum



Software Engineering (c) 2005, Koch, Störrie, Wirsing LMU München

„Extreme Programming“ (XP) Ursprung

- Erstmals formuliert Mitte der 90´er von Kent Beck & Ward Cunningham.
- Entstanden aus dem C3-Projekt (Lohnbuchhaltung bei Chrysler).
- Basiert auf den Prinzipien „simplicity, communication, feedback & courage“.
- Besteht aus 12 „key practices“.
- 2001 einigen sich einige „Gurus“ darauf, ihre Ansätze künftig als „Agile“ zu vermarkten statt „Lightweight“. Noch besser wäre „Adaptiv“ gewesen.

Software Engineering (c) 2005, Koch, Störrie, Wirsing LMU München

XP – Practices

A – Projektmanagement

- Planning Game
- Small Releases
- 40-hour Week (sustained pace)

B – Architektur

- Simple Design
- Refactoring

C – Dokumentation

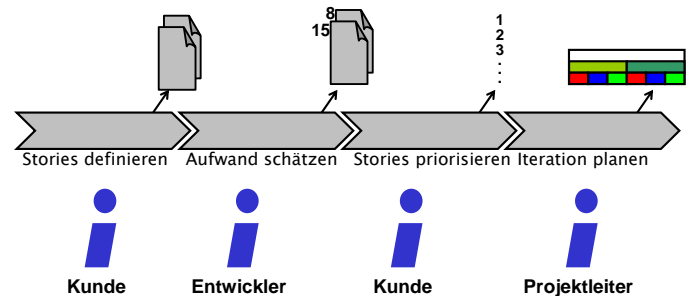
- Coding Standard
- Metaphor

D – Spezifikation und Validation

- Testing (Test First)
 - Pair Programming
 - On-Site Customer
 - Collective Ownership
 - Continuous Integration
- Dies sind Vorschläge, die auf die Einzelsituation angepasst werden müssen (-> „Tailoring“)

XP-Practices

A (Projektmanagement): Planning Game



XP-Practices

A (Projektmanagement): Planning Game

- Probleme
 - Der Kundenvertreter vertritt den Kunden, nicht den Nutzer. Von der Zufriedenheit des Nutzers aber hängt der Erfolg der Software ab!
 - Kunde ist nicht willig oder fähig zur Kooperation.



Copyright © 2003 United Feature Syndicate, Inc.

XP-Practices

A (Projektmanagement): *sustainable pace*

- XP fordert *sustainable pace*
 - 40-Stunden Woche, 2 Wochen Urlaub
 - Fortbildung
 - größere Verantwortung und Eigenständigkeit der Mitarbeiter.
- Das hört sich vielleicht etwas merkwürdig an, aber in den USA
 - ...sind Arbeitszeiten und Urlaubsanspruch nicht gesetzlich geregelt;
 - ...ist die Tayloristische Arbeitsteilung sehr stark verbreitet;
 - ...gibt es keine starken Arbeitnehmervertreter
 - ...herrscht ein geringes durchschnittliches Bildungsniveau;
 - ...ist der Quereinstieg die Regel, nicht die Ausnahme.
- Hierzulande sieht das – meistens – anders aus.

XP-Practices

B (Architektur):

- Simple Design
 - Refactoring
 - (Restrukturierung)
- spezifisch für OO-Sprachen
- erfordert starke Werkzeugunterstützung – die gibt es aber inzwischen

XP-Practices

C (Dokumentation):

- Coding Standard
 - Metaphor (gemeinsame Begriffswelt)
- „Doku is for wheenies“
- ...aber gut, um in ein Projekt reinzukommen,
- oder einen Review zu machen,
- oder um dem Kunden-Management in die Hand zu drücken,
- oder was auch immer der Grund ist, ein System verstehen zu wollen, wenn man nicht von Anfang an dabei war, und die „alten Hasen“ nicht da sind, oder unwillig sind (z.B. „innere Kündigung“).

XP-Practices

D (Spezifikation und Validation):

- Testing (Test First)
- On-Site Customer
- Continuous Integration
- Collective Ownership
- Pair Programming



Software Engineering (c) 2005, Koch, Störle, Wirsing LMU München

XP

Beobachtungen

- Die XP-Practices bündeln bekannte sinnvolle Techniken:
 - Coding Standards
 - Metaphor
 - sustained pace
 - test first
 - pair programming
 - continuous integration
 - customer on-site
- Wichtig ist gute Zusammenarbeit im Team

Software Engineering (c) 2005, Koch, Störle, Wirsing LMU München

Das adaptive Paradigma

Vor- und Nachteile

Vorteile

- Gut einsetzbar bei unklaren Zielen und sich ändernden Anforderungen/Umgebung
- Verspricht besseres Kosten/Nutzen-Verhältnis
- Vermutlich durchschnittliche Code-Qualität besser

Nachteile

- Ergebnis ist nicht vorhersagbar
 - Qualitätseigenschaften können nicht garantiert werden
 - Oft nicht nachvollziehbar, wie eine Funktion zustande kommt
- 80–90% aller Software läuft auf eingebetteten Systemen.
 - Das bezieht sich z.B. auf Maschinen und Anlagen jeder Art und Größe, Verkehrsmittel, Militärische Anwendungen, Medizinische Geräte.
 - Hier sind Fehler fatal – ein „agiler“ Prozess kommt nicht (oft) in Frage.
- Refactoring geht nicht bei nicht oo-Sprachen.
 - Aber ein Großteil (ca. 90%) der bestehenden Applikationen weltweit ist in Cobol, PL1, Assembler und C, und OO-Sprachen sind nicht immer optimal.

Software Engineering (c) 2005, Koch, Störle, Wirsing LMU München

Literatur

- Noack: *Techniken der OO-Sw.-Entwicklung*, Springer, 2001
- Humphrey: *Managing the Software Process*. CMU-Press/Addison Weseley, 1990
- Noack: *Eine Werkbank für den Zuschnitt von oo Softwareprozessen*. Wirtschaftsinformatik 4 2002
- Bunse, v. Knethen: *Vorgehensmodelle kompakt*. Spektrum, 2002
- Dröschel, Wiemers: *Das Vorgehensmodell '97*, Oldenbourg, 1997
- Coldewey: *Ein Überblick über die agile Entwicklung* Objektspektrum 1/2002, S. 69–77
- Beck: *Extreme Programming*, Addison Weseley, 1998

Software Engineering (c) 2005, Koch, Störle, Wirsing LMU München