Software-Engineering

Vorlesung 12 vom 17.01.2005

Sebastian Iwanowski
FH Wedel

Projektmanagement

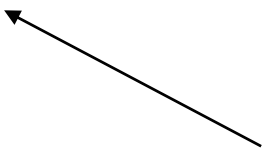
Zeitliche Organisation des Projektablaufs

- Wasserfallmodell
- Prototyping
- Spiralmodell

} Grobe Zeitplanung für das Gesamtprojekt

- RUP: Rational Unified Process
- XP: eXtreme Programming

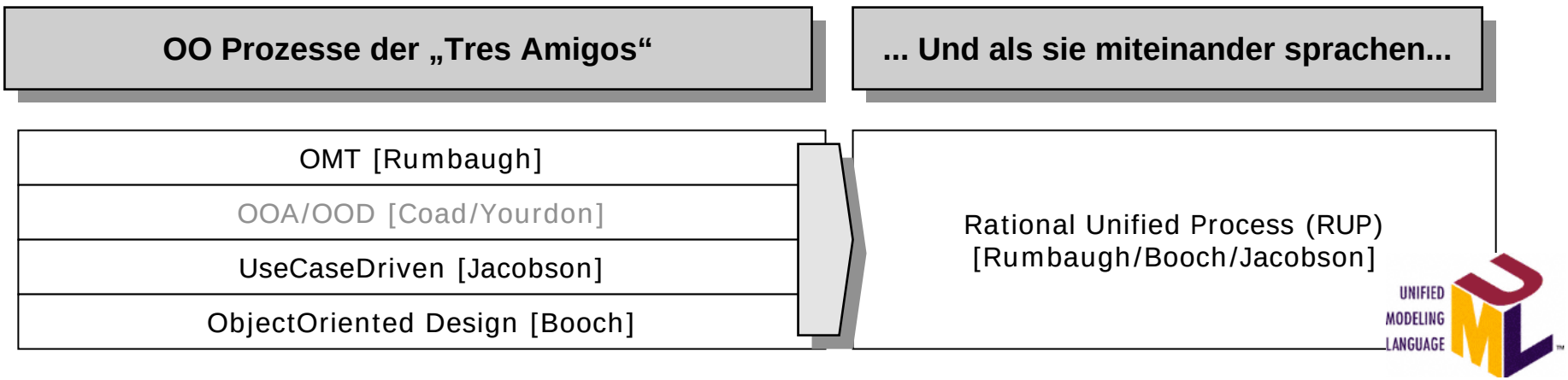
} Detaillierte Zeitplanung für Teilphasen



mit Forderungen an Personalorganisation

RUP: Rational Unified Process

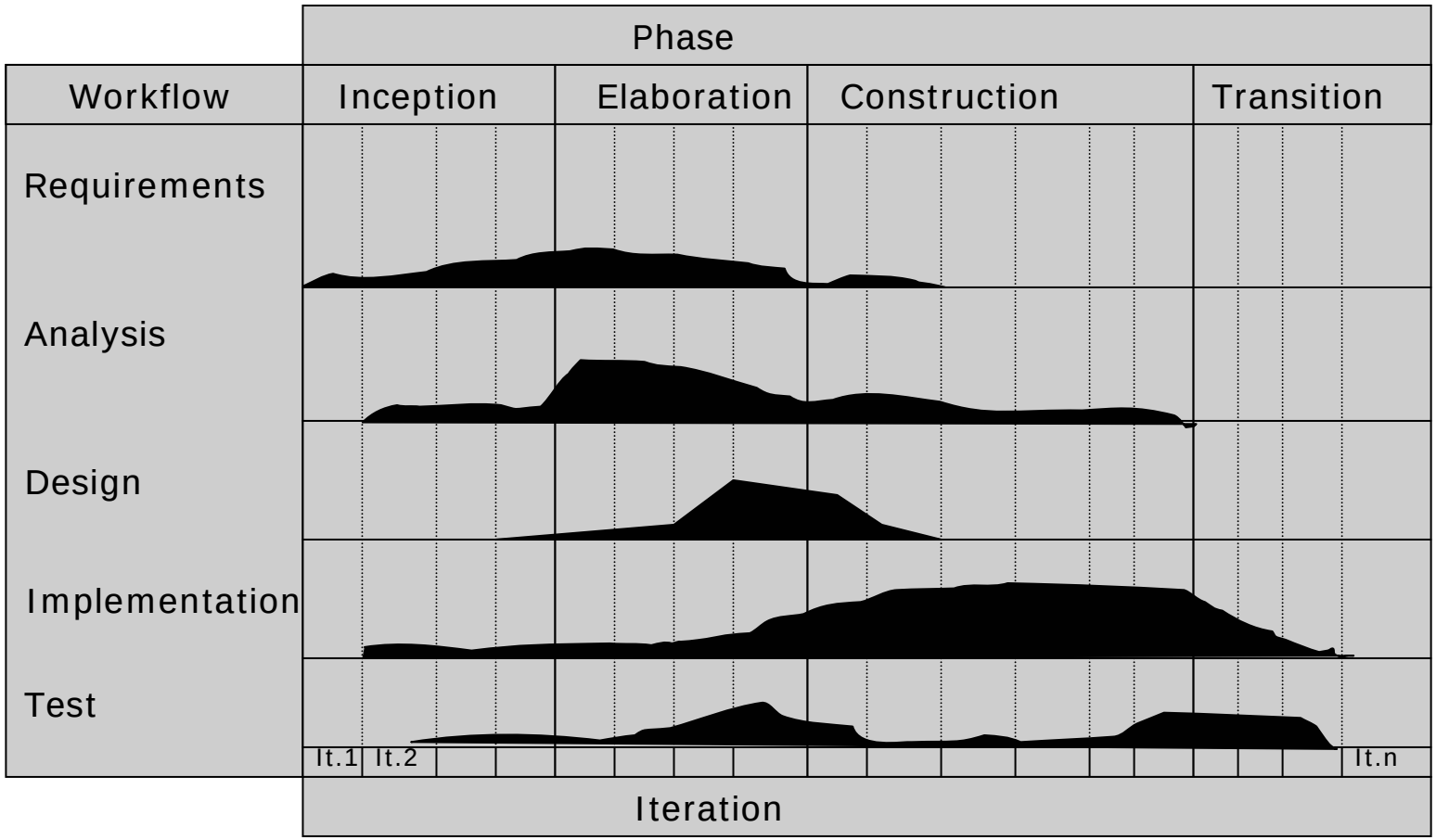
Historie



RUP ist eine Anleitung,
wie man UML in der Projektorganisation bestmöglich einsetzt.

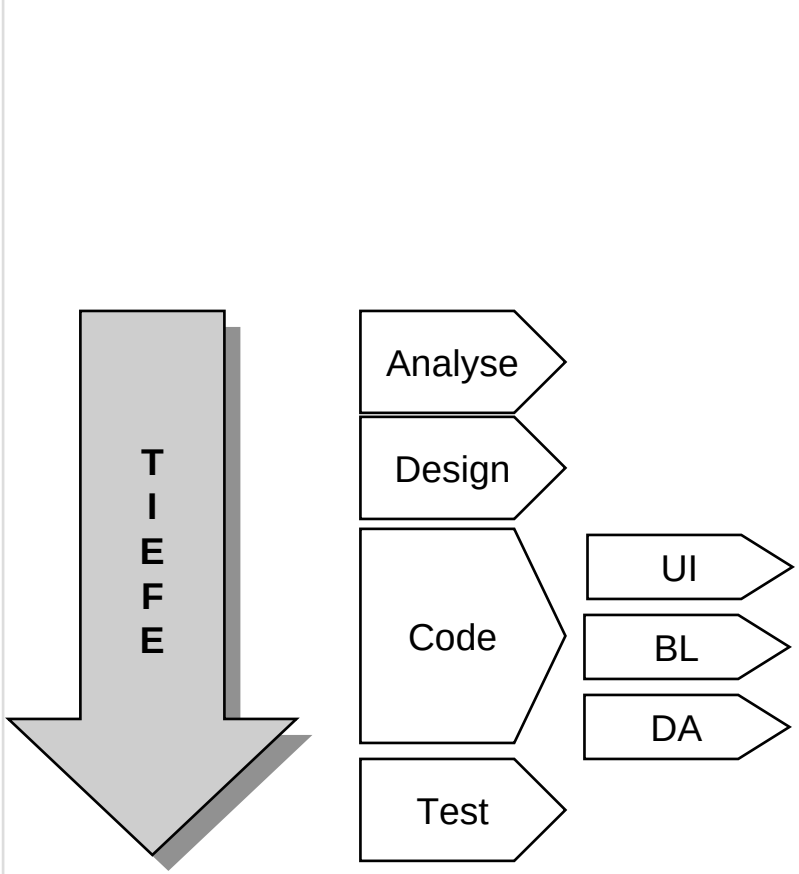
RUP: Rational Unified Process

Aufteilung der Arbeitsschritte in bestimmte Phasen



RUP: Rational Unified Process

Zeitliche Einordnung in Breiten-/Tiefenraster



Lieber Projektleiter,

das ist DEIN Problem, wir glauben nicht an eine allgemeingültige Lösung.

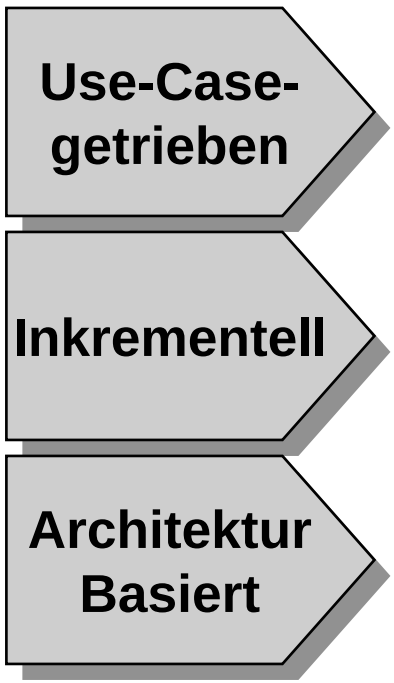
Wenn Du aber verschiedene Schwerpunkte setzt - Risikominimierung,
Analyse, Architektur, Code, Test, etc - helfen wir Dir gerne:

Die Workflows sagen Dir, was Du planst (welche Ergebnisse Du produzierst).

Die Phasenbeschreibung sagt Dir, wie Du planst (worauf Du achten mußt).

RUP: Rational Unified Process

Wesentliche Bausteine und Prinzipien



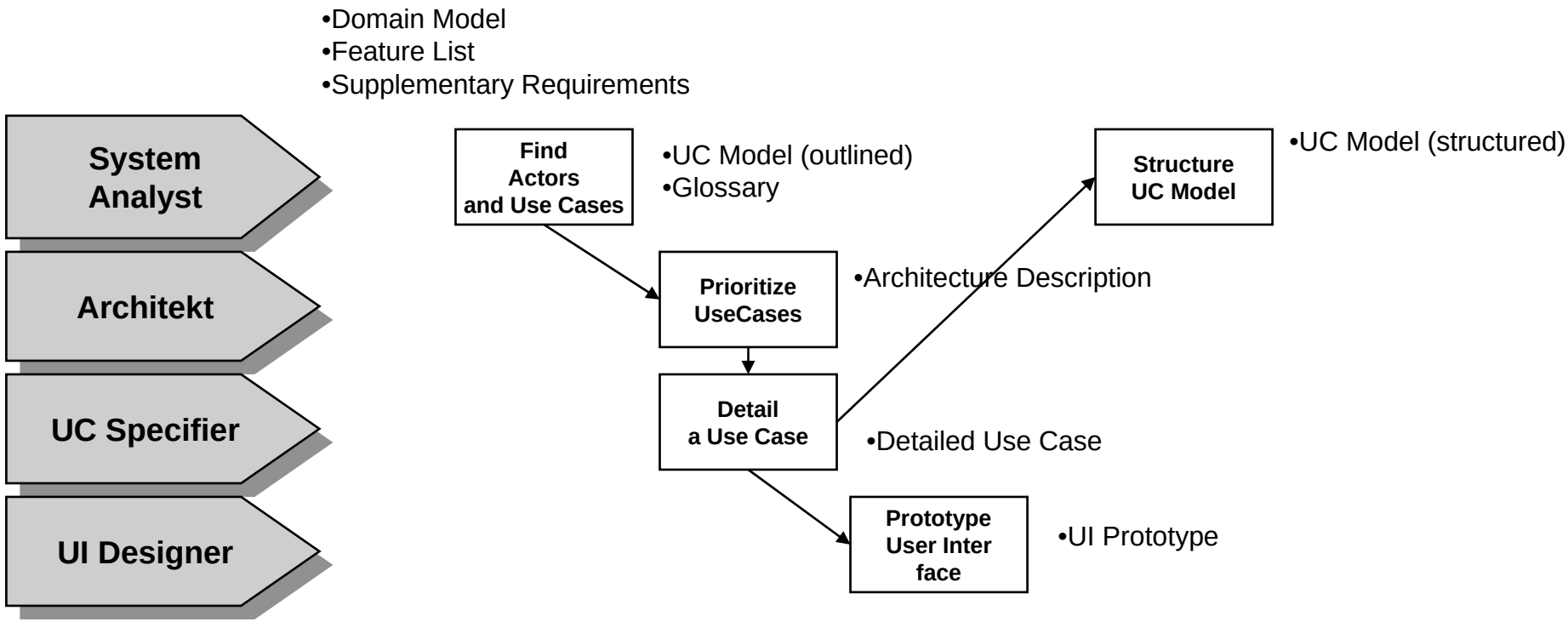
- Use cases bilden die Basis für alle Phasen incl. Test
- Die Planung erfolgt nach Use-Case-Paketen und Fertigungstiefe

- Das System wird in Iterationen errichtet
- Jede Iteration kann etwas mehr als die Vorgängeriteration

- Für kritische Use Cases wird eine Musterlösung erstellt
- Diese Lösung dient als „Schema F“ für die weiteren Use Cases
- Diese Lösung heißt „Architektur“

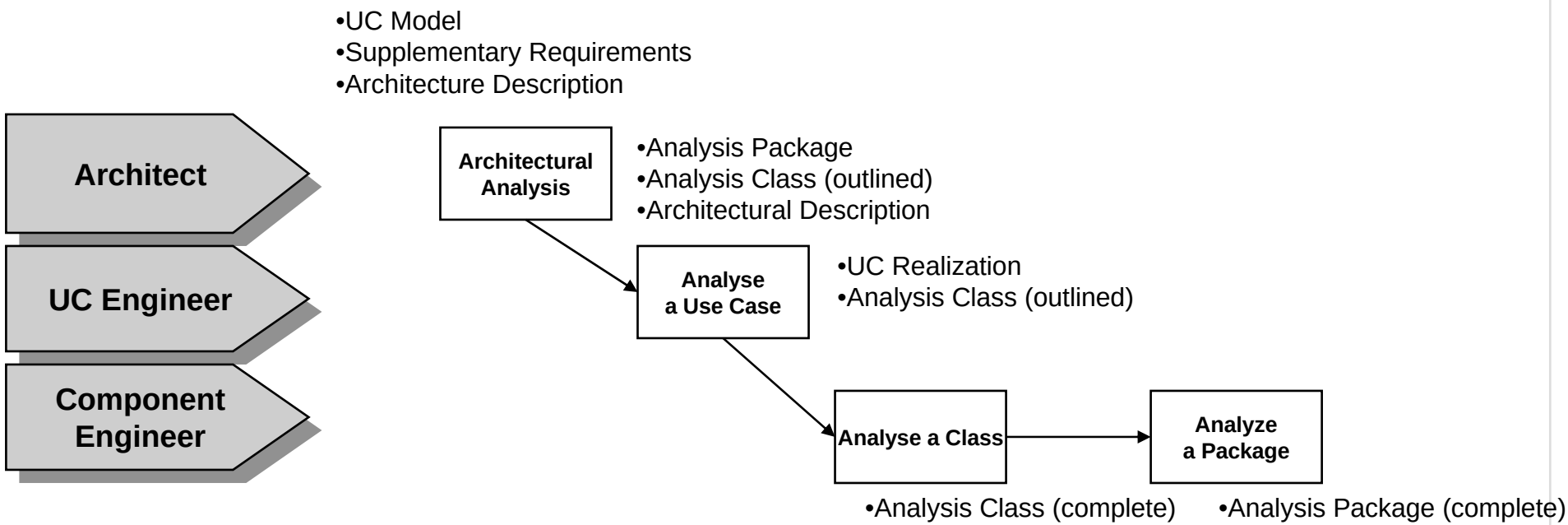
RUP: Rational Unified Process

Requirements Workflow



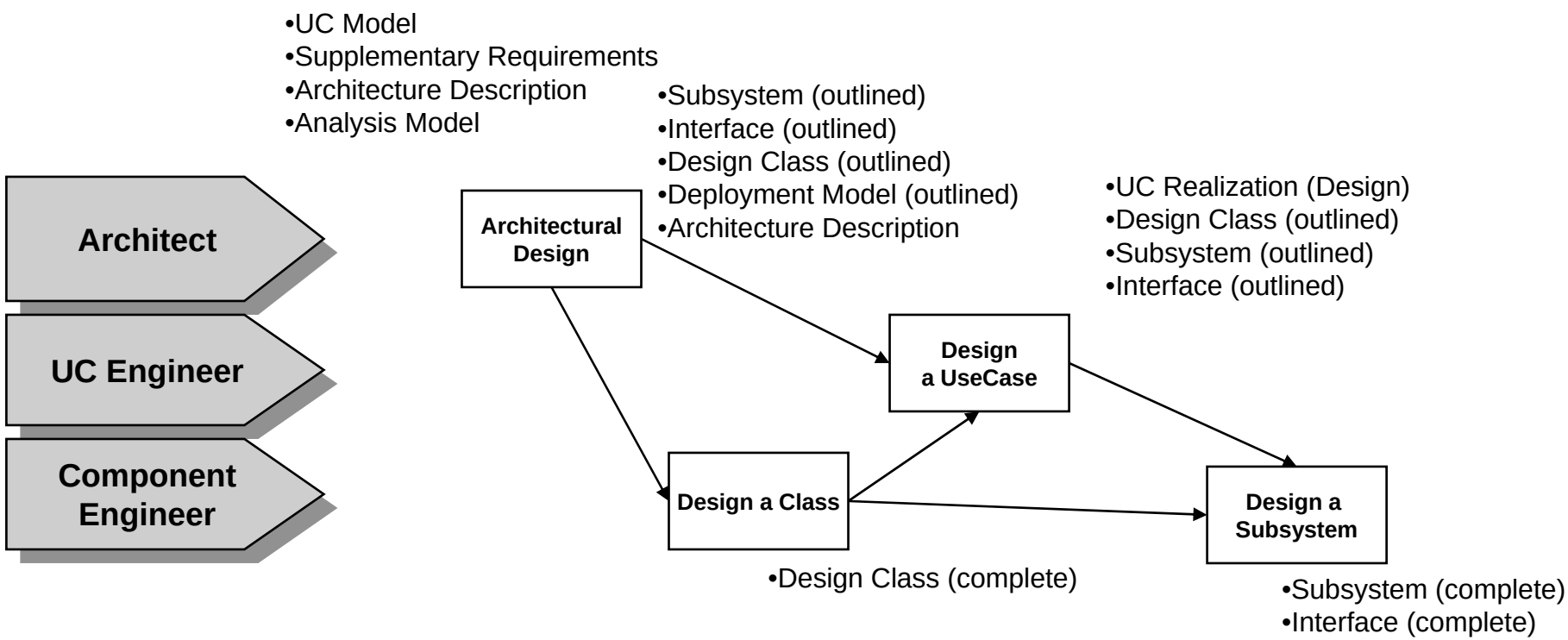
RUP: Rational Unified Process

Analysis Workflow



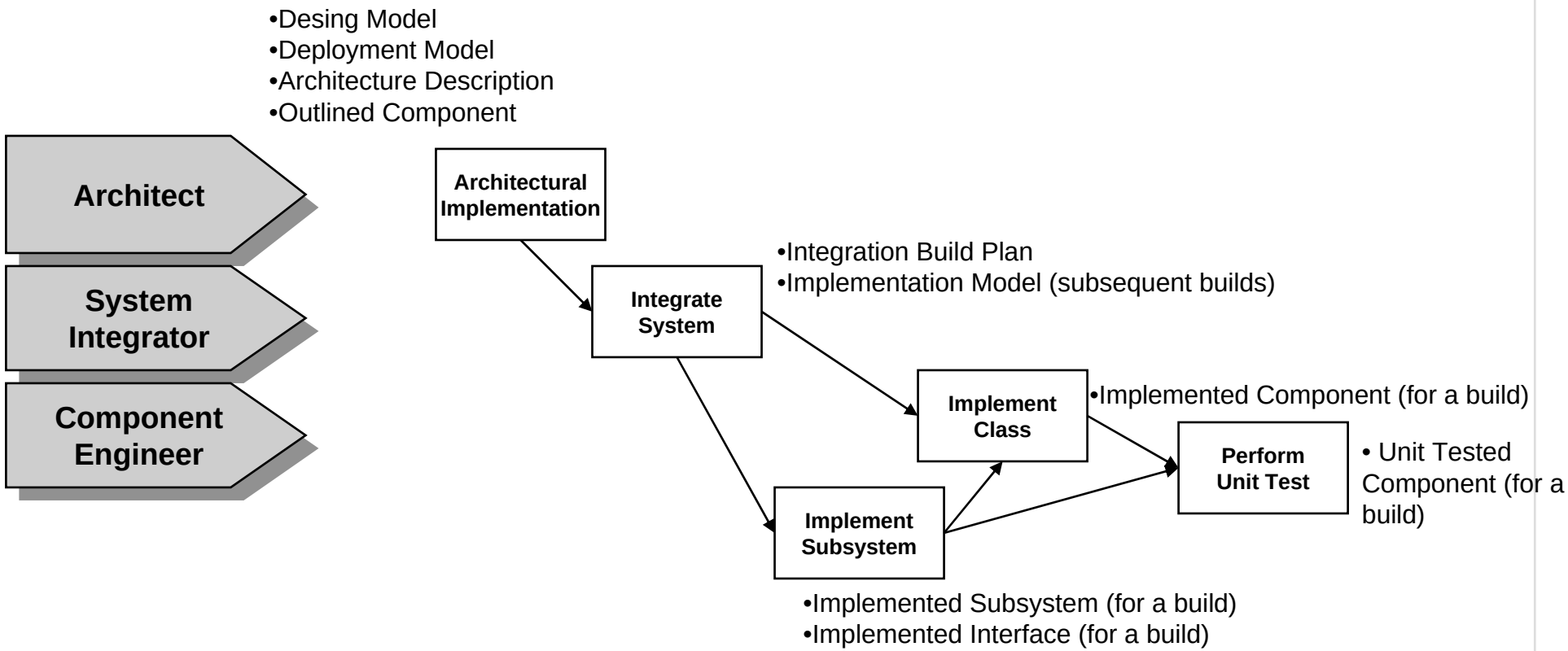
RUP: Rational Unified Process

Design Workflow



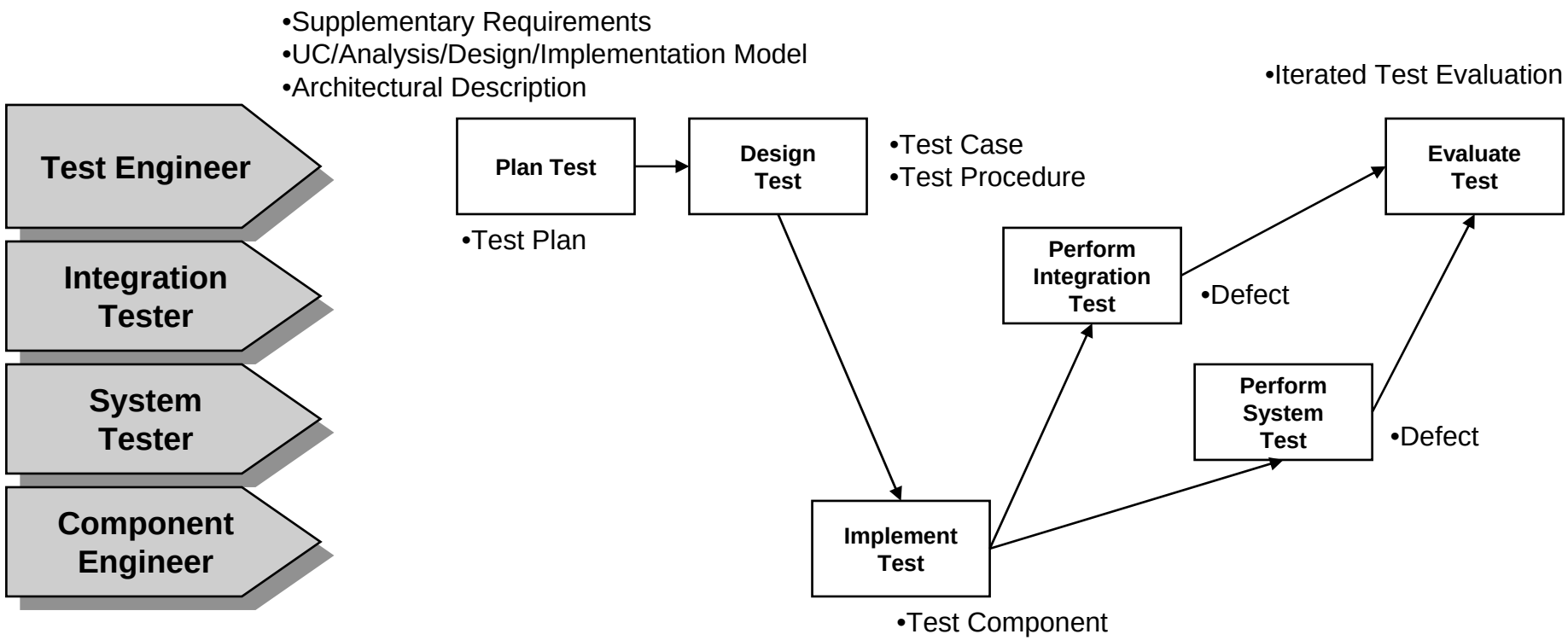
RUP: Rational Unified Process

Implementation Workflow



RUP: Rational Unified Process

Test Workflow



XP: eXtreme Programming

Historie

- propagiert durch Kent Beck (Smalltalk-Softwareentwickler) 1999
- weitere Weggefährten: Martin Fowler, Erich Gamma, Ralph Johnson

Wesentliche Prinzipien

- XP stellt die **Programmierung** in den Vordergrund – der Code ist das, wofür bezahlt wird
- XP nimmt Moving Targets als den Normalfall an. Änderungen werden nicht bekämpft, sondern übernommen: **embrace change**
- Bei XP steht im Verhältnis Auftraggeber und Auftragnehmer **Vertrauen** im Vordergrund ...

XP: eXtreme Programming

Bestandteile von XP

Code Reviews sind gut:

- Also werden Sie dauernd gemacht.
- **Pair Programming**
- Es gibt informelle **Coding Standards**

Permanentes Testen ist gut:

- Also werden die Testfälle ständig auf dem Stand gehalten und ausgeführt.
- Use Cases dienen auch als Testscripts für **Unit Testing.**
- **Testfälle werden gebaut, bevor codiert wird.**

Integrationstests sind wichtig:

- Deshalb wird permanent integriert und wieder getestet: **Continuous Integration**

XP: eXtreme Programming

Bestandteile von XP

Einfachheit ist gut:

- **„Do the simplest thing that might possibly work“**

Design ist wichtig:

- Wird deshalb während des Codierens durch **Refactoring** ständig verbessert

Kurze Iterationen sind wichtig:

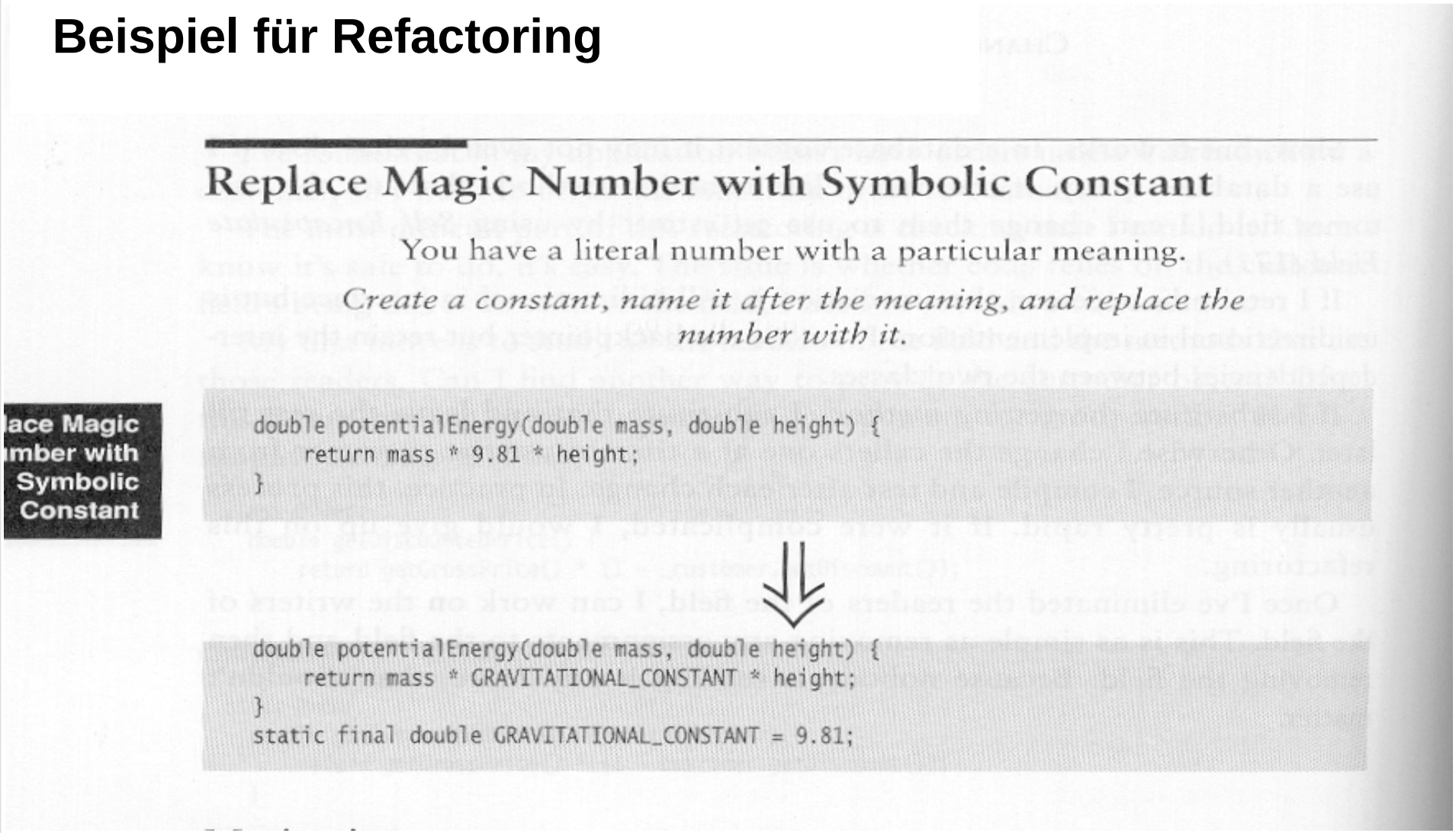
- Also werden die Iterationen extrem kurz gemacht
- 10 Tage, dann wieder „Planning Game“

Feedback durch den Kunden ist wichtig:

- Also ist der Kunden im selben Raum: **On site customer**

XP: eXtreme Programming

Beispiel für Refactoring



Replace Magic Number with Symbolic Constant

You have a literal number with a particular meaning.

Create a constant, name it after the meaning, and replace the number with it.

lace Magic
mber with
Symbolic
Constant

```
double potentialEnergy(double mass, double height) {
    return mass * 9.81 * height;
}
```

⇓

```
double potentialEnergy(double mass, double height) {
    return mass * GRAVITATIONAL_CONSTANT * height;
}
static final double GRAVITATIONAL_CONSTANT = 9.81;
```

Quelle: Martin Fowler

XP: eXtreme Programming

Voraussetzungen für XP:

- kleines Entwicklungsteam (2-10)
- gute Leute, die nicht anfangen zu hacken
- geringe Änderungskosten: Grundlage Refactoring
- eine Software-Entwicklungsumgebung, die permanente Tests erlaubt
- einfaches Design – keine gigantischen Frameworks

XP: eXtreme Programming

Achtung !

Manche benutzen XP als Ausrede um nicht zu dokumentieren ..

- Im Chrysler C3 Projekt wurde nicht dokumentiert
- Das hat später durch Wechsel von Teammitgliedern zu Problemen geführt.
- Außerdem wurde das Projekt inzwischen abgebrochen - angeblich aus „politschen Gründen“.

Manche sagen, sie machen XP:

- Nur leider ist der Kunden nicht greifbar
- Damit kann es nicht mehr funktionieren und bleibt dann lediglich eine Ausrede für schlechte Architektur und mangelnde Dokumentation

RUP ↔ XP

RUP versus XP oder RUP kombiniert mit XP ?

siehe http://www.netobjectdays.org/pdf/00/slides/barchfeld.pdf

Einiges spricht für die Kombination:

- Use cases helfen der Verständigung mit dem Kunden und erzeugen das notwendige fachliche Verständnis für die Programmierer.
- Kleine Inkremente erhöhen die Planungssicherheit und schaffen Möglichkeiten zu einer adaptiven Vorgehensweise.
- Code-Unit-Tests sind das Wesentliche, um feststellen zu können, ob Inkremente wirklich Inkremente sind, d.h. nachweisbare Resultate aufweisen.

auch hier: Dogmatik schadet !

Personalorganisation

Am Anfang eines Projekts:

Zusammenstellung von Teams

Bereitstellung von benötigten Ressourcen

Aufbau einer Organisationsstruktur

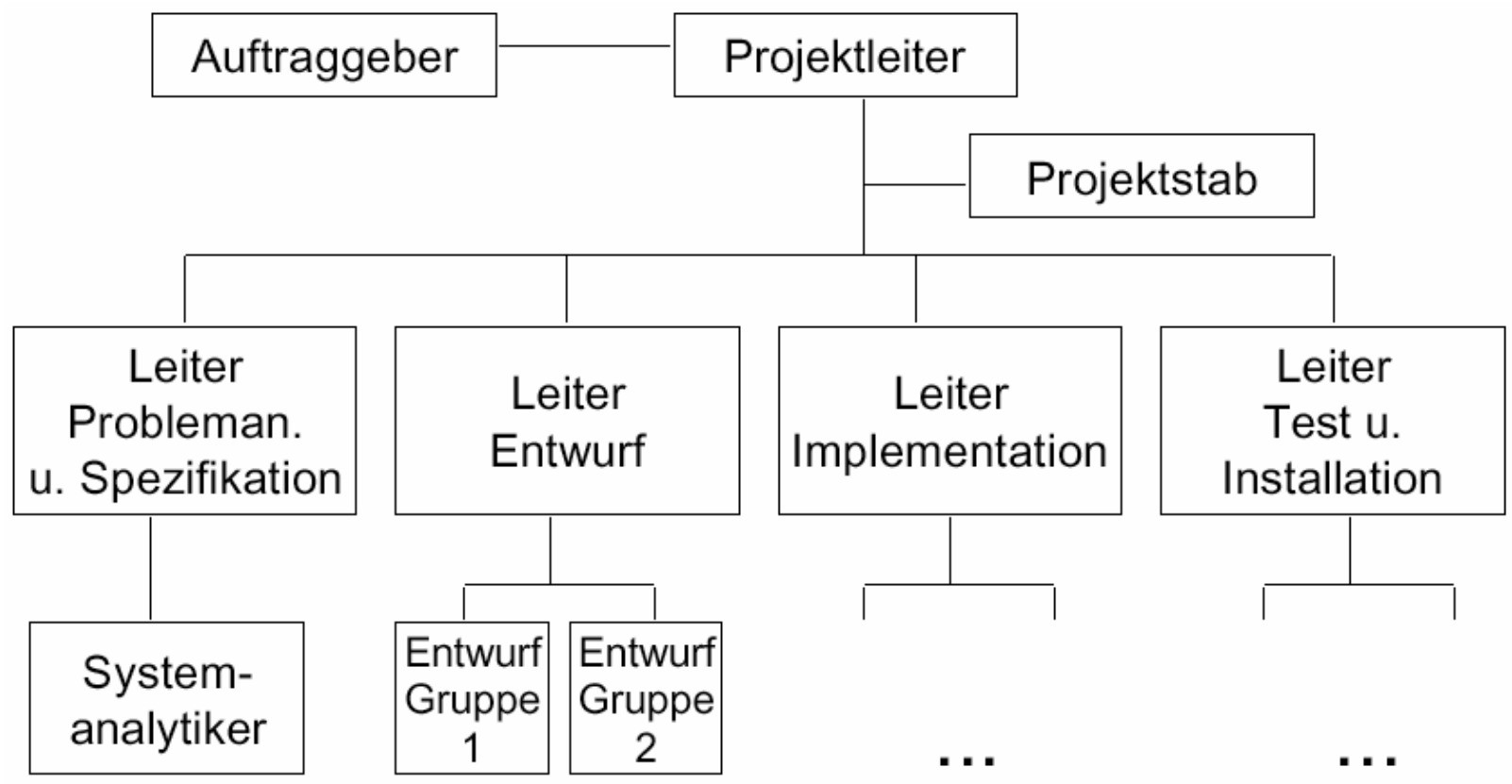
- Zuordnung von Teilaufgaben/Phasen zu Teams oder einzelnen Personen
- Genaue Festlegung der Aufgaben, Rechte und Pflichten aller am Projekt beteiligten Personen (Zuständigkeiten)

Typen von Organisationsstrukturen

- Klassische hierarchische Struktur
- Chef-Programmierer-System

Personalorganisation

Klassische hierarchische Struktur



Personalorganisation

Klassische hierarchische Struktur

Probleme:

- Projektleiter zu weit von Programmierung entfernt
- Mehrstufigkeit behindert Kommunikation
- Aufstieg in Hierarchie bis zur Inkompetenz

Personalorganisation

Chef-Programmierer-System

Verzicht auf Projektleiter, der nicht an Systementwicklung beteiligt ist

Einsatz von sehr guten Spezialisten, die mit hoher Eigenverantwortung arbeiten

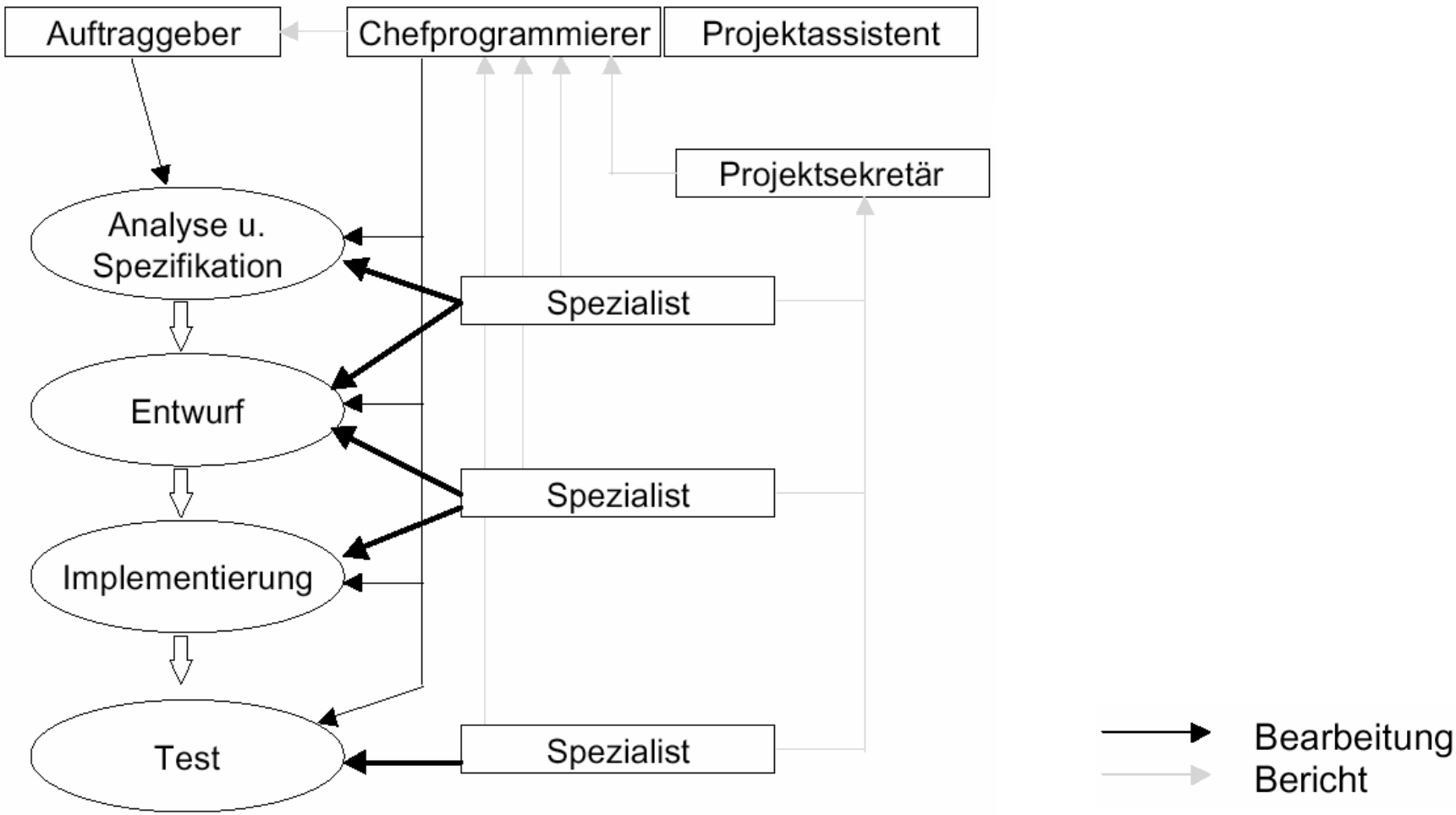
Beschränkung der Teamgröße

Zusammensetzung der Teams:

- Chef-Programmierer
- Projektassistent
- Projektsekretär
- Spezialisten (Systemanalytiker, Programmierer, Testspezialisten)

Personalorganisation

Chef-Programmierer-System



Personalorganisation

Chef-Programmierer-System

Vorteile

- Chefprogrammierer kann durch direkte Einbindung Kontrollfunktion besser wahrnehmen
- Geringere Kommunikationsschwierigkeiten
- Kleinere (Spezialisten-)Teams sind produktiver

Nachteile

- Beschränkung auf kleine Teams
- Anforderungen an Chefprogrammierer nahezu unerfüllbar
- Stellung des Projektsekretärs problematisch

Zusammenfassung Software-Engineering

2 Grundlegende Begriffe und Prinzipien

Systeme und Modelle, Prinzipien der Abstraktion, Zerlegung und Perspektivenbildung

2 3 Softwareplanung

Lastenheft und Pflichtenheft: Zweck, wesentliche Merkmale, Unterschiede

3 4 5 Systemanalyse

Prozessorientierte Modellierungsmethoden: Funktionsbaum, Datenflussdiagramm, Entscheidungstabelle/-baum, Abgrenzung zu Kontrollflussdiagrammen
Datenflussorientierte Modellierungsmethoden: Entity-Relationship-Modellierung, Objektorientierte Modellierung (UML), wesentliche Begriffe und Prinzipien, Unterschiede, Interpretierung und Modellierung einfacher Beispiele in einer der beiden Modellierungstechniken, Konversionen zwischen diesen Techniken

5 6 Softwareentwurf

Abgrenzung zur Systemanalyse, Entwurfsmethoden top-down und bottom-up, Leitlinien für die Modularisierung

Zusammenfassung Software-Engineering

6 7 8 CASE-Tools

UML: Unterschiede zwischen Systemanalyse und Softwareentwurf, Klassendiagramme, Use-Case-Diagramme, Sequenzdiagramme, Zustandsübergangsdiagramme, Aktivitätsdiagramme, Interpretation und Modellierung kleiner Beispiele
Aris: Aris-Haus mit den wesentlichen Sichten, EPKs (mit Erweiterung), Zusammenspiel von EPKs mit UML-Bausteinen in Aris
Eignung von UML und Aris für die verschiedenen Entwicklungsphasen von Software

8 9 Aufwandsabschätzung

Zusammenhang von Kosten und Zeit bei Software, verschiedene Messgrößen
Basismethoden: Gewichtungsmethode, parametrische Gleichungen, Multiplikatormethode, Analogiemethode, Relationsmethode, Kennzahlenverfahren, Prozentsatzverfahren (mit Einordnung in übergeordnete Ziele)
Allgemeine Schätzverfahren: Function-Point-Methode (mit Kenntnis des allgemeinen Verfahrens, Anwendungsmöglichkeiten, Vorteilen und Nachteilen), COCOMO (nur grundsätzliches Prinzip, Bewertung dieser Methode)

Zusammenfassung Software-Engineering

9 10 11 Qualitätsmanagement

Qualität im Wettbewerb mit anderen Entwicklungskriterien, Klassifizierung der verschiedenen Qualitätskriterien für Software: Unterschiede und gegenseitige Widersprüche, statische und dynamische Qualitätssicherungsverfahren
Dynamische Verfahren im Detail: Blackbox und Whitebox-Tests, Unterschiede
Whitebox-Tests im Detail: Kontrollflussbezogene Verfahren, Überdeckungskriterien, datenflussbezogene Verfahren (defs/uses)
Blackbox-Test-Typen: Definitionen und die wesentlichen Vor- und Nachteile

11 12 Projektmanagement

Wasserfallmodell, Prototyping, Spiralmodell: Definitionen und die wesentlichen Vor- und Nachteile
RUP, XP: Wesentliche Prinzipien, Unterschiede
Vorteile vom Chefprogrammiererorganisation zur streng hierarchischen Organisation

Das war die Vorlesung Software-Engineering:

Reaktion auf die Kritik und Anregungen