Qualitätsmanagement

- BiBi 5. Klasse
  - 2 SE Qualitätsmanagement
  - 2.1 Testing
  - 2.2 Patterns / Antipatterns
  - 2.3 Refactoring / Code Smells

BAT / KAUF
2015/2016

2. QM - Motivation

Kleine Bugs - Große GAUs
Ein paar falsche Zahlen oder Zeilen und schon passiert‘s:

- Explosion Ariane 5 (1996)
- Verlorengegangene Venus-Sonde Mariner1 (1962)
- Koffer-Debakel am Flughafen Denver
- Neues Computer System bei US Fed (Fehlbuchungen 4 Mrd $)
- Kundendaten frei im Internet bei Credit Suisse (Image-Schaden)
- Wall-Street-Crash (1987)
- Spendable Geldautomaten bei Postbank-Service Card (2002)

2. QM - Definition

Qualität [DIN 55350, Teil 11]

Qualität ist die Gesamtheit von Eigenschaften und Merkmalen eines Produkts oder einer Tätigkeit, die sich auf deren Eignung zur Erfüllung gegebener Erfordernisse bezieht.

Software-Qualität [DIN ISO 9126]

ist die Gesamtheit der Merkmale und Merkmalswerte eines SW-Produkts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen.

2. QM - Definition

Qualitätsmanagement (quality management)

- aufeinander abgestimmte Tätigkeiten zum Leiten und Lenken einer Organisation bezüglich Qualität.
- Leiten und Lenken bezüglich Qualität umfassen üblicherweise das Festlegen der Qualitätspolitik und der Qualitätsziele, die Qualitätsplanung, die Qualitätslenkung, die Qualitätssicherung und die Qualitätsverbesserung.

Qualitätsmanagementsystem, QM-System (quality management system)

- Managementsystem zum Leiten und Lenken einer Organisation bezüglich der Qualität. (EN ISO 9000:2015)

2. QM - Definition

EN ISO 9001

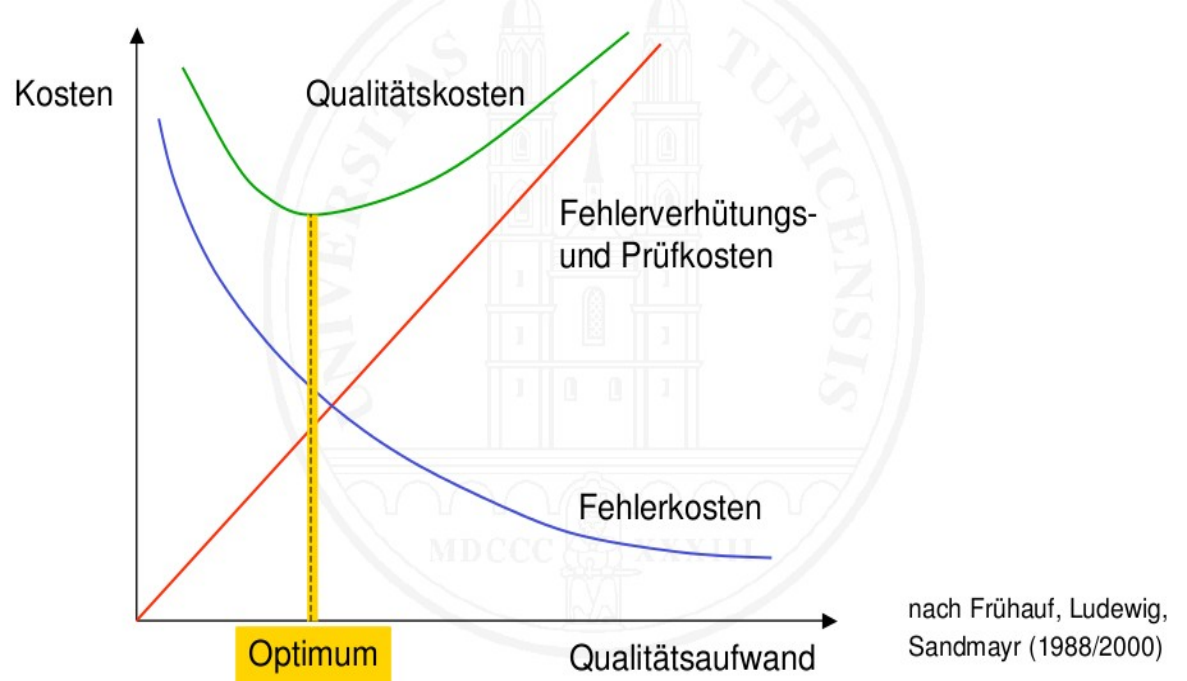
- legt die Mindestanforderungen an ein Qualitätsmanagementsystem (QM-System) fest, denen eine Organisation zu genügen hat, um Produkte und Dienstleistungen bereitstellen zu können, welche die Kundenerwartungen sowie allfällige behördliche Anforderungen erfüllen. Zugleich soll das Management einem stetigen Verbesserungsprozess unterliegen.

2. QM – Besonderheit / Kosten

Besonderheiten von Software-Qualitätsmanagement

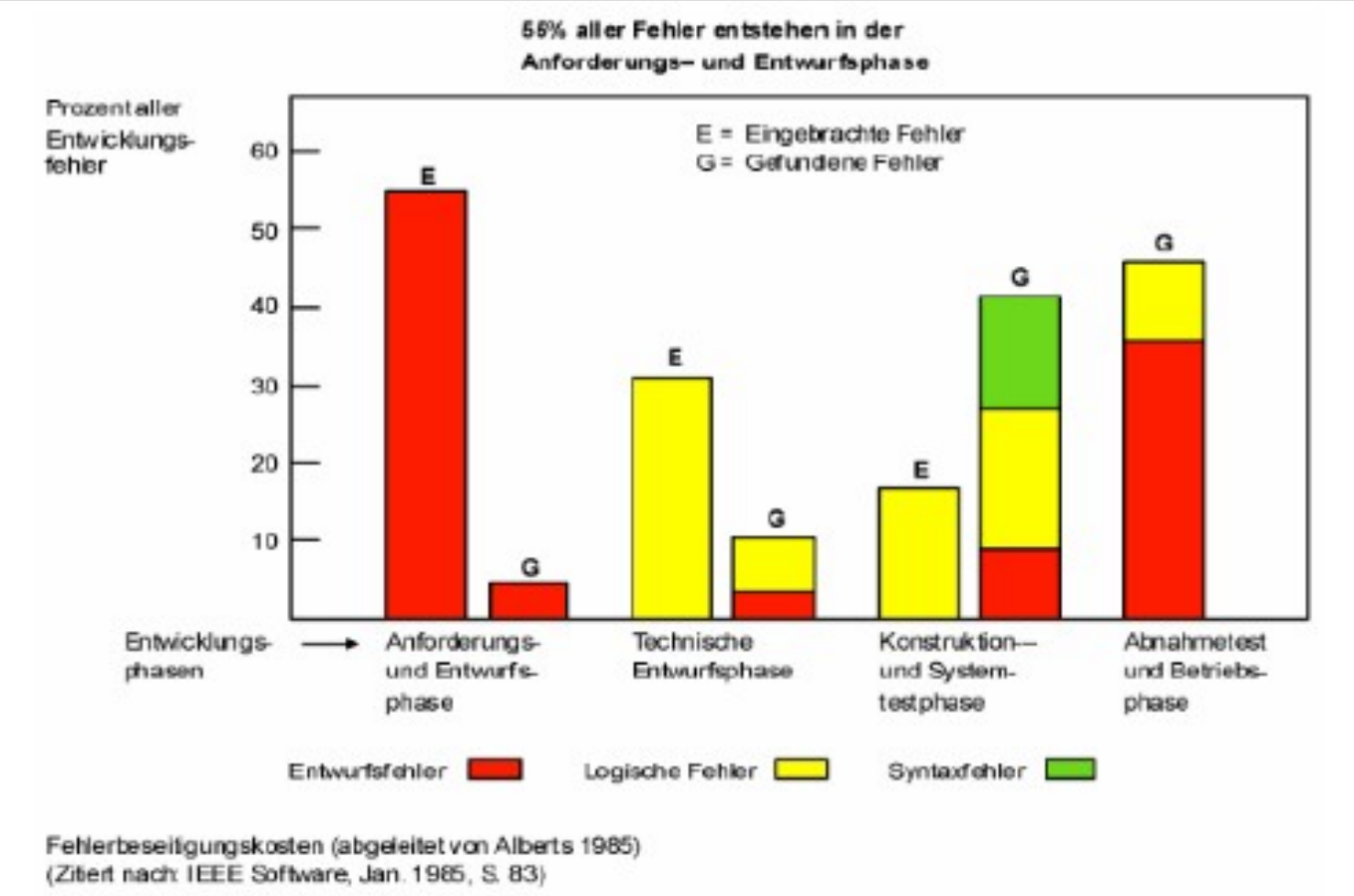
- Nur Entwicklung, keine Produktion
- Keine tradierten Standards
- Immateriell: schwierig zu messen und zu prüfen
- Spezifische Mess- und Prüfverfahren erforderlich
- Rückkopplung wird nur ansatzweise beherrscht

Kosten



2. QM – Besonderheit / Kosten

Fehleraufteilung pro Phase / Fehlerbeseitigungskosten

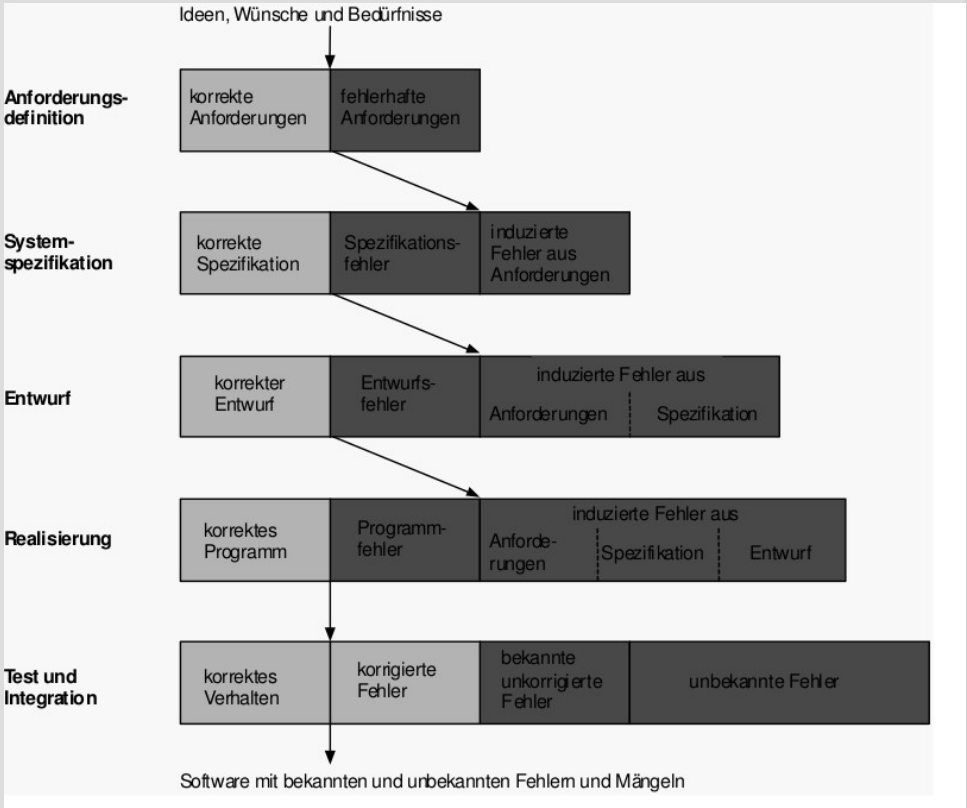


Fehlerbeseitigungskosten (abgeleitet von Alberts 1985)
(Zitiert nach: IEEE Software, Jan. 1985, S. 83)

2. QM – Besonderheit / Kosten

Prinzip der frühzeitigen Fehlerbehebung/Entdeckung

- Bei Abweichungen von den Anforderungen nach der Programmierung, muss nicht nur das Programm, sondern auch der Entwurf geändert werden
- Noch aufwendiger werden Modifikationen, wenn Fehler erst im Betrieb des fertiggestellten und freigegebenen Produktes festgestellt wurden
- Ein Fehler ist...
  - jede Abweichung von den Anforderungen des Auftraggebers
  - jede Inkonsistenz in den Anforderungen



2. QM - Grundsätze

1) Qualität muss erzeugt werden, sie kann nicht „erprüft“ werden
2) Qualität bezieht sich immer auf Produkte und auf Prozesse
3) Qualitätsverantwortung ist untrennbar verbunden mit Sach-, Termin-und Kostenverantwortung
4) Das Qualitätswesen erbringt Dienstleistungen und ist verantwortlich für die Ermittlung (Messung) der Qualität
5) Das Qualitätswesen muss einen unabhängigen Berichterstattungspfad haben, der bis zur Geschäftsleitung geht
6) Die Mitarbeiter müssen über die Qualität ihrer Arbeit orientiert werden

2. QM - TQM

Totales Qualitätsmanagement (TQM)

- Es ist möglich, Qualität ins Zentrum des unternehmerischen Handelns zurücken:

  Totales Qualitätsmanagement macht Qualität zum Unternehmensprinzip

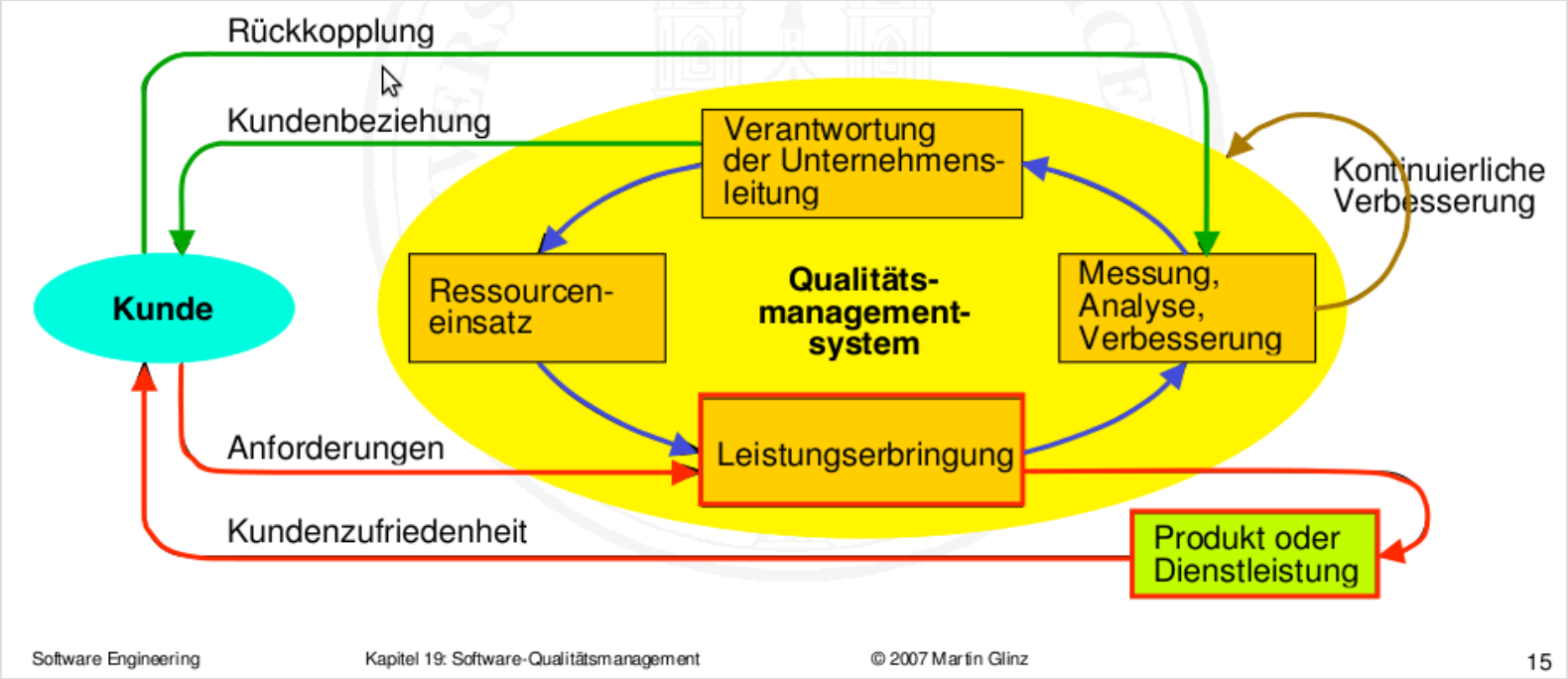
- Totales Qualitätsmanagement (TQM) – eine Führungsmethode, welche Kundenzufriedenheit als oberstes Unternehmensziel postuliert. Qualität wird in den Mittelpunkt gestellt, und alle Mitglieder des Unternehmens ins Qualitätsmanagement eingebunden.

- Alle übrigen Unternehmensziele werden vom Ziel der Kundenzufriedenheit und den damit verbundenen Qualitätsanforderungen abgeleitet.

2. QM - Prinzipien

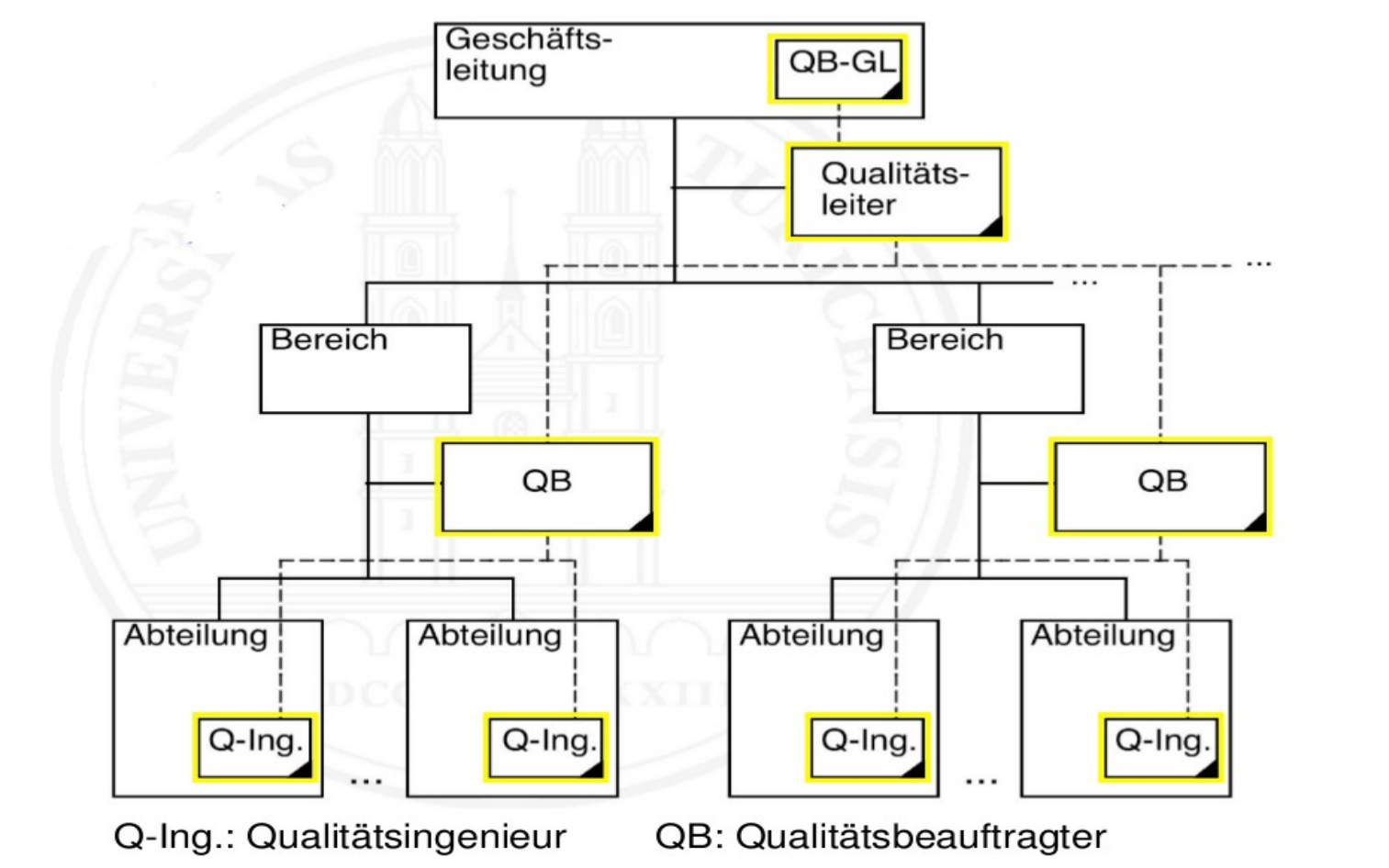
Totales Qualitätsmanagement (TQM)

- Orientiert an Selbstverantwortung aller Beteiligten
- Kundenzufriedenheit
- Prozessorientiert, systemischer Ansatz zur Realisierung
-



2. QM – Aufbauorganisation 1

Das Qualitätswesen – eine Sekundärorganisation mit den Fachleuten für Qualität

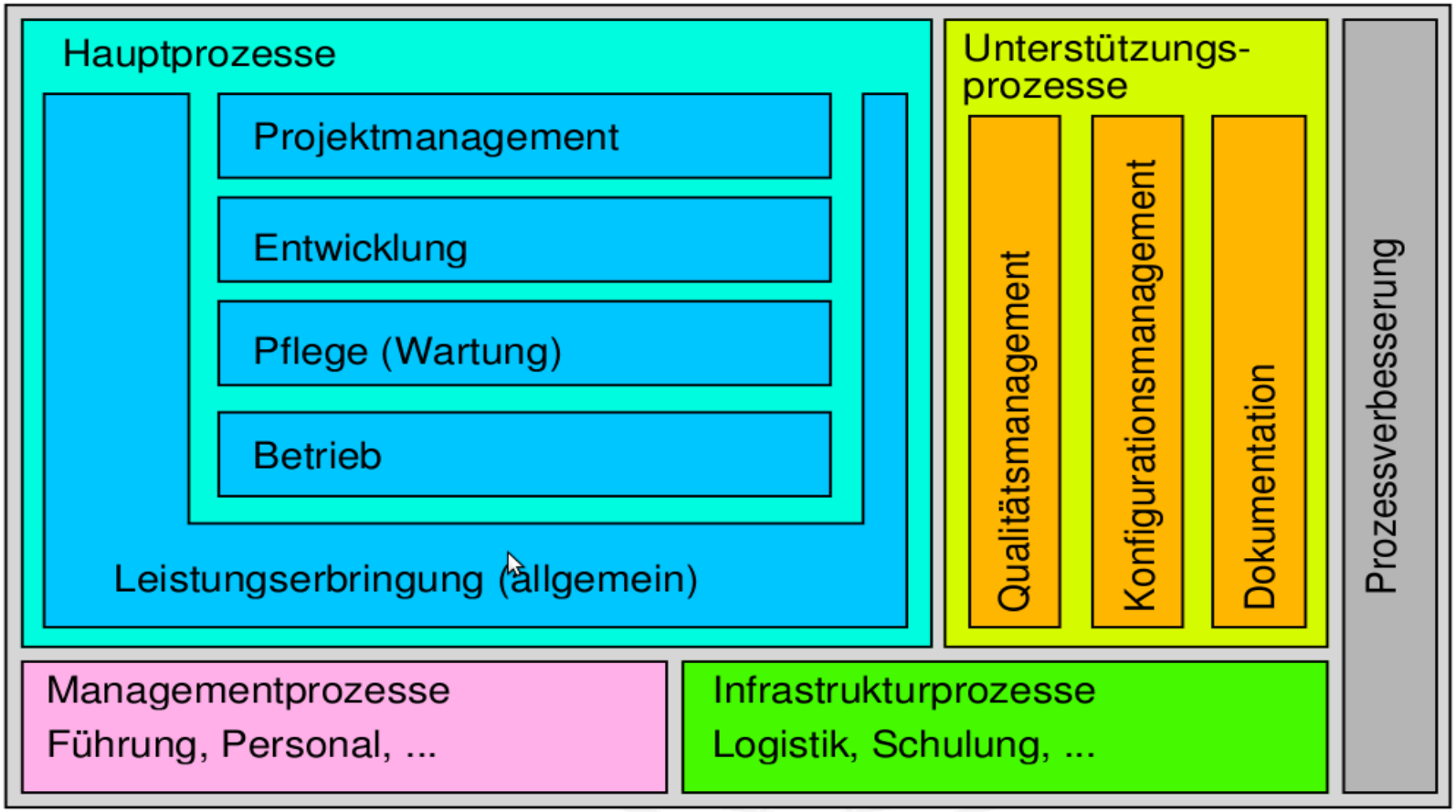


2. QM – Aufbauorganisation 2

Einbeziehung aller Mitarbeiterinnen und Mitarbeiter
Verankerung in der Primärorganisation

- Die Qualitätsfachleute bilden eine Sekundärorganisation im Unternehmen
- Diese Sekundärorganisation ...
  - hat das notwendige Fachwissen über alle Qualitätsbelange
  - erbringt Dienstleistungen im Bereich Qualität (z.B. Messung / Auswertung von Kenngrößen)
  - hat einen unabhängigen Berichtspfad für Qualitätsbelange bis hinauf in die Geschäftsleitung
  - ist verantwortlich für Pflege, Weiterentwicklung und Verbesserung des Qualitätsmanagementsystems

2. QM – Ablauforganisation 1



Prozessorientierte Ablauforganisation in einem Softwareunternehmen

2. QM – Ablauforganisation 2

Das Qualitätsmanagementsystem regelt alle qualitätsrelevanten

- Kompetenzen
- Verantwortlichkeiten
- Beziehungen

- Qualitätsaufgaben in die Unternehmensprozesse integriert
- Möglichst wenig Qualitätsaufgaben separat geregelt

2. QM –

- generell: planen – lenken – verbessern

2. QM – Verfahren 1

generell: planen – lenken – verbessern

Qualitätsplanung
Definition der Qualitätsziele: Das wollen wir erreichen!

Qualitätslenkung
konstruktiv: so müssen wir arbeiten!
analytisch: haben wir richtig gearbeitet? (Qualitätsprüfung)

Qualitätsverbesserung
Wie gut funktioniert was?
Wo gibt es Schwachstellen?
Das wollen wir verbessern!

2. QM – Verfahren 2

Qualitätsplanung (quality planning)

- Teil des Qualitätsmanagements, der auf das Festlegen der Qualitätsziele und der notwendigen Ausführungsprozesse sowie der zugehörigen Ressourcen zur Erfüllung der Qualitätsziele gerichtet ist. (ISO 9000:2000)
- Qualitätsziele bestimmen
- Kein Qualitätsmanagement ohne eine saubere, quantifizierte Spezifikation der Anforderungen.
- Qualitätsplanung heißt
  - Im Allgemeinen: Aufbau und Dokumentation des QM-Systems, allgemeine Qualitätsziele
  - Im Speziellen: Festlegung der Qualitätsziele für individuelle Projekte

2. QM – Verfahren 3

Qualitätslenkung (quality control)

- Teil des Qualitätsmanagements, der auf die Erfüllung von Qualitätsanforderungen gerichtet ist. (ISO 9000:2000)
- Konstruktive Maßnahmen (Lenkung präventiv)
- Analytische Maßnahmen (Qualitätsprüfung erkennend, nachträglich)
- Qualitätslenkung
  - Im Allgemeinen:
    Methoden, Sprachen, Werkzeuge
    Ausbildung
    Vereinheitlichung der Arbeitsweise
  - Im Speziellen: Maßnahmen der Projektführung zur Erreichung dergeplanten Qualität
-

2. QM – Verfahren 4

Qualitätslenkung - Konstruktive Maßnahmen

- Fehlerverhindernde / fehlervermeidende Prozesse definieren
- Prüf- und Korrekturverfahren in die Prozesse integrieren
- Prüfergebnisse zur Verbesserung des Prozesses verwenden
- Eine systematische, ingenieurmäßige Vorgehensweise, welche die Erreichung gegebener Qualitätsanforderungen garantiert, gibt es für Software bis heute nicht.
- Konstruktive Maßnahmen werden so weit als möglich eingesetzt, umdas generelle Qualitätsniveau zu heben
- Rigorose Qualitätsprüfung (und Behebung der festgestellten Mängel) während aller Phasen der Entwicklung ist heute das Mittel zur Sicherstellung der konkreten Qualitätsanforderungen an Software.

2. QM – Verfahren 5

Qualitätslenkung - Analytische Maßnahmen:
Qualitätsprüfung

- Prüfung der Produkte
  Zwischen- und Endergebnisse überprüfen
  Statische Prüfung
    - Review
    - Statische Analyse
    - Formale Programmverifikation
- Dynamische Prüfung
    - Test
    - Simulation
    - Prototypen
- Prüfung der Prozesse
    - Audits (systematische Inspektion eines QM-Systems durch Experten)
    - Prozessbeurteilung

2. QM – Verfahren 6

Qualitätsverbesserung (**quality improvement**)

- Teil des Qualitätsmanagements, der auf die Erhöhung der Fähigkeit zur Erfüllung von Qualitätsanforderungen gerichtet ist. (ISO 9000:2000)
- Behebung der bei der Produktprüfung gefundenen Qualitätsmängel
  - Notwendig zur Erreichung von Produktqualität
  - Häufig jedoch nur Symptombekämpfung
- Modifikationen im Entwicklungsprozess und im Qualitätsmanagementsystem aufgrund von
  - Auswertung von Fehlerursachen
  - Resultaten von Audits
  - Messungen
- Prozessverbesserung

2. QM – Verfahren 7

Qualitätssicherung (**quality assurance, QA**)

- Teil des Qualitätsmanagements, der auf das Erzeugen von Vertrauen darauf gerichtet ist, dass Qualitätsanforderungen erfüllt werden. (ISO 9000:2000)
- Regelmäßige Überprüfung der Wirksamkeit des Qualitätsmanagement-systems durch Experten in Audits
- Publikation von qualitätsrelevanten Messgrößen
- Dokumentation (und teilweise Offenlegung) der Prozesse und Qualitätsverfahren
- Zertifizierung des Qualitätsmanagementsystems
- Aktionsprogramme zur Verbesserung der Prozesse für die Entwicklung,Pflege und Verwaltung von Software Software Engineering

2. QM – Dokumentation

Qualitätshandbuch (**quality manual**)
dokumentiert das Qualitätsmanagementsystem

- **QM-Plan** (oder Qualitätsplan, quality plan)
  dokumentiert das QM für ein spezifisches Projekt oder Produkt
- **Anforderungsspezifikation** (requirements specification)
  dokumentiert die zu erfüllenden Anforderungen
- **Verfahrens- und Arbeitsanweisungen** (procedures)
  beschreiben die Durchführung von Prozessen und Verfahren im Detail
- **Leitfäden** (guidelines)
  geben Empfehlungen und Vorschläge zur Vorgehensweise
- **Aufzeichnungen** (records)
  weisen ausgeübte Tätigkeiten oder erzielte Ergebnisse nach

2. QM – Modelle 1

DIN EN ISO 9000 ff.

Orientierung bietet die DIN EN ISO 9000er Serie, die weltweit Anerkennung findet und der Zertifizierung dient. Sie gibt Normen vor, wie ein QM-System aufgebaut sein sollte. Die Normen sind allgemein formuliert und gelten branchenübergreifend. Der Leitfaden **ISO 9000-3 stellt allerdings eine Umsetzung der Formulierungen in eine Terminologie der Softwarebranche dar**. Jedoch wird stets nur beschrieben, dass etwas geregelt werden soll, nicht, wie.

- Wichtige Anforderungen der ISO 9000-3 sind: Festlegung der Forderungen des Auftraggebers, Planung der Entwicklung, Design und Implementierung, Test und Validierung, Wartung, Konfigurationsmanagement, Lenkung der Dokumente, Messungen, Werkzeuge und Techniken.
- Weitere elementare Punkte sind das Festhalten des QM-Systems in einem QM-Handbuch, die Gesamtverantwortung der Unternehmensleitung, die Akzeptanz durch alle Mitarbeiter, ausreichende Kommunikation und die Schaffung von Teilverantwortlichkeiten. Prinzipiell sind die Anforderungen ähnlich umfassend wie das EFQM-Modell (s. Was ist Qualität), das Grundlage des European Quality Awards ist und ebenfalls eine QM-Norm darstellt.
- Wird vom QM das gesamte Unternehmen mit allen Teilbereichen erfaßt, spricht man von einem TQM

2. QM – Modelle 2

- **EN ISO 9000**
  - Erläutert werden die Grundlagen für Qualitätsmanagementsysteme und die in der Normenreihe EN ISO 9000 ff. verwendeten Begriffe. Die europäische Norm ISO 9000:2000 wurde in drei offiziellen Fassungen in englischer, deutscher und französischer Sprache veröffentlicht. Auch der prozessorientierte Ansatz des Qualitätsmanagements wird erklärt, basierend auf dem nach William Edwards Deming benannten Demingkreis (engl. auch Deming Cycle oder PDCA).
  - Die ISO 9000:2000 wurde im Jahr 2005 überarbeitet, um einheitliche Begriffsdefinitionen für die Normen ISO 9001:2000 und ISO 19011:2002 erweitert und als ISO 9000:2005 im Dezember 2005 veröffentlicht.

EN ISO 9001

- EN ISO 9001 legt die Anforderungen an ein Qualitätsmanagementsystem (QM-System) für den Fall fest, dass eine Organisation ihre Fähigkeit darlegen muss, Produkte bereitzustellen, welche die Anforderungen der Kunden und allfällige behördliche Anforderungen erfüllen, und anstrebt, die Kundenzufriedenheit zu erhöhen.
- Diese Norm beschreibt modellhaft das gesamte Qualitätsmanagementsystem und ist Basis für ein umfassendes Qualitätsmanagementsystem.

2. QM – Modelle 3

Capability Maturity Model (CMM) 1/3

- Ein anderes Modell ist das Capability/Process Maturity Model (Carnegie Mellon Software Engineering Institute (SEI). Es ist ein aus 5 Stufen bestehendes Schema, in das sich TQM = Total Quality Management Unternehmen einordnen lassen. Die Stufen reichen von „Initial“ (= kaum Planung und Standards) bis „Optimierend“ (umfassendes QM) und zeigen die jeweils einzuleitenden Maßnahmen zur Erreichung der nächsten Stufe auf.
- Der Aufstieg um eine Stufe benötigt ca. 1 bis 3 Jahre, und es wird geschätzt, dass sich die Mehrheit aller Unternehmen auf den ersten beiden Stufen befindet.
- Anzustreben seien Reifegrade 2 bis 3, während Stufen 4 und 5 angesichts des Aufwandes für nur sehr wenige Unternehmen von Interesse seien

2. QM – Modelle 4

Capability Maturity Model (CMM) 2/3

CMMI beschreibt den Grad der Reife eines einzelnen Prozessgebiets durch so genannte „**Fähigkeitsgrade**“ (capability levels). Ein Fähigkeitsgrad bezeichnet den Grad der Institutionalisierung eines einzelnen Prozessgebiets. Die Fähigkeitsgrade sind (seit Version 1.3):

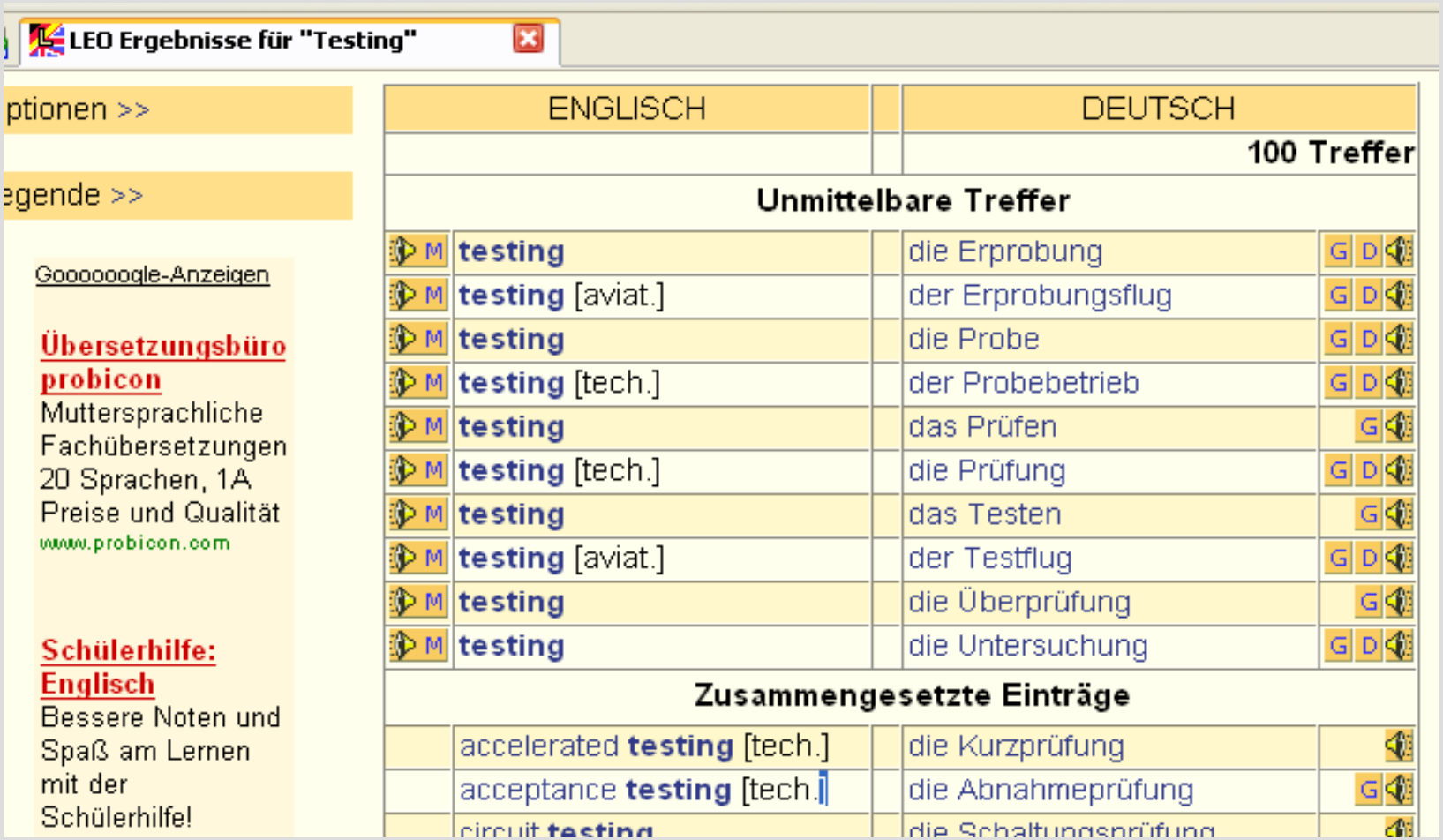
- 0 - Incomplete
  Die Arbeit wird so durchgeführt, dass die fachlichen Ziele (in CMMI "Specific Goals" genannt, z. B. bei der Projektplanung ein Projektplan) nicht erreicht werden.
- 1 - Performed
  Die Arbeit wird so durchgeführt, dass die fachlichen Ziele erreicht werden.
- 2 - Managed
  Die Arbeit wird geführt.
- 3 - Defined
  Die Arbeit wird mit Hilfe eines angepassten Standardprozesses durchgeführt und die Arbeitsweise verbessert.

2. QM – Modelle 5

Capability Maturity Model (CMM) 3/3

- Neben den Fähigkeitsgraden eines einzelnen Prozessgebiets definiert CMMI „**Reifegrade**“ (maturity levels). Ein Reifegrad umfasst eine Menge von Prozessgebieten, die mit dem zum Reifegrad korrespondierenden Fähigkeitsgrad etabliert sein müssen. Jeder Reifegrad ist ein Entwicklungsplateau in der Prozessverbesserung der Organisation. CMMI bietet damit eine Hilfe für die Verbesserung, indem es die Prozessgebiete bezüglich der Verbesserung priorisiert. Die Reifegrade sind:
- 1 - Initial Keine Anforderungen. Diesen Reifegrad hat jede Organisation automatisch.
- 2 - Managed Die Projekte werden geführt. Ein ähnliches Projekt kann erfolgreich wiederholt werden.
- 3 - Defined Die Projekte werden nach einem angepassten Standardprozess durchgeführt, und es gibt eine organisationsweite kontinuierliche Prozessverbesserung.
- 4 - Quantitatively Managed Es wird eine statistische Prozesskontrolle durchgeführt.
- 5 - Optimizing Die Arbeit und Arbeitsweise werden mit Hilfe einer statistischen Prozesskontrolle verbessert.

2.1 Testing



LEO Ergebnisse für "Testing"

ptionen >>

egende >>

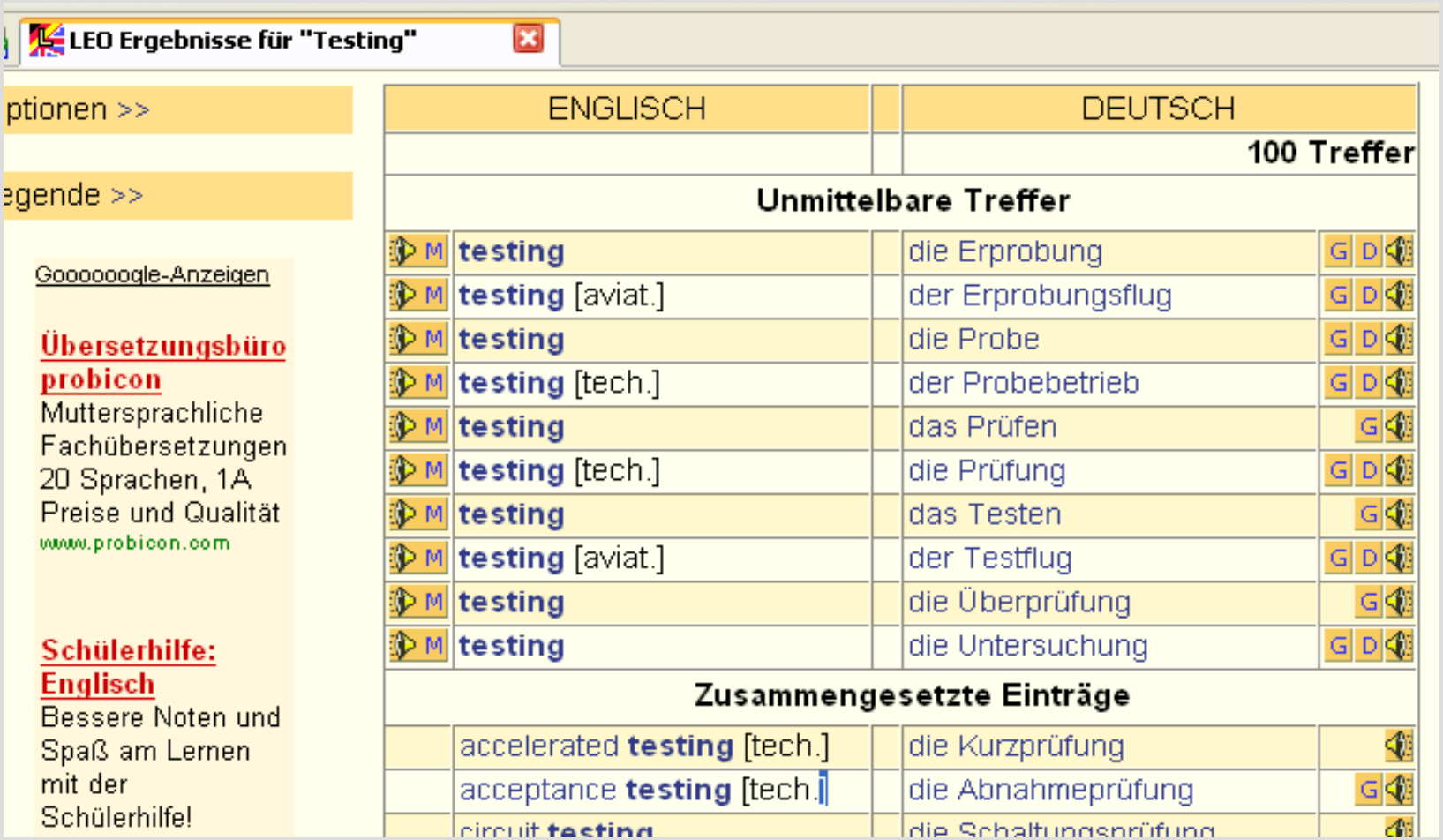
Gooooooogle-Anzeigen

Übersetzungsbüro probicon
Muttersprachliche Fachübersetzungen
20 Sprachen, 1A Preise und Qualität
www.probicon.com

Schülerhilfe: Englisch
Bessere Noten und Spaß am Lernen mit der Schülerhilfe!

ENGLISCH		DEUTSCH	
			100 Treffer
Unmittelbare Treffer			
M **testing**		die Erprobung	G D
M **testing** [aviat.]		der Erprobungsflug	G D
M **testing**		die Probe	G D
M **testing** [tech.]		der Probebetrieb	G D
M **testing**		das Prüfen	G
M **testing** [tech.]		die Prüfung	G D
M **testing**		das Testen	G
M **testing** [aviat.]		der Testflug	G D
M **testing**		die Überprüfung	G
M **testing**		die Untersuchung	G D
Zusammengesetzte Einträge			
accelerated **testing** [tech.]		die Kurzprüfung	
acceptance **testing** [tech.]		die Abnahmeprüfung	G
circuit **testing**		die Schaltungsprüfung	

2.1 Testing



LEO Ergebnisse für "Testing"

ptionen >>

egende >>

Gooooooogle-Anzeigen

Übersetzungsbüro probicon
Muttersprachliche Fachübersetzungen
20 Sprachen, 1A Preise und Qualität
www.probicon.com

Schülerhilfe: Englisch
Bessere Noten und Spaß am Lernen mit der Schülerhilfe!

ENGLISCH		DEUTSCH	
			100 Treffer
Unmittelbare Treffer			
M **testing**		die Erprobung	G D
M **testing** [aviat.]		der Erprobungsflug	G D
M **testing**		die Probe	G D
M **testing** [tech.]		der Probebetrieb	G D
M **testing**		das Prüfen	G
M **testing** [tech.]		die Prüfung	G D
M **testing**		das Testen	G
M **testing** [aviat.]		der Testflug	G D
M **testing**		die Überprüfung	G
M **testing**		die Untersuchung	G D
Zusammengesetzte Einträge			
accelerated **testing** [tech.]		die Kurzprüfung	
acceptance **testing** [tech.]		die Abnahmeprüfung	G
circuit **testing**		die Schaltungsprüfung	

2.1 Testing
Wieso

- Qualitätssicherung eines neu erstellten oder geänderten Softwareprodukts
- Erhöhung der Verlässigkeit, Senkung der Gefahr von Unglücksfällen
- Grundsätzlich: Vergleich und Dokumentation von Testergebnissen mit erwartendem Ergebnis

2.1 Testing
Kennzeichen

- Software Life Cycle
  - Definition zu welchem Zeitpunkt im SEP
  - Defintion des Automatisierungsgrades
- Kosten/Nutzen-Effekt
- Testwerkzeuge
  - Eigentwicklung
  - Lizenz/Kosten
  - Einarbeitung für effizienten und effektiven Einsatz
- Wartungsaufwand
  - Anpassungen an Änderungen

2.1 Testing
Automatisierungsgrad

- Automatisch
  - Toolunterstützung
  - Nightly build Integration
  - Set-up und tear-down Arbeiten
- Halbautomatisch
  - Toolunterstützung
  - Ablauf / Erweiterung / Überprüfung durch Testperson
- Manuell
  - Testabteilung / Testmanager
  - Testdrehbuch

2.1 Testing
Arten – BlackBox Tests

Black-Box-Test bezeichnet eine Methode des Software-Tests, bei der die Tests ohne Kenntnisse über die innere Funktionsweise des zu testenden Systems entwickelt werden. [Wikipedia]

- Spezifikation wird berücksichtigt
- Implementierung wird nicht berücksichtigt
- Ziel: Übereinstimmung eines Software-systems mit seiner Spezifikation überprüfen

2.1 Testing
Arten – Whitebox/Unit Tests

Der Begriff White-Box-Test (auch Glass-Box-Test) bezeichnet eine Methode des Software-Tests, bei der die Tests mit Kenntnissen über die innere Funktionsweise des zu testenden Systems entwickelt werden. [Wikipedia]

Im Gegensatz zum Black-Box-Test ist für diesen Test also ein Blick in den Quellcode gestattet, d.h. es wird am Code geprüft. [Wikipedia]

2.1 Testing
Arten – Übersicht

Testbezeichnung	Beschreibung	SEP-Phase
Regression Test	Test nach SW oder Config Änderungen. Kann durch Functional Tests oder Unit Tests gemacht werden.	Entwicklung
Unit-Test	Bestimmte SW-Einheiten (Units), also Methoden, werden getestet. = Whitebox Test	Entwicklung
Whitebox Test	Test mit Verständnis des Codes	Entwicklung
Integration Test	Integrierter Test von Gruppen von SW-Modulen	Test
Blackbox Test	Test ohne Kenntnis des Codes. SW System als „Blackbox“. Prüfung gegen Spezifikation	Test
System Test	Im Scope der Blackbox Tests. Wird auf einem kompletten, integrierten SW-System durchgeführt um zu prüfen, ob alle SW-Anforderungen erfüllt sind und die SW stabil ist.	Test
Usability Testing	Testen auf Benutzerfreundlichkeit. Kann auch schon am Prototyp gemacht werden.	Konzeption, Test
UAT	Verifizieren, dass eine Lösung die Benutzeranforderungen erfüllt.	Einführung
OAT	„conduct operational readiness (pre-release) of a product, service or system“	QM/Test

2.1 Testing
Arten –Integrationstests

- Aufeinander abgestimmte Reihe von Einzeltests
- voneinander abhängige Komponenten eines komplexen Systems im Zusammenspiel miteinander testen
- Unit-Test wurden bereits erfolgreich durchgeführt

2.1 Testing
Arten – Regressionstests

- v. lat. Regression = Rückschritt
- Nebenwirkungen von Modifikationen in bereits getesteten Teilen der Software aufzuspüren (Seiteneffekte)

- Soll-Ergebnis wird mit Ist-Ergebnis eines alten Testfalls verglichen

2.1 Testing
Arten – Systemstests

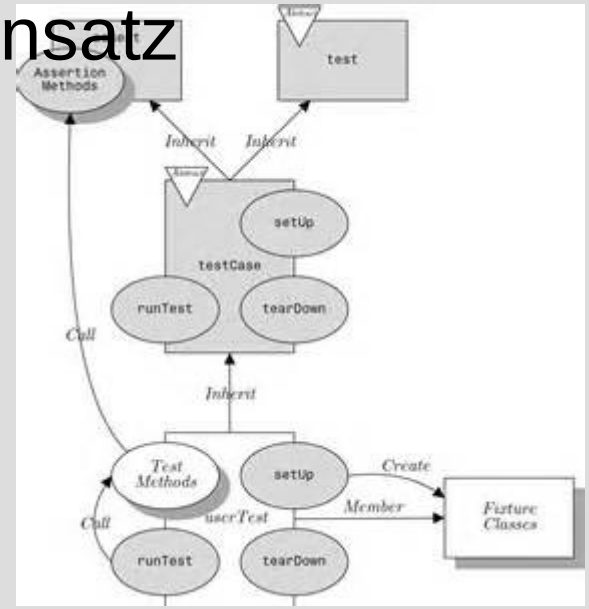
- Testphase, bei der das gesamte System gegen die Spezifikation getestet wird
- Test eines Gesamtsystems gegen seine Anforderungen.
- Durchführung durch die entwickelnde Organisation
- Funktionale Systemtests: funktionale Qualitätsmerkmale auf Korrektheit und Vollständigkeit
- Nicht funktionale Systemtests: nicht funkt. Qualitätsmerkmale wie Sicherheit, Benutzbarkeit, Interoperabilität, Zuverlässigkeit

2.1 Testing
Arten – Belastungstests

- Vorbereitung
  - Aufzeichnung des Testdurchlaufes
  - Verwendung von Scripts/Tools/Sniffern ...
- Testlauf/Durchführung des Tests
  - Der aufgezeichnete Test wird n-mal parallel durchgeführt
  - Max. Belastung wird ausgetestet
  - Verhalten bei max. Last untersucht (Broken Links, Response time ...)
- Auswertung
  - Graph. Auswertungen (z. Bsp. Mit JMeter)

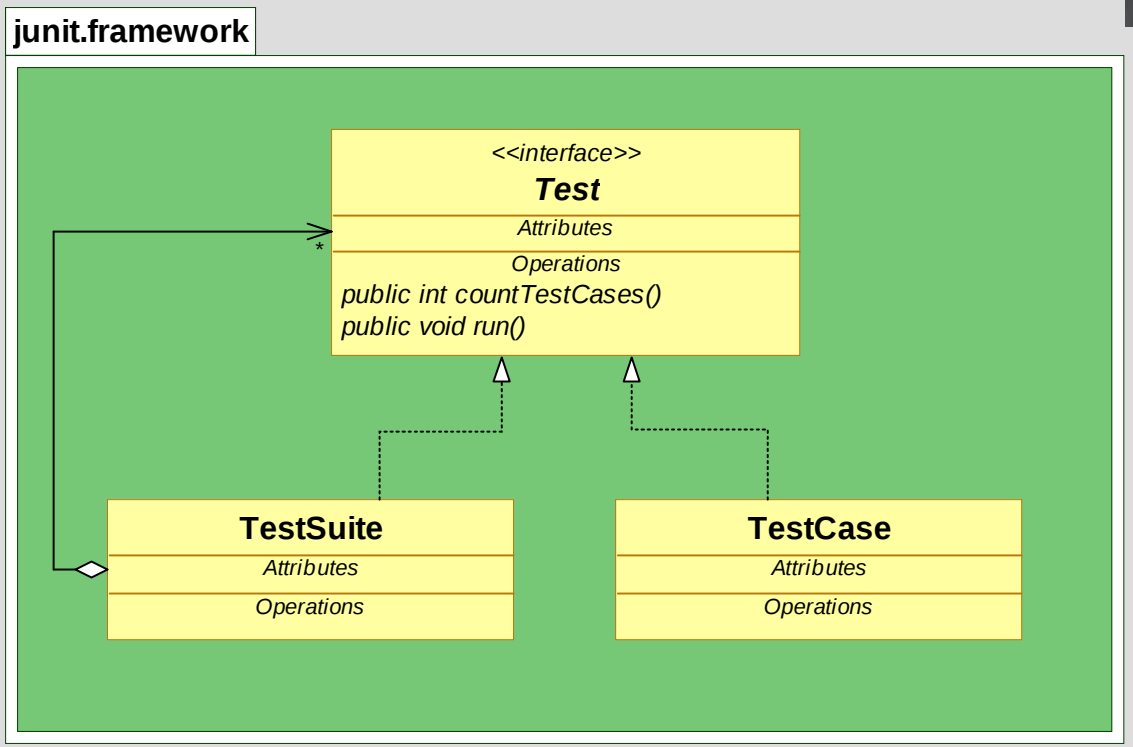
2.1 Testing
Konkrete Implementierung - JUnit

- Bieten einen Ansatz für Komponententests
- Werden von vielfältigen Frameworks in vielen Programmiersprachen unterstützt
- Entkoppeln Tests von der zu testenden Komponente
- Fordern und fördern den Test-First-Ansatz
- Ermöglichen automatisierte Tests



2.1 Testing
Konkrete Implementierung - JUnit

- „Rot/Grün Tests“



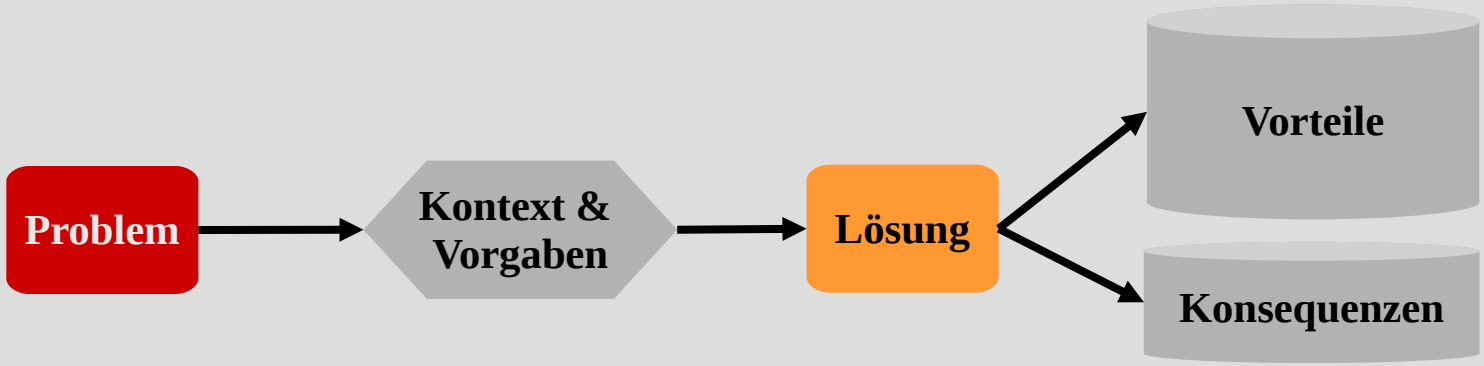
2.1 Testing

Konkrete Implementierung - JMeter

- Systemtests – Automatische Browsertests / Aufzeichnung von Browserinteraktionen (Blackbox-Tests)
- Performance / Lasttests : Aufgezeichnete Interaktionen können mehrfach parallel (mit Zeitverzögerungen) abgespielt werden

2.2 Pattern Allgemein

- Patterns sind bewährte Lösungen, die beschrieben werden, weil sie in der Praxis unabhängig voneinander an vielen Stellen zu finden sind
- Vor der Anwendung steht das Erkennen des Patterns
  - Man sieht, was man weiß!



2.2 Pattern
Geschichte

- Die “Gang of Four” (Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides.) behandelten nur Entwurfsmuster, d.h. Patterns, die in der Design-Phase genutzt werden (Design Patterns - Elements of Reusable Object-Oriented Software 1995).
- Martin Fowler publizierte eine Reihe von Analyse-Patterns (mit dem Untertitel “Reusable Object Models”).
- Coplien und Beck führten gleichzeitig sogenannte (sprachspezifische) “Idiome” ein.
- Die “Siemens-Gang” (Buschmann et.al.) fügte außerdem Architektur-Patterns zur Entwicklung von Software-Architekturen hinzu.
- Ebenfalls von Coplien stammen die ersten Organisations-Patterns.

2.2 Pattern
Klassifikation

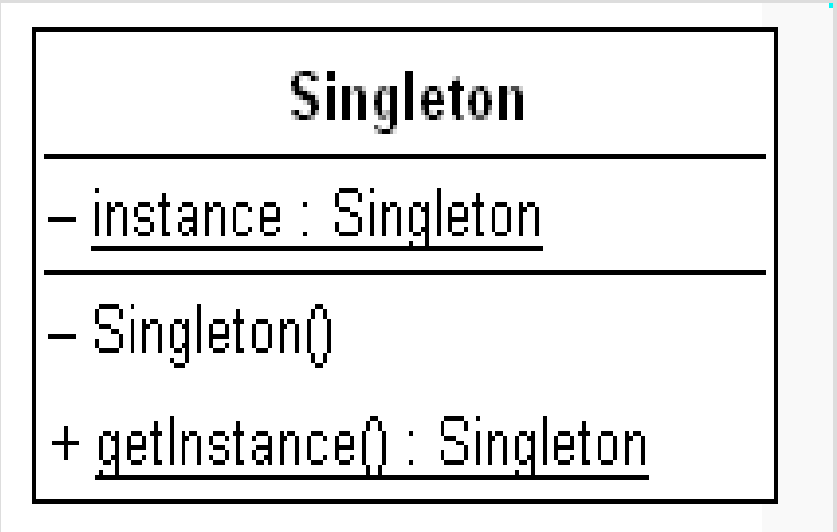
- Erzeugungsmuster
  - Erzeugungsmuster abstrahieren Objekterzeugungsprozesse
- Strukturmuster
  - Strukturmuster fassen Klassen und Objekte zu größeren Strukturen zusammen
- Verhaltensmuster
  - Verhaltensmuster beschreiben die Interaktion zwischen Objekten und komplexe Kontrollflüsse

2.2 Pattern
Pattern: Singleton

- Singleton-Pattern [Quelle: Wikipedia]

  - Kategorie der Erzeugungsmuster (engl. Creational Patterns)
  - Es existiert nur genau **ein** Objekt der Klasse
  - Es gibt einen globalen Zugriff auf dieses Objekt

  Verwendung:
  - Zentrales Protokoll-Objekt, das Ausgaben in eine Datei schreibt
  - Druckaufträge, die zu einem Drucker gesendet werden



2.2 Pattern
Pattern: Singleton

- Singleton-Pattern [Quelle: Wikipedia]

- Vorteile:
  - Das Muster bietet eine Verbesserung gegenüber globalen Variablen.
  - Das Einzelstück kann durch Unterklassenbildung spezialisiert werden.
  - Sollten später mehrere Objekte benötigt werden, ist eine Änderung leicht(er) möglich.
- Nachteile:
  - 'globale Variablen' für OO
  - Probleme mit Scope – z. Bsp. In Clusterlösungen / Classloader in Webapplications
  - Parallelverarbeitung – es darf nur eine Instanz erzeugt werden (thread-save)
  - Ressourcenfreigabe ist schwierig

2.2 Pattern
Pattern: Singleton

```
public class Singleton
{
    private static Singleton oSingletonInstance = null;

    /**
     * Default-Konstruktor, der nicht außerhalb dieser Klasse
     * aufgerufen werden kann
     */
    private Singleton()
    {
        // Dieser Konstruktor kann nicht verwendet werden
    }

    /**
     * Statische Methode, liefert die einzige Instanz dieser
     * Klasse zurück
     */
    public static Singleton getInstance()
    {
        if (oSingletonInstance == null)
        {
            oSingletonInstance = new Singleton();
        }
        return oSingletonInstance;
```

Klassische Singleton-Implementierung
Nicht Thread-safe!

2.2 Pattern
Pattern: Singleton

```
public class Singleton
{
    private static Singleton oSingletonInstance = new Singeton();

    /**
     * Default-Konstruktor, der nicht außerhalb dieser Klasse
     * aufgerufen werden kann
     */
    private Singleton()
    {
        // Dieser Konstruktor kann nicht verwendet werden
    }

    /**
     * Statische Methode, liefert die einzige Instanz dieser
     * Klasse zurück
     */
    public static Singleton getInstance()
    {
        return oSingletonInstance;
    }
}
```

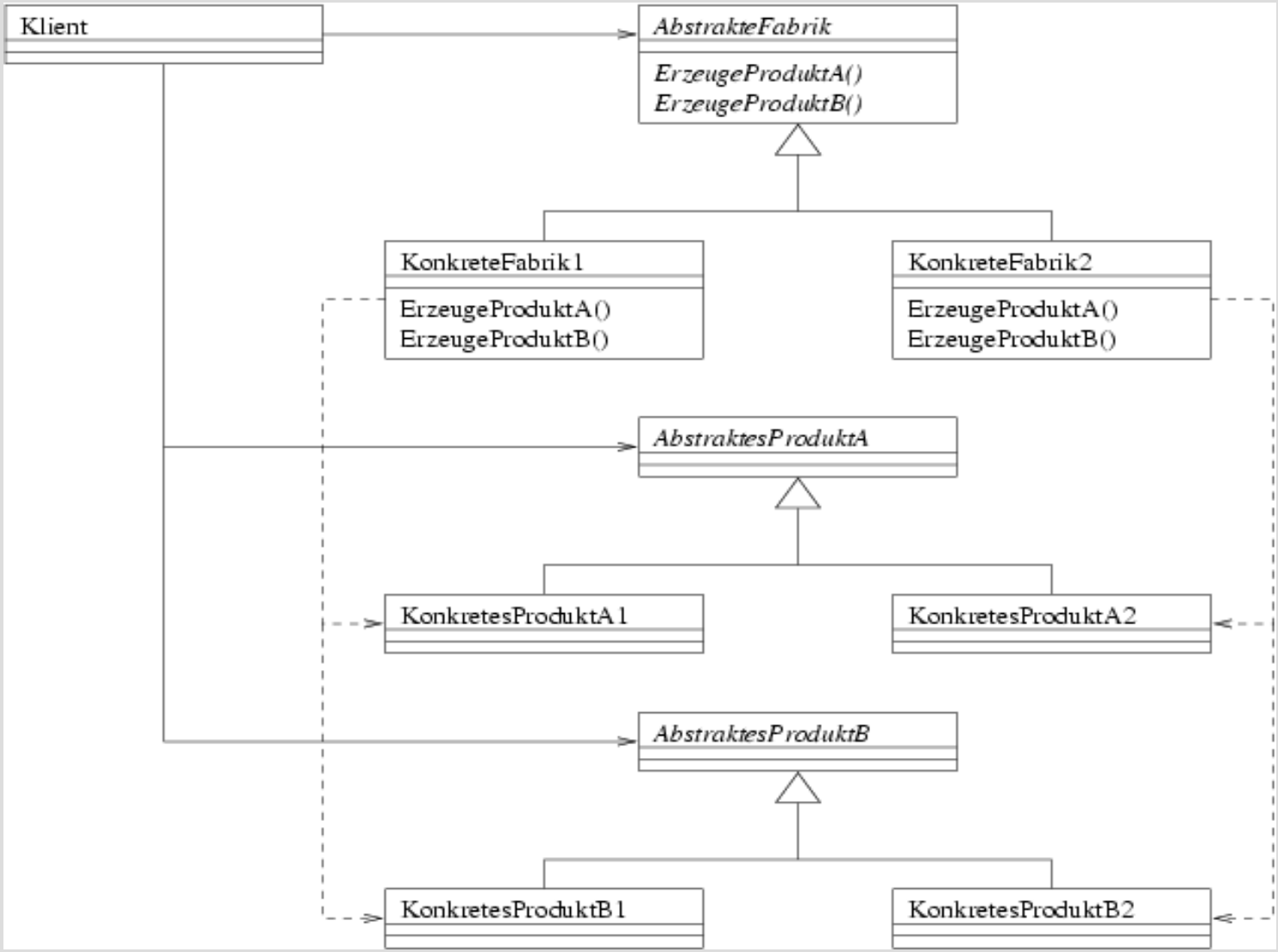
Thread-safe, da beim 2. Zugriff durch die JVM eine Instanz angelegt wird.

2.2 Pattern
Pattern: Factory

- Factory
  - Kategorie der Erzeugungsmuster (Creational Patterns)
  - Es stellt eine Schnittstelle zur Erzeugung einer Familie von Objekten bereit
  - Die konkreten Klassen der zu erzeugenden Objekte werden dabei nicht festgelegt (aus Client/Anwendersicht)
- Vorteile:
  - Konkrete Klassen werden isoliert (z. Bsp. Datenbankzugriffsklassen)
  - Austausch von Produktfamilien ist einfach möglich
- Nachteile:
  - Neue Produktarten lassen sich schwer hinzufügen, da in allen konkreten Fabriken Änderungen vorzunehmen sind.

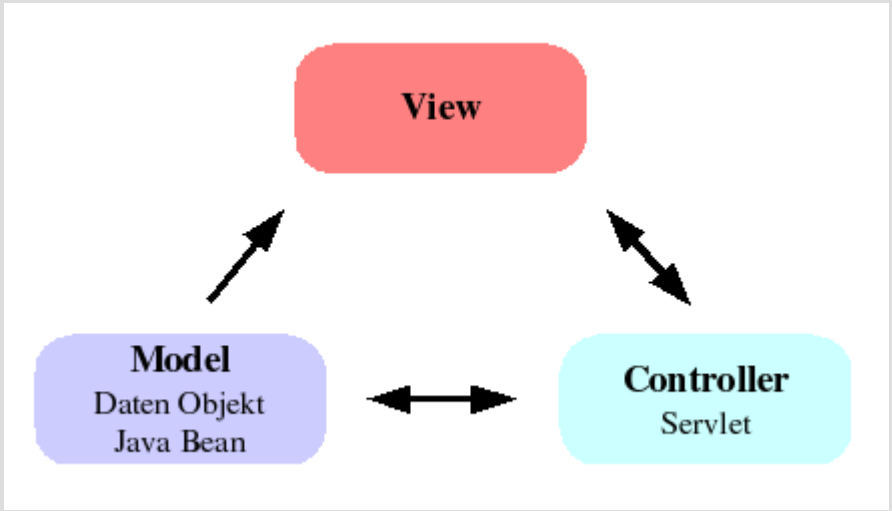
2.2 Pattern

Pattern:Factory



2.2 Pattern
Pattern: MVC

- Model, View Controller
  - Entwurfsmuster für die UI-Programmierung
  - Trennung in Schichten
  - Model und Controller sind teilweise nur 'schwach' getrennt



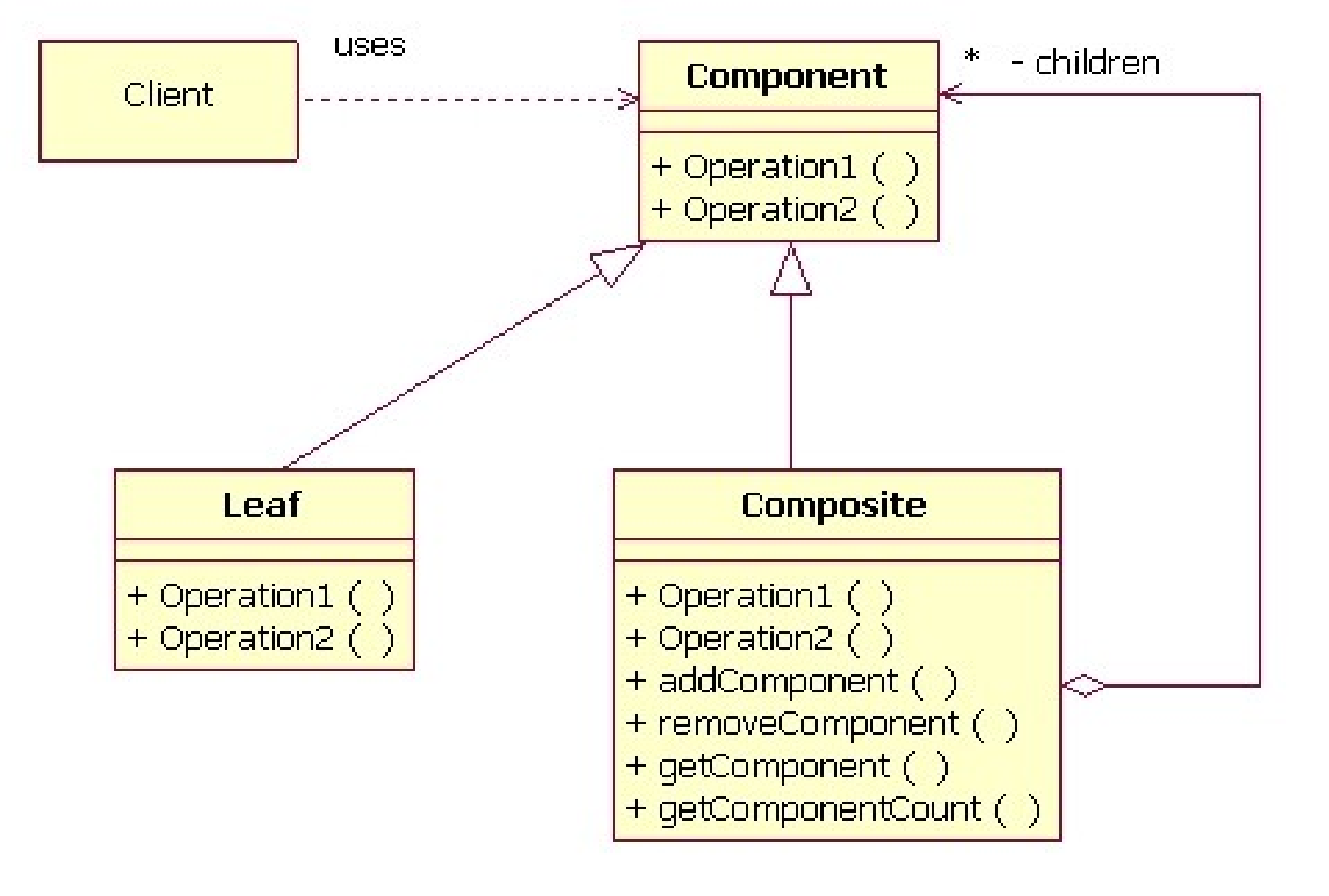
2.2 Pattern

Pattern: Composite

- Ein Composite enthält folgende Bestandteile:
  - Eine (oft abstrakte) Basisklasse, die sowohl zusammengesetzte als auch elementare Objekte repräsentiert.
  - Diese Basisklasse wird auch als "Component" bezeichnet.
  - Alle elementaren Objekte sind aus dieser Basisklasse abgeleitet.
  - Daraus abgeleitet gibt es mindestens eine Containerklasse, die in der Lage ist, eine Menge von Objekten der Basisklasse aufzunehmen.

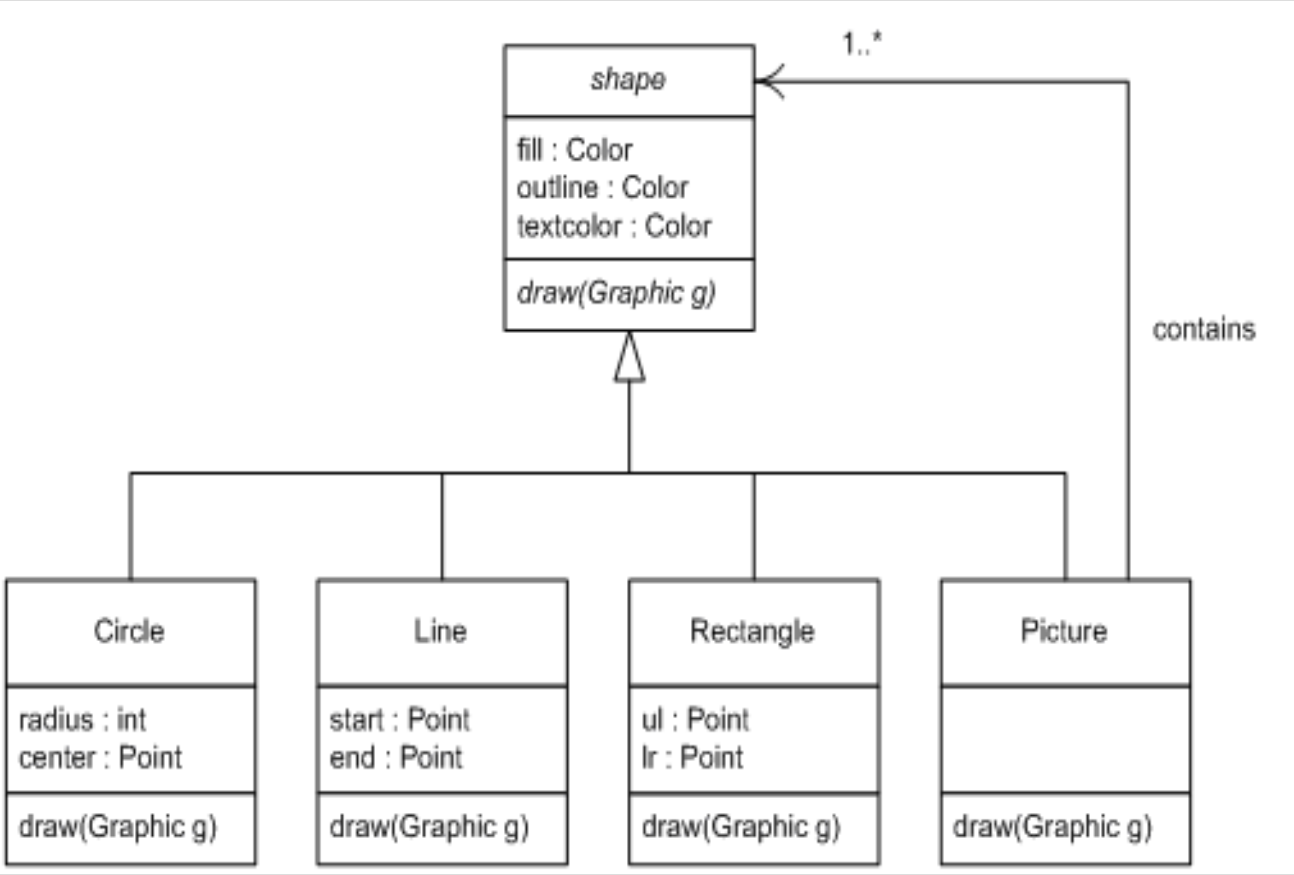
2.2 Pattern

Pattern: Composite - Beispiel



2.2 Pattern

Pattern: Composite – Beispiel II

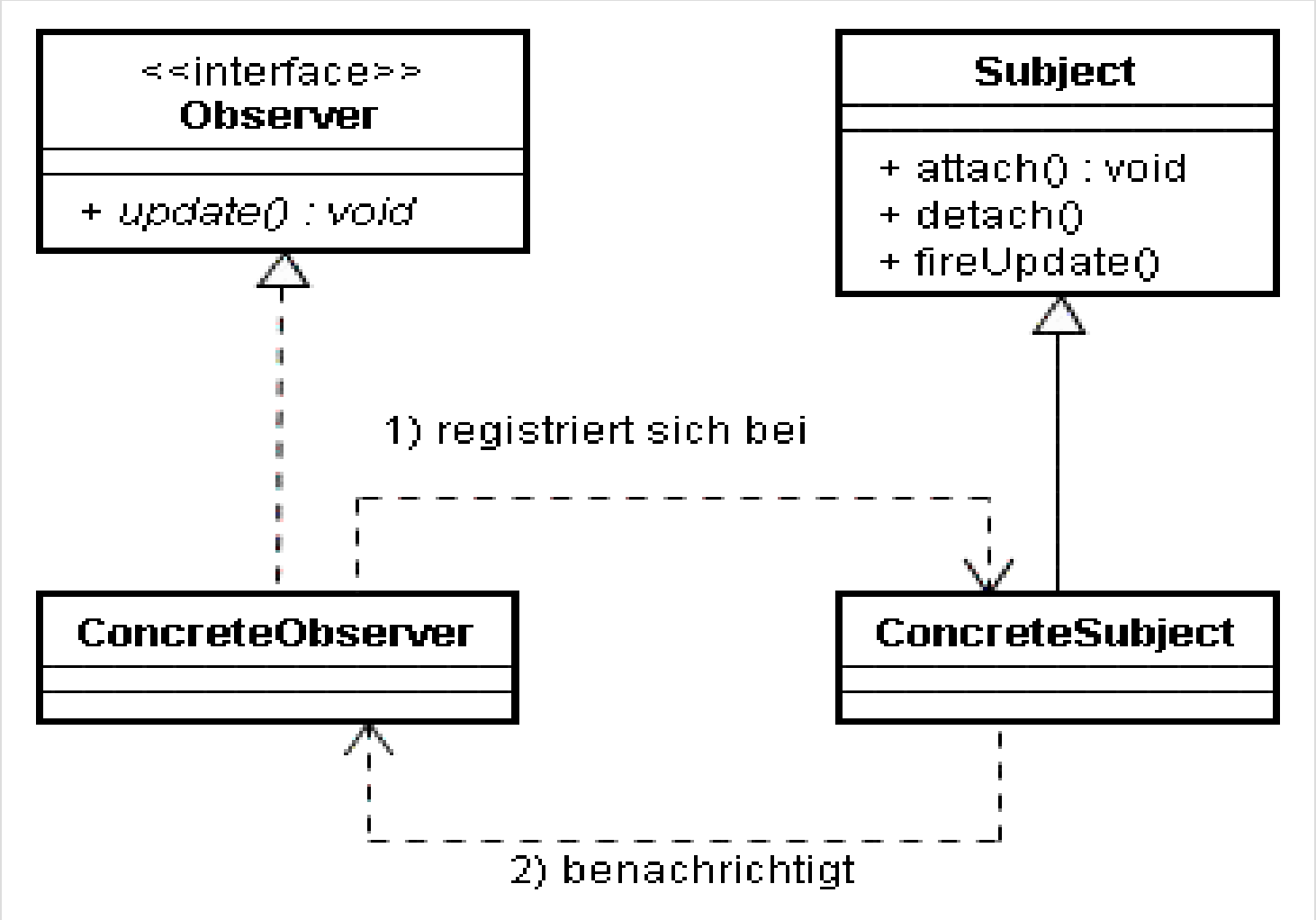


2.2 Pattern
Pattern: Observer

- Ein Observer ist ein Design-Pattern, das eine Beziehung zwischen einem Subject und seinen Beobachtern aufbaut.
  - Als Subject wird dabei ein Objekt bezeichnet, dessen Zustandsänderung für andere Objekte interessant ist.
  - Als Beobachter werden die Objekte bezeichnet, die von Zustandsänderungen des Subjekts abhängig sind; deren Zustand also dem Zustand des Subjekts konsistent folgen muss.

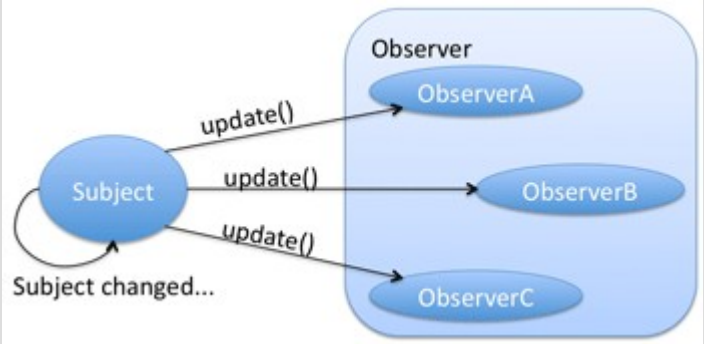
2.2 Pattern

Pattern: Observer - Beispiel



2.2 Pattern

Pattern: Observer – Beispiel II



w.philipphauer.de/study/se/design-pattern/observer.php

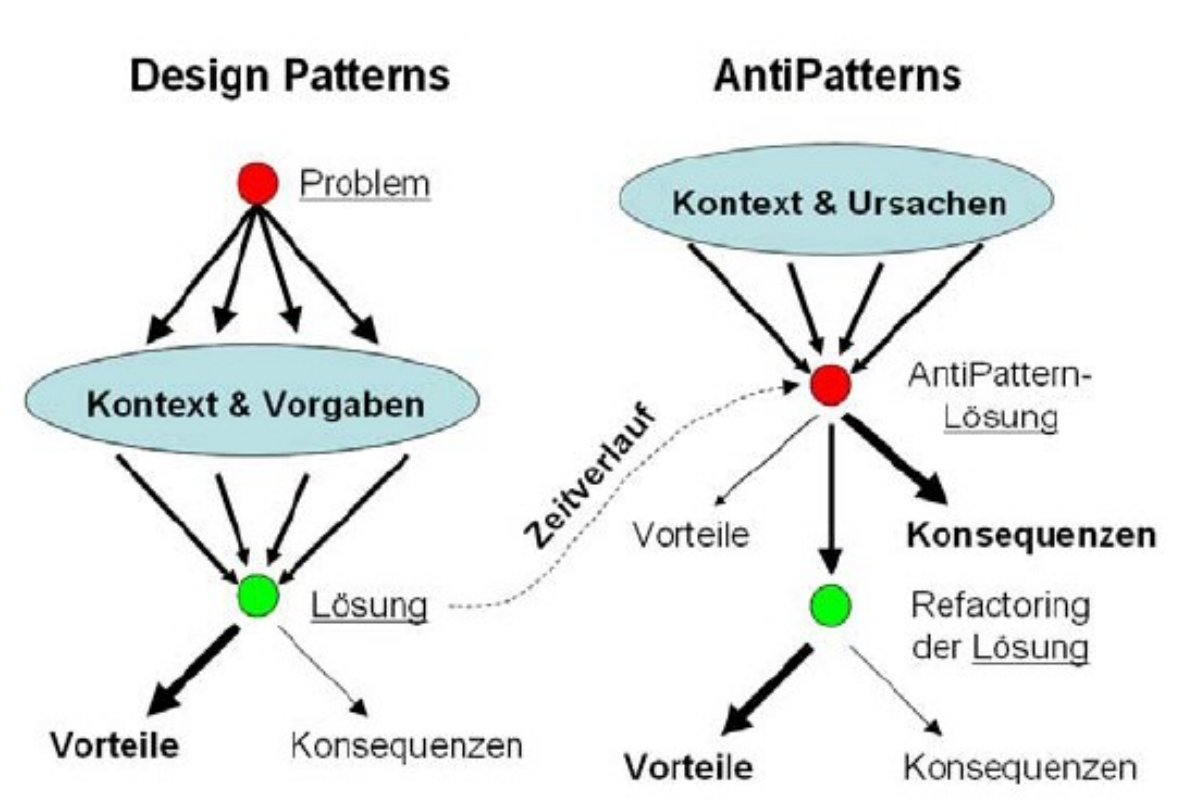
2.2 Antipattern
Probleme mit Design-Patterns

- Anwendbarkeit oft schwierig zu entscheiden, da sehr abstrakt
- Unangemessener und falscher Einsatz

- Weitere Gründe für negative Konsequenzen von Lösungen:
  - Übereilte Entscheidungen führen zu schlechtem Design.
  - Gleichgültigkeit gegenüber der Qualität der Lösung.
  - Engstirnigkeit. Weigerung, allgemein anerkannte Praktiken zu verwenden.
  - Faulheit führt zu der Lösung mit dem geringsten Aufwand.
  - Gier führt zu einem Übermaß an Detailliertheit und Komplexität.
  - Ignoranz führt zu fehlendem Verständnis.
  - Stolz beeinträchtigt Wiederverwendung von Komponenten, die jemand anderer bereits entwickelt hat.

2.2 Antipattern

- AntiPatterns fokussieren die Probleme, die sich aus der Anwendung bestimmter Lösungsmuster ergeben.
- AntiPatterns bieten Lösungen für fehlerhafte Lösungen...



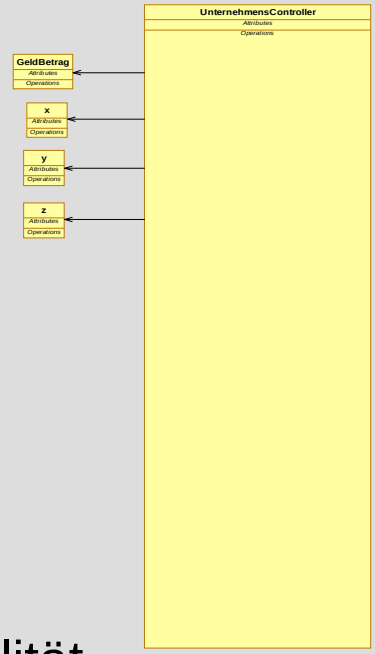
2.2 Antipattern
Antipattern: Blob

- Problem:
  - Eine einzelne Klasse (der “Blob”) monopolisiert die Funktionalität!
  - “...Und das ist das eigentliche Herzstück unserer Architektur!”

- Typische Ursachen:
  - Fehlende (nicht nur: objekt-orientierte) Architektur!
  - Mangelhafte Durchsetzung der Architektur!
  - Die Architektur folgt (implizit) der (prozeduralen) Anforderungs-spezifikation!

2.2 Antipattern

Antipattern: Blob



Lösung:

- Aufspalten des Blob:
  - Identifikation zusammengehörender Teilfunktionalität.
  - Verschiebung der funktionalen Blöcke an ihren “logischen Ort”.
- Entkoppeln des Blob:
  - Verschieben bzw. Entfernen von Beziehungen zu anderen Klassen, die durch die Verschiebung der Funktionalität obsolet geworden sind.
- Auslagern transienter Funktionalitäten in Utility-Klassen.

2.2 Pattern / Antipattern

Antipattern: Lava Flow

- Problem:
  - Der Code ist mit ungenutzten Fragmenten (“Dead Code”) durchsetzt, die aus früheren Entwicklungszweigen “übriggeblieben” sind.
    - Erhöhung der Komplexität ➔ Schlechtere Wartbarkeit
    - Erhöhung des Ressourcenverbrauchs (Ladezeit, Speicherbedarf)
    - Probleme mit der Dokumentation
  - “Wird wohl nicht mehr gebraucht, aber ich bin mir nicht sicher...”

- Typische Ursachen:
  - Fehlende Architektur!
  - Fehlendes / nicht durchgesetztes Konfigurationsmanagement!
  - Unkontrollierte Weiterentwicklung der Architektur / “gewachsenes” Design!
  - Fehlendes Refactoring

2.2 Antipattern

Antipattern: Lava Flow

- Lösung:
  - Konsistente und praktizierte Architektur.
  - Konfigurationsmanagment.
  - Stabile Schnittstellen zwischen den Software-Modulen.
  - Oft nützlich:
    Versionsmanagement-Systeme wie SCCS, PVCS, VSS...

2.2 Antipattern
Antipattern: Golden Hammer

- Problem:
  - Eine wohlbekannte Technik wird als Patentlösung für alle Probleme genutzt - auch, wo es nicht paßt!
    - Die Architektur wird durch das Werkzeug bestimmt...
    - Machbarkeit = Machbarkeit mit diesem Werkzeug...
    - Abhängigkeit von einem Lieferanten!
  - “Unsere Datenbank ist unsere Architektur!”
    - “Ich habe nur einen Hammer, und deshalb ist alles andere ein Nagel...”

- Typische Ursachen:
  - Hohe Investitionen in ein bestimmtes Werkzeug...
  - Vorherige Erfolge in der Anwendung des Werkzeuges...
  - Die Anwendung des Werkzeuges wird “von oben” “empfohlen”...

2.2 Antipattern
Antipattern: Golden Hammer

- Lösung:
  - Die Evaluation neuer Technologien sollte integrale Aufgabe der Enwicklungsabteilung sein!
  - Orientierung auf offene Standards!
  - Aktive Investition in die Fortbildung der Mitarbeiter...
  - Oft hilfreich:
    Anwerben von Mitarbeitern mit unterschiedlichen technologischen (und ggf. auch fachlichen) Hintergründen...

2.2 Antipattern

Antipattern: Golden Hammer

2.3 Refactoring

Wenn es stinkt, wickle es.

2.3 Refactoring
Definition

- Eine Software umstrukturieren, ohne ihr beobachtetes Verhalten zu ändern, indem man eine Reihe von Refaktorisierungen anwendet
- Eine Änderung an der internen Struktur einer Software, um sie leichter verständlich zu machen und einfacher zu verändern, ohne ihr beobachtetes Verhalten zu ändern

Refactoring ist mehr als nur bereinigen von Quellcode. Es bedient sich zur Bereinigung ganz konkreter Techniken

2.3 Refactoring
Zeitpunkt im SEP

- Wo steht Refactoring im Software-Entwicklungsprozess?
  - Metapher der zwei Hüte (nach Kent Beck)
    - Alternierend wird ständig zwischen Funktionalität hinzufügen und refactoring abgewechselt
  - Tip 4 aus dem Buch *Der pragmatische Programmierer* [Hun03] besagt: Akzeptieren Sie keine zerbrochenen Fensterscheiben. … Reparieren Sie alles, sobald es entdeckt wird.“ Hierzu kann Refactoring ideal eingesetzt werden
  - Ebenfalls aus [Hun03] ist Tip 47: „Refaktorisieren Sie früh und häufig“

- Refactoring sollte integraler Bestandteil des Entwicklungsprozesses sein

2.3 Refactoring
Vorgehensweise

- Wie refaktorisiert man?
  - Schreiben Sie zunächst für den Quellcode den Sie anfassen umfassende Unit-Tests
    - Da sich das Verhalten der Software nicht ändern soll, können Vergleichswerte aus Durchläufen der bestehenden Software ermittelt werden
    - Beachten Sie auch „was nicht passieren soll“ bei den Tests
  - Testen Sie, die Tests müssen alle erfolgreich durchlaufen
  - Analysieren Sie wie die Software verbessert werden soll
  - Fügen Sie weitere passende Tests ein
  - Testen Sie, die Tests für die Änderungen müssen fehlschlagen
  - Führen Sie die geplante Änderung durch
  - Testen Sie, die Tests müssen wieder alle erfolgreich durchlaufen

2.3 Refactoring
Arten

- Welche Arten von einfachem Refactoring gibt es?
  - Methoden zusammenstellen
    - Typisches Bsp.: Methode extrahieren
  - Eigenschaften zwischen Objekten (Klassen) verschieben
    - Typisches Bsp.: Methode verschieben
  - Daten organisieren
    - Typisches Bsp.: Wert durch Objekt ersetzen
  - Bedingte Ausdrücke vereinfachen
    - Typisches Bsp.: Bedingung zerlegen
  - Methodenaufrufe vereinfachen
    - Typisches Bsp.: Parameter ergänzen
  - Umgang mit der Generalisierung
    - Typisches Bsp.: Feld nach oben verschieben

2.3 Code Smells

Allgemein I

- Die nichtfunktionalen Anforderungen an die Software werden im wesentlichen durch Softwaredesign realisiert.
- Die Qualität des Designs lässt sich nicht direkt messen.
- Gutes Design ist wie ein gut geschriebener Text, wenn man ihn liest, gefällt einem der Stil.
- Bei geschriebenen Texten gibt es viele Stilbrüche die man vermeiden sollte, trotzdem kann ein Text ohne solche Stilbrüche einen schlechten Stil haben, während Text mit Stilbrüchen trotzdem gefallen kann.
- CodeSmells entsprechen den Stilbrüchen in Texten.

2.3 Code Smells

Allgemein II

- CodeSmells stellen einen Indikator für schlechtes Design dar.
- CodeSmells liefern nur sehr begrenzt Hinweise auf die Ursachen des schlechten Designs.

2.3 Code Smells
Ursachen

- Spekulative Allgemeinheit
- Temporäre Felder
- Nachrichtenketten
- Vermittler
- Unangebrachte Intimität
- Unvollständige Bibliotheksklassen
- Datenklassen
- Ausgeschlagenes Erbe
- Kommentare

2.3 Code Smells

Ursache: Copy-Paste

- „Nummer Eins der Gestanksparade“ [Fowler 1]
- Entsteht aus *Copy-Paste-Sessions*
- Änderungen müssen in allen Kopien durchgeführt werden
- Fehler treten gleich im Rudel auf
- Entsteht sowohl klassen- wie auch methodenübergreifend
- Nicht immer leicht zu erkennen

Zusammenfassend lässt sich sagen:

- Duplizierter Code kann Indiz sein sowohl für schlechtes Methodendesign, wie auch für schlechtes Klassendesign
- Wartungs- und Weiterentwicklungskosten steigen überproportional
- Bei der Weiterentwicklung werden oft neue Fehler erzeugt, da man den Überblick über den duplizierten Code verliert

2.3 Code Smells

Ursache: Lange Methoden

- Lange Prozeduren sind schwer zu verstehen
- Lange Methoden sind schwer zu warten
- Entwickler scheuen sich lange Methoden anzufassen
- Fehler sind in langen Methoden nur schwer zu lokalisieren und oft noch schwerer zu beheben

Zusammenfassend lässt sich sagen:

- Lange Methoden sind ein Indiz für schlechtes Design
- Wartungs- und Weiterentwicklungskosten steigen überproportional an
- Bei der Weiterentwicklung entsteht oft duplizierter Code, da man den nicht-verstandenen Teil nicht im Original anfassen möchte

2.3 Code Smells

Ursache: Große Klassen

- Große Klassen sind ein indirekter Indikator für andere übel riechende Ursachen
- Klassen die zuviel zu tun haben, enthalten oft zu viele Instanzvariablen
  => duplizierter Code ist nicht weit entfernt
- Große Klassen enthalten oft Methoden (=Lösungen) die man dort nicht vermutet
  => Wiederverwendung wird erschwert weil man nicht die schon existierende passende Lösung findet
  => Duplizierter Code, ohne Copy-Paste

2.3 Code Smells

Ursache: Divergierende Änderung & Schrotkugeln

- Zwei Seiten derselben Medaille
- Divergierende Änderungen liegen vor, wenn mehrere einfache fein granulare Anforderungsänderungen immer wieder zu Änderungen ein und derselben Stelle/Klasse führen
- Schrotkugeln liegen vor, wenn eine einfache fein granulare Anforderungsänderung zu Änderungen an mehreren Stellen führt.

→ Fazit:

- Indiz für schlechtes Klassendesign
- Weiterentwicklung ist teuer

2.3 Code Smells

Ursache: Neid

- Liegt vor, wenn eine Methode mehr an einer fremden Klasse interessiert ist, als an der eigenen
- Offensichtlich liegt die Methode in der falschen Klasse

→ Fazit:

- Wiederverwendung wird erschwert, weil man nicht die schon existierende bzw. passende Lösung findet
  => Duplizierter Code, ohne Copy-Paste

Quellen

- Qualitätsmanagement in der Software-Entwicklung
  Prof. Dr. Thomas Kudraß / HTWK Leipzig
  Seminar PC-Ware AG
- Martin Glinz, Harald Gall
  Software Engineering / Kapitel 19 / Software-Qualitätsmanagement
- Qualitätsmanagement für die Softwarebranche (wie wird Qualität prüfbar?) Referatszusammenfassung von Marc Reinecke / Bonn 2003