

Einführung in Software Engineering

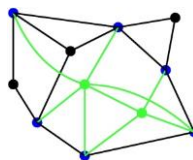
Barbara Paech, Marcus Seiler

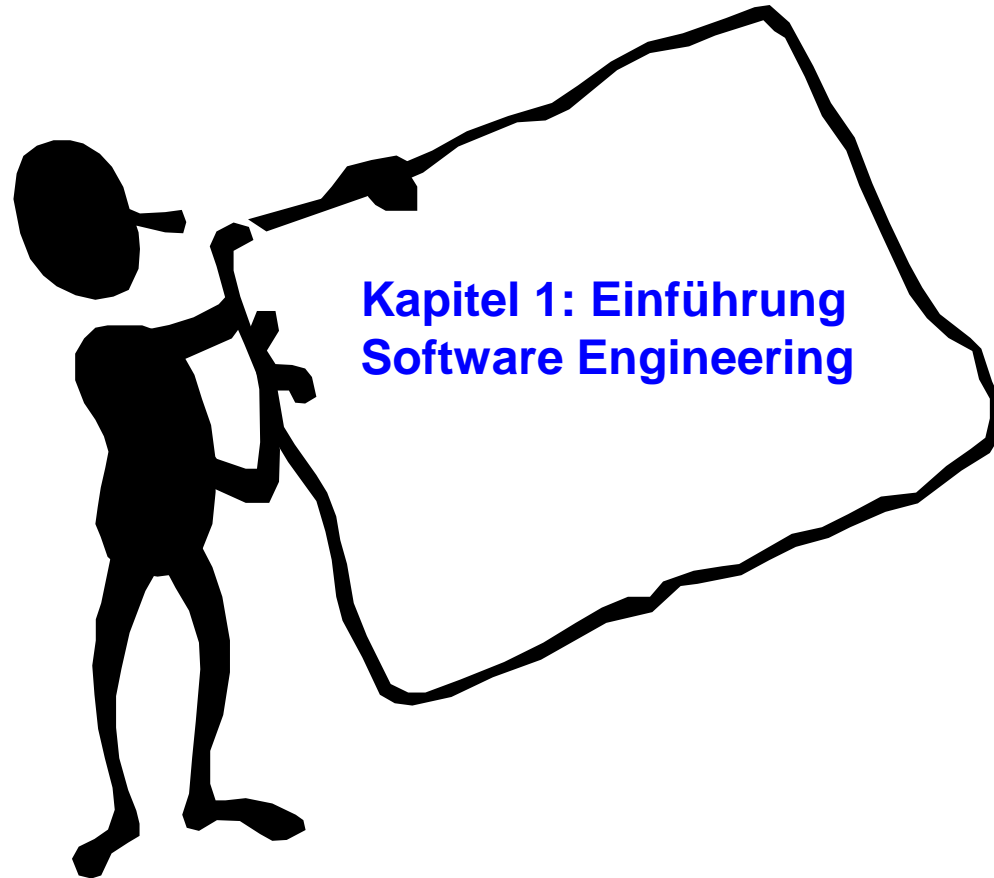
Institute of Computer Science
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
<http://se.ifi.uni-heidelberg.de>
paech@informatik.uni-heidelberg.de



RUPRECHT-KARLS-UNIVERSITÄT HEIDELBERG

- Prof. Dr. Barbara Paech
 - Since 12 years in HD
 - before Fh IESE, Kaiserlautern
 - 8 finished PhD students
 - 7 ongoing PhD students
- **Profile Quality Engineering through Software Engineering Inteligence**
- **Products**
 - SE teaching and consulting
 - Requirements Engineering Method TORE
 - Rationale-based CASE-Tool UNICASE (with TU München)





1. Einführung

Motivation

Organisatorisches

Wdh. Objektorientierung

Software Engineering ist

Entwicklung

- Qualitativ hochwertiger
- Großer Programmsysteme
- Mit vielen Beteiligten
- Unter Kosten- und Zeiteinschränkungen





Software-Qualität ist schwierig

- SWE-Charakteristik variiert in Abhängigkeit von den Zielen
[Weinberg, Schulmann, 1974]

Optimierungsziel	Aufwand	Anzahl Anweisungen	Speicherbedarf	Klarheit Programm	Klarheit Output
Aufwand	1	4	4	5	3
Anzahl Anweisungen	2-3	1	2	3	5
Speicherbedarf	5	2	1	4	4
Klarheit Programm	4	3	3	2	2
Klarheit Output	2-3	5	5	1	1

Was heißt...

■ Groß (Lines of Code, LOC)

- Typische APP (Android)
 - 3.000-20.000 LOC
- Typische Geschäftssoftware
 - z.B. HTTP Apache Server (120.000) - SAP Netweaver (240.000.000)



■ Viele Beteiligte

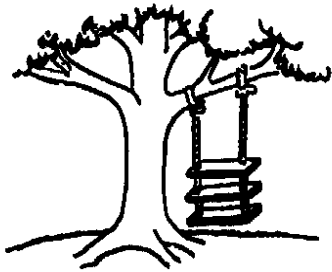
- NutzerInnen
 - APP Millionen
 - SAP Hunderttausende
- Entwicklungsteam
 - APP 1-20 (ca. 5 Kern)
 - HTTP Apache ca. 100 (ca 20 Kern)

Gewinn!

Kosten!



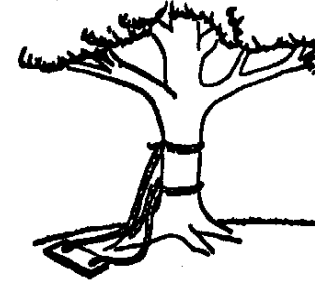
Kommunikation ist schwierig!



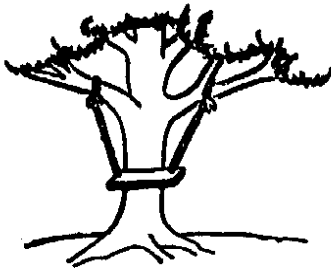
**As proposed by the
project sponsor**



**As specified in the
project request**



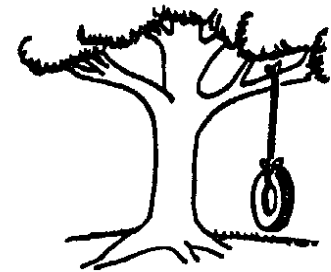
**As designed by the
senior analyst**



**As produced by the
programmers**



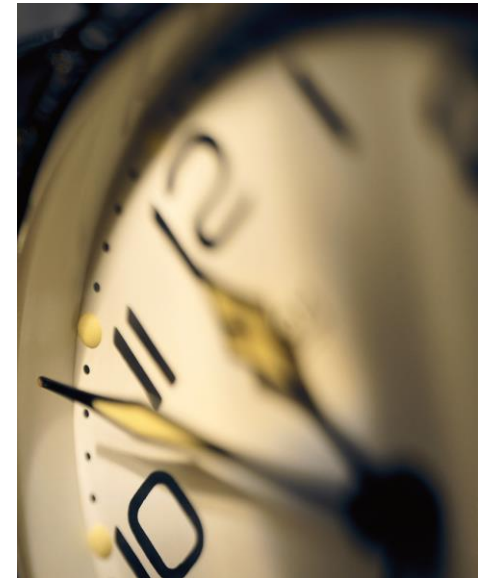
**As installed at
the user's site**



What the user wanted

■ Entwicklungs-Kosten und Zeit

- Kosten in der SWE
 - keine Fertigungskosten! => Kosten = Zeitaufwand
- **Achtung:** Erstentwicklung vs. **Weiterentwicklung!**
 - Ca. 60% der Gesamtkosten für Wartung!
- APP
 - Wenige Tage oder Wochen
 - 16.000-500.000€ (1)
- Geschäftssoftware
 - Wochen bis Jahre



(1) = <http://www.appadvisors.de/2011/07/was-ap-entwicklung-in-deutschland-kostet/>

Syer, Mark D., et al. "Revisiting prior empirical findings for mobile apps: an empirical case study on the 15 most popular open-source Android apps."
" CASCON. 2013.

Revisiting Prior Empirical Findings For Mobile Apps: An Empirical Case Study on the 15 Most Popular Open-Source Android Apps

Mark D. Syer, Meiyappan Nagappan, Ahmed E. Hassan
Software Analysis and Intelligence Lab
School of Computing, Queens University
{mdsyer, mei, ahmed}@cs.queensu.ca

Bram Adams
Lab on Maintenance, Construction and Intelligence of Software
Génie Informatique et Génie Logiciel, École Polytechnique de Montréal
bram.adams@polymtl.ca

Abstract

Our increasing reliance on mobile devices has led to the explosive development of millions of mobile apps across multiple platforms that are used by millions of people around the world every day. However, most software engineering research is performed on large desktop or server-side software applications (e.g., Eclipse and Apache). Unlike the software applications that we typically study, mobile apps are 1) designed to run on devices with limited, but diverse, resources (e.g., limited screen space and touch interfaces with diverse gestures) and 2) distributed through centralized "app stores," where there is a low barrier to entry and heavy competition. Hence, mobile apps may differ from traditionally studied desktop or server side applications, the extent that existing software development "best practices" may not apply to mobile apps. Therefore, we perform an exploratory study, comparing mobile apps to commonly studied large applications and smaller applications along two dimensions: the size of the code base and the time to fix defects. Finally, we discuss the impact of our findings by identifying a set of unique software engineering challenges posed by mobile apps.

Copyright © 2013 Mark D. Syer. Permission to copy is hereby granted provided the original copyright notice is reproduced in copies made.

1 Introduction

App stores (e.g., Google Play and Apple App Store) have changed the software development world by providing a platform for the rapid growth of mobile apps (i.e., software applications for smartphones and tablets). Since 2007, mobile apps have exploded into a multi-billion dollar market [1, 2] and become hugely popular among consumers and developers. Mobile app downloads have risen from 7 billion in 2009 to 15 billion in 2010 [2] and are projected to reach 74 billion in 2016 [1]. Simultaneously, the number of mobile apps has also increased: Google Play now hosts over 1 million mobile apps [3].

Despite the ubiquity of mobile devices and the popularity of mobile apps, few software engineering researchers have studied them [4, 5, 6, 7]. During the thirty-five years following Fred Brooks' seminal text, The Mythical Man Month [8], the software engineering community has proposed and evaluated several ideas of how high quality, successful software is developed and maintained. These "software engineering concepts" aim to tie aspects of software artifacts (e.g., size and complexity) [9], their development (e.g., number of changes) [10] and their developers (e.g., developer experience) [11] to definitions of quality (e.g., post-release defects). However, such software engineering concepts have primarily been evaluated against large-scale projects [12].



The CRASH Report 2014-2015 (CAST Research on Application Software Health)

The Global State of Structural Quality in IT Applications



<http://www.castsoftware.com/research-labs/crash-reports>

Kernfragen der Softwareentwicklung

- ✓ **Qualität:** Wie stellen wir sicher, dass das Softwaresystem **tut, was es soll?**
- ✓ **Beteiligte:** Wie erreichen wir, dass das Softwareprodukt **für die NutzerInnen nützlich** ist?
- ✓ **Beteiligte:** Wie erreichen wir, dass das Softwaresystem **effizient** zu entwickeln und insbesondere **für neue EntwicklerInnen verständlich und langfristig weiterzuentwickeln** ist?
- ✓ **Kosten/Zeit:** Wie stellen wir sicher, dass das Softwaresystem mit den **vorgegebenen Ressourcen** (Geld, Technologie, Leute) **im Zeit- und Kostenrahmen fertiggestellt** wird?





CHAOS SUMMARY FOR 2010

Copyright © 2010 The Standish Group International, Inc.

3

- Challenged = nicht im Plan
- Zeitüberschreitung bis zu 63%, Kostenüberschreitung bis zu 45 %

Wie ist die Entwicklung zu gestalten?

- **Hängt von den Zeit- und Rahmenbedingungen ab**
 - Wer macht was wie? (Rollen, Ergebnisse, Werkzeuge)
 - Kernfragen
 - Wie wird Qualität sichergestellt?
 - Wie werden NutzerInnen einbezogen?
 - Wie wird effiziente Entwicklung und Weiterentwicklung (Wissenübergabe) unterstützt?
 - Wie wird Zeit/Kosten-Rahmen eingehalten?

- **Wie läuft das bei SWE im Kleinen, z.B. APP-Entwicklung?**

SWE Techniken: Welche kennen Sie?

	SWE im Kleinen (APP)	SWE im Großen (zusätzlich)
Rollen	Tester	
Ergebnisse	Code	Diagramme
Werkzeuge	Code-Bibliothek	
Qualität	Test	
Einbezug NutzerInnen		
Wissensweiter- gabe, effiziente Entwicklung	Code-Dokumentation	
Zeit/Kosten		

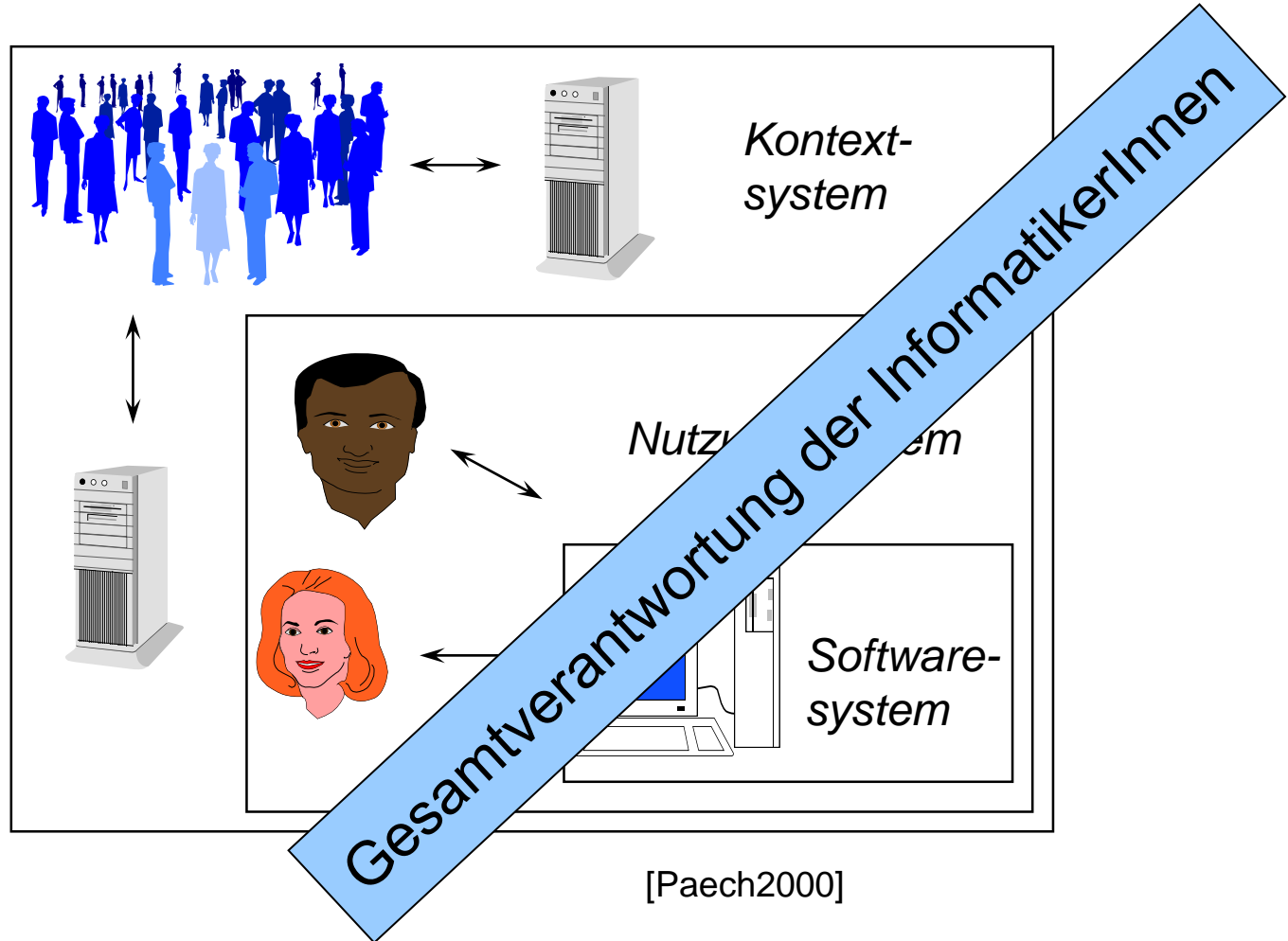
SWE: Was hilft zum Projekterfolg?

Details zur Standish Group Chaos-Umfrage (Reihenfolge der Erfolgsfaktoren laut Umfrage, blau = Bezug zu Anforderungen):

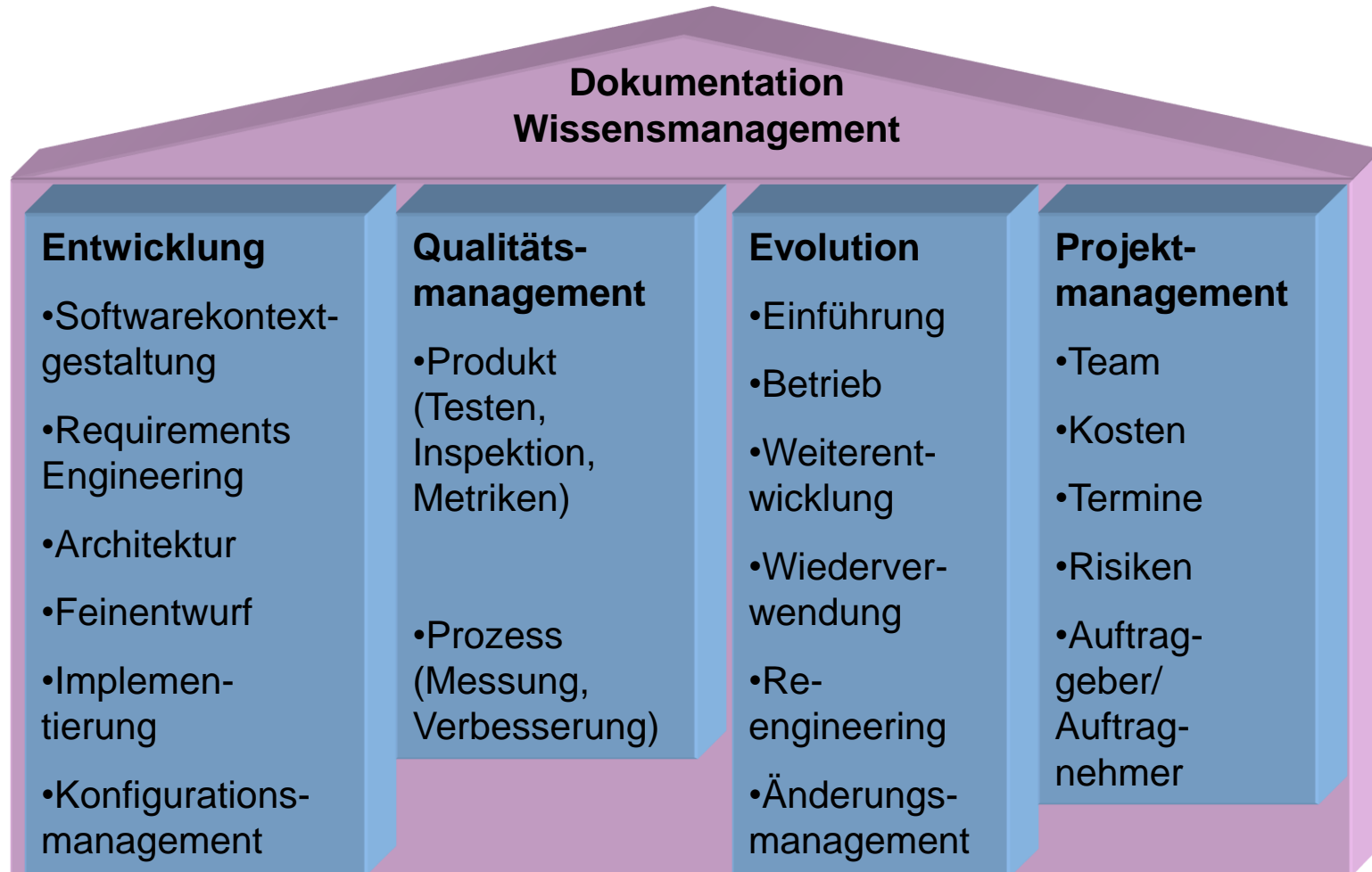
1. Management Unterstützung
2. Einbezug von NutzerInnen
3. Klare Geschäftsziele
4. Reife
5. Reduzierter Scope
6. Agile Prozesse
7. Erfahrene ProjektmanagerInnen
8. Kompetentes Team
9. Durchführung
10. Standard SW Infrastruktur

Quelle: Standish Group 2010

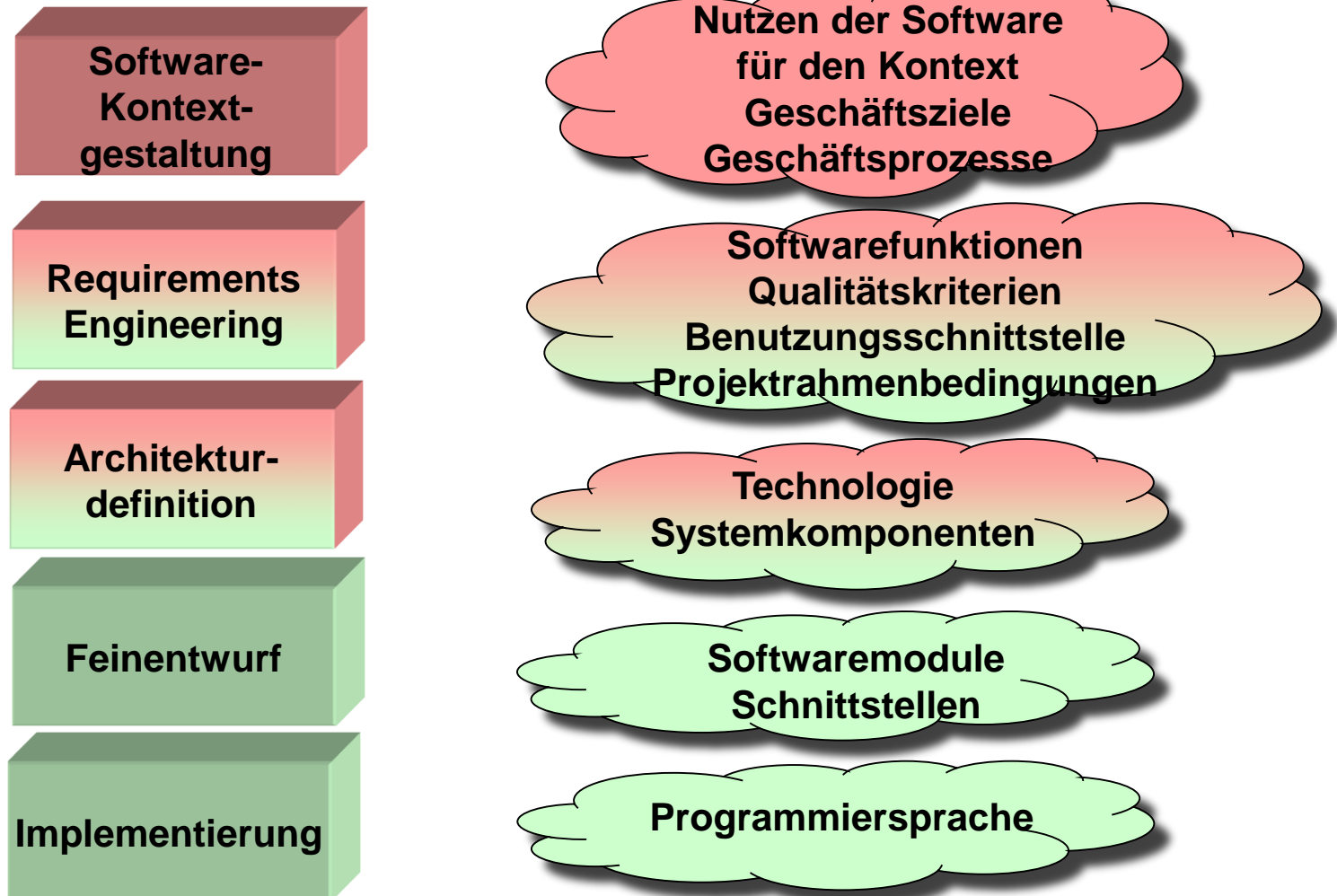
Gestaltungsbereiche der SW-Entwicklung



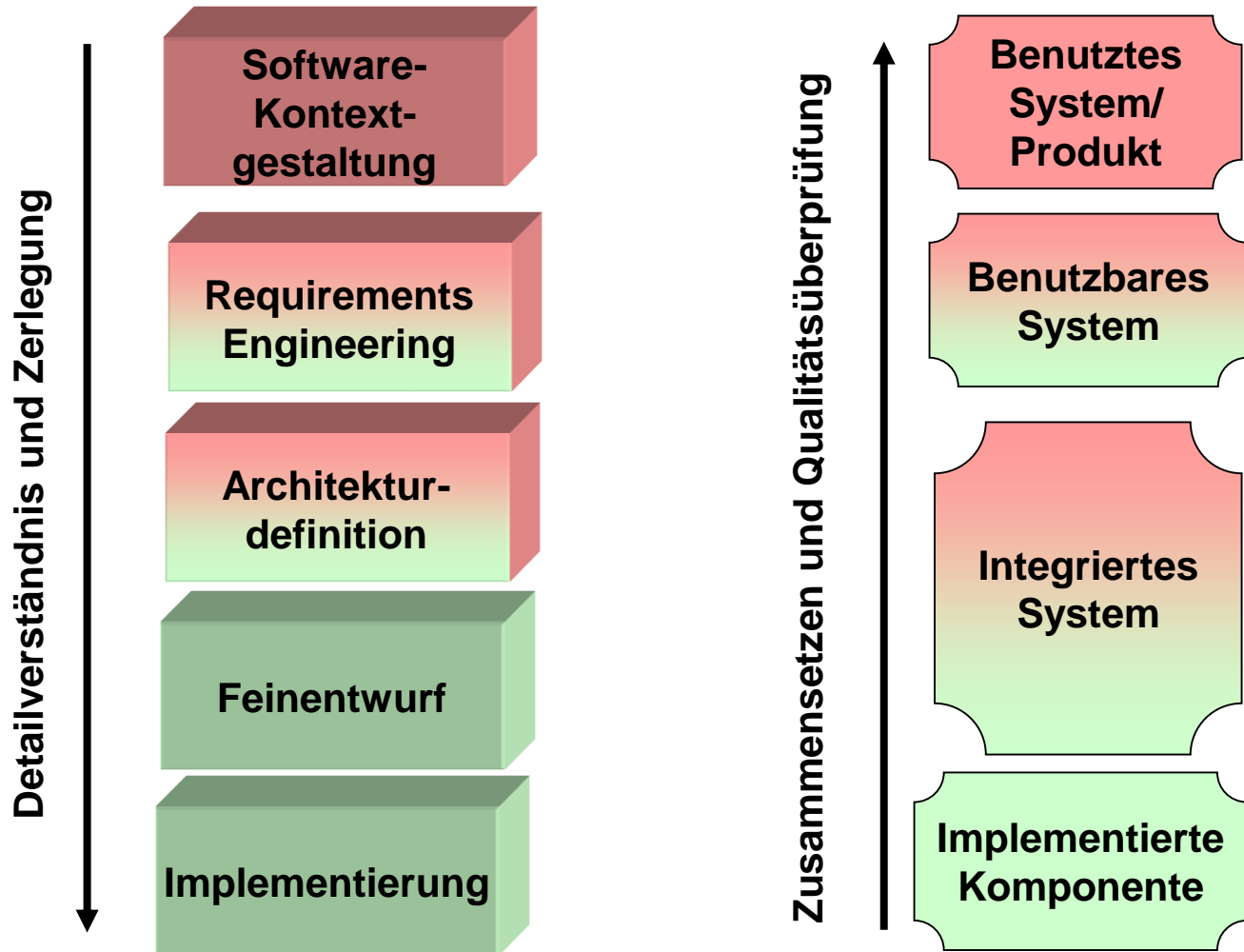
Aufgabenbereiche des Software Engineering

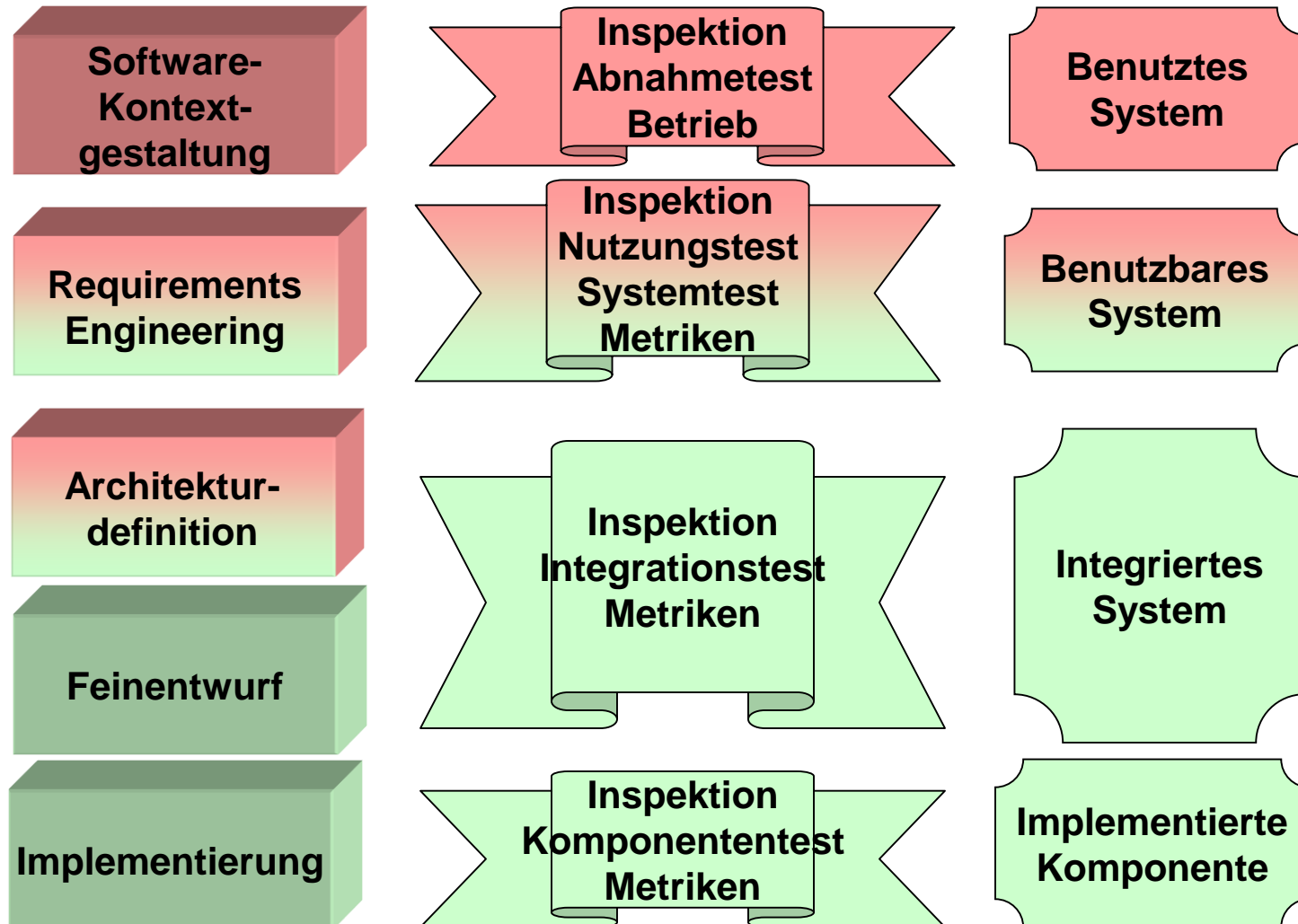


Aktivitäten und Gestaltungsentscheidungen



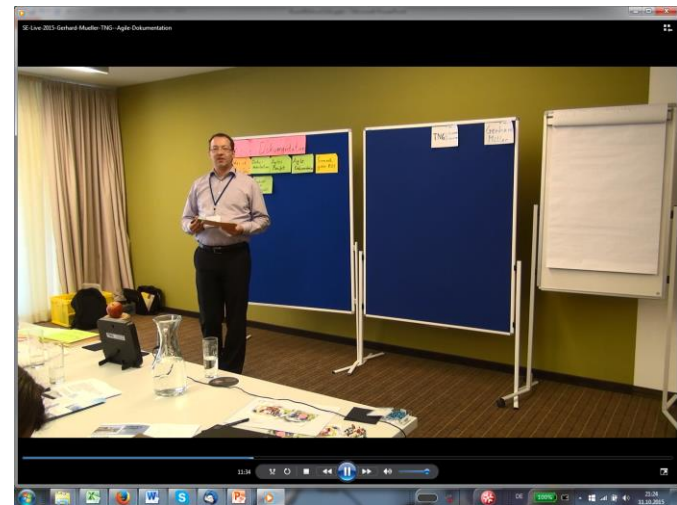
Aktivitäten und Ergebnisse der Entwicklung





Video Agile Dokumentation

- Gerhard Müller
- <http://www.tngtech.com/>
- <https://www.youtube.com/watch?v=MkYHspL2K8g>



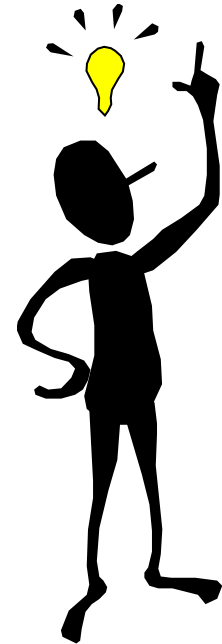
1. Einführung

Motivation

Organisatorisches

Wdh. Objektorientierung

- Verständnis für die **Komplexität** der Softwareentwicklung
 - Wissen über die **Interessen und Aufgaben** der Projektbeteiligten
 - Kenntnis der **Grundprinzipien** der Softwareentwicklung
-
- Inhalte
 - Begriffe und Konzepte
 - Best Practices
 - eigenständige Erfahrung von wichtigen Techniken (durch Übungen)



Überblick über Inhalte der Vorlesung und Übung

	SWE im Kleinen	SWE im Großen (zusätzlich)	Übungen
Weiter-entwicklung, Vorgehen	Issue Tracker, Agiles Vorgehen (XP)	Änderungsmgt., Projektmgt, Agiles Vorgehen (Scrum), Nicht-agiles Vorgehen	Projekt
Requirements Engineering	Textuelle Anforderungen	Aufgaben, Rollen, Persona Use Cases, Domänendaten, UI-Struktur, Dialoge, Sichten, Qualitätsanforderungen	Projekt
Architekturdefinition		Architektur und Architekturmuster	
Feinentwurf		Modellierung mit UML, OOAD, Entwurfsmuster, Rationale	Projekt
Implementierung	Code-Richtlinien, Versionsmanagement		Projekt (Java, GUI, Eclipse)
Qualitätssicherung	Paarprogrammierung, Fehlermgt., Metriken, Komponententest, Integrationstest, Systemtest	Inspektionen, Usability-Test	Projekt

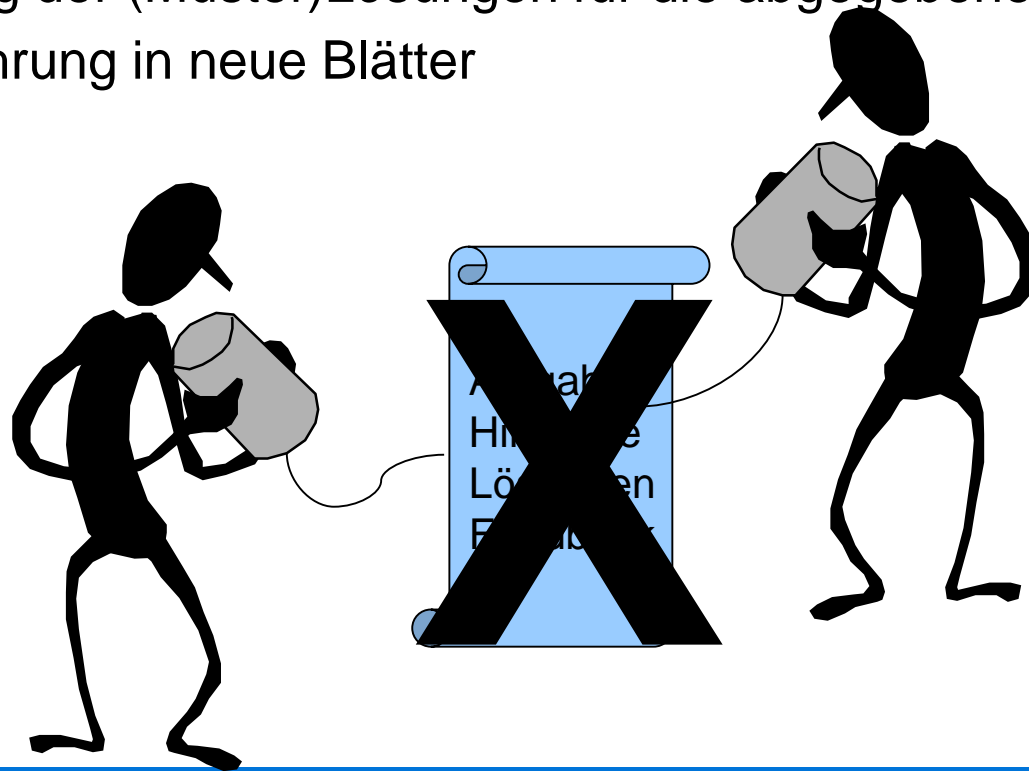
- **IPI** führt ein in die Programmierung im Kleinen und in die Objektorientierung (mit C++)
- Beides wird in ISW **vorausgesetzt!**
- **ISW** führt ein in die Programmierung im Großen (Team, KundInnen, Qualität). Insbesondere wird mit **komplexer Java-Software** (EMFStore) gearbeitet (weiterentwickelt). Dazu wird eine **komplexe IDE** verwendet (Eclipse).

- Dienstag vormittag 1. Teil **Vorlesung**: Neue Konzepte
 - Interaktive Erarbeitung
- Dienstag nachmittag 2. Teil **meistens Zentralübung**: Technologien (manchmal Vorlesungsstoff)
- **Gruppenübung** (Hausaufgaben) **siehe Hinweisblatt**
 - Aufgaben teilweise alleine, teilweise als 2er oder 4er Team zu bearbeiten
 - **ERSTE WOCHE**: Übungsmöglichkeit für JAVA und Eclipse
 - Passwort für Moodle: isw1516
- Optional: Anfängerpraktikum als Blockpraktikum im Februar/März (12 Tage): **Bearbeitung eines großen Softwaresystems im Team**

■ Altes Modell

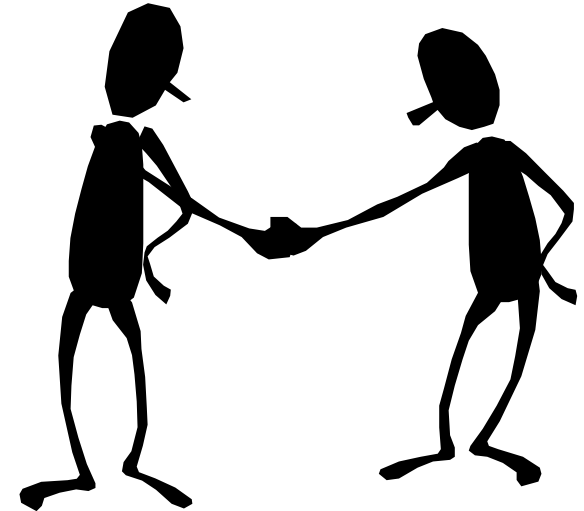
- Aufgabenblatt
- Abgabe der Lösung
- Besprechung der (Muster)Lösungen für die abgegebenen Blätter
- Kurze Einführung in neue Blätter

■ Ineffizient



■ Neues Modell

- **Präsenzübungen**
- Betreute Ein- und Erarbeitung der neuen Aufgaben
- Abgabe der Lösung
- **Präsentation der Lösung bei den Tutoren (Testat)**

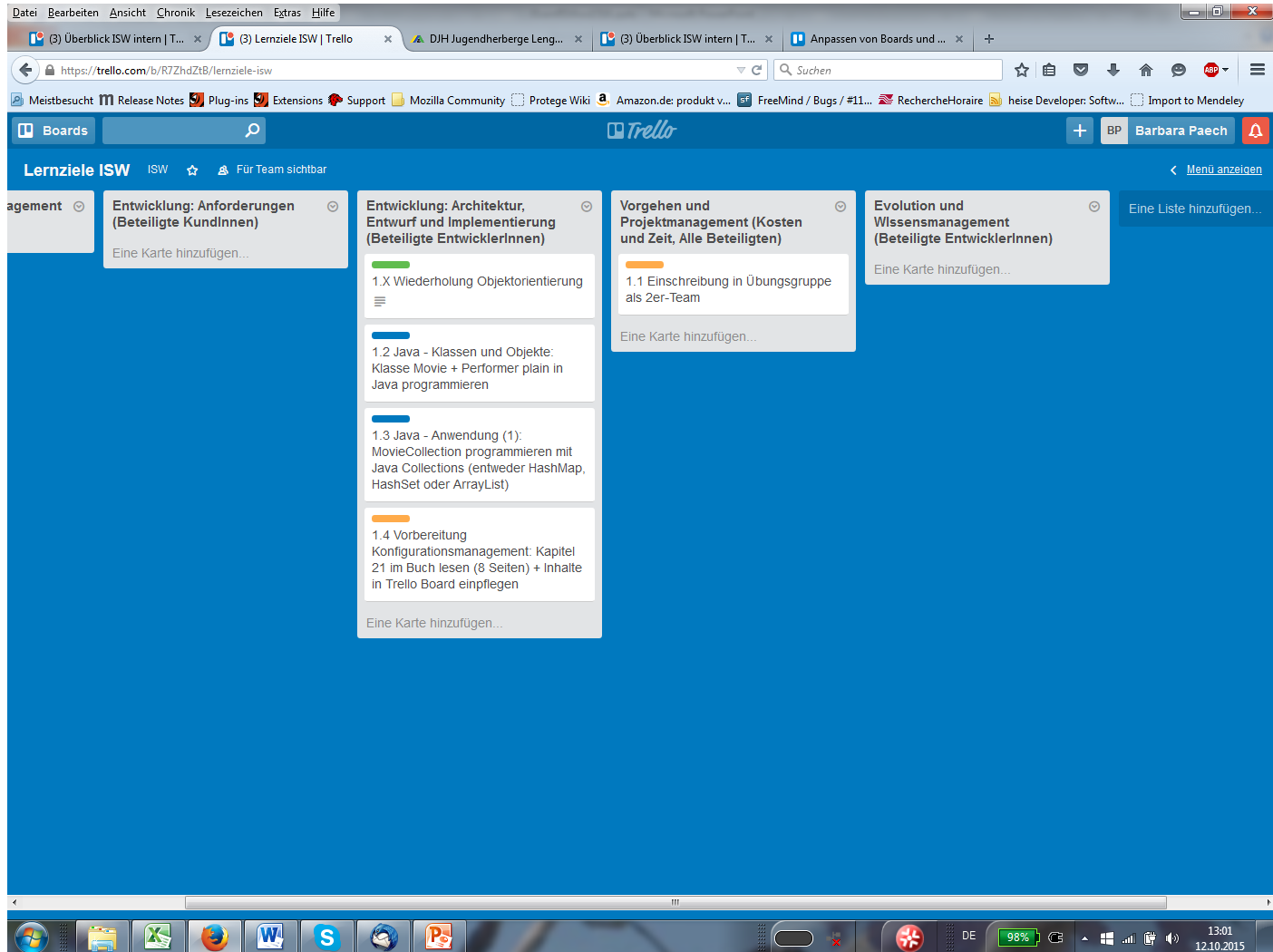


- Nicht bei allen Aufgaben, aber bei allen Pflichtaufgaben

- Ich erkläre **Folien** in der Vorlesung (kein Skript, große Teile im Buch Ludewig/Lichter nachzulesen).
- Sie bearbeiten **Arbeitsblätter** während der Vorlesung (insbes. für Modellierung), wir diskutieren Lösungen.
- **Inverse Teaching**: Sie bereiten Stoff zuhause vor (mit Leitfragen, Nutzung von Trello), wir diskutieren das in der Vorlesung.
- **Klare Definition von Lernzielen**: Sie können die Lernziele im Trello-Board „Lernziele ISW“ verfolgen (und konstruktiv kommentieren).
- Marcus Seiler erklärt neue Technologien in der **Zentralübung**. Dort üben Sie die Grundlagen dazu.

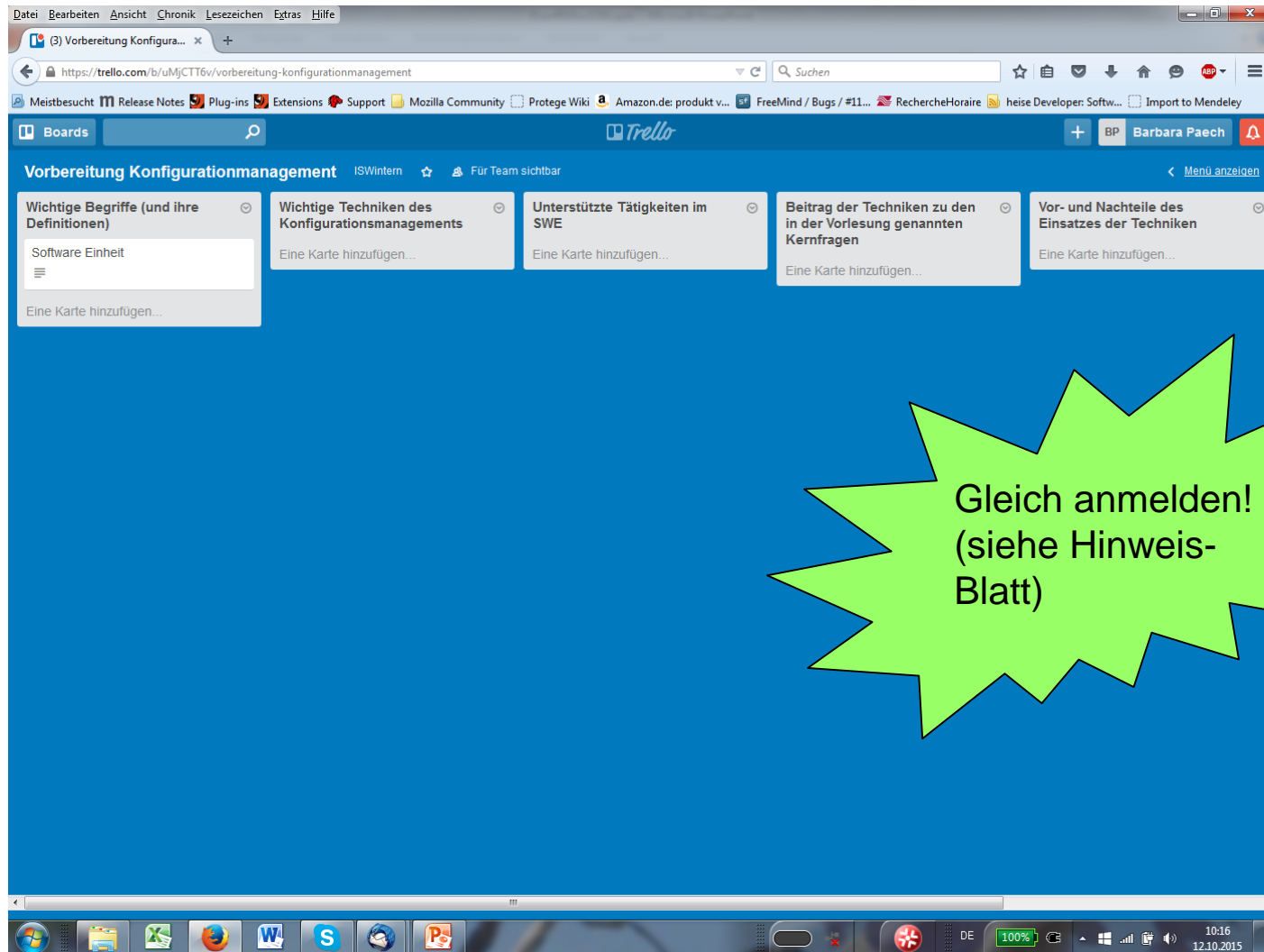
Organisation: Interaktion Studierende Lehrende (2)

- Sie bearbeiten **Übungsblätter** zuhause und geben alles in Moodle ab.
- Sie bearbeiten Teile der Hausaufgaben in der **Präsenzübung**.
- Sie präsentieren Ihre Lösung für **Testaufgaben** dem Tutor und erhalten direkt Feedback.
- Sie erhalten **Feedback** zu den anderen Aufgaben über Moodle.
- Sie können **Verständnisfragen** zu Übungsaufgaben in der Zentralübung und der Übung sowie im Forum stellen und erhalten schnellstmöglich (**Wochenende!**) antworten.
- Weitere **Bemerkungen** zu den Übungsaufgaben können Sie im Trello-Board „Lernziele ISW“ machen.



The screenshot shows a Trello board named "Lernziele ISW" with the following structure:

- Board Name:** Lernziele ISW
- Columns (Lists):**
 - Entwicklung: Anforderungen (Beteiligte KundInnen)**
 - Card: Eine Karte hinzufügen...
 - Entwicklung: Architektur, Entwurf und Implementierung (Beteiligte EntwicklerInnen)**
 - Card: 1.X Wiederholung Objektorientierung
 - Card: 1.2 Java - Klassen und Objekte: Klasse Movie + Performer plain in Java programmieren
 - Card: 1.3 Java - Anwendung (1): MovieCollection programmieren mit Java Collections (entweder HashMap, HashSet oder ArrayList)
 - Card: 1.4 Vorbereitung Konfigurationsmanagement: Kapitel 21 im Buch lesen (8 Seiten) + Inhalte in Trello Board einpflegen
 - Card: Eine Karte hinzufügen...
 - Vorgehen und Projektmanagement (Kosten und Zeit, Alle Beteiligten)**
 - Card: 1.1 Einschreibung in Übungsgruppe als 2er-Team
 - Card: Eine Karte hinzufügen...
 - Evolution und Wissensmanagement (Beteiligte EntwicklerInnen)**
 - Card: Eine Karte hinzufügen...
- Right Side:** A button "Eine Liste hinzufügen..." and a "Menu anzeigen" link.
- User:** Barbara Paech
- Bottom Bar:** Windows taskbar showing icons for various applications and system status (13:01, 12.10.2015).





- 8 ECTS (4+2 SWS; 240 Stunden Aufwand)
- Erfolgreiche Teilnahme an Übungen
 - Geht nicht in Note ein, aber **Voraussetzung** für die Teilnahme an der schriftlichen Prüfung
 - Erfolgreich bedeutet
 - Alle Testaufgaben erfolgreich bearbeitet (das sind die Programmier- und Projektaufgaben, die man in der Klausur nicht sinnvoll abfragen kann. Ca. 50% der Hausaufgaben). Testat besteht aus erfolgreicher Erklärung der Lösung gegenüber den Tutoren (einzeln oder im Team).
 - Die anderen Hausaufgaben dienen der Vorbereitung auf die Vorlesung (inverse Teaching) und der Einübung von Techniken (wie sie auch in der Klausur vorkommen können).
- Schriftliche Prüfung am Ende der Vorlesung (11. Februar, Uhrzeit noch unklar)
- Note: aus der schriftliche Prüfung

- Entwicklung und Management großer Informationssysteme
- Wissenschaftliches Arbeiten in Datenbanken und Software Engineering
- <http://www.informatik.uni-heidelberg.de/index.php?id=265>

1. Einführung

Motivation

Organisatorisches

Wdh. Objektorientierung

- Qualitätssicherung durch wichtige Eigenschaften von Programmiersprachen
 - Strukturelemente zur Konstruktion modularer Programmeinheiten
 - Trennung von Schnittstelle und Implementierung
 - Mächtiges Typsystem mit strenger Typprüfung
 - Syntax, die zur Lesbarkeit des Codes beiträgt
 - Automatische Zeigerverwaltung
 - Ausnahmebehandlung

- Sprachparadigma
- Welche technische Umgebung ist gegeben?
 - Legacy-System? Gibt es eine Vorgabe des Unternehmens?
- Welche Bibliotheken werden benötigt?
- Wie gut ist die Werkzeugunterstützung?
 - Compiler, Debugger, IDE, ...
- Welche Kenntnisse/Vorlieben besitzen die Programmierer?

- Hinweis: Ranking aktuelle Programmiersprachen
 - <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>
 - <http://redmonk.com/sogady/2012/09/12/language-rankings-9-12/>

- Auch imperative Programmierung genannt
- Beispiele: Modula-2, Pascal, C, Fortran
- Ideen zum Modul-Konzept teilweise vorhanden
- Komfortable Definition von Datenstrukturen
- Trennung von Datenstruktur und Funktionen macht Wartbarkeit schwieriger
- **Fazit:**
 - Für kleinere und mittlere Systeme geeignet

- Beispiele: LISP, Haskell, ML, heute Clojure, Scala, XSLT
- Seiteneffektfrei und dadurch leicht verstehbar
- Sehr mächtiges Typsystem
- Patternmatching auf Argumenten
- Effiziente Definition von Datenstrukturen und Funktionen
 - `data Tree = Leaf(Int) | Node(Tree,Int,Tree)`
- Funktionen höherer Ordnung (Funktionen auf Funktionen anwendbar)
 - `twice f x = f(f(x))`
- Kompakte Formulierung

- **Fazit:**
 - Effektive und elegante Programmierung, aber langsamere Ausführungszeiten
 - Geeignet für schnelle Entwicklung, skaliert nicht für große Programmsysteme
 - Schwierigkeiten mit interaktiven Systemen (z.B. GUI) umzugehen

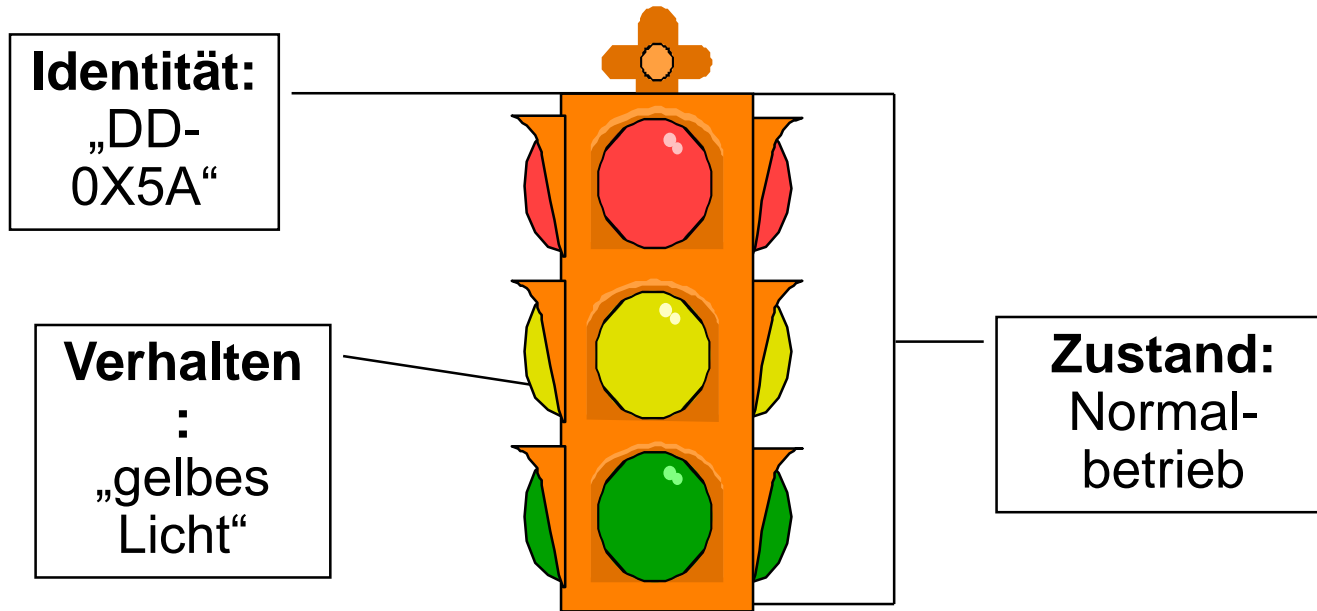
- Beispiel: Java, C++, Oberon, Modula-3, Smalltalk, C#, ...
- Die Merkmale der OO:
 - Objekte (Klassen) als Kapseln von Daten und Funktionen
 - Objekte dynamisch erzeugen: Objektidentität
 - Vererbung (in ihren verschiedenen Ausprägungen)
- OO Sprachen subsumieren prozedurale Programmierung (Java 8 auch etwas funktionale Programmierung)
- Es erfordert aber eine wesentlich andere Vorgehensweise, um Vorteile der OO zu nutzen:
 - Vererbung als Mechanismus zur Anpassung und Verbesserung der Wiederverwendung
 - „Gutes Design“, um Wartbarkeit und Erweiterbarkeit zu stützen
 - Codingstandards für Lesbarkeit
- **Fazit:**
 - OO ist heute das Mittel der Wahl für große Projekte, effizienter geht es aber oft mit Spezialsprachen

- **Logikprogramme:** Prolog
 - Logische Aussagen in Hornklauselform als Programm
- **Visuelle Programmierung**
 - a) Komposition des Programms aus Bausteinen
 - b) Modellierung z.B. mit ausführbaren Statecharts
- **Programmiersprachen mit inhärentem Verteilungskonzept**
 - für massiv verteilte Systeme
- **Skriptsprachen**
 - Beispiel: Ruby, PHP, JavaScript, Python, Tcl/Tk, Perl
 - Meist kein festes Typsystem
 - effektive Module zur Textbearbeitung (z.B. reguläre Ausdrücke)
 - Neu: immer bessere Integration mit anderen Sprachen (z.B. Python + Java)
- Weitere Spezialsprachen: HTML, XML, SQL, ...

- **Kooperationsfähigkeit** erlaubt die gleichzeitige Verwendung mehrerer Sprachen:
 - Durch Middleware: Corba, .NET, Embedded SQL (z.B. in Java)
 - Java Server Pages (JSP) = Java + HTML
 - Modul-Import über API's (Python in Java)
- Fehlende Spracheigenschaften werden durch **standardisierte Klassenbibliotheken** abgedeckt:
 - I/O wurde in C als erstes durch Bibliotheksfunktionen standardisiert
 - Threads/Nebenläufigkeit wird in Java über die Bibliothek realisiert
 - Kommunikation, Datenspeicherung wird nicht über Sprachprimitive, sondern Bibliotheksfunktionen angeboten
 - Security (z.B. Java Sandbox oder Verschlüsselung)

Konzepte der Objektorientierung: Was ist ein Objekt?

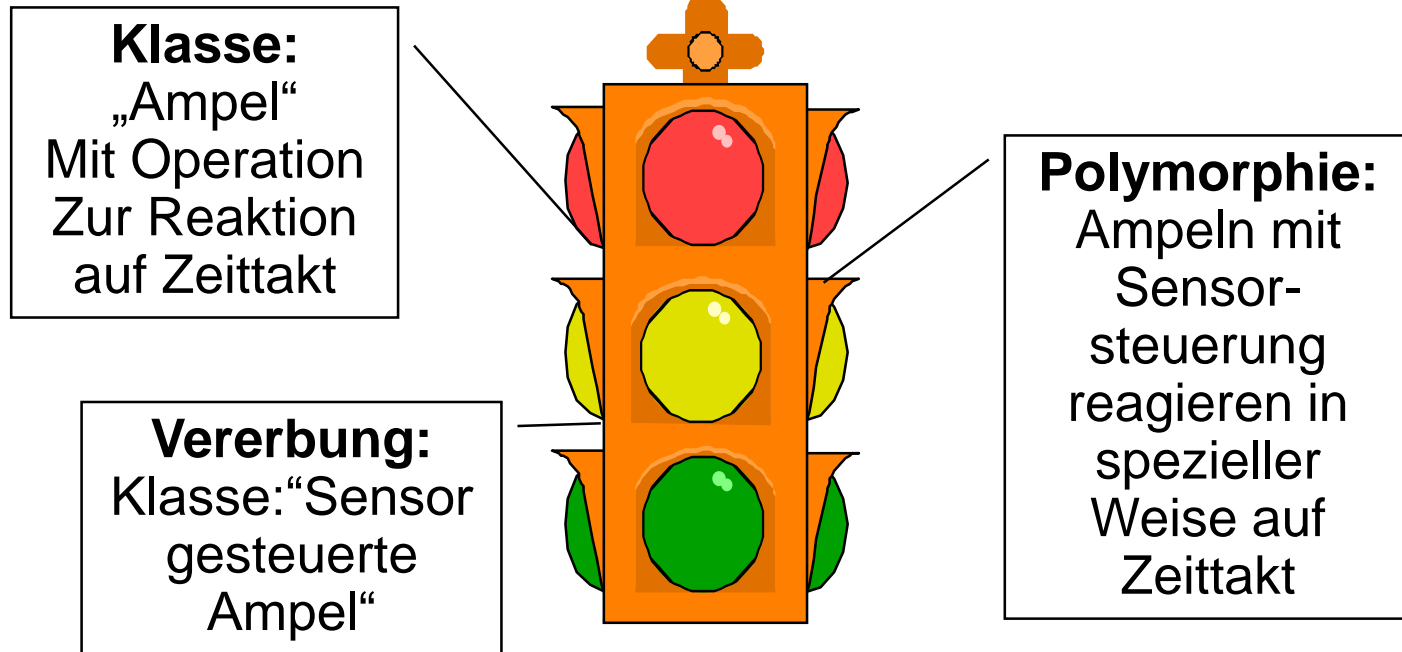
- Ein System besteht aus vielen Objekten.
- Ein Objekt hat ein definiertes **Verhalten**
 - Definierte **Methoden**
 - Methode wird beim Empfang einer **Nachricht** ausgeführt.
- Ein Objekt hat einen inneren **Zustand**
 - Zustand des Objekts ist **Privatsache**.
 - Resultat einer Methode hängt vom **aktuellen Zustand** ab.
- Ein Objekt hat eine eindeutige **Identität**
 - Identität ist **unabhängig von anderen Eigenschaften**.
 - Es können mehrere verschiedene Objekte mit identischem Verhalten und identischem inneren Zustand im gleichen System existieren.



- ◆ Wie ist das Objekt bezeichnet?
- ◆ Wie verhält es sich zu seiner Umgebung?
- ◆ Welche Informationen sind „Privatsache“ des Objekts?

- Ein **Objekt** gehört zu einer **Klasse**.
 - Die Klasse schreibt das Verhaltensschema und die innere Struktur (Attribute, Operationen) ihrer Objekte vor.
- Klassen besitzen einen 'Stammbaum', in dem Verhaltensschema und innere Struktur durch **Vererbung** weitergegeben werden.
 - Vererbung bedeutet **Generalisierung** einer Klasse zu einer Oberklasse.
- **Polymorphie**: Eine Nachricht kann verschiedene Reaktionen auslösen, je nachdem zu welcher Unterklasse einer Oberklasse das empfangende Objekt gehört

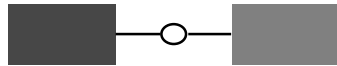
Beispiel: Klasse und Objekt



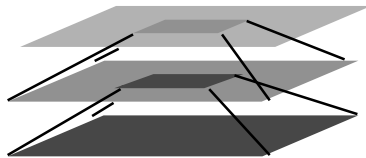
- ◆ Welcher Begriff beschreibt das Objekt?
- ◆ Welche Begriffshierarchie wird verwendet?
- ◆ Wie hängt das Verhalten des Objektes von der Hierarchie ab?



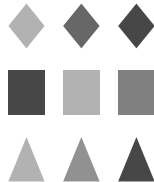
Zuständigkeitsbereiche



Klare Schnittstellen



Hierarchie



Baukastenprinzip

- Lokale Kombination von Daten und Operationen, gekapselter Zustand
- Definiertes Objektverhalten, Nachrichten zwischen Objekten
- Wiederverwendung durch Vererbung und Polymorphie (Spezialisierung), Klassenschachtelung
- Wiederverwendung durch Benutzung vorgefertigter Klassenbibliotheken, Anpassung durch Spezialisierung (mittels Vererbung)

Allgemein

Ludewig L, Lichter H (2013) *Software Engineering*, dpunkt

- Sommerville I (2012) *Software Engineering*, Addison Wesley

und Angaben zu jedem Kapitel: hier für Kapitel 1

- CRASH-Report 2015
- Paech B (2000) *Aufgabenorientierte Softwareentwicklung, Integrierte Gestaltung von Unternehmen, Arbeit und Software*, Springer Verlag
- Syer et al (2013) Revisiting Prior Empirical Findings For Mobile Apps: An Empirical Case Study on the 15 Most Popular Open-Source Android Apps
- Weinberg G, Schulmann E (1974) Goals and Performance in Computer Programming, *Human Factors* 16, p.70-77