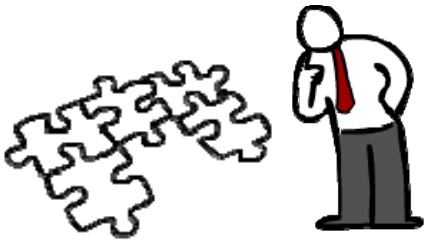
Vorlesung Software-Engineering I

... im 3. und 4. Semester.

01. Einführungsteil - Überblick



http://wwwlehre.dhbw-stuttgart.de/~sto/public/Modulplaene/T2INF2003.pdf

SW-Engineering I – Qualifikationsziele und Kompetenzen

- Die Studierenden kennen die **Grundlagen des Softwareerstellungsprozesses**.
- Sie können eine vorgegebene Problemstellung **analysieren** und rechnergestützt Lösungen **entwerfen, umsetzen** und **dokumentieren**.
- Sie kennen die **Methoden** der jeweiligen Phasen und können sie **anwenden**.
- Sie können Lösungsvorschläge für ein gegebenes Problem konkurrierend **bewerten** und **korrigierende Anpassungen** vornehmen.
- Die Studierenden können sich mit Fachvertretern über Problemanalysen und Lösungsvorschläge, sowie über die Zusammenhänge der einzelnen Phasen **austauschen**.
- Sie können einfache Softwareprojekte **autonom** entwickeln oder bei komplexen Projekten effektiv in einem **Team** mitwirken.
- Sie können ihre Entwürfe und Lösungen **präsentieren** und **begründen**.
- In der Diskussion im Team können sie sich kritisch mit **verschiedenen Sichtweisen** auseinandersetzen und diese erläutern.
- Sie **bewerten** die eingesetzten Technologien und schätzen ihre Folgen ab.
- Die Studierenden können sich selbsständig in **Werkzeuge** einarbeiten.
- Sie verbinden den Softwareentwicklungsprozess mit Techniken des Projektmanagement und beachten während des Projekts **Zeit- und Kostenfaktoren**.

http://wwwlehre.dhbw-stuttgart.de/~sto/public/Modulplaene/T2INF2003.pdf

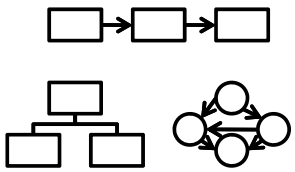
Lehr- und Lehrinhalte SW-Engineering I

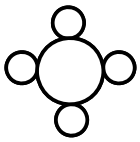
- **Vorgehensmodelle**
- **Phasen** des SW-Engineering und deren Zusammenhänge
- **Analyse**: Lastenheft
- **Spezifikation:** Pflichtenheft, Anwendungsfälle
- **Methoden** zur Repräsentation von Algorithmen
- Datenmodellen, Funktionsweisen, Zustands- und Regelabhängigkeiten
- **Entwurf:** SW-Architektur, Systementwurf, Schnittstellenentwurf, Klassendiagramme
- **Implementierung** und **Test**
- Codierrichtlinien und **Codequalität**, systematisches **Testen**, Testarten und Testdurchführung, Installation und Einführung
- Phasenspezifische werden die verschiedenen Arten der **Dokumentation** behandelt.

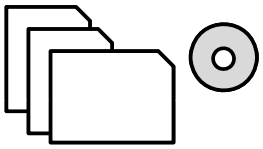
Labor/Gruppenarbeit:

Ein **komplexes Problem** wird als **Projekt** mit allen Phasen von den Studierenden erarbeitet und dokumentiert.

Parallele Inhaltsebenen







Standards

- Vorgehensmodelle
- Methoden, Modelle
- Verfahren, Notationen
- Projektmanagement

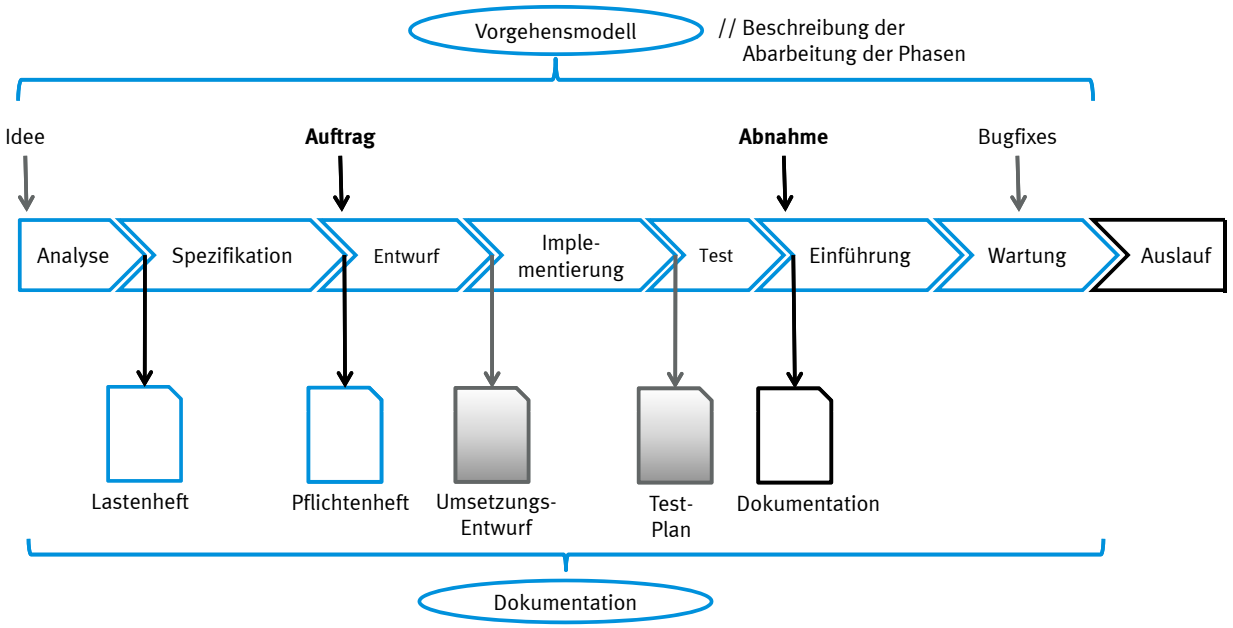
Best Practice

- Refactoring
- Pattern
- Technologien
- Werkzeuge

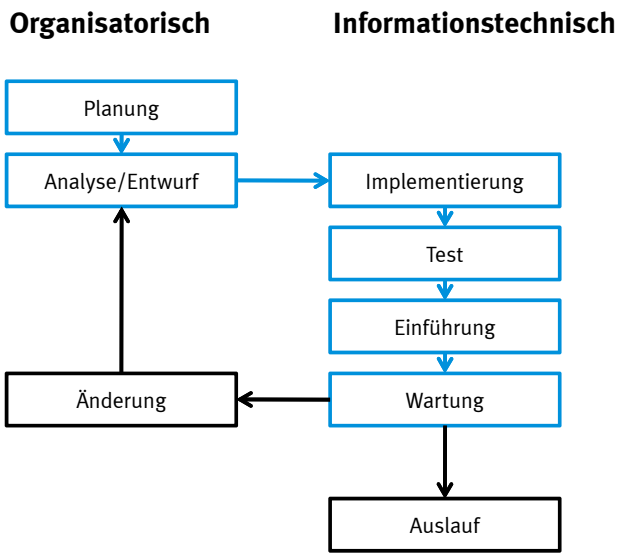
Reale Welt

- Soft Skills
- Beispiele
- Erfahrungen
- Teamplay

Phasenmodell der Software-Entwicklung



Software-Lebenszyklus



Software-Engineering, warum?

Definitionen:

Software:
- immateriell, nicht greifbar
- schnell änderbar

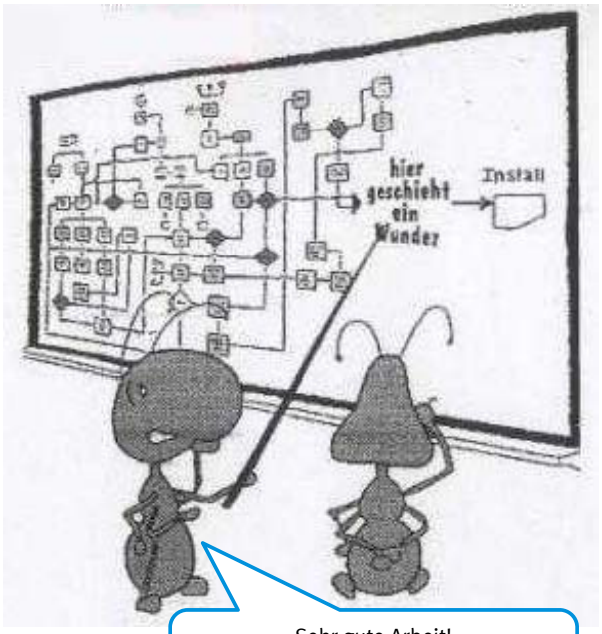
=› komplex, aufwändig, teuer in der Erstellung

Engineering:
- Ingenieurmäßiges Vorgehen
- praxisgerecht, bewährt

=› formalisiert, nachvollziehbar, zielgerichtet

=› (vollständig, berechenbar, vorhersagbar)

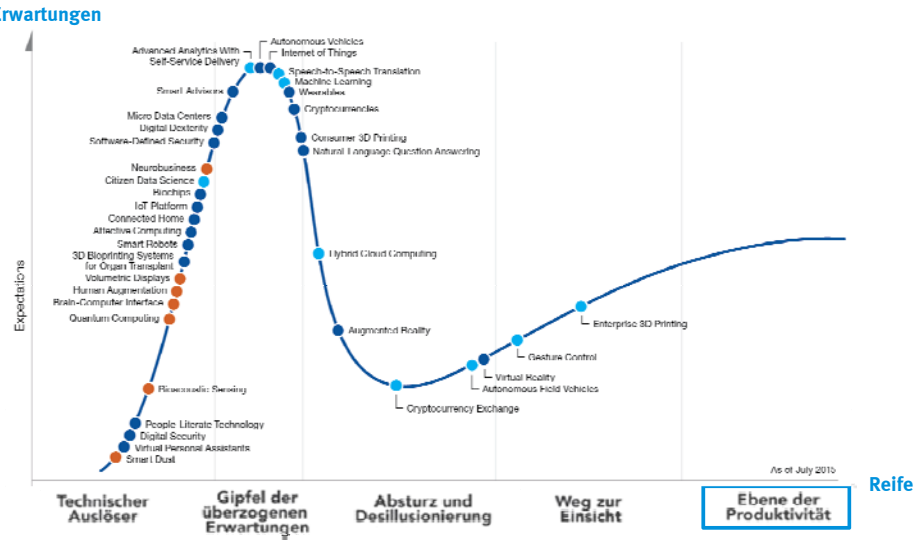
Ziel: Erfolgreich bessere Software erstellen!



Sehr gute Arbeit!
Aber sollten wir **hier** vielleicht nicht noch **ein wenig detaillierter** werden...?

http://de.wikipedia.org/wiki/Hype-Zyklus

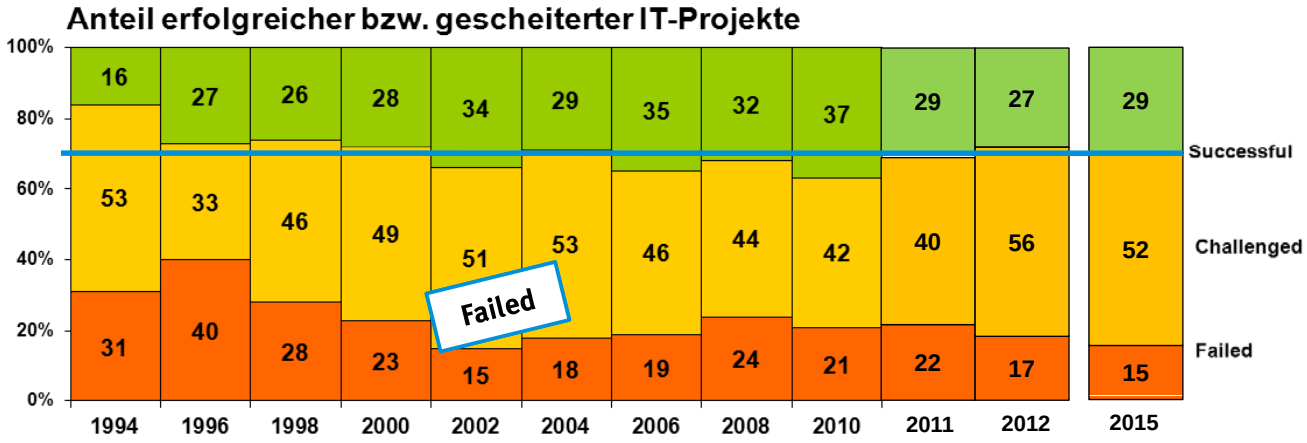
The Gartner Hype Cycle – Technologien im Mainstream



Quelle: Gartner-Report

http://www.standishgroup.com/
http://de.wikipedia.org/wiki/Chaos-Studie

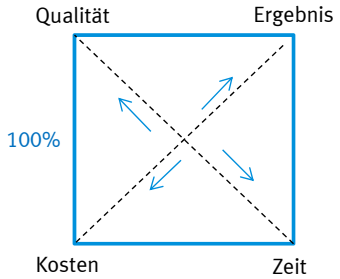
Standish Group - Chaos Report

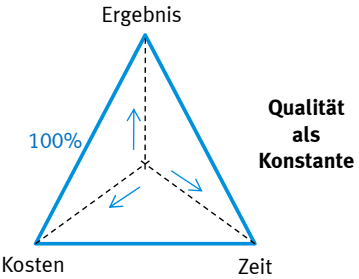


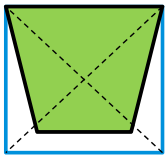
Successful	Funktion, Qualität, Kosten und Termine eingehalten und in Produktion genommen.
Challenged	In Produktion genommen, aber Funktion, Qualität, Kosten oder Termine nicht eingehalten.
Failed	Nicht in Produktion genommen.

Quelle: Standish Group – Chaos-Report

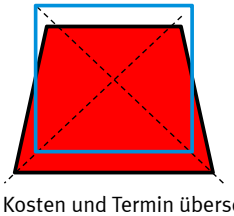
Das Teufelsquadrat und das magische Dreieck



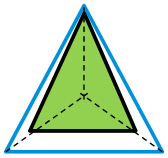




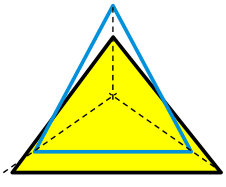
Qualität und Ergebnis mit weniger Kosten und vor dem Termin erreicht.



Kosten und Termin überschritten, Ergebnis mit entsprechender Qualität aber noch nicht erreicht.



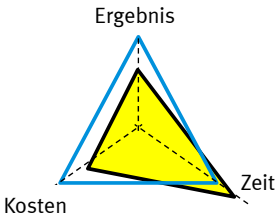
Ergebnis mit weniger Kosten und vor dem Termin erreicht.



Kosten und Termin überschritten, Ergebnis aber noch nicht erreicht.

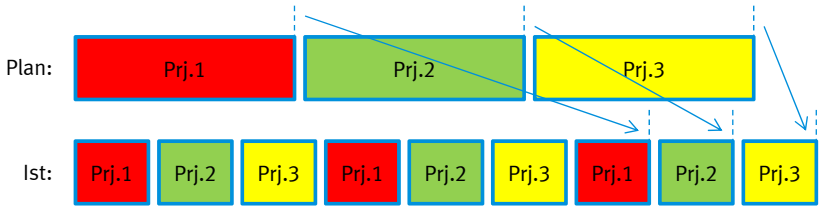
schädliches Multitasking

Typischer Projektstatus:



Termin überschritten,
Ergebnis noch nicht erreicht,
aber noch Geld (Kosten)übrig.

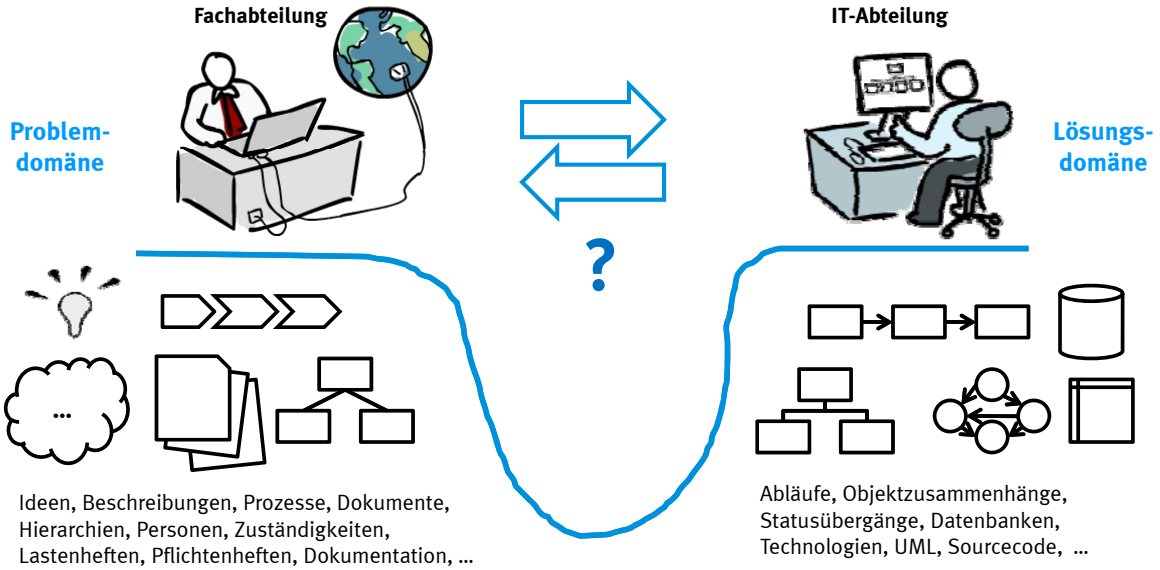
Typischer Projektverlauf :



Mehrere Projekte werden parallel bearbeitet.
Folge: keines der Projekte wird zum geplanten Termin fertig.

Wo wird diese Erkenntnis schon anerkannt?
- Agile Techniken -› Fokussierung
- Projekt-Management -› CCPM (Engpassmanagement)
- Gesunder Menschenverstand bzw. eigene Erfahrung

Der Abgrund zwischen Anwender und Programmierer



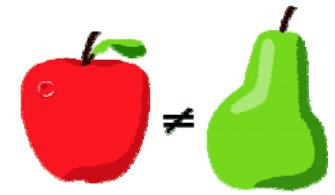
Ideen, Beschreibungen, Prozesse, Dokumente,
Hierarchien, Personen, Zuständigkeiten,
Lastenheften, Pflichtenheften, Dokumentation, ...

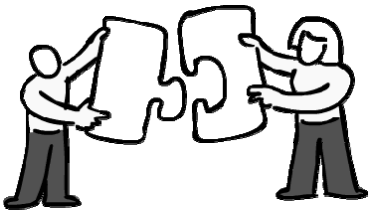
Abläufe, Objektzusammenhänge,
Statusübergänge, Datenbanken,
Technologien, UML, Sourcecode, ...

Gegensätze erkennen und differenzieren können

- Lasten- und Pflichtenheft
- Auftraggeber und Auftragnehmer
- Anwender und Programmierer
- Problem und Lösung
- Problemdomäne und Lösungsdomäne
- Anforderungen und Umsetzung
- Implementierung und Test
- Implementierung und Dokumentation
- Entwurf und Umsetzung
- Bug und Feature Request
- ...

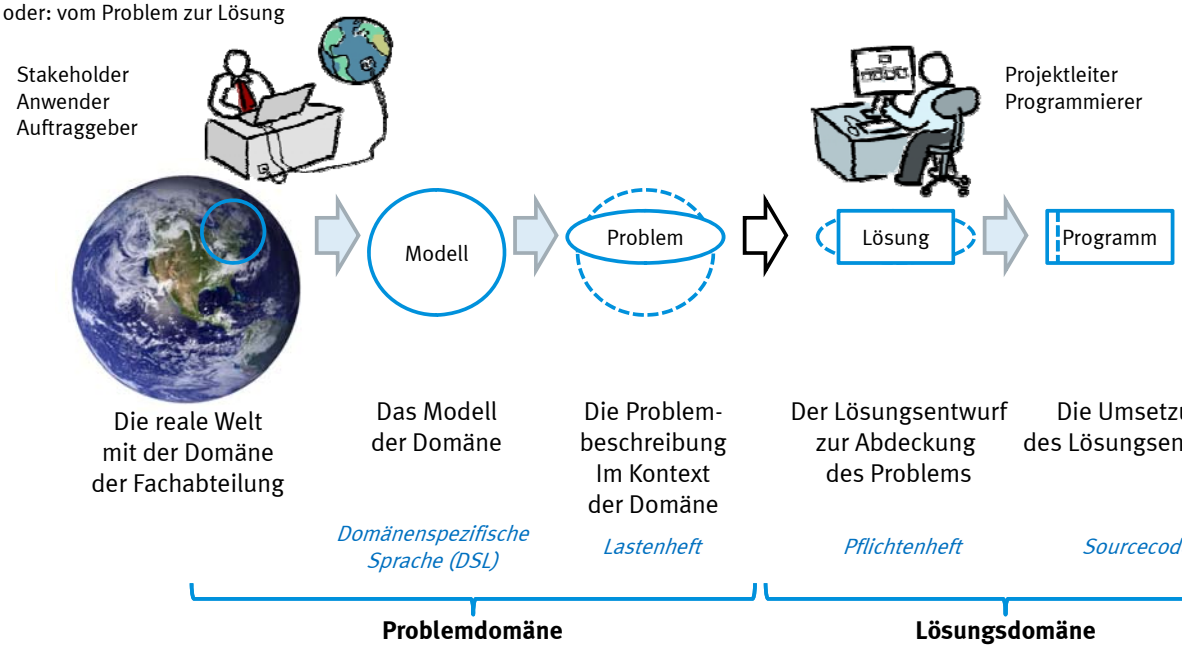
=› Zielgerichtet und angemessen handeln





Domänen und ihre Sprachen

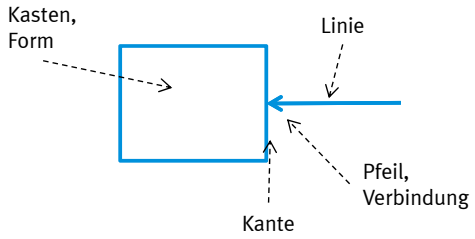
oder: vom Problem zur Lösung



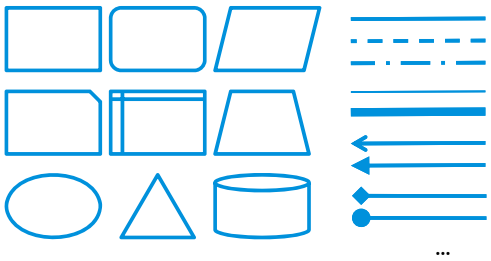
von Kästchen und Linien

BPMN
UML

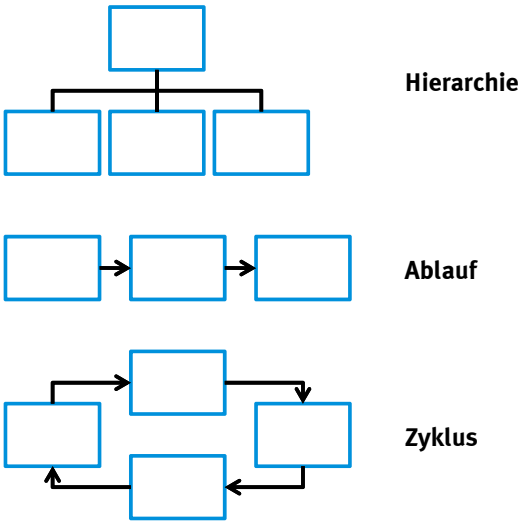
...oder „von Kanten und Pfeilen“:



Typen (Auswahl):

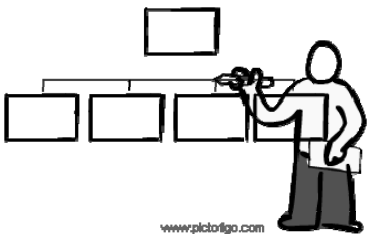


Anordnungen (typisch):



von Notationen und Syntax

Textuell formalistische Beschreibungen:
Lastenheft, Pflichtenheft, Anforderungen, Aufgaben, Definitionen, Umsetzungsenwürfe, Sourcecode, Kommentare, Dokumentationen.



Zielsetzung bei allen Diagrammen und textuellen Beschreibungen:

Welchen Grad der Vollständigkeit wollen wir erreichen?

Nur zur Dokumentation?	-› Ablage = Papierkorb!
Verständlich für Auftraggeber und Programmierer?	**-› praxisgerechter Ansatz**
Für automatische Codegenerierung?	-› SWE II, theoretische Informatik

Zusammenfassung:

- Unser Ziel ist: Erfolgreich bessere Software erstellen!
- Wir müssen uns mit dem Phasenmodell der SW-Entwicklung auseinandersetzen
- Nicht jede neue Technologie (Hype) ist auch sofort einsetzbar.
- Fachabteilung (Anwender) und IT-Abteilung (Programmierer) verstehen sich nicht
  -› Anforderung an uns (IT-Abteilung): Verstehen lernen!
- Das Scheitern von SW-Projekte hängt von verschiedenen Faktoren ab
  (Abgrund zwischen Fachabteilung in IT, Projekt-Multitasking, Qualität, Ergebnis, Aufwand, Zeit)
- Eine vollständige 1:1 Abbildung von Problem zur Lösung ist meist nicht möglich
  -› problemgerechte Annäherung anstreben
- Grad der Vollständigkeit von Formalismen (grafisch oder textuell)
  -› praxisgerechter Ansatz: für Anwender und Programmierer verständlich und ausreichend

Fragen

Bilder von:
pictofigo
www.pictofigo.com