

Vorlesung "Softwaretechnik"

Einführung

Lutz Prechelt

Freie Universität Berlin, Institut für Informatik

- SWT: Probleme/Aufgaben/Ziele
 - Klären OB, WAS, WIE
 - Realisieren, validieren
- Was hat das mit mir zu tun?
- Kosten, Nutzen, Qualität
- Grobunterteilungen der SWT:
 - Prozess, Produkt
 - Prinzip, Methode, Werkzeug
- menschliche/technische Aspekte
- Arten von SWT-Situationen
 - Art der Benutzer
 - Größe des Projekts
 - Rolle von Domänenwissen, Näherungslösungen, Effizienz, Verlässlichkeit

- Hauptquelle zur Vorlesung:
Bernd Brügge, Allen Dutoit:
Objektorientierte Softwaretechnik mit UML, Entwurfsmustern und Java
Pearson Studium.
 - Derzeit nur auf Englisch oder gebraucht zu kaufen.
 - Deckt aber nur ca. 1/3 unseres Stoffes ab.
- Pragmatisches und praktisch orientiertes Buch.
- Jedes andere Softwaretechnik-Lehrbuch ist aber ebenfalls hilfreich
 - als 2. Perspektive

- Organisatorische Details siehe Veranstaltungswebseite
- **Miniskript**
 - siehe Webseite



Was ist Software?

- **Software:**

Menge von Programmen oder Daten zusammen mit begleitenden Dokumenten, die für ihre Anwendung notwendig oder hilfreich sind

- Ein Begriffssystem für die Softwaretechnik /Hesse et al. 84/

- **Software:**

Computer programs, procedures, rules, and possibly associated documentation and data pertaining to the operation of a computer system

- IEEE Standard Glossary of Software Engineering Terminology /ANSI 83/.
- *To pertain to:* betreffen, sich beziehen auf

Was ist Softwaretechnik (SWT)?

Es gibt viele Definitionen. Hier zwei Beispiele:

- **Software Engineering:**

The systematic approach to the development, operation, maintenance, and retirement of software

- IEEE Standard Glossary of Software Engineering Terminology /ANSI 83/
- *approach*: (hier:) Behandlung eines Themas, Methode

- **Softwaretechnik (syn.: Software Engineering):**

Fachgebiet der Informatik, das sich mit der Bereitstellung und systematischen Verwendung von Methoden und Werkzeugen für die Herstellung und Anwendung von Software beschäftigt

- /Hesse et al. 84/

Andere Definitionen erwähnen zusätzlich z.B. folgende Aspekte:

- Ingenieurmäßiges Vorgehen (d.h. auf Basis wissenschaftlicher Erkenntnisse), evtl. auch Quantifizierbarkeit
- Herstellung schließt ein:
 - Ermittlung der Anforderungen
 - Pflege und Fortentwicklung
- Meist ändern sich die Anforderungen bereits während der Herstellung ("Agile Software-Entwicklung")

Welche Probleme versucht die SWT zu lösen?

- "Am Schluss hat sich rausgestellt, dass wir zur Hälfte Sachen gebaut hatten, die der Kunde nicht brauchte, aber dafür mit dem Rest nie genau das richtige."
 - → Anforderungsbestimmung
- "Wir haben den Projektplan um zwei Drittel überzogen, weil das Produktmanagement noch während des Projekts andauernd Änderungen verlangt hat."
 - → Anforderungsbestimmung, Anforderungsmanagement, Projektmanagement
- "Die Fehler, die uns am meisten Zeit gekostet haben, haben wir alle schon ziemlich früh gemacht."
 - → Anforderungsbestimmung, Software-Entwurf

Welche Probleme versucht sie zu lösen? (2)

- "Die Hälfte der Zeit beim Implementieren geht dafür drauf, dass die Leute irgendwas untereinander abstimmen."
 - → Software-Entwurf
- "Die meisten Defekte sind keine Programmierfehler, sondern Missverständnisse."
 - → Software-Entwurf, konstruktive Qualitätssicherung
- "Wenn *der* das Team verlässt, können wir wieder von vorn anfangen."
 - → Entwurfsdokumentation, Projektmanagement

Welche Probleme versucht sie zu lösen? (3)

- "Unsere Software wäre wirklich gut, wenn sie nur nicht so wahnsinnig viel Speicher bräuchte."
 - → Software-Entwurf, Implementierung
- "Das Schlimmste ist, dass wir beim nächsten Projekt wieder die gleichen Sachen falsch machen werden."
 - → Prozessmanagement
- "Unsere Software ist wie Bananen: Sie reift beim Käufer."
 - → Qualitätssicherung
- "Termingerecht fertig gestellte Software gibt es nicht."
 - → Projektmanagement

- Helfen zu klären **OB** (Kosten/Nutzen-Analyse)
 - Was könnte SW hier leisten? Und welchen Nutzen brächte das?
 - Was würde sie kosten?
- Klären **WAS** (Anforderungen)
 - Was sollte die SW tun? (funktionale Anforderungen)
 - Welche sonstigen Eigenschaften sind nötig? (nichtfunktionale A.)
- Klären **WIE** (Entwurf)
 - Wie sollte die SW aufgebaut sein? (Architektur, Modulzerlegung)
- **Bauen** (Implementierung)
- **Validieren** der Lösung (Analytische Qualitätssicherung)
 - Hat die SW die erwünschten Eigenschaften?
- **Verwalten** des Vorgehens (Management)
 - Den ganzen Prozess, mit dem obige Aufgaben gelöst werden, gestalten (z.B. planen, Qualitätsmängeln vorbeugen etc.)

Alle 6 Aufgaben sind eng miteinander verwoben

Wer ist beteiligt? ("Stakeholders")

- Kund/inn/en
 - das sind die Leute, die das Geld geben
 - evtl. viele, mit unterschiedlichen Bedürfnissen
- Benutzer/innen
 - das sind die Leute, die später die Software einsetzen
 - nur manchmal mit Kund/inn/en identisch
 - meistens viele, mit unterschiedlichen Wünschen/Bedürfnissen
- Manager/innen
 - das sind die Leute, die während der Entwicklung die organisatorischen Entscheidungen fällen
 - hoffentlich nur eine/r (aber meist mit weiteren Vorgesetzten)
 - Alternative: Selbstorganisation
- Entwickler/innen
 - das sind die Leute, die die Software definieren und bauen
 - in der Regel mehrere ("Programmieren im Großen")
 - in **verschiedenen Rollen** (d.h. m. verschiedenartigen Aufgaben)

u.a.m.

Was hat das alles mit mir zu tun? (Gute Zeiten)

- Mögen Sie Freizeit?
 - Gehen Sie gern mal aus oder sitzen mit Freunden zusammen?
 - Möchten Sie eines Tages Ihre Kinder beim Aufwachsen begleiten?
- Möchten Sie auch an etwas anderes denken können als an Software?
- Haben Sie gern Erfolgserlebnisse?
 - Oder wäre Ihnen egal, wenn das, was Sie die letzten 18 Monate lang erarbeitet haben, in den Papierkorb geworfen wird?
 - Oder wenn Sie an vielen Arbeitstagen keinerlei greifbare Fortschritte erzielen?

Richtig (oder eben falsch) angewandte Softwaretechnik entscheidet erheblich mit über Ihre Lebensqualität!

Was ich Ihnen mit dieser Vorlesung sagen möchte:

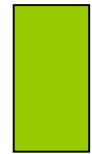
- **Beherrschung von Technologie ist nicht ausreichend, um Probleme zu lösen**
 - Sie ist nötig, aber nicht ausreichend
- **Beherrschung von Technologie ist nicht ausreichend für beruflichen Erfolg**
 - Sie ist wichtig und hilfreich, aber nicht ausreichend

Was hat das alles mit mir zu tun? (2)

- Indische SW-Entwickler verdienen ca. US\$ 14.000/Jahr

<https://www.averagesalarysurvey.com/software-developer/india>

- Sind genauso gut ausgebildet wie Sie es sind
 - Vielleicht besser: Die indische Firma Tata TCS war die erste der Welt, die firmenweit CMMI-Prozessreifestufe 5 erreicht hat
- Deutsche verdienen ca. das Vierfache



- Wenn Sie nur Technologie können, ist das direkte Konkurrenz
 - Ebenso Leute in Polen, Tschechien, Ungarn, China, Russland, Ukraine, Rumänien, etc.

- Was könnten Sie demgegenüber für Vorteile bieten?
 - Sie verstehen die deutsche Kultur. (*Tun Sie das?*)
 - Sie verstehen deutsche Geschäftsprozesse (*Tun Sie das?*)
 - Sie stärken einen Weltmarktführer vor Ort (*Verstehen Sie die?*)
 - "[Hidden Champions](#)" (Hermann Simon)

- Sie werden **nicht** dafür bezahlt
 - dass Sie einen Hochschulabschluss haben
 - dass Sie etwas wissen
 - dass Sie irgendwelche Probleme lösen
- sondern dafür
 - dass Sie solche Probleme lösen, bei denen der Nutzen der Lösung höher ist als die Kosten der Lösung
 - am besten sogar viel höher
- Also lernen Sie bitte:
 1. Probleme und ihre Wichtigkeit verstehen
 2. Probleme *effizient* lösen
 3. Kosten und Nutzen von Technologie und Methoden abschätzen.

Urteilskraft!

- Die Softwaretechnik war für eine Software dann gut, wenn
 - der erzielte **Nutzen** viel höher ist als
 - die aufgewendeten **Kosten**
 - und das auch künftig weiter gilt (bis die SW in Ruhestand geht)
- (Dies gilt für professionell durchgeführte SW-Entwicklung,
 - die wirtschaftlichen Maßstäben unterliegt.
 - Auch für viele Open-Source-Entwicklungen.)

Wie maximiert man den **Nutzen**?

- **Wertvolle Anforderungen** aufdecken und umsetzen
- dazu passende SW **hoher Qualität** produzieren

Wie minimiert man die **Kosten**?

- Software mit möglichst wenig Arbeitsaufwand fertigstellen
 - Schlüsseltechniken: **Wiederverwendung, Qualitätssicherung, Risikomanagement**
- Künftige Kosten vermeiden
 - keine hohen Kosten für **Defektkorrekturen**
 - erreicht durch hohe Qualität
 - keine hohen Kosten bei **späteren Änderungen**
 - erreicht durch gute Wartbarkeit, ebenfalls ein Aspekt von Qualität
- **Qualität spielt also eine zentrale Rolle in der SWT**

Externe Qualitätseigenschaften (d.h. aus Benutzersicht)

- Benutzbarkeit
 - Bedienbarkeit, Erlernbarkeit, Robustheit, ...
- Verlässlichkeit
 - Zuverlässigkeit, Verfügbarkeit, Sicherheit, Schutz
- Brauchbarkeit
 - Angemessenheit, Geschwindigkeit, Skalierbarkeit, Pflege, ...
- ...

(Man kann diese Listen auch deutlich anders machen.)

Interne Qualitätseigenschaften (d.h. aus Entwicklersicht)

- Zuverlässigkeit
 - Korrektheit, Robustheit, Verfügbarkeit, ...
- Wartbarkeit
 - Verstehbarkeit, Änderbarkeit, Testbarkeit, Korrektheit, Robustheit
 - dafür nötig sind z.B. Strukturiertheit, Dokumentiertheit, Flexibilität, etc.
- Effizienz
 - Speichereffizienz, Laufzeiteffizienz, Skalierbarkeit
- ...

Interne Q. sind Mittel zum Zweck!

Wie schafft man hohe Qualität?

- Analytische Qualitätssicherung
 - **Produktorientiert:** Konkrete Qualitätsmängel aufdecken und nachbessern
 - Statisch durch: Durchsichten, Inspektionen, Modellprüfung etc.
 - Dynamisch durch: Test
- Konstruktive Qualitätssicherung
 - **Prozessorientiert:** Qualitätsmängel von vornherein durch das Vorgehen vermeiden
 - "Vorbeugen ist besser als Heilen"
 - Durch: Prozessmanagement



Keine Gegensätze: Ergänzen sich!

Was bewirkt schlechte Softwaretechnik?

- Aus Sicht der Kund/inn/en:
 - Geldverschwendung
 - Bei Firmen zusätzlich: Risiko für das Unternehmen, Abnahme der Konkurrenzfähigkeit
- Aus Sicht der Benutzer/innen:
 - Frustration
 - Produktivität/Sicherheit/Komfort niedrig
- Aus Sicht der Entwickler/innen und Manager/innen:
 - Überstunden
 - Frustration, Zerstörung der Motivation
 - Störung der beruflichen Fortentwicklung



1. Nach **Perspektive**

- Produkt und Prozess

2. Nach **Lebensdauer des Wissens**

- Prinzip, Methode, Verfahren, Werkzeug

3. Nach **Art der Aktivität**

- Anforderungsermittlung, Entwurf, Implementierung, Validierung, Inbetriebnahme, Management

4. Nach **betroffenem Bereich**

- Technische Aspekte, menschliche Aspekte

- Siehe nachfolgende Folien (nur zu 1, 2, 4)

- Details siehe im Miniskript

Größte und gebräuchlichste Unterteilung der SWT

Produkt:

- Gesamtheit der greifbaren **Arbeitsergebnisse**
 - meist genannt "Dokumente" oder "Artefakte"
 - z.B. Programmcode, Code für Test/Build/Installation etc., Benutzerdokumentation, interne technische Dokumentation, etc.

Prozess:

- 1. Für ein Projekt: Gesamtheit aller konkreten **Abläufe**
- 2. Allgemein: Gesamtheit der zugrunde liegenden allgemeinen Konzepte u. Verfahrensweisen ("Prozessmodell")
 - Rollen, Aktivitäten, Methoden, Werkzeuge, etc.

Bitte nie miteinander verwechseln!

- Prinzip: Ein Grundsatz, an dem man sein Handeln orientiert
 - Sehr abstrakt; sehr allgemein; allgemein gültig
 - **Bleibt lebenslang richtig und nützlich!**
- Methode: Planmäßig angewandte und begründete Vorgehensweise zum Erreichen festgelegter Ziele
 - Abstrakt; meist auf einen Bereich spezialisiert
 - Bleibt **einige Jahre** (ca. 10-30) lang relevant
- Verfahren: Präzise und recht konkrete Vorgehensvorschrift
 - Konkret oder leicht konkretisierbar; für sehr spezifischen Bereich; mit spezifischem Ergebnis und Zweck
- Werkzeug: Computerprogramm, das ein Verfahren automatisiert oder eine Methode unterstützt
 - Bleibt evtl. nur **kurze Zeit** relevant

1. Softwaretechnik wird von Menschen betrieben

- Menschliche Eigenschaften sind wichtige Treiber für die Softwaretechnik
 - Alle Beteiligten: Entwickler, Benutzer, Kunden, Manager (und -innen)
- z.B. haben Menschen Beschränkungen (etwa bei Wissen, Kurzzeitgedächtnis, Kommunikationsvermögen), die entscheidend für die Gestaltung der Methoden der SWT sind
- Außerdem zeigen Menschen variables Verhalten: sie sind manchmal ängstlich, aggressiv, beeinflussbar, einfalllos, eitel, faul, fleißig, ignorant, kreativ, müde, mutig, souverän, redselig, schweigsam, wachsam, u.a.m.

2. Software wird von Menschen benutzt

- → sozio-technisches System
- Beide Aspekte werden von vielen Softwareingenieuren und Projektmanagern leider gern ignoriert
 - **Technische Aspekte sind nicht alles!**

Welche Arten von SWT-Situationen gibt es?

Warum will man das wissen?

- Die meisten Methoden der Softwaretechnik sind nur für manche Situationen oder Softwarearten ideal
 - Und für die anderen problematisch oder gar unbrauchbar
- **Bitte immer im Auge behalten!**

Einteilung nach mehreren Kriterien (siehe nächste Folien):

- Nähe und Anzahl der Kunden
- Art der Benutzer, Art der Benutzung
- Größe/Komplexität der SW und des Projekts
- Wie kritisch ist (nichttechnisches) Domänenwissen?
- Müssen Näherungslösungen verwendet werden?
- Wie kritisch ist Effizienz?
- Wie kritisch ist Verlässlichkeit?

- Fall 1: Individualsoftware
 - Es gibt nur einen Kunden. Dieser gibt die SW in Auftrag.
 - Der Kunde ist konkret und meist für Auskünfte über die Anforderungen wenigstens einigermaßen zugänglich.
 - Beispiel: Betriebliches Informationssystem
- Fall 2: Standardsoftware oder eigener Web-Dienst
 - Es gibt (potentiell) viele Benutzer und Kunden. Evtl. Millionen.
 - Sehr viele der Benutzer sind selbst Kunden.
 - Diese sind sehr verschieden in Bezug auf Wissen, Wünsche, Vorlieben, Bedürfnisse, Geduld, Risikofreude, Lern- und Innovationsbereitschaft.
 - Es ist schwierig, dieses Spektrum gut genug zu verstehen, um die "richtigen" Anforderungen für die SW zu ermitteln.
 - Beispiele: Computerspiel; SAP ERP; Textverarbeitung; öffentliche Webanwendungen

Art der Benutzer:

- Fall 1: Geschulte, professionelle Benutzer
 - Beispiel: SW-Ingenieure (für ein API-basiertes SW-Produkt)
 - Beispiel: Fluglotsen (für ein Flugsicherungssystem)
- Fall 2: Gering oder gar nicht ausgebildete Benutzer
 - Beispiele: MS Word, Pkw-Navigationssystem
 - Solche SW ist viel schwieriger gut hinzubekommen:
Erfordert gutes Verständnis für Benutzbarkeitsgestaltung

- Fall 1: Klein
 - z.B. zwei Entwickler für wenige Wochen
 - Viele Methoden der SWT sind unnötig (evtl. gar kostenschädlich)
 - Verletzung der *notigen* Methoden macht den Erfolg zwar mühselig, aber nicht unmöglich
- Fall 2: Groß (evtl. sogar verbunden mit HW-Entwicklung)
 - z.B. Hunderte (oder Tausende) Entwickler über mehrere Jahre
 - Kosten z.B. 10.000 Personenjahre, entspricht ca. 1 Milliarde EUR
 - Beispiele: MS Windows, SAP ERP, Lucent 5ESS, Linux-Kern
 - Kompetent angewandte SWT ist die einzige Chance für Erfolg
 - Eine riesige Palette von Methoden ist zu erwägen
 - Projektmanagement wird extrem bedeutsam
- Und natürlich ein Kontinuum dazwischen
 - Ferner wichtig: Lebensdauer der SW

Wie kritisch ist (nichttechnisches) Domänenwissen ?

- Fall 1: Eine rein softwaretechnische Domäne
 - Beispiel: Betriebssystemkern, Compiler
 - Hier verstehen die SW-Ingenieur/inn/e/n ohne fremde Hilfe die Anforderungen recht gut
 - Fall 2: Eine Domäne mit komplexer Fachlichkeit
 - Beispiel: SW zur Berechnung des Kreditausfallrisikos einer Bank
 - Hier sind die SW-Ingenieur/inn/e/n bei den Anforderungen naiv
 - Die Anforderungen müssen von Fachspezialisten geliefert werden
 - Kommunikationsproblem!
- Faustregeln:
 1. Wenn es eine interaktive Benutzungsschnittstelle hat, ist es nie rein technisch
 2. Die Domäne ist immer vielschichtiger als man denkt

Müssen Näherungslösungen verwendet werden?

- Fall 1: Nein, nirgends
 - Man kann die funktionalen Anforderungen genau erfüllen (zumindest im Prinzip)
 - Das ist bislang der Normalfall
- Fall 2: Niemand weiß, wie man das Problem komplett und präzise lösen kann
 - Beispiele: Spracherkennung, Video-nach-Text-Transkription, autonomes Fahrzeug, alles was "intelligent" ist
 - Solche Aufgaben sind bei der Qualitätssicherung schwieriger und verlangen andere Herangehensweisen

Wie kritisch ist Effizienz?

- Fall 1: Hauptspeicher, Massenspeicher, Netzbandbreite, Akkuleistung und CPU-Leistung sind im Überfluss vorhanden
 - Beispiel: Textverarbeitungsprogramm auf modernem PC
- Fall 2: Manche Ressourcen sind knapp und müssen sparsam eingesetzt werden
 - Beispiel: SW für ein billiges Gerät (z.B. 20-EUR-Handy)
 - Beispiel: SW für ein mobiles Gerät (Akkukapazität!)
 - Beispiel: SW für eine numerische Simulation (z.B. Wetter, Klima)
 - Beispiel: SW für Echtzeitanwendungen (z.B. Computerspiel, Fabrikationssteuerung)
 - Beispiel: SW f. Systems-of-Systems
 - Erfordert ganz anderen Softwareentwurf

Wie kritisch ist Verlässlichkeit?

- Verlässlichkeit =
Zuverlässigkeit + Sicherheit + Verfügbarkeit + Schutz
- Fall 1: Wenn die SW versagt, kann ein Mensch das abfedern
 - Beispiel: Fehlerhafte Silbentrennung in der Textverarbeitung
 - Beispiel: Unsinnige Vorschläge eines Routenplaners
 - Diesen Fall gibt es bestenfalls graduell.
Fast immer gilt mehr oder weniger stark:
- Fall 2: Wenn die SW versagt, geschieht etwas Schlimmes
 - Beispiel: Autopilot eines Verkehrsflugzeugs
 - Beispiel: Abstands-Tempomat im Auto
 - Beispiel: Kontoführungssoftware einer Bank
 - Beispiel: Steuerung einer automatischen Insulinpumpe



msn.com 2005:
Hagesund → Trondheim

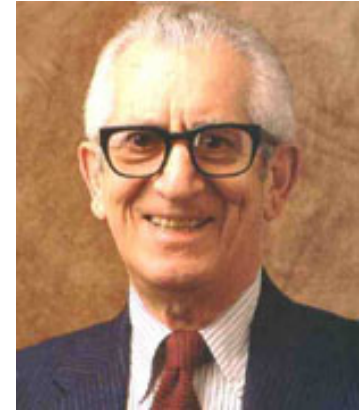
Reprise: Welche Arten von SWT-Situationen gibt es?

- Nähe und Anzahl der Kunden
- Art der Benutzer, Art der Benutzung
- Größe/Komplexität der SW und des Projekts
- Wie kritisch ist (nichttechnisches) Domänenwissen?
- Müssen Näherungslösungen verwendet werden?
- Wie kritisch ist Effizienz?
- Wie kritisch ist Verlässlichkeit?

Wichtig, denn:

- Die meisten Methoden der Softwaretechnik sind nur für manche Situationen oder Softwarearten ideal
 - Und für die anderen problematisch oder gar unbrauchbar
- **Bitte immer im Auge behalten!**

- Taxonomie kognitiver Lernziele in Bezug auf einen Lerngegenstand (nach Bloom et al.):



- | | |
|-------------------|----------------------------------|
| 1. Kennen | Gleich wiedergeben |
| 2. Verstehen | Anders wiedergeben |
| 3. Anwenden | Folgern in konkreten Fällen |
| 4. Analysieren | Zerlegen; Verstehen der Struktur |
| 5. Synthetisieren | Gezielt neue Strukturen bilden |
| 6. Beurteilen | Brauchbarkeit bewerten |
- Sind nicht streng hierarchisch (obwohl so gedacht)
 - Stufen 1 und 2 sind ingenieurmäßig weitgehend wertlos
 - Erfolgreiche SWT braucht an sehr vielen Stellen bei genügend vielen Beteiligten mindestens Stufe 5
 - und für Kostenminimierung sehr oft Stufe 6

Lernziele für diese Vorlesung

- Prinzipien und Zusammenhänge:
 - verstehen, anwenden, analysieren, synthetisieren, beurteilen
- Methoden:
 - verstehen, anwenden, analysieren (grob), beurteilen (teilweise)
- Werkzeuge (kommen kaum vor):
 - verstehen, anwenden
- Anders gesagt:
 - Schulen Sie Ihre **Urteilkraft** für die grundsätzlichen Fragen der Softwaretechnik
 - Verstehen Sie **das Wesen der Dinge** im Lehrstoff, die Details sind weniger von Interesse
 - In der Mathematik:
"Kann man sich bei Bedarf herleiten"
 - In der Softwaretechnik:
"Will man konkret sowieso oft anders machen"

Nicht-Lernziele für diese Vorlesung

- Kenntnis aller Details

Nachbemerkung: Lernstil

- Im Studium sollten Sie vor allem verstehen:
 - Theoretische und konzeptionelle Grundlagen
 - Deren Eigenschaften, Vor-/Nachteile, Anwendungsbereiche
 - Zusammenhänge
- NICHT übermäßig wichtig ist:
 - Ihre genaue Abschlussnote
- Warum?
 - Ihr Berufsleben dauert ca. 40 Jahre
 - In dieser Zeit veraltet ihr Spezialwissen mindestens 4 mal
 - Grundlagenwissen bleibt wirksam und hilft beim Weiterlernen
 - Abschlussnote interessiert nur bei der ersten Bewerbung erheblich

- Lernen Sie:
 - Generische Konzepte, Grundlagen
 - Kriterien f. Anwendbarkeit/Nützlichkeit: Urteilskraft
 - Ggf. aktiv nachfragen bei Dozenten
 - Übertragen von Wissen in neue Gebiete
 - Technisches Detailwissen auf wenigen Gebieten
 - Gebiete, die Ihnen Spaß machen
 - Da aber massenhaft
- Beste Methode dafür:
 - Glauben Sie nichts, hinterfragen Sie alles
 - "Stimmt das wirklich?", "Warum?", "Immer?"
 - Suchen Sie Lücken, Fehler und Unsauberkeiten
 - Diskutieren Sie Abwägungen mit Anderen
 - Praxis, Praxis, Praxis!

Eigenverantwortung!

Danke!

Diesen Foliensatz bitte nacharbeiten!