

Software Engineering I

1. Einführung

Prof. Dr. Eckhard Kruse

DHBW Mannheim


Organisation

- Software Engineering I: WS+SS 2022/2023
 - Vorlesung: 96 h (WS 6 h, SS 4 h pro Woche, inkl. je 1h begl. Selbststudium)
 - **Selbststudium: 174 h!**
 - Termine: s. Kalender, inkl. etwas Puffer
- Vorlesung + Übungen + Teamarbeit: Software-Engineering-Projekt
- Fragen: Am besten direkt in der VL
 - eckhard.kruse@dhbw-mannheim.de, Raum 344 B, Tel. (0621) 4105 1262
- Verteilung der Folien
 - nach jeder Vorlesung per E-Mail-Verteiler
- Leistungsnachweis
 - Programmentwurf
 - 1 Note (Ende SS): Gesamtergebnisse aus Projektarbeiten (Code, Doku, Präsentationen usw.), aus WS+SS (50/50)
- Literatur („further reading“), z.B.:
 - Ludewig, Lichter: *Software Engineering: Grundlagen, Menschen, Prozesse, Techniken*
 - Ian Sommerville: *Software-Engineering* (englisch)

Software macht's möglich ...

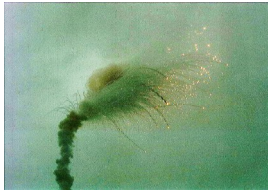


Software macht's möglich ...



4.6.1996: Fehlgeschlagener Start der ersten Ariane 5
Finanzieller Schaden: ca. 500 Mio. US\$

4. Juni 1996: Erster Start der Ariane 5



- Während des Fluges läuft ein unnötiges Kalibrierprogramm, welches von der alten Software der Ariane 4 übernommen wurde.
- Die gemessenen Werte der Ariane 5 überschreiten die in der Ariane-4-Software vorgesehenen Wertebereiche → Variablenüberlauf
- Die Exception wird erkannt → Fehlerbehandlung: Anhalten des Steuerungscomputers, Umschalten auf zweites redundantes System.
- Im zweiten System läuft die gleiche Software → identische Fehlerbehandlung ...

Die Softwarekrise ...

- Software wird immer komplexer.
- Der Aufwand für Softwareentwicklung und Testen steigt rasant.
- Kleine Fehler können große Folgen haben.

Folgen: Software wird nicht termingerecht fertig, Kosten laufen aus dem Ruder und die Software-Qualität ist schlecht.



Wir haben eine „Softwarekrise“!

Der Begriff „Softwarekrise“ wurde Ende der 1960er geprägt!

Die Softwarekrise geht weiter!

- Software wird immer komplexer.
- Der Aufwand für Softwareentwicklung und Testen steigt rasant.
- Kleine Fehler können große Folgen haben.

Folgen: Software wird nicht termingerecht fertig, Kosten laufen aus dem Ruder und die Software-Qualität ist schlecht.

- Software wird „überall“ eingesetzt → Fehler in technischen Systemen sind immer häufiger auf Software-Fehler zurückzuführen
 - Autos, Unterhaltungselektronik, Haushaltsgeräte, Telefone...
- Software wird immer wichtiger in sicherheitskritischen Anwendungen
 - Verkehr (fly/drive-by-wire, Automatisierung), Industrie (z.B. Chemie, Pharma, ...), medizinische Anwendungen usw.
- Fehlerhafte Software ist als Normalität akzeptiert:
 - Updates/Patches als Standard, Kunde=Tester
 - Fehlerfreies Produkt von Anfang an: ist das überhaupt noch ein Ziel?

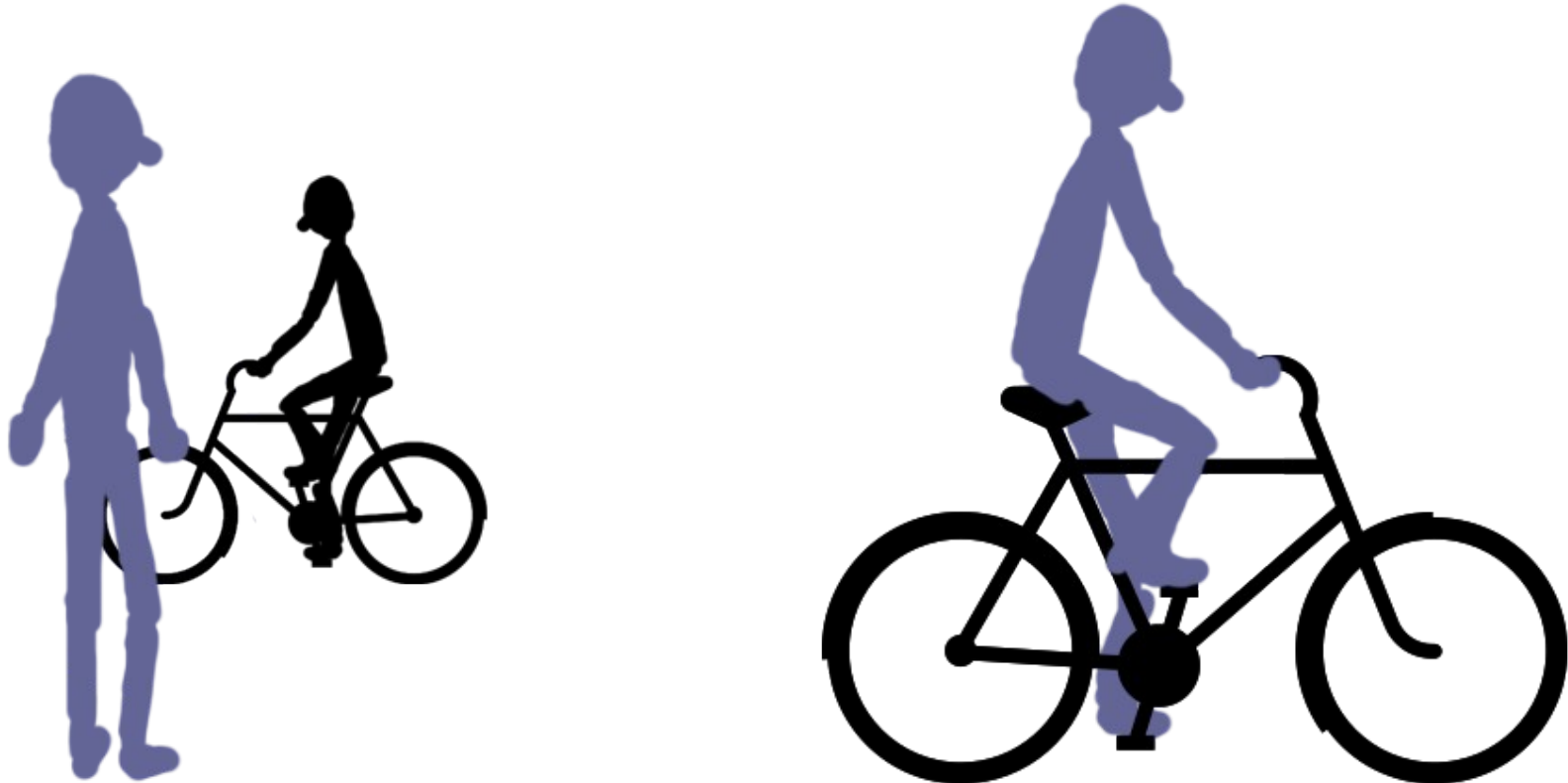
Software-Engineering als Rettung

Software-Engineering:

„Zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresystemen unter Berücksichtigung von Kosten, Zeit und Qualität“

[s. H. Balzert, Lehrbuch der Software-Technik, Spektrum Akademischer Verlag, Heidelberg 2001]

Wie lernt man Software-Engineering?

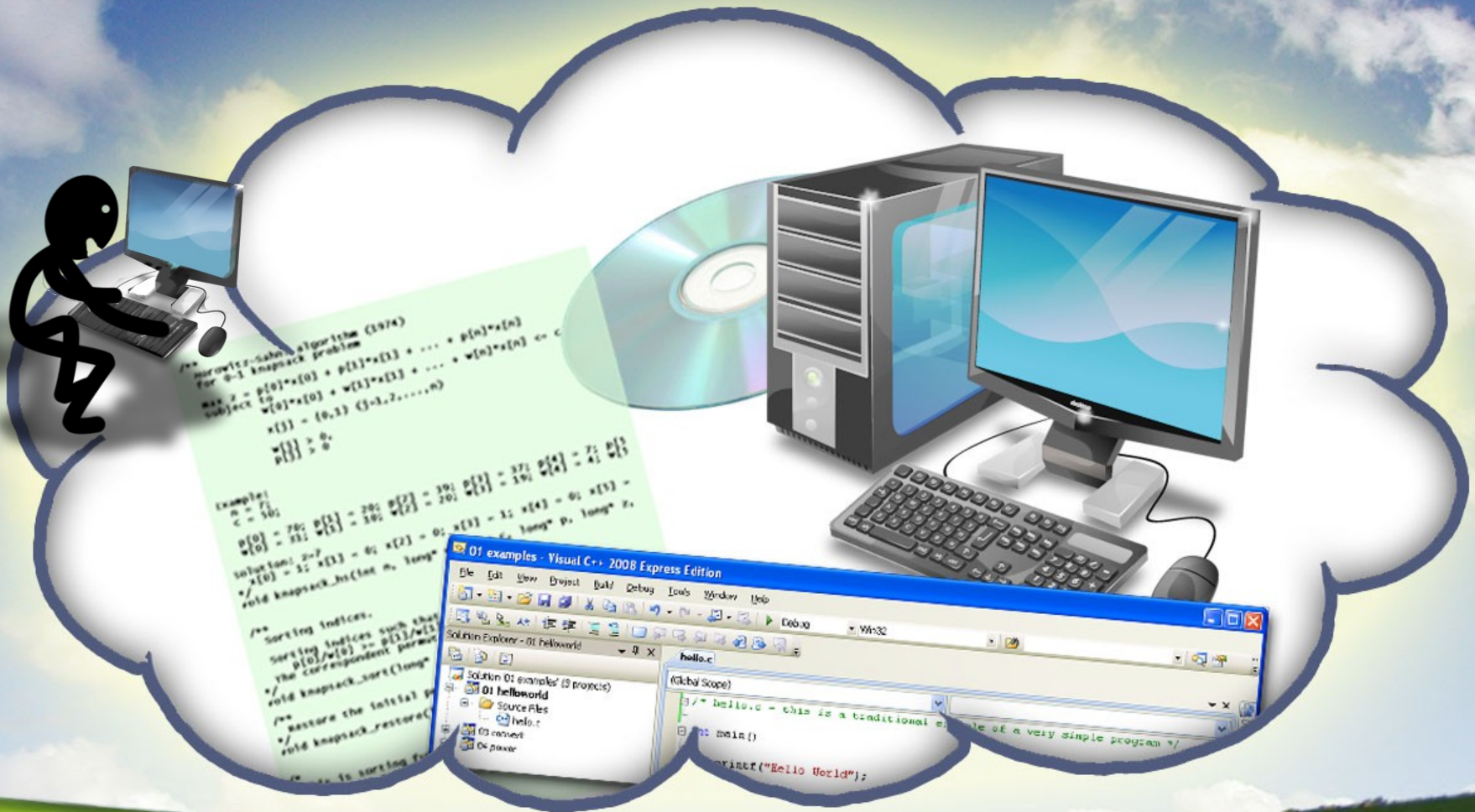


Eigentlich selbstverständlich...

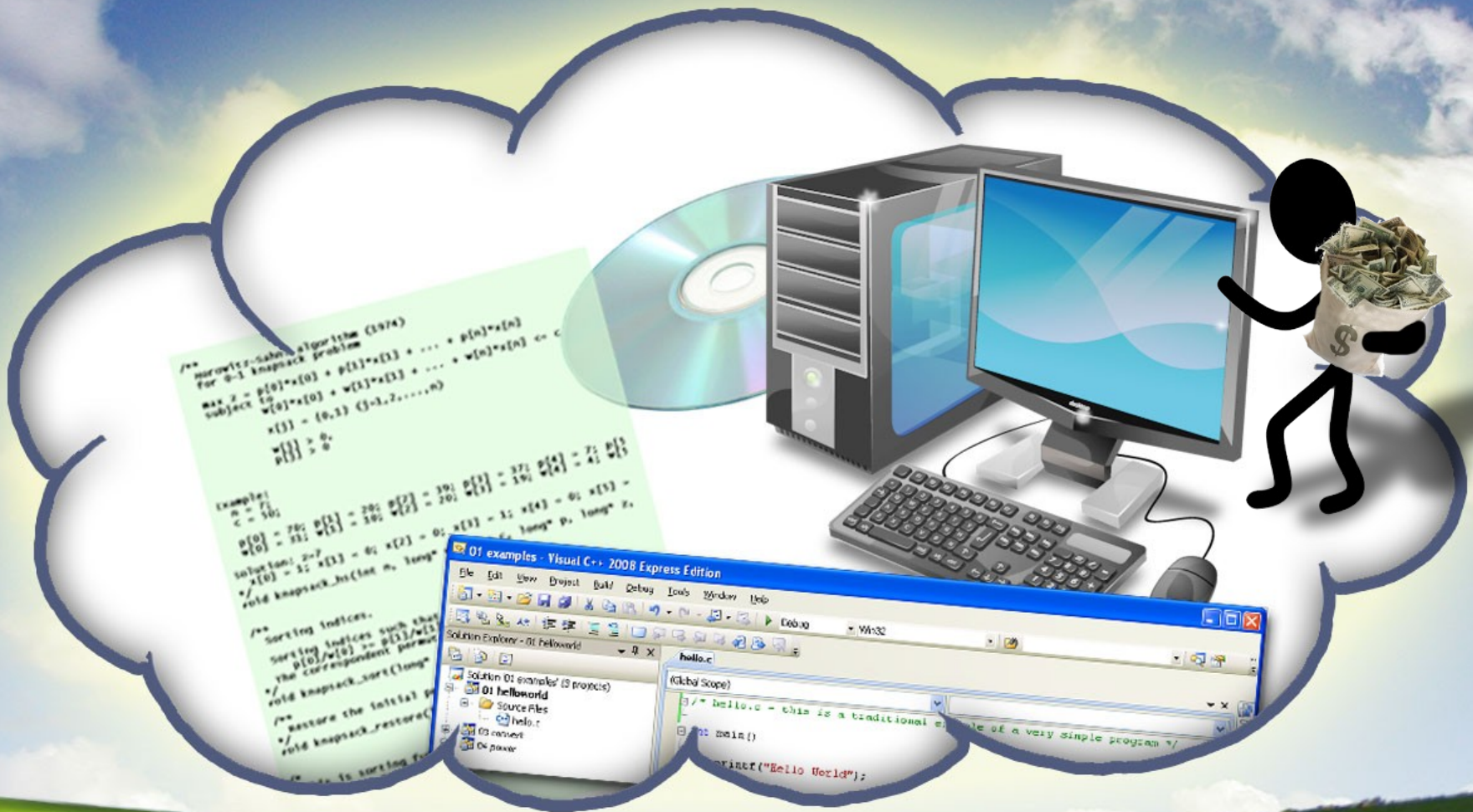
Bewährte Regeln für effizientes gemeinsames Arbeiten, Besprechungen usw. → gilt auch für diese Vorlesung:

- Pünktlichkeit (Vorlesungsbeginn, Pausenende)
- Anwesenheit: von Körper + Geist
- Anzahl gleichzeitig redender Personen ≤ 1 (Ausnahme Teamarbeit)
- Konzentration auf das Geschehen
 - Laptops nur ggf. zum Mitschreiben und für die Übungen
- Handys ausschalten, keine Telefonate
- ggf. Feedback zum Arbeitsprozess
 - Stoff zu schnell / zu langsam? Pausenbedarf?

Software-Engineering = Programmieren?



Was möchte eigentlich der Kunde?



Was möchte der Kunde?



Was möchte der Kunde?

Ein Programm?

Nein, eine zuverlässig funktionierende Lösung für seine Aufgaben!

z.B.

- *eine Möglichkeit für seine Kunden Zimmer bequem übers Internet zu buchen*
- *eine Steuerung seiner Produktion für maximalen Durchsatz und Qualität*
- *eine sichere, zentrale Verwaltung aller in der Firma anfallenden Daten.*
- ...
- *einen finanziellen Nutzen, der die Kosten der Software deutlich übersteigt.*

Klarer Business case

Return on investment

Qualität
(Bedienbarkeit,
Performance,
Zuverlässigkeit...)

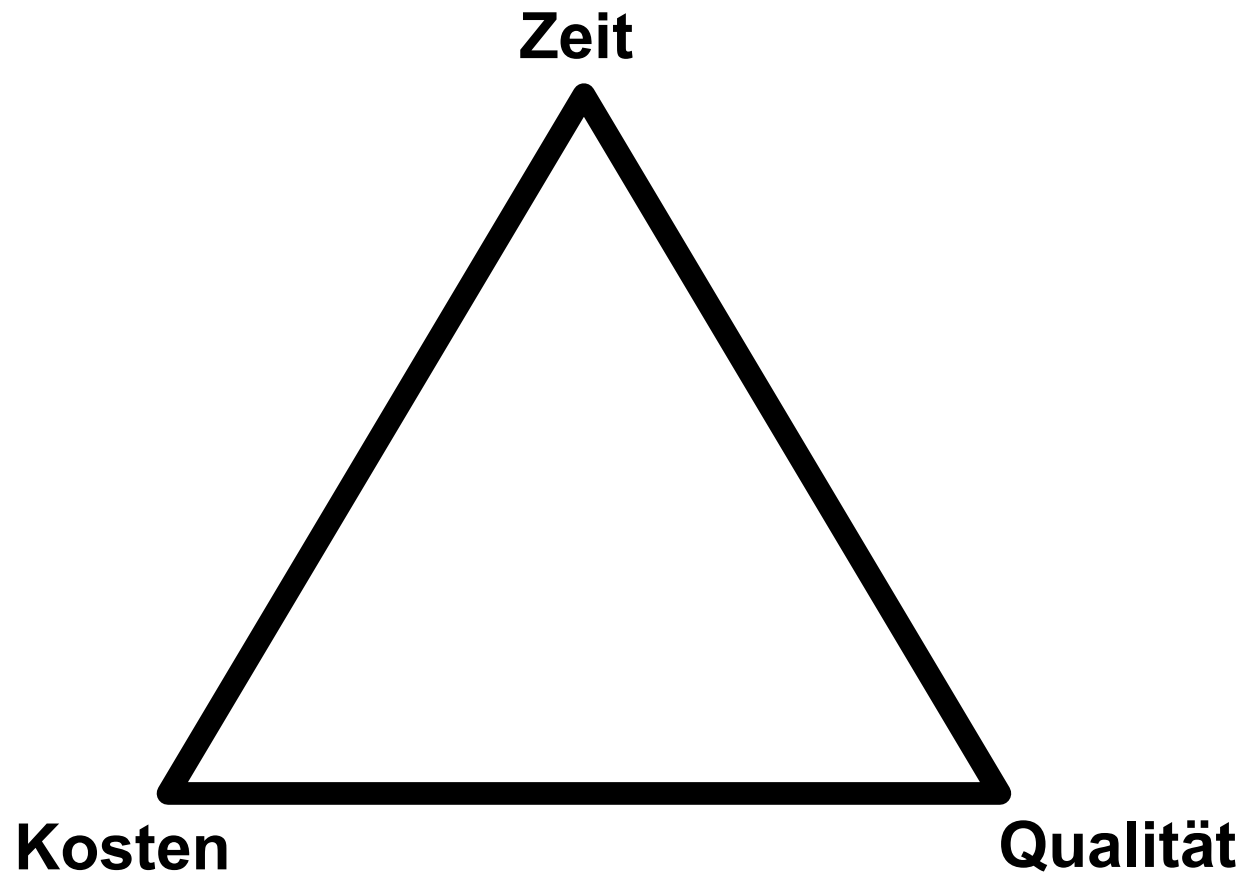
Total cost of ownership
→ Kosten über gesamte
Systemlaufzeit

Software-Engineering:

„**Zielorientierte** Bereitstellung und systematische Verwendung von Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresystemen **unter Berücksichtigung von Kosten, Zeit und Qualität**“

[s. H. Balzert, Lehrbuch der Software-Technik, Spektrum Akademischer Verlag, Heidelberg 2001]

Das magische Dreieck



Projektaufgabe - Was möchte der Kunde?



Software-Engineering-Projekt

P.1 Bildung von Projektteams

- a) Bilden Sie Projektteams für die Themen.
- b) Die Teams sind die Lieferanten für das gewählte Thema.
- c) Die Teams agieren außerdem als Kunden für ein anderes Team.
- d) Vorschlag: Benennen Sie pro Team einen Projektleiter für die Lieferantenrolle und einen für die Kundenrolle. (Kann ggf. auch noch während der Projektlaufzeit geändert werden.)
- e) Teamzusammensetzung per E-Mail an Dozenten.

Hinweise zur Teambildung:

- Sind die Teams ausgewogen? (Neben den Einzelteams sollte auch der Kurs zusammen als Team funktionieren.)
- Akzeptieren Sie Kompromisse. Im Berufsleben kann man sich seine Teams meist nicht aussuchen.

Übung

Das magische Dreieck

- a) Überlegen Sie, aus welchen anderen Lebensbereichen Sie das „magische Dreieck“ aus Kosten, Zeit, Qualität kennen.
- b) Überlegen Sie, wie sich folgende Entscheidungen in einem SW-Projekt auf die drei Faktoren auswirken könnten:
 - 1. Verdopplung des Umfangs der Systemtests
 - 2. Outsourcing in ein Niedriglohnland
 - 3. Erweiterung des Funktionsumfangs in einer frühen Projektphase
 - 4. Erweiterung des Funktionsumfangs in einer späten Projektphase
 - 5. Einstellung zusätzlicher, aber unerfahrener Mitarbeiter
 - 6. Zusätzliche, regelmäßige Treffen für Projektreviews mit dem Kunden
- c) Nennen Sie weitere Beispiele für potenzielle Entscheidungen und Abwägungen bzgl. der drei Faktoren – insbesondere im Hinblick auf die eigenen Projektideen.

http://en.wikipedia.org/wiki/Project_triangle

"... There are also numerous spinoffs to this triangle, the most common including:

- *College: Work, Sleep, Play – Pick two.*
- *Men: Handsome, High-Earner, Faithful – Pick two.*
- *Women: Single, Sane, Sexy, Smart – Pick any three. (also called The four S's of dating)*
- *Operating System: Fast, Efficient, Stable - Choose two.*
- *Bicycle Parts: Strong, Light, Cheap - Pick any two.*
- *Opensource Software Development: Speed/Time, Inclusiveness/Openness, Quality*
- *Schedule, Scope, Resources – Pick two.*

..."

Übung

Was ist ein gutes Team?

Nennen Sie Kriterien für ein gutes Projektteam.

- a) Warum muss eine Gruppe gleichgesinnter Freunde nicht notwendigerweise ein gutes Projektteam sein?
- b) Welche Kompetenzen sind für SW-Projekte wichtig und sollten im Team vorhanden sein?
- c) Schwieriges Team = viel Lernpotenzial. Erläutern Sie diese Aussage.

- Eine Gruppe gleichgesinnter Freunde muss nicht unbedingt ein optimales Projektteam sein.
 - Projektteams leben von Vielfalt und verschiedenem Wissens-/Interessenmix.
 - Außenstehende können neue Sichtweisen (gegen die Betriebsblindheit) einbringen.
 - zu viel Harmonie kann "einlullen" und Risiken werden möglicherweise nicht gesehen
- Wertvolle Kompetenzen: Analysieren, verhandeln, erklären, hinterfragen, dokumentieren, schlichten, energetisieren, programmieren, tüfteln, ...
- Im Berufsleben gibt es immer wieder schwierige Teams. Hier lässt sich ggf. das Arbeiten in solchen Teams in einem geschützten Rahmen (ohne schlimme Konsequenzen) erlernen.

Software:

- Alles 'Nichtphysische' im Computersystem: Computerprogramme, Prozeduren, Regeln, Dokumentation, Daten

Softwaresystem:

- Ein System ist ein Ausschnitt aus der realen Welt bestehend aus Teilen, die in Beziehung stehen
SW-System: Ein System, dessen Komponenten aus Software bestehen

Software-Produkt:

- Ein Produkt ist ein in sich abgeschlossenes, i.a. für einen Auftraggeber bestimmtes Ergebnis eines erfolgreich durchgeführten Herstellungsprojektes.
- SW-Produkt: Produkt, das aus Software besteht.

Zeitplan (Vorschlag)

Woche

- 1 Teams, Themen, Definition Kunden
- 1-2 Lastenheft
- 2-4 Vorstudie(n) + Mockups, Pflichtenheft → Auftrag
- 4-5 Entwurf/Architektur, (grobe) Projektplanung+Aufwandsschätzung
- 5-10 Moduldesign + Implementierung
- 8-10 Testspezifikation + Testen
- 11 Übergabe, Abgabe Projektstagebücher
(Die Arbeitsergebnisse gehen in die bewertete PL nach dem SS ein!)

Übung

Warum ist Software besonders?

Überlegen Sie, worin sich das Entwickeln von Software von traditionellen Engineeringprojekten, die der Entwicklung von „hardware-basierten“ Produkten dienen, unterscheiden könnte.

- a) Was sind die Besonderheiten des Endprodukts „Software“?
- b) Welche Arbeitsschritte während der SW-Entwicklung sind zusätzlich/spezifisch für Software, welche Schritte könnten entfallen?
- c) Was ist einfacher bei der Entwicklung von Software?
- d) Was könnte Software-Engineering schwieriger machen?

Softwarebesonderheiten

- Fertigung/Produktionskosten (= DVD brennen / auf Server zum Download stellen) spielen praktisch keine Rolle (und sind kein Kostenfaktor) → prinzipiell leicht änderbar.
- Qualität schwer messbar
- Zuverlässigkeit (von Komponenten) lässt sich nicht wie bei Hardware einfach in Ausfallraten (z.B. MTBF = mean time between failure) angeben.
- Praktisch nicht 100%ig testbar
- „Alterung“: Zugrunde liegende Betriebssysteme/Technologien entwickeln sich schnell → große Auswirkung auf Softwareprodukt. Kontinuierlicher Aufwand für Aktualisierung
- Patches / Updates über Internetdownload möglich

SW-Engineering I: Ziele

s. Studienplan:

- Theoretische Grundlagen des Software-Erstellungsprozesses kennen.
- Komplexe Problemstellungen analysieren und rechnergestützte Lösungen umsetzen und dokumentieren können.
- Projektphasenmodell und Methoden der Phasen kennen und anwenden können.
- Ergebnisse der jeweiligen Phasen in ihren Inhalten und Zielrichtungen erfassen und dokumentieren.
- Konkrete Ergebnisse innerhalb der einzelnen Projektphasen mit geeigneten Tools erarbeiten.
- Gruppendynamische Prozesse bei der Bearbeitung größerer Aufgaben innerhalb von Projektgruppen erfahren.

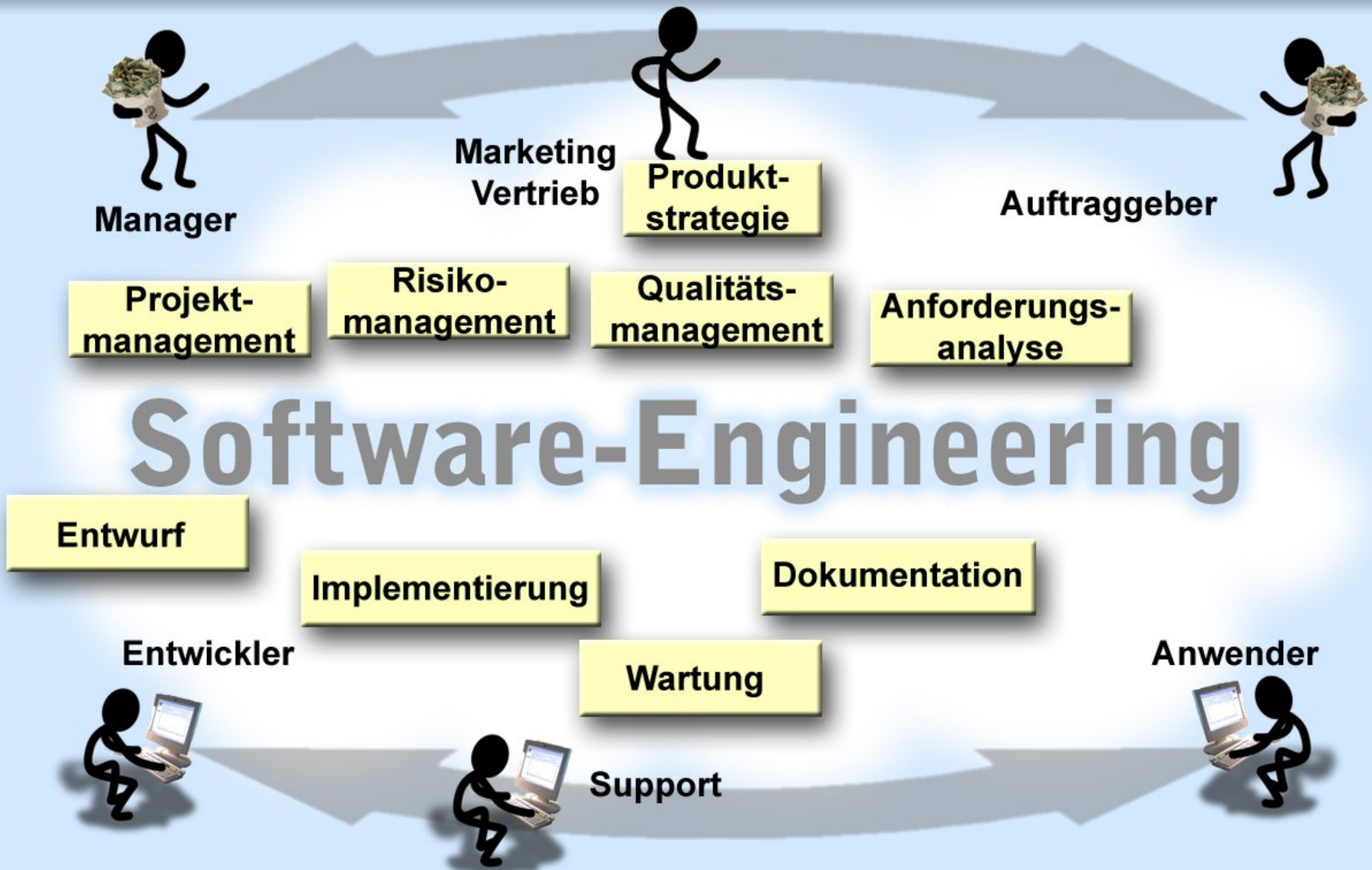
SW-Engineering Umfeld



Software-Engineering



SW-Engineering: Viele Aufgaben

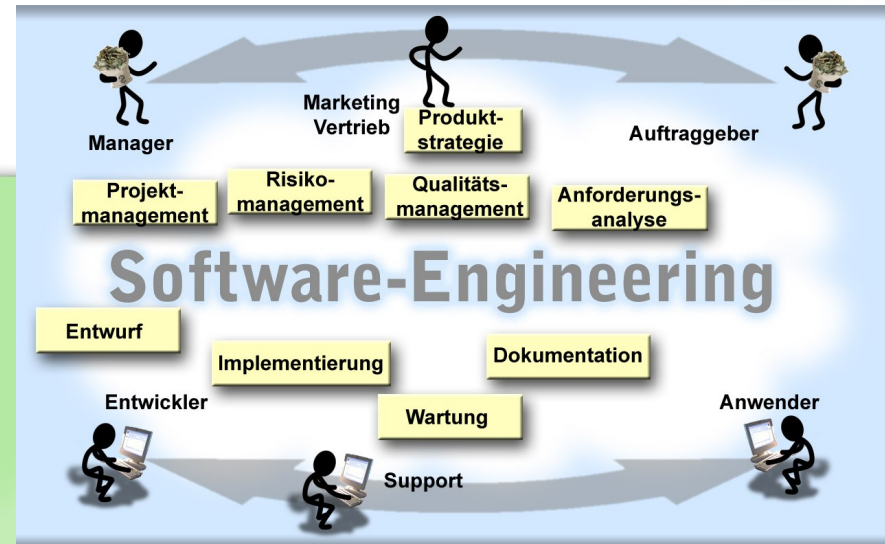


Übung

Aufgaben im SW-Engineering

Diskutieren Sie die Rollen und Aufgaben im Umfeld des Software-Engineerings.

- Was sind Merkmale der verschiedenen Rollen: Welche Qualifikation müssen entsprechende Mitarbeiter haben? Was für Aufgaben müssen sie typischerweise wahrnehmen?
- Können Sie die genannten Rollen weiter differenzieren? Welche Rollen/Unterrollen könnte es noch geben?
- Können Sie die Aufgabenbereiche näher charakterisieren? Gibt es weitere Aufgabenbereiche / Unterbereiche?



- Planen
- Verwalten
- Kommunizieren
- Analysieren
- Spezifizieren
- Dokumentieren
- Entwerfen
- Programmieren/Implementieren
- Kontrollieren (Qualität)
- Testen
- Konfigurieren
- Installieren
- Instandhalten

- **Analyst:** Probleme erfassen, kommunizieren, reden+aufschreiben, Kenntnis der Anwendungsdomäne
- **Modellierung/Designer:** Abstraktionsfähigkeit, UML, Design Patterns, Erfahrung, Informationsmodelle
- **UI Designer:** HTML, Usability, Gestalterische Fähigkeiten
- **Datenbankexperte:** Technische Kenntnisse, Modellierung
- **Architekt:** Programmiererfahrung, Entwurfsmuster, kommunizieren
- **Entwickler (für Programmiersprache X)** C/Java/...-Kenntnisse, Erfahrung!, guter Programmierstil
- **Techn. Schreiber:** Klare schriftl. Ausdrucksweise, Einfühlungsvermögen in den Leser, Erfahrung mit System-/Benutzer-Dokumentation
- **Tester** Kritische Einstellung, Sonderfälle erkennen, Erfahrung in Testplanung, Testtools
- **Qualitätsmanager:** Organisieren, kommunizieren, kritisch hinterfragen, planen
- **Projektleiter:** Organisieren, kommunizieren, präsentieren, entscheiden, Kompromisse aushandeln, schlichten, motivieren, Blick fürs Ganze, Zielorientierung, Erfahrung mit Vorgehensmodellen, Planungstools

(Ohne Anspruch auf Vollständigkeit!)

Beispiel: Rational Unified Process

