Software Engineering

Einführung

Authors of slides:
Norbert Siegmund
Janet Siegmund
Oscar Nierstrasz
Sven Apel

Organisatorisches: Übersicht

- Durchführung:
  - Vorlesung: Prof. Dr.-Ing. Janet Siegmund
  - Übung/Praktikum: Shadi Saleh
  - Startet in der zweiten Woche!
- Webseiten im OPAL
  - Passwort: sews19/20
  - Bitte Termin für Gastvorlesung abstimmen!

Organisatorisches: Termine

- Vorlesung: 09:15-10:45, 1/346 (Neu: A12.346)
- Ausfalltermine: bisher keine, falls Ausfall siehe Webseite/OPAL

- Übungen/Praktikum:
  - Wöchentlich
  - Donnerstag: 17:15 (für 4h Praktikum); 1/B202 (Neu: A11.202)
  - Freitag: 7:30 (für 2h Übung/2h Praktikum); 1/346 (Neu: A12.346)
  - Startet in der zweiten Woche (24. bzw. 25.10.)

Übung I

- Es wird ein durchgehendes Projekt über die ganze Vorlesung geben
- Ziel dieses Projektes ist es, die typischen Phasen im Softwarelebenszyklus durchzuspielen
- Das Projekt wird in der nächsten Übung bekannt gegeben
- Es wird mindestens eine Klausurfrage zum Projekt geben

Übung II

- Für jede Lebensphase wird es einen Meilenstein geben
- Alle 2 bis 3 Wochen muss ein Meilenstein eingereicht werden
- Für die Implementierung:
  - Es muss nicht das ganze Projekt implementiert werden
  - Sie können sich eins/wenige Features raussuchen und diese/s dann implementieren
  - Sie dürfen sich gern mit anderen Teams abstimmen und ihre Features am Ende integrieren
  - Sie dürfen sich eine objekt-orientierte Programmiersprache aussuchen (bitte begründen)
  - Sie dürfen vorhandene Libraries benutzen (bitte begründen)

Übung III

- Das Projekt wird in Teams mit 4 bis 6 Studierenden bearbeitet
- Jede Woche gibt es eine kurze Präsentation zum Fortschritt:
  - Was ist seit der letzten Woche passiert?
  - Was waren die Herausforderungen und Probleme? Wie wurden sie gelöst?
  - Was lief gut? Was lief nicht gut?
  - Was haben Sie gelernt? Was würden Sie beim nächsten Mal anders machen?
- Jedes Teammitglied soll mindestens einmal eine Fortschrittspräsentation halten

Industrievorlesung(en)!

- (nicht gesichert) Jonas Hecht (codecentric AG)
  - IT-Consultant
  - Java-Entwickler
  - Spring & Spring-Boot
  - Docker, etc.
  - **Für Termin abstimmen!**

codecentric

- Prof. Dr.-Ing. Thomas Leich (METOP GmbH)
  - Geschäftsführung
  - Software Engineering in der Praxis
  - Software Engineering mit global verteilten Teams

METOP MENSCH | TECHNIK ORGANISATION | PLANUNG

Organisatorisches: Einordnung

- Zuhörer: Bachelorstudierende
  - Gibt es Masterstudenten?
- Einordnung:
  - Vorlesung: Theoretische Grundlagen, kleine Beispiele, Diskussionen
  - Übung: Praktische Beispiele, Stoff orientiert sich an Erfahrungen und Problemen bei der Bearbeitung eines typischen, realen Software-Projekts
- Klausur:
  - 90 Minuten

Literatur

- *Software Engineering*. Ian Sommerville. Addison-Wesley Pub Co; ISBN: 020139815X, 7th edition, 2004
- *Software Engineering: A Practioner's Approach*. Roger S. Pressman. McGraw Hill Text; ISBN: 0072496681; 5th edition, 2001
- *Using UML: Software Engineering with Objects and Components*. Perdita Stevens and Rob J. Pooley. Addison-Wesley Pub Co; ISBN: 0201648601; 1st edition, 1999
- *Designing Object-Oriented Software*. Rebecca Wirfs-Brock and Brian Wilkerson and Lauren Wiener. Prentice Hall PTR; ISBN: 013629825 7; 1990

- Aber:
  - Kein einzelnes Buch für den gesamten Stoff der Vorlesung
  - Zu jedem Thema gibt es Empfehlungen
  - Vieles auch im Internet gut nachzulesen

Ablauf einer Vorlesung

- Lernziele (+ Fragen zu vorherigen Themen)
- Interaktive Methoden
  - Murmelgruppe
  - Fragen
  - Quiz

- Evtl. mögliche Klausurfragen am Ende der Vorlesung

Was ist Software Engineering?

Software Engineering vs. Informatik

- Informatik beschäftigt sich mit der Theorie und den Grundlagen von Computersystemen

- Software engineering beschäftigt sich mit praktischen Themen der Entwicklung und Auslieferung “guter” Software

Warum Software Engineering?

Naive Sicht:

Problemspezifikation —Coding→ Finales Programm

Aber:

- Woher kommt die Spezifikation?
- Wie korrespondiert die Spezifikation zu den Nutzeranforderungen?
- Wie entscheidet man, wie das Programm strukturiert wird?
- Wie weiß man, dass das Programm tatsächlich den Spezifikationen entspricht?
- Wie weiß man, dass das Programm immer korrekt arbeitet?
- Was macht man, wenn die Nutzeranforderungen sich ändern?
- Wie teilt man Aufgaben auf, falls man mehr als ein 1-Personen Team hat?

Gründe zur Entstehung: Softwarekrise

- Kosten für Software überstiegen Kosten für Hardware
- Nur 34% der Softwareprojekte erfolgreich abgeschlossen
- Während der NATO Software Engineering Konferenz 1968 geprägt

- "[The major cause of the software crisis is] that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem." – Edsger Dijkstra, 1972

Beispiele gescheiterte Projekte: Ariane 5

- Ariane 5 Flight 501: 4. Juni, 1996
- Selbstzerstörung nach 39 Sekunden
- ~ 500 million US-$ Schaden

Quelle(FU): https://www.ima.umn.edu/~arnold/disasters/ariane.html

Therac-25

- Medizinische Strahlentherapie
- Durch SW-Fehler zu hohe Strahlendosis, mehrere Tote
- Ein einziger Entwickler schrieb Software und benutzte vorhandene Komponenten der Vorgänger, hatte aber wenig Erfahrung in diesem Bereich

Image source (FU): http://cr4.globalspec.com/blogentry/19025/Failure-of-the-Therac-25-Medical-Linear-Accelerator
https://hackaday.com/2015/10/26/killed-by-a-machine-the-therac-25/

Apollo 11 PGNCS

- Während der Mondlandung:
- Unerwartete “executive overflow” Alarms
  - 13% der Computerressourcen wurden durch einen Fehler im Radar verbraucht (5x Alarm + Neustart der Software)

Image source (PD): https://en.wikipedia.org/wiki/Apollo_11#Lunar_descent

Mariner I

- Erste US Venus Sonde
- Rakete zerstörte sich 295s nach Start selber aufgrund eines HW+SW Bugs
- https://www.edn.com/electronics-blogs/edn-moments/4418667/Mariner-1-destroyed-due-to-code-error--July-22--1962

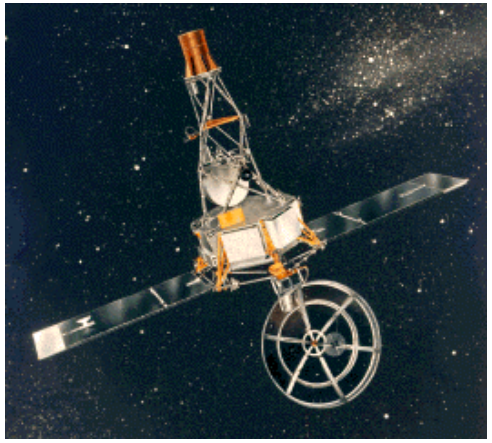


Image source (PD): http://nssdc.gsfc.nasa.gov/nmc/spacecraftDisplay.do?id=MARIN1

“Morris Worm”

- Erster “Computervirus” – per Zufall
- Entworfen als ein Forschungswerkzeug zum Zählen der Computer im Internet (1989)
- Durch Bug verbreitete er sich selber
  - 10% des Internetverkehrs
  - Ein PC konnte mehrmals infiziert werden
- 10k-10.000k$ Schaden

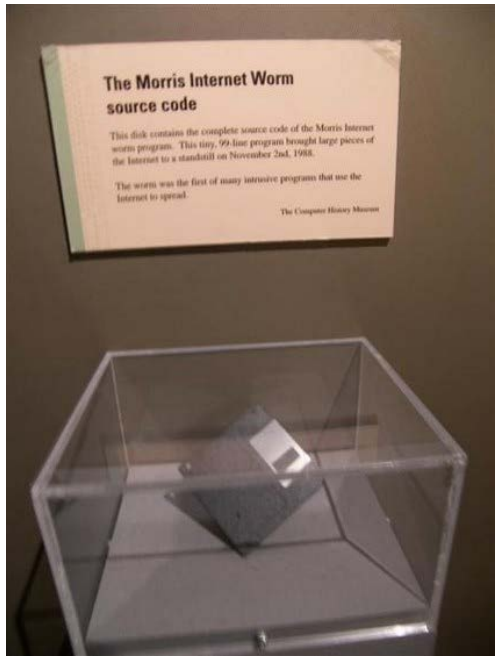


Image source (CC): https://en.wikipedia.org/wiki/Morris_worm#/media/File:Morris_Worm.jpg

Mars Climate Orbiter

- Verloren 1999 beim Anflug auf den Mars
- ~ 330 Millionen-US-$ Schaden
- Grund: Einheitenfehler im Navi (metrisch vs. imperial)

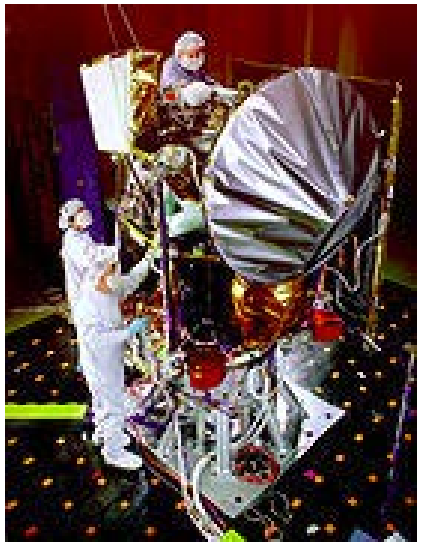


Image source (PD): https://en.wikipedia.org/wiki/Mars_Climate_Orbiter

Mars Polar Lander

- Verloren bei der Marslandung 1999
- ~ 330 Millionen-US-$ Schaden
- Fehler in der Software interpretierte Vibrationen bei dem Ausfahren der Landebeine als tatsächliche Landung und schaltete Triebwerke zu früh ab

Image source (PD): https://en.wikipedia.org/wiki/Mars_Polar_Lander

Patriot Missile Disaster

- Fehler im Zielsuchsystem führte zum Verfehlen der abzufangenden Rakete und somit zum Tod von 28 Menschen während des Golfkriegs 1991
- Grund: Fehlerhafte Zeitberechnung durch ungenaue arithmetische Berechnungen (Kommastellen wurden nach 24Bit abgeschnitten, was zu Ungenauigkeiten führte)

Image source (CC): https://en.wikipedia.org/wiki/MIM-104_Patriot

AT&T Telefonnetzwerk-Crash

- AT&T Netzwerk kollabierte am 15.1.1990
- 11 h Ausfall, ~ 60 Millionen-US-$ Schaden

> "The fault was in the code" of the new software (routine update) that AT&T loaded into front-end processors of all 114 of its 4ESS switching systems in mid-December, said Larry Seese, AT&T's director of technology development.

- http://www.phworld.org/history/attcrash.htm

Image source (FU): http://www.att.com/gen/press-room?pid=6158&cat=46&u=484

Los Angeles Flugkontroll-Crash

- Flugkontrolle in LAX stürzte am 30.4.2014 ab
- 10 cancelled und 500 delayed Flüge
- US-Spionageflugzeug verwirrte Software
- https://www.theguardian.com/world/2014/may/06/u-2-spy-plane-crashes-los-angeles-air-traffic-control

Image source (CC): https://www.flickr.com/photos/usnavy/7261361906/

2003 Northeast Blackout

- Über 2 Tage Stromausfall in USA/Kanada
- ~ 10 Opfer, 45/10 Millionen Menschen betroffen
- SW Bug verhinderte lokalen Alarm, sodass sich das Problem ausbreitete

Image source (CC): https://en.wikipedia.org/wiki/Northeast_blackout_of_2003

Corrupted Blood Incident

- Unbeabsichtigte “Seuche” in WoW, 2005, 1 Woche
- Wurde später bei Epidemiologen verwendet, um die Ausbreitung von Seuchen zu studieren

Image source (FU): https://en.wikipedia.org/wiki/File:WoW_Corrupted_Blood_Plague.jpg

Hamburg-Altona Railway Switch

- Computer “upgrade” des Switch-Systems
- Ergebnis: Gesamter Bahnhof musste für 2 Tage schließen, 100 000s Passagiere betroffen
- Stackoverflow + Fehler im Code zum Behandeln des Overflows

Image source (CC): https://en.wikipedia.org/.../media/File:Bahnhof_Hamburg_Altona.JPG

Mars-Sonde

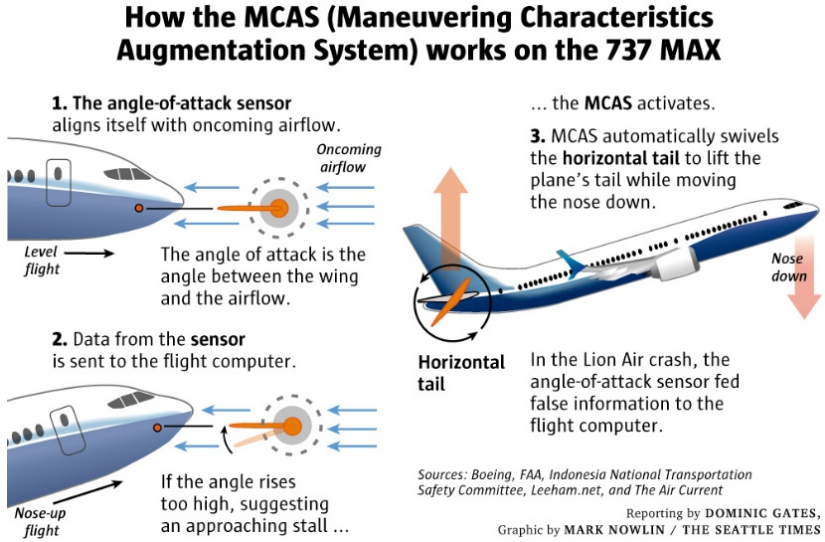
Computer war schuld an "Schiaparelli"-Crash

Das Rätsel um den Absturz der Marssonde "Schiaparelli" ist gelöst. Ein Untersuchungsbericht erklärt, warum der Forschungsroboter aus fast vier Kilometer Höhe ungebremst aufschlug.

ESA/ ATG medialab/dpa

Boing 737 Max

- https://www.seattletimes.com/business/boeing-aerospace/failed-certification-faa-missed-safety-issues-in-the-737-max-system-implicated-in-the-lion-air-crash/
- Softwaresystem MCAS (Maneuvering Characteristics Augmentation System) hatte Sicherheitsprobleme und war abhängig von einem einzelnen Sensor
- 346 Menschen starben; 32$ Mrd. Aktienverlust, 8$ Mrd. Kosten

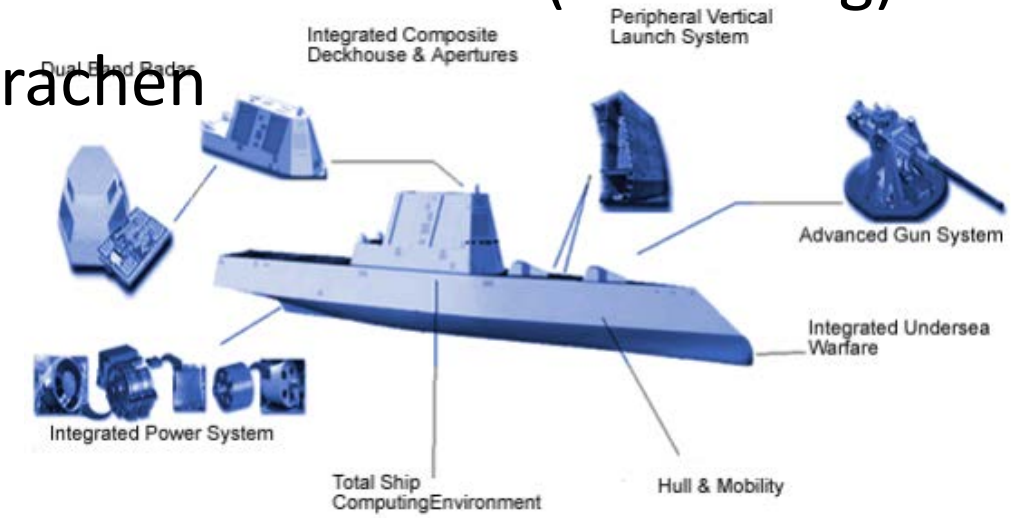


Beispiele für gescheiterte Projekte II

- SAGE System (1951) zur Suche nach Bomber; zur Auslieferung in 1963 nicht mehr benötigt
- Toll Collect: geplanter Start Aug. 2003, tatsächl. Start in Jan. 2006; 3.5 Mrd€ Verlust
- Kaliforniens Driver and Vehicle Licensing Agency brach SW Project nach 6 Jahren und 45 M$ ab
- FBI Virtual Case File brach 2005 nach 3 Jahre und 170 M$ SW Projekt ab
- London Stock Market brach Taurus Projekt 1993 nach 11 Jahren und 800 M£ (13200 %) ab

Extreme Komplexität (Beispiel)

- DDX U-Boot
- Viele eingebettete Systeme
- Zusammen 30.000.000.000 Zeilen Code (Schätzung)
- In 142 Programmiersprachen



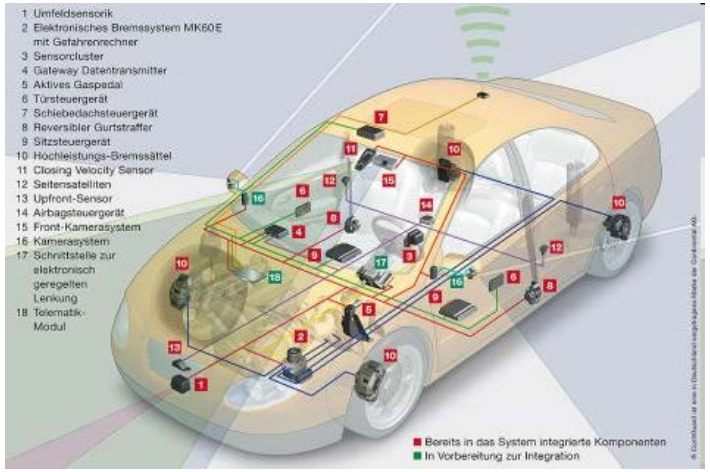
Aufgabe: Gründe für Scheitern

- Warum scheitern Softwareprojekte?

Gründe für Scheitern

- Budget aufgebraucht
- Zeit aufgebraucht
- Ineffizienz von Software
- Schlechte Qualität von Software
- Software erfüllt Anforderungen nicht
- Projekte waren nicht mehr zu managen und Quelltext schwierig zu warten
- Software wurde nie ausgeliefert

Lösung: Software Engineering



Begriff: Software Engineering

- Begriff wurde auf der NATO-Konferenz (1968) geprägt
- Idee: Softwareentwicklung in Anlehnung an Ingenieursdisziplin (Engineering)

- Im Folgenden: Definitionen
  - Geben einen Einblick über verschiedene Sichtweisen auf diese Disziplin

Definition: Software Engineering (1)

„state of the art of developing quality software on time and within budget “

- Aktuellster Stand der Technik:
  - Entscheidet Community
  - Lebenslanges lernen
- Ressourcenbeschränkung

Definition: Software Engineering (2)

„multi-person construction of multi-version software “ -- Parnas

- Teamwork
- Erfolgreiche Software-Systeme müssen sich weiterentwickeln: Änderung ist der Normalfall, nicht die Ausnahme

Definition: Software Engineering (3)

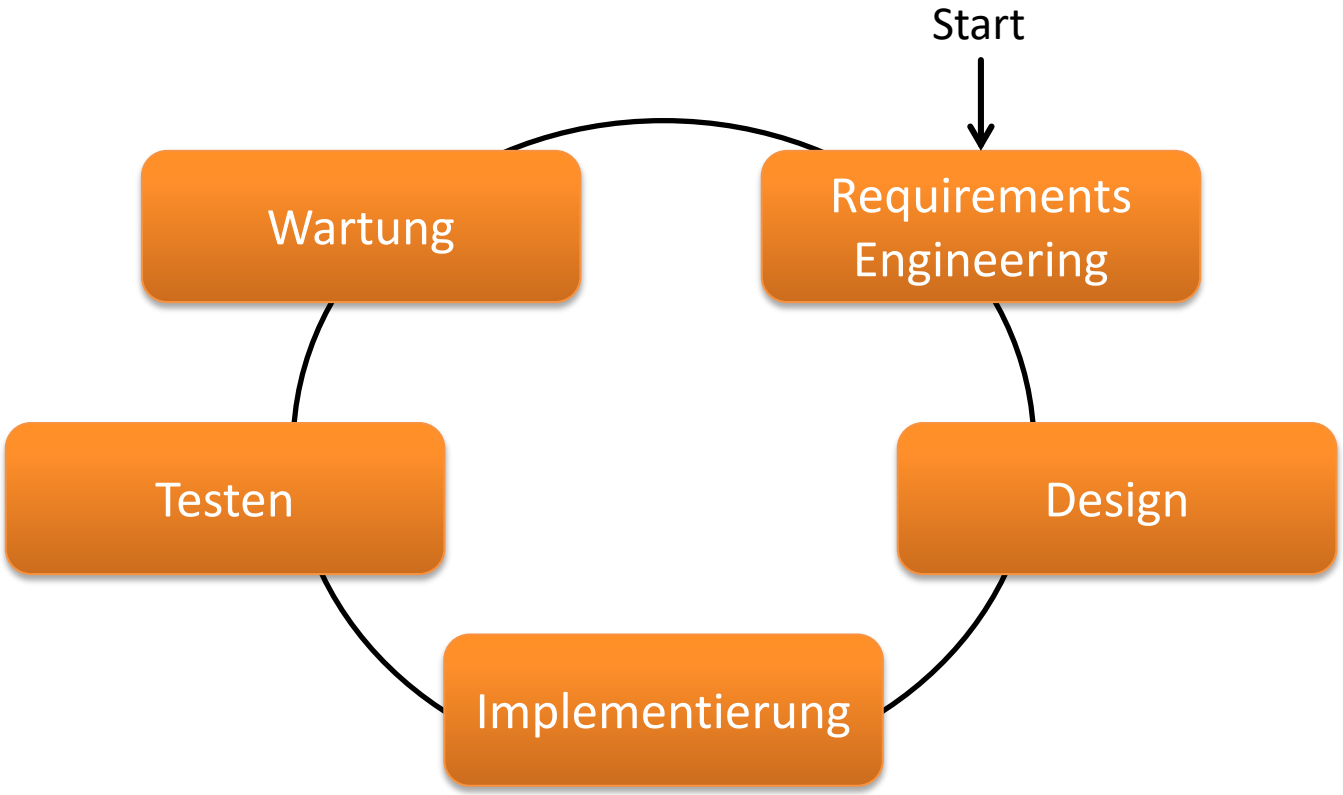
„software engineering is an engineering discipline that is concerned with all aspects of software production“ -- Sommerville

- Software ist ein Produkt, das für einen bestimmtem Kunden entwickelt wird
- Nicht nur Programmierung, sondern alle Aspekte des Softwarelebenszyklus

Aufgabe

- Welche Prozesse gehören zum Softwarelebenszyklus?
- Welche Rolle spielt die Programmierung dabei?

Softwarelebenszyklus - Einordnung



- Programmierung ist nur ein kleiner Teil von Software-Entwicklung

Abgrenzung: Ingenieursdisziplin

- Software hat praktisch keine physischen Eigenschaften
- Keine zugrundeliegende physikalische Theorie
- Kein Verfall, trotzdem existiert nach 10 Jahren typischerweise keine einzige Zeile des ursprünglichen Quellcodes mehr
- Software kann (fast) beliebig komplex werden

Was ist „Gute“ Software?

- Maintainable (Wartbar)
- Dependable (Verfügbar, Zuverlässig)
- Efficient (Effizient)
- Usable (Nutzbar, Anwendbar)

Fundamentale Prinzipien und Konzepte

- Prinzipien
  - Divide and conquer
  - Simplicity
  - Rigorousness
- Konzepte
  - Modularität und Struktur
  - Abstraktion und Generalisierung
  - Design for change
  - Separation of concerns
  - Stepwise refinement
  - Information hiding

Zusammenfassung

- Grundlegende Vorstellung von Software Engineering haben
- Notwendigkeit des Software Engineerings erkennen
- Einordnung des Anteils der Programmierung an der Entwicklung von Software

Was Sie mitgenommen haben sollten

- Was ist Software Engineering?
- Nennen/Erklären Sie X mögliche Gründe für das Scheitern von Software-Projekten
- Was ist die Softwarekrise? Warum trat sie auf?
- Nennen/Erklären Sie die Prozesse im Software-Lebenszyklus.
- Was sind die Gemeinsamkeiten/Unterschiede von Software Engineering zur klassischen Ingenieursdiziplin?

Literatur

- Beliebiges Software-Engineering Buch
- Internet