Software Engineering: Wartung und Qualitätssicherung

16. Oktober 2019

1 Software-Entwicklung, Wartung und (Re)Engineering

Erstellte Software muss oft geändert werden, entweder aufgrund von geänderten Anforderungen oder neuen Anforderungen, welche eingebaut werden müssen.

1.1 Einleitung

1.1.1 Geschichte

- Softwarekrise 1968
- Nato **Working Conference on Software Engineering**
- Zuordnungen
  - Praktische Informatik
  - Theoretische Informatik
  - Projektplanung
  - Organisation
  - Psychologie
  - ...

1.1.2 "Software-Technik" Definition

Software-Engineering(Software-Technik) ist nach **Entwicklung**, **Pflege** und **Einsatz**.
Eingesetzt werden:

- Wissenschaftliche Methoden
- Wirtschaftliche Prinzipien
- Geplante Vorgehensmodellen
- Werkzeuge
- Quantifizierbare Ziele

1.1.3 50 Jahre nach Beginn der SSoftware-Krise"

- 19% aller Projekte sind gescheitert, früher 25%
- 52% aller Projekte sind dabei zu scheitern, früher 50%
- 29% aller betrachteten IT-Projekte sind erfolgreich, früher 25%

Hauptgründe fürs Scheitern der Projekte:
Unklare Anforderungen und Abhängigkeiten sowie Problemen beim Änderungsmanagement.

1.2 Software-Qualität

Ziel der Software-Technik ist die effiziente Entwicklung messbar qualitativ hochwertiger Software.

1.2.1 Qualitätsdefinition

Qualität ist der Grad, in dem ein System, eine Komponente oder ein Prozess die Kundenerwartungen und Kundenbedürfnisse erfüllt.

1.2.2 Softwarequalität

Softwarequalität ist die Gesamtheit der Funktionalitäten und Merkmale eines Softwareprodukts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen.

1.2.3 Qualitätsmerkmale

- Funktionalität

1.2.4 Nichtfunktionale Merkmale

- Zuverlässigkeit (Reliability)
- Benutzbarkeit (Usability)
- Effizienz (Efficiency)
- Änderbarkeit (Maintainability)
- Übertragbarkeit (Portability)

1.2.5 Prinzipien der Qualitätssicherung

- **Qualitätszielbestimmung**: Auftraggeber und Auftragnehmer legen vor Beginn der Software-Entwicklung gemeinsames Qualitätsziel für Software-System mit nachprüfbaren Kriterienkatalog fest (als Bestandteil des abgeschlossenen Vertrags zur Software-Entwicklung)
- **Quantitative Qualitätssicherung**: Einsatz automatisch ermittelbaren Metriken zur Qualitätsbestimmung (objektivbare, ingenieursmäßige Vorgehensweise)
- **Konstruktive Qualitätssicherung**: Verwendung geeigneter Methoden, Sprachen und Werkzeuge (Vermeidung von Qualitätsproblemen)
- **Integrierte, frühzeitige, analytische Qualitätssicherung**: Systematische Prüfung aller erzeugter Dokumente (Aufdeckung von Qualitätsproblemen)
- **Unabhängige Qualitätssicherung**: Entwicklungsprodukte werden durch eigenständige Qualitätssicherungsabteilung überprüft und abgenommen (verhindert u.a. Verzicht auf Testen zugunsten Einhaltung des Entwicklungsplans)

1.2.6 Konstruktives Qualitätssicherung zur Fehlervermeidung

:

- Technische Maßnahmen
  - Sprachen (UML, Java)
  - Werkzeuge (UML-CASE-TOOL)
- Organisatorische Maßnahmen
  - Richtlinien (Gliederungsschema für Pflichtenheft, Programmierrichtlinien)
  - Standards (für verwendete Sprachen, Dokumentformate, Management)
  - Checklisten

1.2.7 Analytisches Qualitätsmanagement für Fehleridentifikation

- **Analysierende Verfahren**: Der "Prüfling"(Programm, Modell, Dokumentation) wird von Menschen oder Werkzeugen auf Vorhandensein/Abwesenheit von Eigenschaften untersucht
  - **Review**: Prüfung durch Menschen
  - **Statische Analyse**: Werkzeuggestützte Ermittlung von Änomalien"
  - **Formale Verifikation**: Werkzeuggestützter Beweis von Eigenschaften
- **Testende Vefahren**: Der "Prüfling"wird mit konkreten oder abstrakten Eingabewerten auf einem Rechner ausgeführt

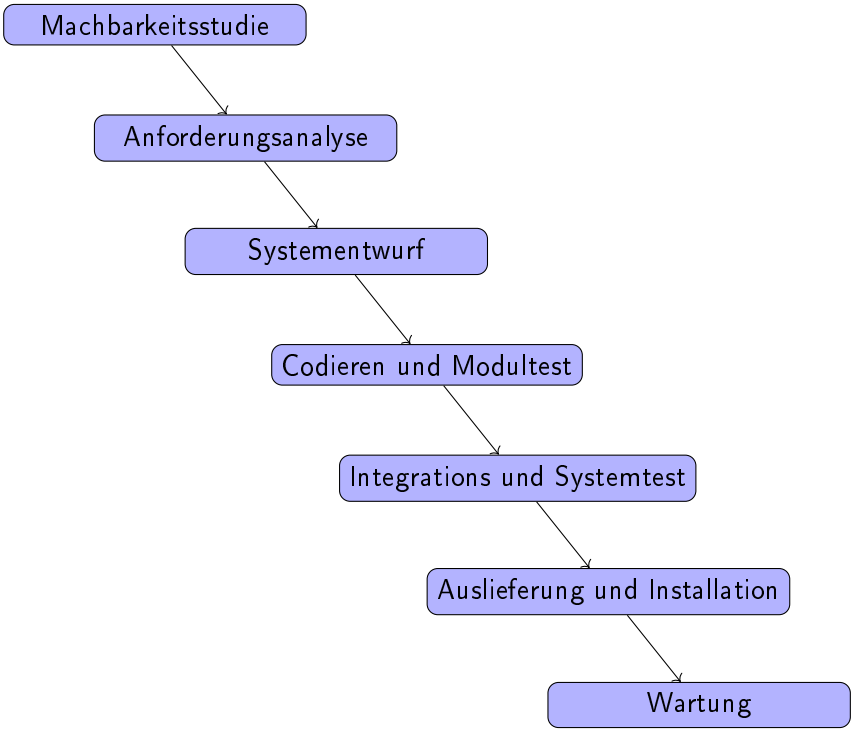
  - **Dynamischer Test**: "normaleÄusführung mit ganz konkreten Eingaben
  - **Symbolischer Test**: Ausführung mit symbolischen Eingaben

1.3 Iterative Softwareentwicklung

Voraussetzung für den sinnvollen Einsatz von Notationen und Werkzeugen zur Software-Entwicklung ist ein:

- **Vorgehensmodell**, das den Gesamtprozess der Software-Erstellung und pflege in einzelne Schritte aufteilt
- Zusätzlich müssen Verantwortlichkeiten der beteiligten Personen in Form von **Rollen** im Software-Entwicklungsprozess klar geregelt sein.

1.3.1 Übersicht der Phasen des Wasserfallmodells



1.3.2 Machbarkeitsstudie (feasability study)

Die Machbarkeitsstudie schätzt Kosten und Ertrag der geplanten Software-Entwicklung ab. Dazu grobe Analyse des Problems mit Lösungsvorschlägen.

- **Aufgaben**

  - Problem informell und abstrahiert beschreiben
  - Verschiedene Lösungsansätze erarbeiten
  - Kostenschätzung durchführen
  - Angebotserstellung

- **Ergebnisse**
  - Lastenheft
  - Projektkalkulation
  - Projektplan
  - Angebot an Auftraggeber

1.3.3 Anforderungsanalyse (requirements engineering)

In der Anforderungsanalyse wird exakt festgelegt, was die Software leisten soll, aber nicht wie diese Leistungsmerkmale erreicht werden.

- **Aufgaben**
  - genaue Festlegung der Systemeigenschaften wie Funktionalität, Leistung, Benutzungsschnittstelle, Portierbarkeit, ... im Pflichtenheft
  - Bestimmen von Testfällen
  - Festlegung erforderlicher Dokumentationsdokumente

- **Ergebnisse**
  - Pflichtenheft = Anforderungsanalysedokument
  - Akzeptanztestplan
  - Benutzungshandbuch (1-te Version)

1.3.4 Systementwurf (system design/programming-in-the-large)

Im Systementwurf wird exakt festgelegt, wie die Funktionen der Software zu realisieren sind. Es wird der Bauplan der Software, die Software-Architektur, entwickelt.

- **Aufgaben**
  - Programmieren-im-Großen = Entwicklung eines Bauplans
  - Grobentwurf, der System in Teilsysteme/Module zerlegt
  - Auswahl bereits existierender Software-Bibliotheken, Rahmenwerke, ...
  - Feinentwurf, der Modulschnittstellen und Algorithmen vorgibt

- **Ergebnisse**
  - Entwurfsdokument mit Software-Bauplan
  - detaillierte(re) Testpläne

1.3.5 Codieren und Modultest (programming-in-the-small)

Die eigentliche Implementierungs- und Testphase, in der einzelne Module (in einer bestimmten Reihenfolge) realisiert und validiert werden.

- **Aufgaben**
  - Programmieren-im-Kleinen = Implementierung einzelner Module
  - Einhaltung von Programmierrichtlinien
  - Code-Inspektionen kritischer Modulteile (Walkthroughs)
  - Test der erstellten Module
- **Ergebnisse**
  - Menge realisierter Module
  - Implementierungsberichte (Abweichungen vom Entwurf, Zeitplan, ...)
  - Technische Dokumentation einzelner Module
  - Testprotokolle

1.3.6 Integrations- und Systemtest

Die einzelnen Module werden schrittweise zum Gesamtsystem zusammengebaut. Diese Phase kann mit der vorigen Phase verschmolzen werden, falls der Test isolierter Module nicht praktikabel ist.

- **Aufgaben**
  - Systemintegration = Zusammenbau der Module
  - Gesamtsystemtest in Entwicklungsorganisation durch Kunden (alpha-Test)
  - Fertigstellung der Dokumentation
- **Ergebnisse**
  - Fertiges System
  - Benutzerhandbuch
  - Technische Dokumentation
  - Testprotokolle

1.3.7 Auslieferung und Installation

Die Auslieferung (Installation) und Inbetriebnahme der Software beim Kunden findet häufig in zwei Phasen statt.

- **Aufgaben**
  - Auslieferung an ausgewählte (Pilot-)Benutzer (Beta-Test)
  - Auslieferung an alle Benutzer
  - Schulung der Benutzer
- **Ergebnisse**
  - Fertiges System
  - Akzeptanztestdokument

1.3.8 Wartung (Maintenance)

Nach der ersten Auslieferung der Software an die Kunden beginnt das Elend der Software-Wartung, das ca. 60% der gesamten Software-Kosten ausmacht.

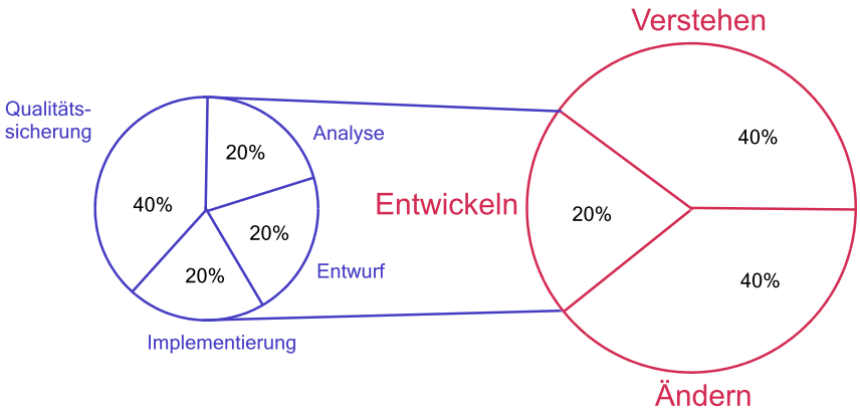
- **Aufgaben**
  - ca. 20% Fehler beheben (corrective maintenance)
  - ca. 20% Anpassungen durchführen (adaptive maintenance)
  - ca. 50% Verbesserungen vornehmen (perfective maintenance)
- **Ergebnisse**
  - Software-Problemberichte (bug reports)
  - Software-Änderungsvorschläge
  - Neue Software-Versionen

1.3.9 Probleme mit dem Wasserfallmodell

- zu Projektbeginn sind nur ungenaue Kosten- und Ressourcenschätzungen möglich
- ein Pflichtenheft kann nie den Umgang mit dem fertigen System ersetzen, das erste sehr spät entsteht (Risikomaximierung)
- es gibt Fälle, in denen zu Projektbeginn kein vollständiges Pflichtenheft erstellt werden kann (weil Anforderungen nicht klar)
- Anforderungen werden früh eingefroren, notwendiger Wandel (aufgrund organisatorischer, politischer, technischer, ... Änderungen) nicht eingeplant

- strikte Phaseneinteilung ist unrealistisch (Rückgriffe sind notwendig)
- Wartung mit ca. 60% des Gesamtaufwandes ist eine Phase

1.3.10 Andere Darstellung der Aufwandsverteilung



1.3.11 Typische Probleme in der Wartungsphase

- Einsatz wenig erfahrenen Personals (nicht Entwicklungspersonal)
- Fehlerbehebung führt neue Fehler ein
- Stetige Verschlechterung der Programmstruktur
- Zusammenhang zwischen Programm und Dokumentation geht verloren
- Zur Entwicklung eingesetzte Werkzeuge (CASE-Tools, Compiler, ...) sterben aus
- Benötigte Hardware steht nicht mehr zur Verfügung
- Resourcenkonflikte zwischen Fehlerbehebung und Anpassung/Erweiterung
- Völlig neue Ansprüche an Funktionalität und Benutzeroberfläche

1.4 Forward-, Reverse- und Reengineering

1.4.1 Software Evolution

- Wünsche
  - Wartung ändert Software kontrolliert ohne Design zu zerstören
  - Konsistenz aller Dokumente bleibt erhalten

- Wirklichkeit
  - Ursprüngliche Systemstruktur wird ignoriert
  - Dokumentation wird unvollständig oder unbrauchbar
  - Mitarbeiter verlassen Projekt

1.4.2 Forward Engineering

Beim Forward Engineering ist das fertige Softwaresystem das Ergebnis des Entwicklungsprozesses. Ausgehend von Anforderungsanalyse (Machbarkeitsstudie) wird ein neues Softwaresystem entwickelt.

1.4.3 Reverse Engineering

Beim Reverse Engineering ist das vorhandene Software-System der Ausgangspunkt der Analyse. Ausgehend von existierender Implementierung wird meist „nur“ das Design rekonstruiert und dokumentiert. Es wird (noch) nicht das betrachtete System modifiziert.

1.4.4 Reengineering

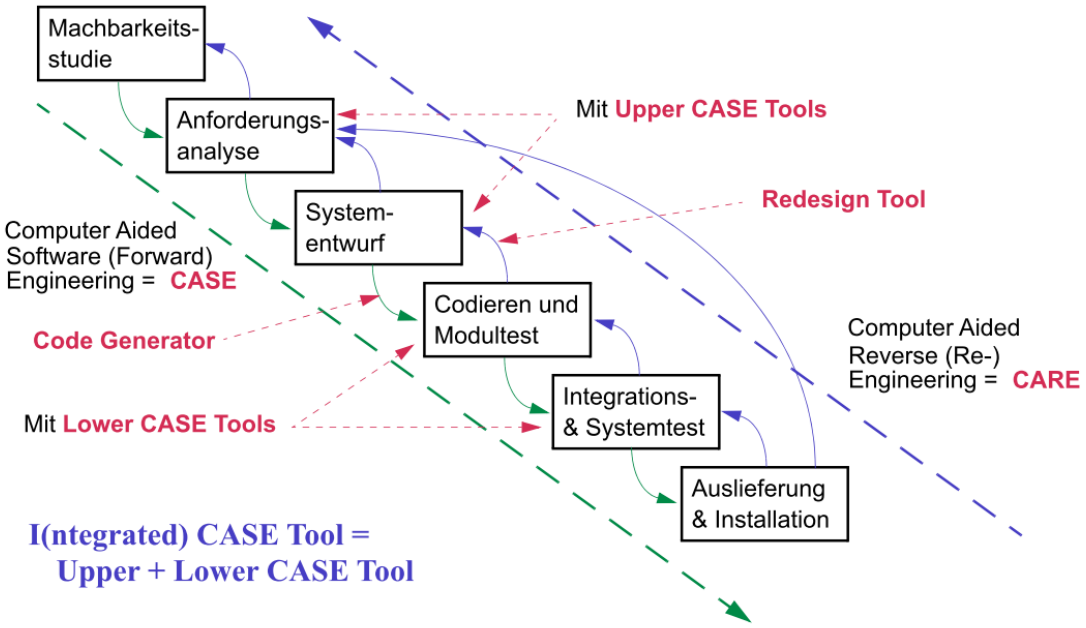
Reengineering befaßt sich mit der Sanierung eines vorhandenen Software-Systems bzw. seiner Neuimplementierung. Dabei werden die Ergebnisse des Reverse Engineerings als Ausgangspunkt genommen

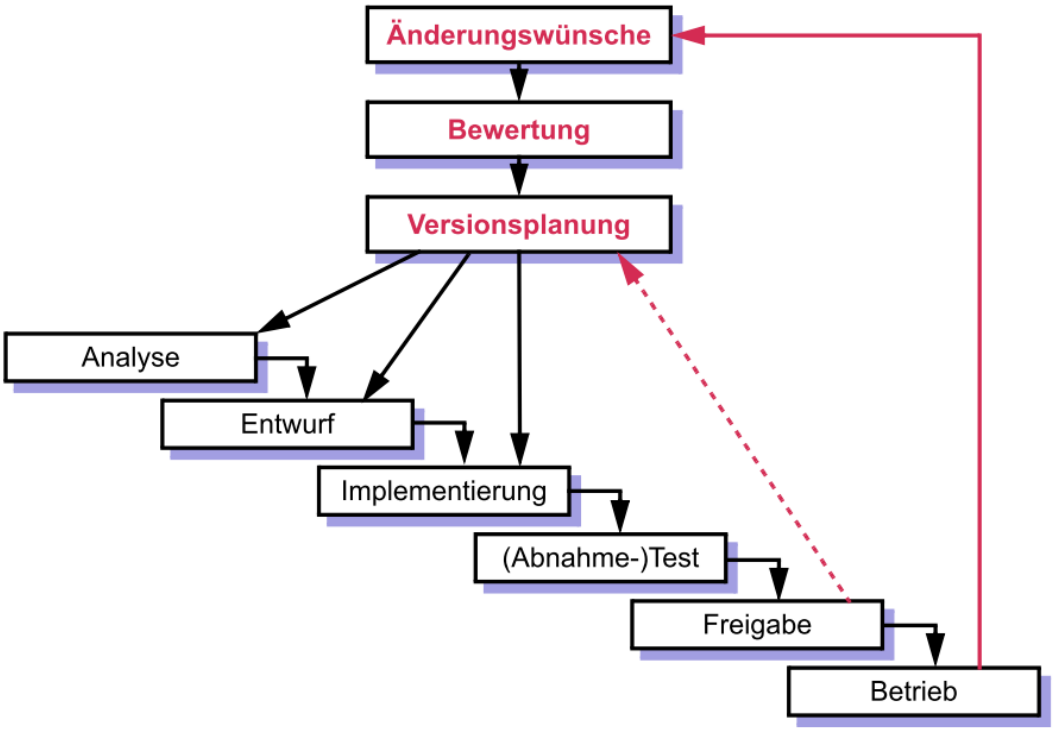
1.4.5 Round Trip Engineering

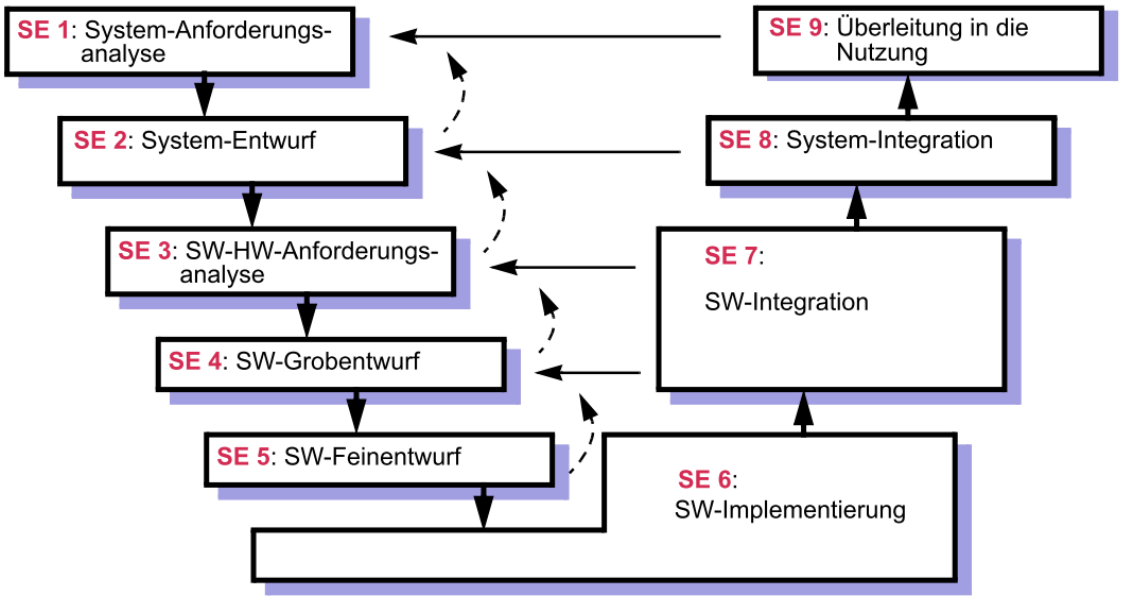
1.4.6 Einfaches Software-Lebenszyklus-Prozessmodell für die Wartung

1.4.7 Das V-Modell

- **Systemanforderungsanalyse**: Gesamtsystem einschließlich aller Nicht-DV-Komponenten wird beschrieben (fachliche Anforderungen und Risikoanalyse)
- **Systementwurf**: System wird in technische Komponenten (Subsysteme) zerlegt, also die Grobarchitektur des Systems definiert
- **Softwareanforderungsanalyse**: Technischen Anforderungen an die bereits identifizierten Komponenten werden definiert
- **Softwaregrobentwurf**: Softwarearchitektur wird bis auf Modulebene festgelegt
- **Softwarefeinentwurf**: Details einzelner Module werden festgelegt
- **Softwareimplementierung**: Wie beim Wasserfallmodell (inklusive Modultest)
- **Software-/Systemintegration:**: Schrittweise Integration und Test der verschiedenen Systemanteile
- **Überleitung in die Nutzung:**: Entspricht Auslieferung bei Wasserfallmodell


Machbarkeits-
studie
Anforderungs-
analyse
System-
entwurf
Codieren und
Modultest
Integrations-
& Systemtest
Auslieferung
& Installation
Mit Upper CASE Tools
Redesign Tool
Computer Aided
Software (Forward)
Engineering = CASE
Code Generator
Mit Lower CASE Tools
Computer Aided
Reverse (Re-)
Engineering = CARE
I(ntegrated) CASE Tool =
Upper + Lower CASE Tool


Änderungswünsche
Bewertung
Versionsplanung
Analyse
Entwurf
Implementierung
(Abnahme-)Test
Freigabe
Betrieb


SE 1: System-Anforderungs-analyse
SE 2: System-Entwurf
SE 3: SW-HW-Anforderungs-analyse
SE 4: SW-Grobentwurf
SE 5: SW-Feinentwurf
SE 6: SW-Implementierung
SE 7: SW-Integration
SE 8: System-Integration
SE 9: Überleitung in die Nutzung