Einführung in die Softwareentwicklung

Seminar: Softwareentwicklung in der Wissenschaft

von Thorsten Lemburg

Gliederung:

1. Einleitung
2. Software
   1. Definition Software (IEEE 610.12)
   2. Feststellungen
3. Softwareentwicklung
   1. Definition Softwareentwicklung
   2. Gesetzmäßigkeiten bei der Softwareentwicklung
   3. Warum ist Softwareentwicklung so schwierig?
   4. Die 13 häufigsten Fehler in der Softwareentwicklung
4. Teilgebiete der Softwareentwicklung
   1. Projektmanagement
   2. Qualitätsmanagement
   3. Risikomanagement
   4. Anforderungserhebung
   5. Systemdesign / technische Konzeption
   6. Implementierung
   7. Softwaretest
   8. Softwareeinführung
   9. Wartung / Pflege
5. Vorgehensmodelle in der Softwareentwicklung
   1. Wasserfallmodell
   2. V-Modell
   3. Weitere Vorgehensmodelle
6. Zusammenfassung
7. Quellen

1. Einführung

Rechner durchdringen heutzutage alle Lebensbereiche und mit ihnen die Software, welche diese Rechner steuert. Viele Selbstverständlichkeiten des Alltagslebens sind ohne Rechner und deren Software nicht mehr möglich. Was würden wir heutzutage ohne PCs, Internet oder Telefone machen? Würden wir wieder auf Briefe und Kuriere umsteigen? Wie bewegen die sich wiederum fort? Mit einem Auto oder mit dem Flugzeug?
Und in der Wirtschaft sieht dies auch nicht anders aus:
Kann sich noch jemand vorstellen Autos mit Hand herzustellen? Und wie funktionieren die Autos denn überhaupt ohne das die Software diese steuert?
Im krassen Gegensatz zu dieser Abhängigkeit steht die Tatsache, dass weltweit die Erstellung von Software nur ungenügend beherrscht wird. Termin- und Kostenüberschreitungen bei Software-Projekten sind die Regel; Software, die Fehler enthält oder sich nicht entsprechend den Vorstellungen und Bedürfnissen der Benutzenden verhält, gehört zum Alltag. Immer wieder kommt es auch vor, dass ganze Projekte scheitern.
Stellvertretend seien 2 Beispiele aus jüngerer Zeit genannt:

Beispiel Ariane 5:

Bild: links: Ariane 5 vor dem Start; rechts: Explosion der Ariane 5
Quelle: http://estb.msn.com

Eine Zusammenfassung der Ereignisse aus dem Wikipedia Artikel:
http://de.wikipedia.org/wiki/Ariane_V88

Der Erstflug der europäischen Schwerlast-Trägerrakete Ariane 5 am 4. Juni 1996 fand unter der Startnummer **V88** statt. Die Rakete trug die Seriennummer **501.** Der Flug endete etwa 40 Sekunden nach dem Start, als die Rakete nach einer Ausnahmesituation in der Software der Steuereinheit plötzlich vom Kurs abkam und sich kurz darauf selbst zerstörte. Vier Cluster-Forschungssatelliten zur Untersuchung des Erdmagnetfelds gingen dabei verloren.

Die Software, die zur Kursabweichung führte, war unverändert von der Vorgängerrakete Ariane 4 übernommen worden, obwohl sie nicht für die geänderte Flugbahn der Ariane 5 geeignet war und nach dem Start der Rakete keinen Zweck erfüllte.

Der Fehlschlag mit einem Gesamtverlust von etwa 290 Millionen Euro führte zu einer einjährigen Verzögerung des Ariane-5-Programms, weshalb vorläufig auf die Ariane 4 ausgewichen wurde. Ein neuer Satz von Cluster-Satelliten wurde vier Jahre nach dem Unglück mit russischen Sojus-Raketen gestartet.

Ablauf:

Der Countdown verlief normal bis sieben Minuten vor Startbeginn, als er wegen nicht erfüllter Sichtbarkeitsbedingungen gestoppt wurde. Nach einer knappen Stunde wurde der Countdown wieder aufgenommen. Bis 36 Sekunden, als sich die Rakete einer Höhe von etwa 3700 m befand, verlief der Flug normal. In den folgenden Sekunden wich die Rakete von ihrem normalen Kurs ab, begann auseinander zu brechen und sprengte sich selbst.

Der Fehler hatte seine Ursache in einem Softwaremodul beider Intertialen Navigationssysteme(INS) der Steuerungseinheit. Bei der Umwandlung einer 64-Bit-Gleitkomma-Variable in eine vorzeichenbehaftete 16-Bit-Ganzzahl kam es zu einem arithmetischen Überlauf. Diese Variable, die für die horizontale Ausrichtung zuständig war, gab die Ausrichtungspräzision der Inertialplattform an und hing mit der horizontalen Geschwindigkeit der Rakete zusammen.

Der unbehandelte Operandenfehler im Programm führte zum Ausfall (Übergang in den „degraded mode“) des Reserve-INS und kurz danach des Haupt-INS, und damit zum vollständigen Verlust von Lenk- und Lageinformationen. Von diesem Zeitpunkt an lieferten die INS an den Flugcomputer keine eigentlichen Flugdaten mehr, sondern im wesentlichen nur noch Diagnoseinformationen.

Der Bordcomputer interpretierte die INS-Diagnoseinformationen als normale Flugdaten, und stellte fälschlicherweise eine große Abweichung von der Flugbahn fest. Daraufhin sendete der Computer an die Düsen und kurz darauf an das Triebwerk der Hauptstufe das Signal zur Schwenkung, um die vermeintliche Abweichung zu korrigieren. Durch die Schwenkung der Düsen wich die Rakete mit über 30 Grad pro Sekunde von ihrem Kurs ab. Die Rakete war dem großen Angriffswinkel der Luftströmung nicht gewachsen und begann angesichts der hohen aerodynamischen Kräfte auseinanderzubrechen. Nachdem die Verbindungen zwischen den Düsen und der Hauptstufe abrissen, leitete die Rakete – wie in diesem Fall vorgesehen – die automatische Selbstzerstörung ein und explodierte. Anschließend aktivierte zusätzlich die Bodenkontrolle den Befehl zur Sprengung, wenngleich die Rakete zu diesem Zeitpunkt bereits zerstört war.

In diesem Beispiel war es eine unbemannte Rakete aber beim nächsten Vorfall könnten Menschen am Board der Rakete sein!

Beispiel CONFIRM-Projekt:

Im CONFIRM-Projekt sollte im Auftrag großer amerikanischer Hotel- und Mietwagenunternehmen ein neues, umfassendes Reservierungssystem entwickelt werden. Nach dreieinhalb Jahren Entwicklungszeit und Investitionen von rund 125 Millionen Dollar wurde das Projekt im Juli 1992 eingestellt, als erkannt wurde, dass die gestellten Anforderungen mit dem gewählten Lösungsansatz nicht erreichbar waren.

Diese beiden Vorfälle können als Paradebeispiele dafür angesehen werden, dass die Tests und die Validierung der Software sowie die Korrektheit immens wichtig sind, wie im Beispiel der Ariane 5. Andererseits sind die gewählten Ansätze und Vorgehensweisen auch mit Bedacht zu wählen. Dazu muss man sich im klaren sein, wo das Projekt hinführen soll und wie man da hin kommt. Um also eine korrekte und gute Software zu entwickeln, muss man sich klar machen, was genau Software eigentlich ist, worauf muss ich in der Entwicklung alles achten und wie gehe ich in meiner Entwicklung vor?

2. Software

2.1 Definition Software (IEEE 610.12)

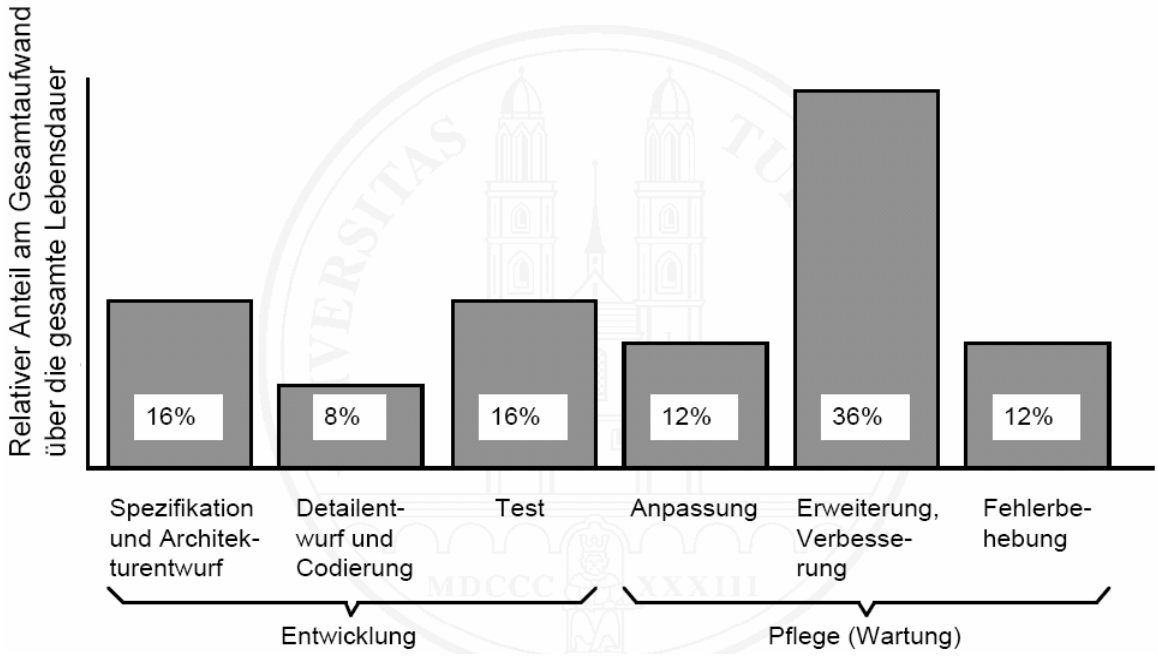
"Software umfasst die Programme, Verfahren, zugehörige Dokumentation und Daten, die mit dem Betrieb eines Computersystems zu tun haben."

Aus dieser Definition lassen sich nun 3 Feststellungen über Software ziehen:

2.2 Feststellungen aus der Definition

Feststellung 1: *"Software umfasst erheblich mehr als nur Programme."*

Der Aufwand für das Schreiben der Programme beträgt in der Regel nur ca. 10-20% (siehe folgendes Diagramm von Boehm) des Gesamtaufwands für die Entwicklung der Software. Da die Programme jedoch der entscheidende Bestandteil von Software sind , wird bei der Schätzung des Aufwands für die Entwicklung viel zu viel nur auf die Programme abgestellt. Dadurch wird der tatsächliche Aufwand um ein vielfaches unterschätzt, ersichtlich in einer Verteilung des Aufwands über die Lebensdauer eines Software-Produktes nach Boehm aus dem Jahr 1971 (Quelle: cs.uni-muenster.de):



Feststellung 2:
"Software ist ein immaterielles technisches Produkt. Man kann Software nicht anfassen."

Die Konsequenzen aus dieser recht trivial erscheinenden Feststellung sind vielfältig:
- Im Gegensatz zu materiellen Produkten gibt es für Software keine natürlichen Grenzen in Form von Materialeigenschaften oder Naturgesetzen. Es gibt einzig die theoretische Grenze der Berechenbarkeit, aber diese wirkt sich in der Praxis kaum aus. Während z.B. ein Brückenbauingenieur bei seinen Konstruktionen Rücksicht nehmen muss auf die Eigenschaften der verwendeten Materialien und auf die Gesetze der Statik und Schwingungsdynamik, ist ein Software-Entwickler grundsätzlich in der Wahl seiner Konstruktionen frei. Software findet ihre praktischen Grenzen nur in den Fähigkeiten ihrer Entwickler. Da diese Grenzen sehr weit gesteckt sind und Menschen außerdem dazu neigen, die Begrenztheit ihres Könnens zu verdrängen, werden bei Software häufiger und leichtsinniger (zu) schwierige Vorhaben angegangen als in anderen technischen Disziplinen.
- Bei der Entwickelung von materiellen Produkten sind Fehler leicht bemerkbar. Wenn beim Bauen das Dach in die Baugrube gesetzt wird, ist dies unschwer als Fehler zu erkennen. Ein Fehler ähnlicher schwere in Software ist nur durch sorgfältige und aufwendige Prüfmaßnahmen zuverlässig erkennbar.
- Gleiches gilt für die Beurteilung des Entwicklungsstands. Der Fertigstellungsgrad einer Software ist oft nur schwer und sehr aufwendig bestimmbar.
- Software ist scheinbar flexibel und leicht zu ändern, da es keine materiellen Produkte gibt, die umgeformt werden müssen. Bei Software mit geringem Umfang trifft dies tatsächlich zu. Jedoch führt die Änderung bei einer umfangreicheren Software oftmals zu sehr vielen Effekten und Konsequenzen, die alle bedacht werden müssen, so dass der Aufwand dieser Änderung oftmals höher ist als für vergleichbare materielle Produkte.

Feststellung 3: *"Software verhält sich (im mathematischen Sinn) unstet. "*

Software ist ein Bestandteil eines digital arbeitenden Rechnersystem. Solche System haben oft die unangenehme Eigenschaft das kleinste Veränderungen (im Extremfall reicht das setzen eines Punktes anstelle eines Kommas) massive Veränderungen im Verhalten des Systems zu Folge haben. Im schlimmsten Fall funktioniert die Software denn gar nicht mehr. Es ist daher schwieriger das wunschgemäße Funktionieren der Software mit einer gegebene Wahrscheinlichkeit zu gewährleisten, wie es zum Beispiel bei dem Bau einer Brücke der Fall ist, denn dort führt das Fehlen einer Schraube noch nicht zum Einsturz der gesamten Brücke. Dort kann man auch genau sagen, wann das fehlen einer Schraube zum Einsturz führt oder nicht. Des Weiteren sind hier Sicherungsmaßnahmen eingebaut, die das Einstürzen der Brücke verhindern sollen.
Bei Software hingegen kann man nicht sagen, ob das setzen eines Kommas statt eines Punktes nun zum Absturz des Programms führt oder nicht. Dies ist von Programm zu Programm auch wieder unterschiedlich, je nachdem wie viel Sorgfalt der Programmierer in die Fehlerbehandlung investiert hat, ob er Fehlerbehandlungsroutinen eingebaut hat oder nicht. Und was diese Routinen dann machen, kann auch in jedem Programm wieder unterschiedlich sein. So könnte in einem Fall die Routine dazu führen, dass das Programm einfach beendet wird oder es könnte auch sein, dass es eine ausführliche Fehlerbeschreibung liefert und Lösungsvorschläge liefert und nicht beendet wird.

3. Softwareentwicklung

3.1 Definition Softwareentwicklung

Softwareentwicklung umfasst alle Tätigkeiten und Ressourcen, die zur Herstellung von Software notwendig sind. Es ist die Umsetzung der Bedürfnisse von Benutzern in Software.
Konkret umfasst dies:
- Spezifikation der Anforderungen
- Konzept der Lösung
- Entwurf und Programmierung der Komponenten
- Zusammensetzung der Komponenten
- Einbindung in vorhandene Software
- Inbetriebnahme der Software
- Überprüfung des Entwickelten nach jedem Schritt

3.2 Gesetzmäßigkeiten bei der Softwareentwicklung

"Die Entwicklung von kleinen Programme unterscheidet sich fundamental von der Entwicklung großer Programme"

Die Entwicklung von großen Programmen stellt dabei den Normalfall dar. Viele Probleme existieren bei der Entwicklung kleiner Programme gar nicht. Die Erfahrung, dass die Entwicklung kleiner Programme recht einfach ist, ist der Hauptgrund für den Trugschluss, dass Softwareentwicklung generell etwas Einfaches sein müsse.

Damit man im folgenden eine Unterscheidung zwischen „kleinen“ und „großen“ Programmen hat ist hier ein kleiner Vergleich zwischen diesen:

kleine Programme	**große Programme**
Programme von bis zu ein paar tausend Zeilen oder auch alle Programme, wo der Benutzer die Übersicht im Kopf behalten kann	Längere Programme, auch bis zu Millionen Zeilen
Für den Eigengebrauch	Für den Gebrauch durch Dritte
Vage Zielsetzung genügt	Genaue Zielbestimmung, d.h. Spezifikation der Anforderungen erforderlich
Ein Schritt vom Problem zum Ziel	Mehrere Schritte:Spezifikation, Konzept der Lösung, Entwurf der Teile, Programmieren der Teile, Zusammensetzen der Teile, Inbetriebnahme

Validierung und nötige Korrekturen finden am Endprodukt statt	Auf jedem Entwicklungsschritt muss ein Prüfschritt folgen, sonst wird das Risiko, dass das Endprodukt unbrauchbar ist, zu groß
Eine Person entwickelt! Keine Koordination und Kommunikation erforderlich	Mehrere Personen entwickeln: Koordination und Kommunikation notwendig
Komplexität der Software ist in der Regel klein. Das heißt es gibt nur wenige Abhängigkeiten, von Klassen oder Modulen zu anderen Komponenten. Das Strukturieren und Behalten der Übersicht ist nicht schwer	Komplexität sehr hoch: explizite Maßnahmen zur Strukturierung und Modularisierung erforderlich, da viele Abhängigkeiten bestehen
Software besteht nur aus wenigen Komponenten	Software besteht aus vielen Komponenten, die eine Maßnahme zur Komponentenverwaltung erfordern
In der Regel wird keine Dokumentation erstellt	Dokumentation dringend erforderlich, damit Software wirtschaftlich betrieben und gepflegt werden kann
Keine Planung und Projektorganisation erforderlich	Planung und Organisation dringend erforderlich für zielgerichtetes und wirtschaftliches entwickeln

Tabelle: Vergleich zwischen „kleinen“ und „großen“ Programmen

Um den Anstieg der Komplexität einmal grafisch darzustellen ist hier mal der Kommunikationsbedarf in Abhängigkeit der beteiligten Personen aufgezeigt:

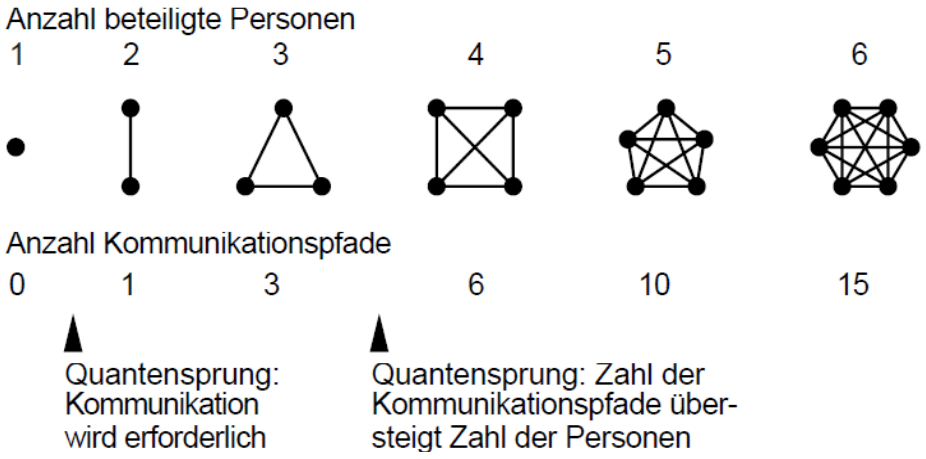


Abbildung: Wachstums und Quantensprünge beim Kommunikationsbedarf
Quelle: ba.db-nico.de

Wie aus dem Diagramm ersichtlich ist, wird schon bei einer geringen Anzahl von Personen die Kommunikation sehr umfangreich, was eine gute Koordination voraussetzt, sonst könnte dies sehr schnell im Chaos enden. Dazu sollten die Kommunikationspfade eingeschränkt werden und nur auf die Nötigsten reduziert werden.

"Software ist einer Evolution unterworfen."

Die Umwelt und die Bedürfnisse der Benutzenden verändern sich ständig. Software, die sich in Gebrauch befindet, bleibt daher ohne ständige Anpassung nicht gebrauchstauglich. Des Weiteren werden immer wieder Fehler entdeckt, die behoben werden müssen. Solche Entwicklungsarbeiten an in Betrieb befindlicher Software nennt man Pflege oder Wartung.
Das Tempo der Software-Evolution ist so hoch, dass häufig schon während der Entwicklung die Entwicklungsziele an veränderte Bedürfnisse angepasst werden müssen.
Die erforderlichen Anpassungen und Erweiterungen bei der Pflege bringen es mit sich, dass Umfang und innere Unordnung von in Gebrauch befindlicher Software ständig zunehmen. Die Pflege einer Software wird daher mit zunehmendem Alter immer schwieriger und teurer.

"Software wird von Menschen gemacht."

Die Fähigkeiten dieser Menschen sowie ihre emotionalen Einstellungen haben einen direkten und erheblichen Einfluss auf die von ihnen entwickelte Software. Hinzu kommt, dass kein Mensch perfekt ist: Jeder macht mal Fehler.

3.4 Warum ist Softwareentwicklung so schwierig?

Wenn man die oben genannten Feststellungen und die ihnen zugrunde liegenden Phänomene analysieren, so stellen man fest, dass es im wesentlichen vier Faktoren sind, welche die Entwicklung so schwierig machen:

- Die Größe der zu lösenden Probleme. Software ist nicht einfacher, als die Probleme die sie löst. Je größer und schwieriger die Software, desto aufwendiger und schwieriger ist die Entwicklung.
- Die Tatsache, das Software ein immaterielles Produkt ist. Die Immaterialität macht das Arbeiten mit Software schwieriger als dasjenige mit materiellen Produkten vergleichbarer Komplexität. Die Risiken sind schwer zu erkennen. Ad-hoc-Vorgehensweisen führen schneller ins Desaster.
- Sich permanent verändernde Ziele aufgrund der Evolution. Schon das Bestimmen und Erreichen fixierter Ziele bei der Entwicklung ist keine leichte Aufgabe. Sich verändernde Ziele machen das ganze nochmal um eine Größenordnung schwieriger.
- Fehler infolge von Fehleinschätzungen (z.B. was im Kleinen geht, geht genauso im Großen). Software-Entwicklung wird daher unbewusst meist als viel einfacher eingeschätzt, als sie tatsächlich ist. Dies führt zu unrealistischen Erwartungen und zu von Beginn an zu tiefen Kosten- und Terminschätzungen. Eine besonders häufige Fehlerursache ist das Denken und Handeln in der Welt von kleinen Programmen, während tatsächlich große Programme zu entwickeln sind. Besonders verheerend wirkt sich die lineare Extrapolation von Erfahrungen mit kleinen auf großen Programmen aus:

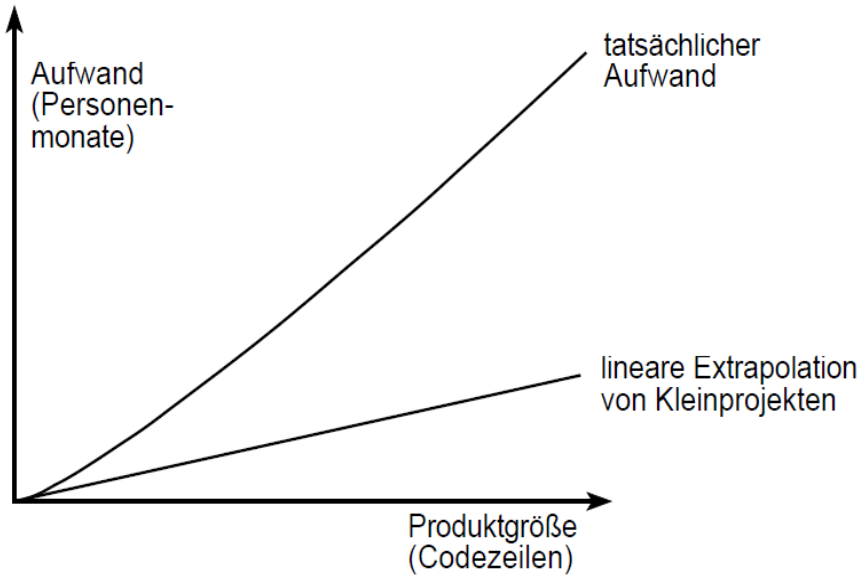


Abbildung: Fehleinschätzung bei linearer Interpolation
Quelle: www.ifi.uzh.ch

3.5 Die 13 häufigsten Fehler in der Softwareentwicklung

1. Es wird mit der Codierung sofort angefangen
2. Es wird nicht systematisch bzw. unzureichend getestet
3. Eine Festlegung der Anforderungen/Qualitätsmerkmale fehlt
4. Standards und Richtlinien werden nicht beachtet
5. Die Dokumentation fehlt bzw. ist veraltet, unzureichend oder nicht adäquat
6. Ein Vorgehensmodell fehlt, bzw. wird nicht verfolgt
7. Eine Abnahme der Phasenergebnisse erfolgt nicht
8. Schlechte Namensvergabe wie z.B. File-, Klassen-, Methoden- und Variablennamen
9. Die Systemarchitektur ist nicht oder nur sehr umständlich erweiterbar (fehlende Datenkapselung, fehlende Modularität)
10. Die Schulung für die Software-Ersteller und -Anwender wird vernachlässigt oder als nicht notwendig angesehen
11. Die Terminvorgaben sind unrealistisch
12. Begriffe werden nicht definiert
13. Die Auswahl der Werkzeuge/Methoden ist unzureichend vorbereitet

Für Ursachen, Folgen und Maßnahmen dieser Fehler lesen sie den Artikel auf folgender Seite:
http://www.theoinf.tu-ilmenau.de/~riebisch/swqs/fehler.html

4. Teilgebiete der Softwareentwicklung

4.1 Projektmanagement

Definition (DIN 69901-5:2009-01):
"Gesamtheit von Führungsaufgaben, -organisation, -techniken und -mitteln für die Initiierung, Definition, Planung, Steuerung und den Abschluss von Projekten."

Definition (Project Management Institute (PMI)):
"Projektmanagement ist die Anwendung von Wissen, Können, Werkzeugen und Techniken auf Projektaktivitäten, um Projektanforderungen zu erfüllen."

Definition (Gesellschaft für Informatik):
"Das Projekt führen, koordinieren, steuern und kontrollieren."

Definition (IPMA Competence Baseline (ICB)):
"Führung der Projektbeteiligten zur sicheren Erreichung der Projektziele."

Textuell sind diese Definitionen sehr verschieden aber inhaltlich sagen sie vom Prinzip her das gleiche aus. Das Projektmanagement besteht aus folgenden Teilgebieten:

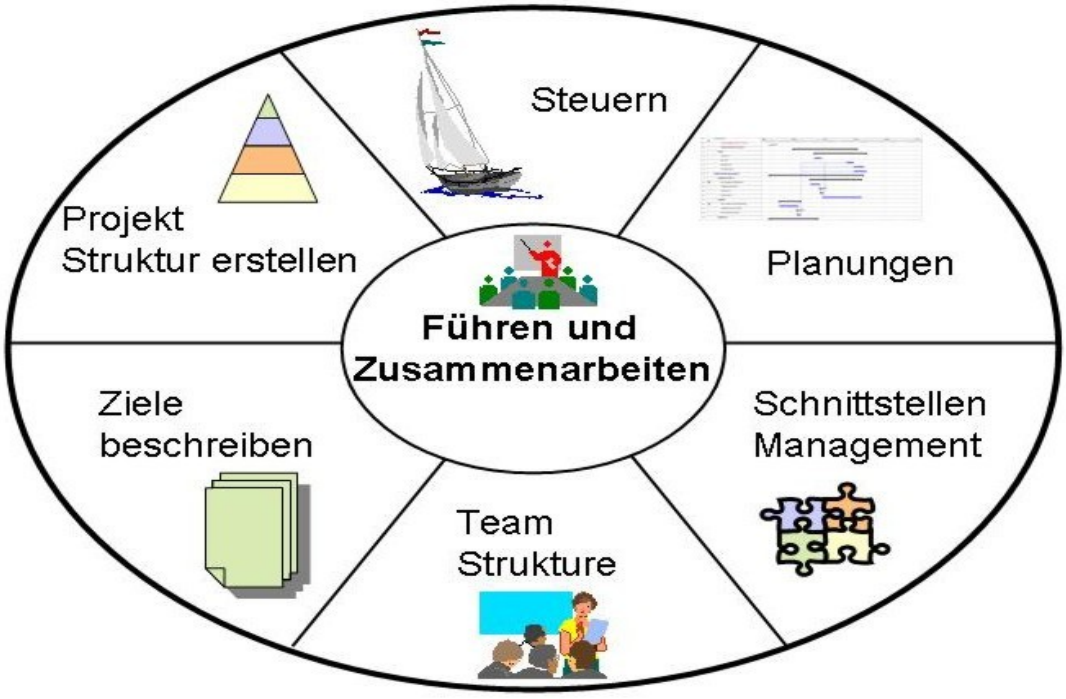
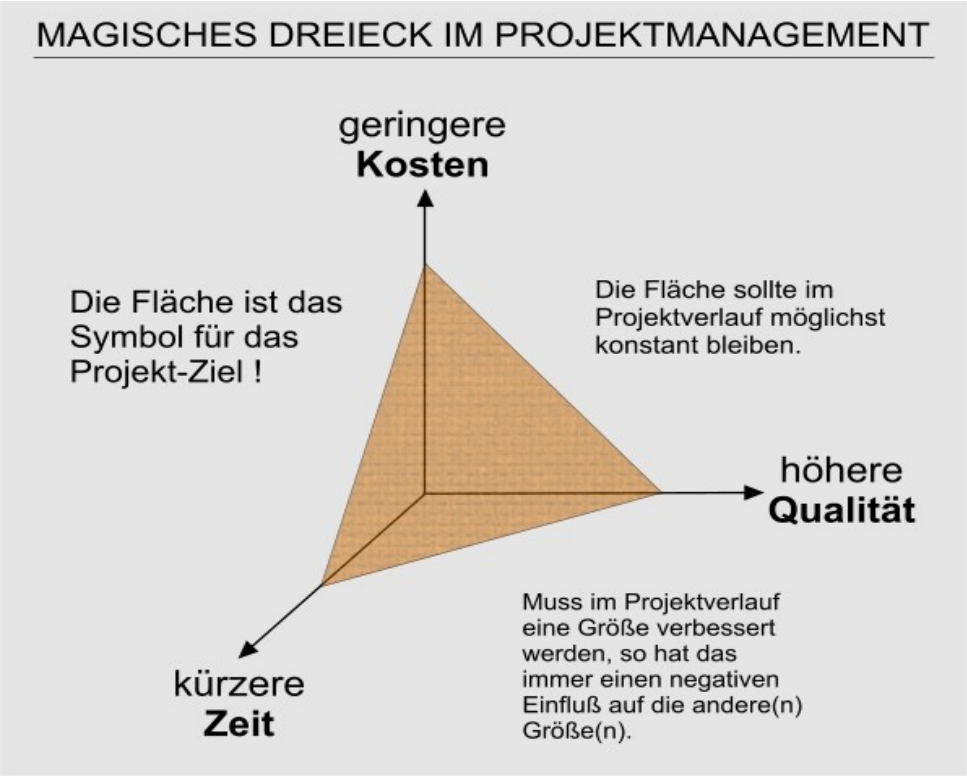


Abbildung: Teilgebiete im Projektmanagement
Quelle: innovationen-machen.de

Der Projektmanager muss eine Übersicht über das ganze Projekt haben, sowie alle Teammitglieder und deren Fähigkeiten kennen, damit er seine Aufgaben effizient ausführen kann.



Quelle: realtime-solutions.de

Des Weiteren sollte der Projektmanager auch immer ein Auge auf das so genannte „magische Dreieck“ haben. Die Änderung einer Größe in diesem Dreieck hat direkten Einfluss auf die anderen Größen. Soll das Projekt zum Beispiel in kürzerer Zeit fertig gestellt werden, so ist sofort erkenntlich dass dies höhere Kosten zur Folge hat oder auf die Qualität Einfluss hat, denn Qualität kostet Zeit. Durch kürzere Zeit muss eventuell auch ein weiterer Programmierer oder dergleichen eingestellt werden, was wiederum höhere Kosten zur Folge hat.
Der Projektmanager sollte darauf achten, dass dieses Dreieck, während des gesamten Projektes „konstant“ bleibt. Er soll also darauf achten, dass das Projekt in der vorgesetzten Zeit, zu den gegebenen Kosten und mit gegebener Qualität fertig gestellt wird, es nicht zu Termin- oder Kostenüberschreitungen kommt.

4.2 Qualitätsmanagement

Qualitätsmanagement bezeichnet alle organisierten Maßnahmen, die der Verbesserung von Produkten, Prozessen oder Leistungen jeglicher Art dienen. Qualitätsmanagement ist eine Kernaufgabe des Managements.
Das Qualitätsmanagement ist auch dafür zuständig die Softwarequalität zu gewährleisten.

Definition Softwarequalität (DIN ISO 9126):
"Unter Softwarequalität versteht man die Gesamtheit der Merkmale und Merkmalswerte eines Softwareprodukts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen"

Bestandteile des Qualitätsmanagements:
- Planung (Plan):
  - Es wird ein Ist-Zustand ermittelt und die Rahmenbedingungen für das Qualitätsmanagement festgelegt. Danach werden Konzepte und Abläufe erarbeitet.
- Lenkung (Do):
  - Die in der Planphase erarbeiteten Ergebnisse werden umgesetzt.
- Sicherung (Check):
  - Auswerten der Qualitätsinformationen (Kosten-Nutzen-Betrachtungen, Überprüfen von gemachten Abnahmen)
- Verbesserung (Act):
  - Aus vorheriger Phase gewonnene Informationen werden für Strukturverbesserungsmaßnahmen und Prozessoptimierung eingesetzt

Es muss also ständig ein Vergleich zwischen dem Ist-Zustand und dem Soll stattfinden, um die Qualität der Software sowie die Durchführbarkeit zu gewährleisten.
So erhalten wir für das Qualitätsmanagement folgenden Kreislauf:

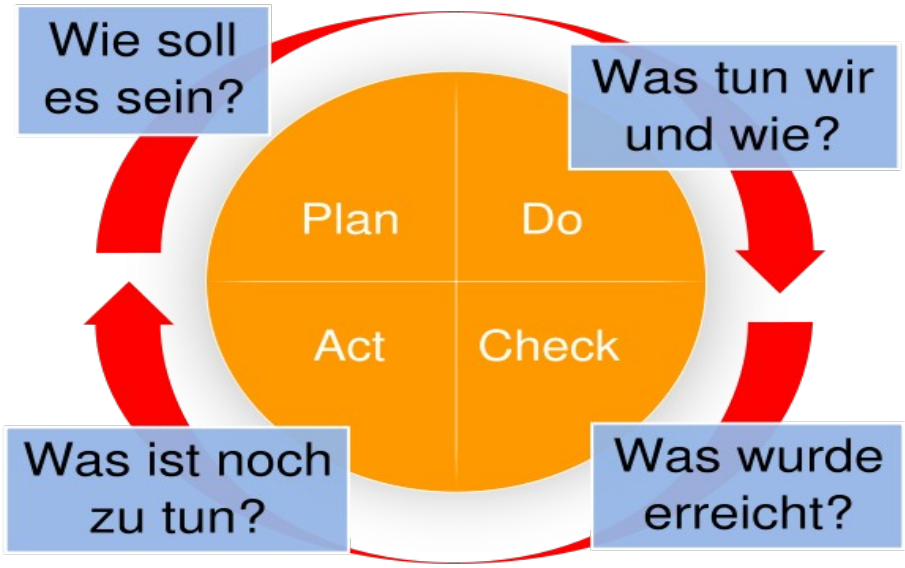


Abbildung: Bestandteile des Qualitätsmanagement
Quelle: commons.wikimedia.org

4.3 Risikomanagement

Risikomanagement ist die systematische Erfassung und Bewertung von Risiken sowie die Steuerung von Reaktionen auf die erkannten Risiken.

Das Risikomangement besteht aus 6 Phasen:

- Managementplanung
  - Hier werden die Verfahren festgelegt, mit denen die folgenden Prozesse arbeiten
- Identifikation
  - Während der Risikoidentifikation werden Risiken mit verschiedenen Methoden identifiziert und dokumentiert
- Qualitative Analyse
  - Die identifizierten Risiken werden qualifiziert, hierzu gehört die Priorisierung der Risiken auf Basis des Einwirkens auf den Projekterfolg.
- Quantitative Analyse
  - Hier wird versucht die Risiken nummerisch exakt zu berechnen
- Planung zur Bewältigung
  - Die Planung der Risikobewältigung ermittelt Gegenmaßnahmen, um die Risiken zu bewältigen oder die Auswirkungen zu minimieren
- Überwachung und Verfolgung
  - Hier wird der Status der Risiken kontinuierlich überwacht.

Dabei gibt es 5 unterschiedliche Risikosteuerungsstrategien:

- Vermeidung
  - Eine vollständige Vermeidung kann nur erreicht werden, indem man risikoreiche Aktivitäten unterlässt. Sinnvoll ist dies nur bei bestandsgefährdenden Risiken.
- Verminderung
  - Diese Strategie setzt darauf die Risiken auf ein Minimum zu reduzieren
- Begrenzung
  - Die Risikobegrenzung versucht, dass die Summe aller Risiken eine definierte Obergrenze nicht überschreitet
- Überwälzung
  - Bei der Risikoüberwälzung wird nicht an einer Vermeidung oder Minderung des Risikos gearbeitet, sondern an einer Weitergabe des Risikos an Dritte. Unterschieden werden kann hier an die Weitergabe an Versicherungsunternehmen oder Vertragspartner.
- Akzeptanz
  - Die Verminderung, Begrenzung und Überwälzung von Risiken kann diese nicht vollständig ausschließen. Das verbleibende Restrisiko muss das Unternehmen akzeptieren und selbst tragen. Die Akzeptanz von Risiken sollte dann gewählt werden, wenn die eben beschriebenen Wege in keiner guten Kosten-Nutzen-Relation stehen

4.4 Anforderungserhebung

Laut **IEEE** kann die Anforderungserhebung in
- Aufnahme
- Analyse
- Spezifikation
- Bewertung

unterteilt werden.
Diese Schritte überlappen sich und werden auch oft mehrfach und iterativ ausgeführt.

Folgenden Kriterien müssen beim Sammeln der Anforderungen erfüllt sein:
- vollständig – alle Anforderungen des Kunden müssen spezifiziert sein
- eindeutig definiert – präzise Definition, um Missverständnisse zu vermeiden
- verständlich beschrieben – damit der Auftraggeber und Entwickler dies lesen und verstehen kann
- atomar – es darf nur eine Anforderung pro Abschnitt beschrieben sein
- identifizierbar – jede Anforderung muss eindeutig identifizierbar sein
- einheitlich dokumentiert – die Anforderungen und Quellen sollten nicht in unterschiedlichen Dokumenten stehen oder unterschiedliche Strukturen haben
- nachprüfbar – die Anforderungen sollten mit Abnahmekriterien verknüpft sein
- rück- und vorwärtsverfolgbar – damit einerseits erkennbar ist, ob jede Anforderung erfüllt wurde und andererseits für jede implementierte Funktionalität erkennbar ist aus welcher Anforderung sie entstanden ist

Nach der Erfassung der Anforderungen findet eine **Strukturierung und Klassifizierung** der Anforderungen statt. Damit erreicht man eine bessere Übersicht, was wiederum das Verständnis der Beziehungen zwischen den Anforderungen erhöht.
Kriterien, die hier zu beachten sind:
- abhängig – Anforderungen müssen darauf analysiert werden, ob sie sich nur gemeinsam realisieren lassen oder Voraussetzungen für andere sind
- zusammengehörig – Anforderungen die zusammengehören, sollten nicht alleine realisiert werden
- rollenbezogen – jede Benutzergruppe hat ihre eigene Sicht auf die Anforderungen

Nach der Strukturierung erfolgt anschließend die **Prüfung und Bewertung der Anforderungen.**
Kriterien hierfür sind wiederum:
- korrekt – Anforderungen müssen insbesondere widerspruchsfrei sein
- machbar – sie müssen realisierbar sein
- notwendig – was nicht vom Auftraggeber gefordert ist, ist auch keine Anforderung
- priorisiert – es muss erkennbar sein, welche Anforderungen am wichtigsten sind
- nutzbar, nützlich – auch bei teilweiser Realisierung soll bereits ein produktives System entstehen
- benutzerfreundlich – die Benutzerfreundlichkeit des Systems muss sichergestellt sein

4.5 Systemdesign / technische Konzeption

Ein Systemdesigner oder in kleineren Projekten auch der Programmierer, legt anhand des Pflichtenheftes die Programmarchitektur fest.
Soweit Standardsoftwareprodukte zum Einsatz kommen, erfolgt in dieser Phase auch eine Spezifikation der geplanten Produkteinbindung oder Produktanpassung.
Für neu zu entwickelnde Software erfolgt der Entwurf des Datenmodells und der einzelnen Funktionen und Algorithmen sowie der Objekt- und Klassenstruktur.
Falls bereits vorhandene Software angepasst werden soll, so wird hier festgelegt, welche Veränderungen und Erweiterungen erforderlich sind.

4.6 Implementierung

In der Implementierungsphase wird die zuvor konzipierte Anwendungslösung technisch umgesetzt, indem Softwareprodukte konfiguriert, vorhandene Software angepasst oder Programme vollständig neu erstellt werden.

4.7 Softwaretest

Ein Softwaretest ist ein Test, der im Rahmen der Softwareentwicklung durchgeführt wird. Er bewertet die Funktionalität der Software gemäß ihrer Anforderungen und misst ihre Qualität. Die aus dem Softwaretest gewonnenen Erkenntnisse werden zur Behebung und Vermeidung von Softwarefehlern herangezogen.

Ein Test hat dabei mehrere Phasen:
- Planung
  - In diesem wird genau festgelegt, was getestet wird, wie es getestet wird, unter welchem Umständen, welche Randbedingungen es gibt, wer testet und alle Rahmenbedingungen, die es für die Tests gibt
- Vorbereitung
  - Aufbauend auf dem Testplan werden hier nun alle Bedingungen und Komponenten bereitgestellt
- Spezifikation
  - Hier werden alle Bedingungen definiert, die nötig sind um einen speziellen Testfall durchführen zu können
- Durchführung
  - Hier wird nun der festgelegte Testfall durchgeführt
- Auswertung
  - Nach der Durchführung findet nun die Auswertung statt, indem man zum Beispiel den Ist-Zustand mit dem Soll-Zustand vergleicht
- Abschluss
  - Hier wird nun entschieden, ob der Test zufriedenstellend verlaufen ist oder nicht und je nachdem werden Maßnahmen eingeleitet

Weiterhin gibt es verschiedene Test, so gibt es zum Beispiel:
- Komponententest – abgegrenzte Teile (Module) der Software werden getestet
- Integrationstest – testet die Zusammenarbeit voneinander abhängiger Komponenten
- Systemtest – Dieser Test testet das gesamte System in Bezug auf die gesamten Anforderungen
- Abnahmetest – Dies ist ein Test durch den Kunden, der das System selbst testet und dann entscheiden kann, ob das System den Anforderungen entspricht

4.8 Softwareeinführung

Die fertiggestellte Software wird auf den Computersystemen des Auftraggebers oder des Betreibers installiert.
Bei größeren Projekten wird das System oftmals auch erst auf wenigen Testsystemen installiert und erst anschließend nach Testläufen auf die anderen Systeme ausgeweitet.
Weiterhin findet hier auch die Einführung in das System für die Benutzer statt. Dazu werden unter anderem Schulungsmaßnahmen angeboten.

4.9 Wartung / Pflege

Nach der Inbetriebnahme einer Software ist eine kontinuierliche Weiterbetreuung durch den Auftragsnehmer üblich. Dies umfasst sowohl eine Unterstützung per Hotline als auch Vor-Ort-Support, was wiederum in einem Vertrag geregelt ist.

Die laufenden Anpassung der Software an sich verändernde Anforderungen und an neue Versionen verwendeter Standardsoftware wird als Softwarepflege bezeichnet und wird üblicherweise auch in Verträgen festgehalten.

5. Vorgehensmodelle in der Softwareentwicklung

Vorgehensmodelle geben in der Softwareentwicklung den Rahmen vor, in welchem der Entwicklungsprozess organisiert wird.
Dabei werden alle Tätigkeiten, die bei der Entwicklung einer Software nötig sind, in einzelne Phasen zerlegt. Es gibt verschiedene, gängige Vorgehensmodelle.

5.1 Wasserfallmodell

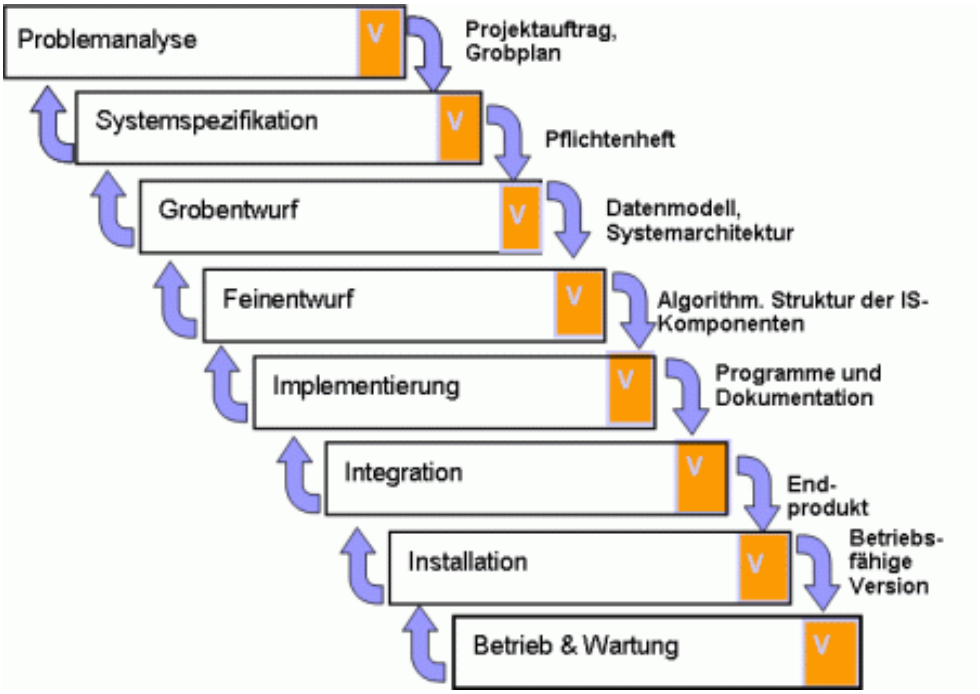


Abbildung: Aufbau des erweiterten Wasserfallmodells
Quelle: winfwiki.wi-fom.de

Das Wasserfallmodell stellt in der Softwareentwicklung die einzelnen Phasen in Form eines Wasserfalls dar. Das Modell zeichnet sich durch einen sequentiellen Charakter aus - es besteht nicht die Möglichkeit, auf eine vorhergehende Phase zurückzugreifen.
Beim erweiterten Wasserfallmodell wurde diese Einschränkung berücksichtigt und korrigiert:
Es ist eine Rückkehr in vorhergehende Phasen möglich. (iteratives Phasenmodell)

Eigenschaften des Wasserfallmodells:

- Aktivitäten sind in der vorgegebenen Reihenfolge und in der vollen Breite vollständig durchzuführen.
- Am Ende jeder Aktivität steht ein fertiggestelltes Dokument, d.h. das Wasserfallmodell ist ein „dokumentgetriebenes“ Modell.
- Der Entwicklungsablauf ist sequentiell
- Es orientiert sich am Top-down-Verfahren.
- Es ist einfach, verständlich und benötigt nur wenig Managementaufwand.
- Eine Benutzerbeteiligung ist nur in der Anfangsphase vorgesehen, anschließend erfolgen der Entwurf und die Implementierung ohne Beteiligung des Benutzers bzw. Auftraggebers. Weitere Änderungen stellen danach Neuaufträge dar.

Vorteile:
- klare Abgrenzung der Phasen
- einfache Möglichkeiten der Planung und Kontrolle
- bei stabilen Anforderungen und klarer Abschätzung von Kosten und Umfang ein sehr effektives Modell

Nachteile:
- Das Modell ist nur bei einfachen Projekten anwendbar
- Unflexibel gegenüber Änderungen und im Vorgehen
- Frühes festschreiben der Anforderungen ist sehr problematisch und kann zu teuren Änderungen führen
- Fehler werden eventuell erst sehr spät erkannt und müssen mit erheblichen Aufwand entfernt werden.

Eine ausführlichere Beschreibung des Wasserfallmodells liefert folgender Artikel:
http://www.scrum-kompakt.de/grundlagen-des-projektmanagements/wasserfall-modell/

5.2 V-Modell

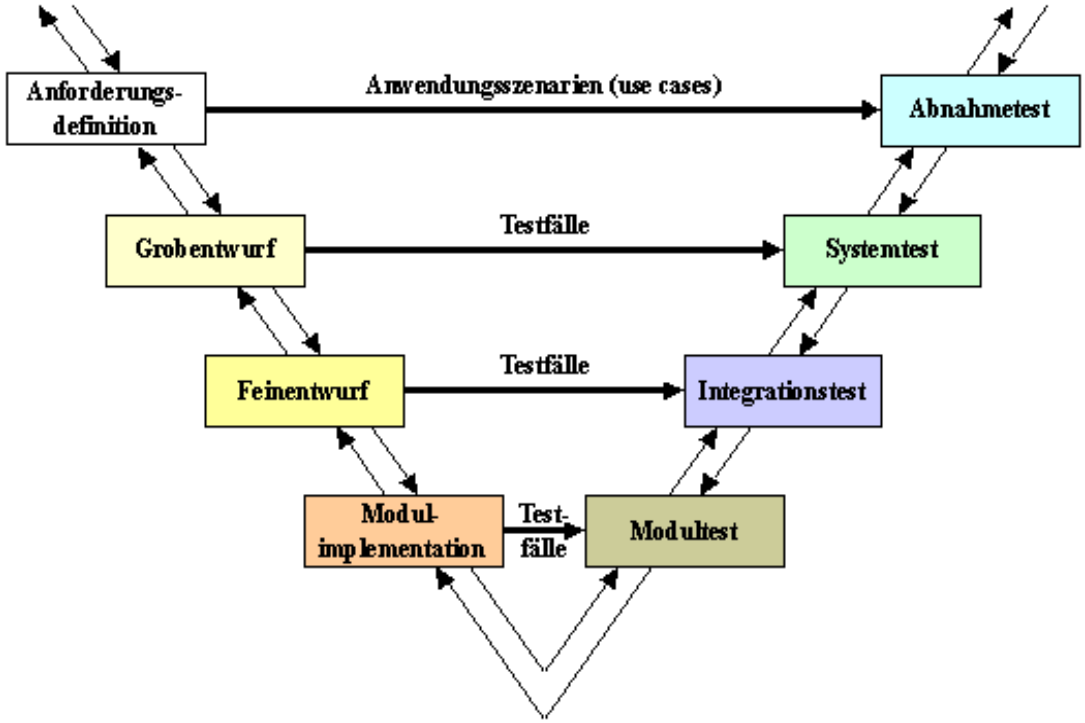


Abbildung: Aufbau des V-Modell
Quelle: it-infothek.de

Das V-Modell ist ein Vorgehensmodell für die Planung und erfolgreiche Durchführung von Systementwicklungsprojekten. Es beinhaltet Projektmanagement und Qualitätssicherung. Der Begriff leitet sich von der V-förmigen Darstellung der Projektelemente aus der Spezifikation und Zerlegung im absteigenden Ast sowie Realisierung und Integration im aufsteigenden Ast her.

Das V-Modell ähnelt dem Wasserfallmodell, erlaubt im Gegensatz zu diesem jedoch die Rückkopplung auf die vorhergehenden Phasen.

Tätigkeitsbereiche des V-Modell:
- Software-Erstellung
- Qualitätssicherung
- Konfigurationsmanagement
- Projektmanagement

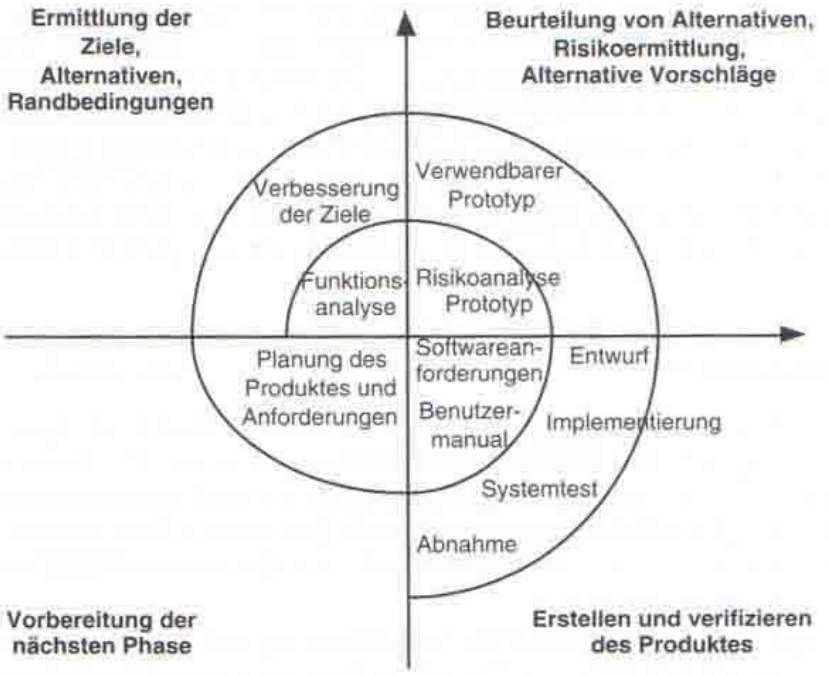
Vorteile:
- Integrierte und detaillierte Darstellung von den Tätigkeitsbereichen
- Generisches Modell mit definierten Möglichkeiten zur Anpassung an projektspezifische Anforderungen
- Gut geeignet für große Projekte

Nachteile:
- Für kleine und mittlere Softwareentwicklungen führt das V-Modell zu einem unnötigen Overhead
- Die im V-Modell definierten Rollen (bis zu 25) sind für gängige Softwareentwicklungen nicht realistisch
- Ohne geeignete Werkzeug-Unterstützung ist das V-Modell nur sehr schwer bis gar nicht handhabbar

Weiter Informationen: http://de.wikipedia.org/wiki/V-Modell
Oder eine ausführliche Beschreibung des gesamten V-Modell liefert die Seite:
http://v-modell.iabg.de
Dort wird das gesamte V-Modell und alle Versionen, die es von diesem gibt, vorgestellt. Des Weiteren wird dort darauf eingegangen, welche Werkzeuge es gibt, um mit dem V-Modell arbeiten zu können.

5.3 Weitere Vorgehensmodelle

5.3.1 Spiralmodell

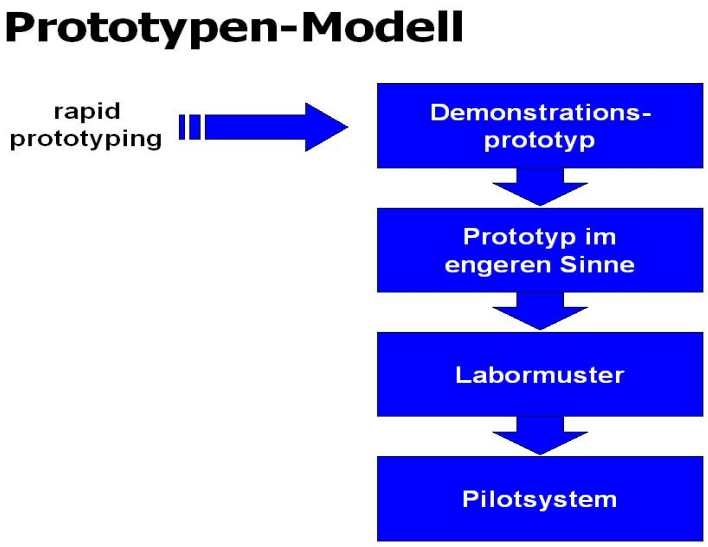


Quelle: imn.htwk-leipzig.de

Das Spiralmodell fasst den Entwicklungsprozess im Software-Engineering als iterativen Prozess auf, wobei jeder Zyklus in den einzelnen Quadranten folgende Aktivitäten enthält:

1. Festlegung der Ziele. Identifikation von Alternativen und Beschreibung von Randbedingungen
2. Beurteilung von Alternativen und eine Risikoanalyse
3. Entwicklung des Prototypen und Überprüfung dieses Produktes durch Tests
4. Planung des nächsten Zyklus

5.3.2 Prototypen-Modell



Quelle: ralfbuerger.de

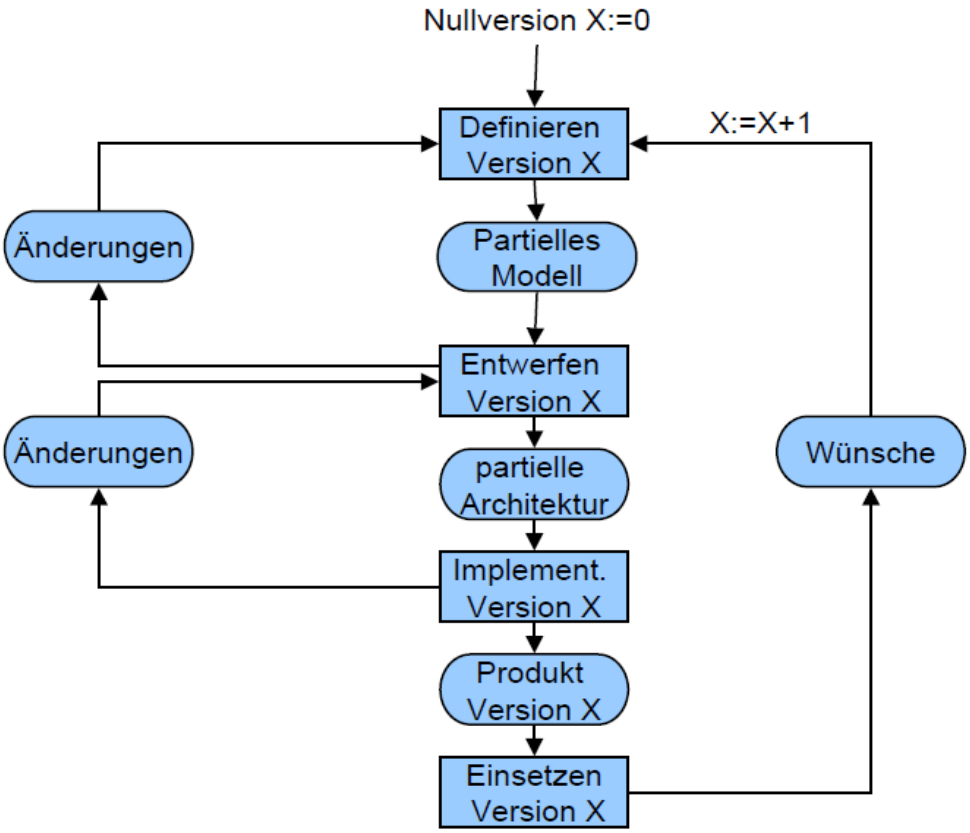
Zitat von http://www.ralfbuerger.de/sse/Vorgehensweise/Modelle.htm

Das "Prototypen-Modell" heißt auch Casey-Modell, womit ein junger Entwicklungsleiter geehrt wird. Dieser kam auf die Idee, einen Prototypen zu bauen, der bereits die gesamte Funktionalität enthält, aber nicht getestet ist, keine Hilfen und keine Plausibilitätskontrollen sowie keine Fehlerbehandlung enthält, weder kommentiert noch dokumentiert ist und auch unter optischen Aspekten völlig anspruchslos ist - also noch kein Produkt ist!

Es geht ganz einfach darum, möglichst schnell ein Stück Software zu haben, das dem Kunden als Basis für weitere Diskussionen dienen kann und den Entwickler spüren lässt, wo die größten technischen Probleme verborgen sind. Ein Prototyp zeigt sehr deutlich, ob der Anbieten seinen Kunden verstanden hat und der Kunde wirklich weiß, was er will.

Heute ist das Prototypen-Modell weiter entwickelt und erzeugt evtl. sogar ein Pilotsystem, das mit den Anwendern des Kunden getestet werden kann. Parallel dazu erfolgt heute auf jeden Fall die Entwicklung des Echtsystems nach einem anderen Modell, denn der Prototyp wird halt weggeworfen.

5.3.3 Evolutionäres Modell



Quelle: www8.informatik.uni-erlangen.de

Bei diesem Modell wird erst der Kern bzw. die Mussanforderungen des Auftraggebers definiert. Anschließend wird als erstes der Produktkern(Nullversion) entworfen und entwickelt. Und die Erfahrungen mit dieser Nullversion führen zu neuen Anforderungen, welche zu einer neuen Version führen.

Merkmale:
- Stufenweise Entwicklung des Produktes, gesteuert durch die Erfahrungen des Endbentuzers
- Gut geeignet, wenn der Auftraggeber seine Anforderungen noch nicht vollständig überblickt
- Die Entwicklung im evolutionären Modell ist codegetrieben

6. Zusammenfassung

Man hat gesehen, dass bei der Softwareentwicklung mehr dran ist, als auf den ersten Blick vielleicht erkannt wird.
Anfangen tut alles schon damit, dass Software nicht nur die Programme selbst umfasst. Dies muss bei der Entwicklung unbedingt bedacht werden, damit Projekte nicht schon von Anfang an zum scheitern verurteilt sein sollen. Die Tatsache, dass man Software nicht anfassen kann, scheint auf den ersten Blick keine Rolle zu spielen, jedoch sind die Konsequenzen hiervon recht umfangreich. So kann die Fehlersuche zum Beispiel sehr viel Zeit in Anspruch nehmen, die gleich zu Anfang mit eingeplant werden sollte.
Berücksichtigt man nun alle Komponenten der Software kann man sich nun an die Entwicklung einer Software machen. Diese wiederum besteht aus einer Vielzahl von Komponenten und unterliegt einigen Gesetzmäßigkeiten, wie zum Beispiel das die Software von Menschen entwickelt wird. Menschen machen Fehler und diese müssen beachtet werden. Des Weiteren kann sich Software sehr unstetig verhalten, denn eine kleine Änderung kann viele unvorhergesehene Änderungen nach sich ziehen. Durch eine gute Strukturierung und ausführliche Definition, welche Abhängigkeiten überall bestehe, kann man dem entgegen wirken.
Ein Blick auf häufig gemachte Fehler schadet auch nicht, denn daraus kann man für sein eigenes Projekt viel lernen und von vorne herein versuchen, diese Fehler zu vermeiden und dadurch das Risiko für sein Projekt zu vermindern.
Bei der Entwicklung sollten alle Teilgebiete berücksichtigt und besetzt werden, dies sogt für eine gute Aufteilung der gesamten Arbeiten im Projekt und man hat dadurch auch eine bessere Übersicht über das gesamte Projekt zu jedem Zeitpunkt der Entwicklung.
Zum Schluss gilt noch zu beachten, dass man sich auf ein Vorgehensmodell einigt und dies auch konsequent einhält. Dabei hat jedes Modell seine Vor- und Nachteile. Diese muss man für sein Projekt abwiegen und je nachdem das beste Modell aussuchen. So eignet sich das Wasserfallmodell eben nur für kleine Projekte, wo von vornherein die Ziele fest und konstant sind.
Zusammenfassend kann man sagen, dass das Gebiet der Softwareentwicklung sehr groß und komplex ist. Es sind eine Menge Dinge zu beachten, damit das Projekt nicht in einer totalen Katastrophe endet. Aber wird dies beachtet, so macht man einen guten Schritt in die Richtung, ein Projekt zum Erfolg zu führen und die Softwareentwicklung in der Zukunft in den Griff zu bekommen, sodass immer weniger Projekte scheitern. Das alles würde für die Zukunft auch dazu beitragen, dass allgemeine Vertrauen in Software zu steigern.

7. Quellen

Wichtige Quellen:

- http://de.wikipedia.org/wiki/Ariane_V88
- http://de.wikipedia.org/wiki/Softwaretechnik
- http://de.wikipedia.org/wiki/Software_Engineering_Body_of_Knowledge
- http://de.wikipedia.org/wiki/Projektmanagement
- http://de.wikipedia.org/wiki/Qualit%C3%A4tsmanagement
- http://de.wikipedia.org/wiki/Softwarequalit%C3%A4t
- http://de.wikipedia.org/wiki/Risikomanagement
- http://de.wikipedia.org/wiki/Anforderungserhebung
- http://de.wikipedia.org/wiki/Softwaretest
- http://de.wikipedia.org/wiki/Wasserfallmodell
- http://de.wikipedia.org/wiki/V-Modell
- http://www.ifi.uzh.ch/groups/req/ftp/se_I/fallstudie_ariane_501.pdf
- http://www.ifi.uzh.ch/rerg/fileadmin/downloads/teaching/courses/software_engineering_ws0607/folien/Fallstudie_Ariane5.pdf
- http://www.ifi.uzh.ch/groups/req/ftp/se_I/kapitel_01.pdf
- http://www.ifi.uzh.ch/groups/req/ftp/se_I/kapitel_02.pdf
- http://www.ifi.uzh.ch/groups/req/ftp/se_I/kapitel_03.pdf
- http://www.wirtschaftsinformatik-24.de/pdf/Softwaretechnik_Software_Engineering.pdf
- http://www8.informatik.uni-erlangen.de/IMMD8/Lectures/SS03/algo2/AlgII.FAU.SS03.Kap21_1.pdf
- http://ti.uni-due.de/ti/de/education/teaching/ss06/pet/folien/Folien%201.pdf
- http://www.theoinf.tu-ilmenau.de/~riebisch/swqs/fehler.html

Andere Quellen:

- http://www-aix.gsi.de/~giese/swr/ariane5.html
- http://archiv.rhein-zeitung.de/on/96/07/24/topnews/fehler_1.html
- http://www.realtime-solutions.de/softwareentwicklung/projektmanagement/index.php
- http://www.innovationen-machen.de/index.php?id=5778
- http://www.it-infothek.de/fhtw/semester_3/se_3_01.html
- http://www.highscore.de/uml/einfuehrung.html
- http://www.computerwoche.de/heftarchiv/1995/28/1115508/
- http://www.computerwoche.de/heftarchiv/1995/27/1115369/
- http://perspektive89.com/2006/12/21/geschichte_der_softwareprogrammierung_freie_software_fur_freiheit_und_gerechtigkeit
- http://www.kreissl.info/st_inhalt.php