Heinz-Nixdorf-Berufskolleg

für Elektrotechnik, Informations- und Telekommunikationstechnik
der Stadt Essen

Grundlagen Anwendungsentwicklung

Theorie-Skript zu den Präsentationsthemen im Fach Deutsch

Softwareprojekte
Softwareentwicklungsmodelle
Grundlagen Softwaretest
ERP-Systeme
Industrie 4.0
Big Data

Name:
Klasse:

Inhalt

Version Schuljahr 2017/18

1. Zuordnung der Präsentationsthemen

Thema	Abschnitt	Anzahl Teammitglieder	Teammitglieder
Projekte und Qualität	**3**	2-4	
Definition Projekt und Proble-matiken bei Software-Projekten	3.1	1-2	
Qualitätskriterien	3.2	1-2	
Softwareentwicklungsmodelle	**4**	14-17	
Wasserfallmodell	4.1 4.2	3	
V-Modell	4.3	2	
Spiralmodell	4.4	1-2	
Extreme-Programming	4.5	1-2	
Scrum	4.6	3	
Kanban	4.7	3	
Life-Cycle	4.8	1-2	
Testen	**5**	10	
Motivation, grundlegende Begriffe	5.1 5.2	2	
Teststufen	5.3	2	
Testarten	5.4	2	
Testmethoden	5.5	2	
ERP-Systeme	**6**	3-5	
Überblick zu ERP-Systemen	6.1	1	
Beispiel: Geschäftsprozessaus-schnitt unterstützt mit dem ERP-System von SAP	6.2	2-4	
Industrie 4.0	**7**	**2**	
Big Data	**8**	**2**	

2. Einleitung

In diesem Skript befassen wir uns mit den Grundlagen der Softwareentwicklung:

- Was sind Projekte?
- Welche Problematiken existieren speziell bei Softwareprojekten?
- Welche Qualitätsanforderungen hat der Kunde an die Software?
- Wie wird Software entwickelt?
  Dabei gehen wir auf unterschiedliche Softwareentwicklungsmodelle ein.
- Wie wird Software getestet?
  Wir lernen Teststufen, Testarten und Testmethoden kennen.

3. Softwareprojekte

3.1. Projekt

Was ist eigentlich ein Projekt und wie geht man speziell an Projekte heran, bei denen es um „Anwendungsprogramme“ geht?

Ein Projekt

- ist eine einmalige Aufgabe
- soll ein bestimmtes Ergebnis erzielen
- erfordert eine Vielzahl von Ressourcen
- und ist zeitlich begrenzt.[1]

Projekte gibt es in vielen Bereichen, beispielsweise ist der Bau des eigenen Hauses oder die Entwicklung und Produktion von Sonnenenergiekollektoren oder Windenergieparks ein Projekt. Problematisch ist, dass auf der einen Seite eine Abschätzung über Dauer und Kosten des Projektes erfolgen muss. Auf der anderen Seite handelt es sich um eine völlig neue Aufgabe.
Aus diesem Grund versucht man, getreu nach dem Prinzip „Teile und Herrsche“, an die Entwicklung von Anwendungen bzw. Software in verschiedene Teilabschnitte oder Phasen einzuteilen, um so die Komplexität zu reduzieren.
Bis heute sind IT-Projekte trotzdem eine Herausforderung, wie die folgende Übersicht zeigt:[2]

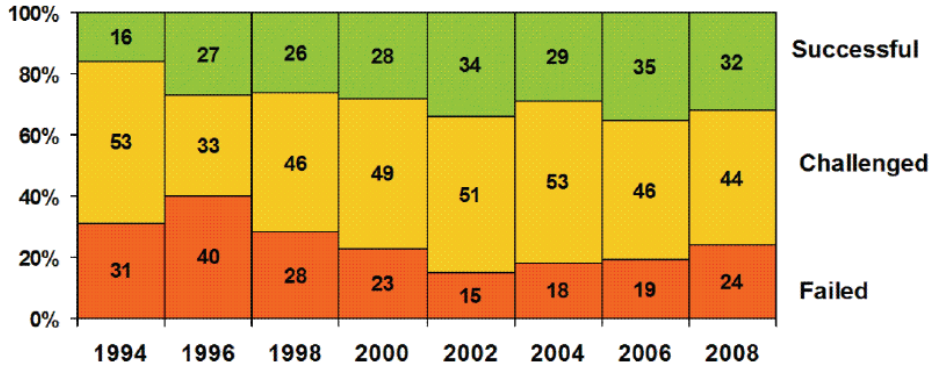


Abbildung: Erfolgsquoten von Softwareprojekten
(successful = erfolgreich, challenged = gefährdet, failed = gescheitert)

[1] Vgl. Andersen, Grude, Haug; Zielgerichtetes Projektmanagement, Ursprüngliche Veröffentlichung 1984 in Norwegen NKI-forlaget, Frankfurt 1997, S. 23.
[2] Quelle: CHAOS Report 2009, Standish Group International, Inc.

Folgende Faktoren[3] tragen zum Misserfolg bei:

Misserfolgsfaktoren (Chaos-Report USA, 1995)	
Unvollständige/ungenaue Anforderungen	13,1%
Mangelnde Einbeziehung der Beteiligten	12,4%
Ressourcenmangel	10,6%
Unrealistische Erwartungen	9,9%
Mangelnde Unterstützung vom Management	9,3%
Sich häufig ändernde Anforderungen/Spezifikationen	8,7%
Mangelhafte Planung	8,1%
Wird nicht mehr benötigt	7,5%
Mangelndes IT-Management	6,2%
Mangelndes Technologiewissen	4,3%
etc.	

Abbildung: Misserfolgsfaktoren

Hierzu wurden in der Historie verschiedene Ansätze entwickelt, die man Softwareentwicklungsmodelle nennt.
Bevor wir uns diesen Modellen zuwenden, soll der folgende Abschnitt verdeutlichen, warum Softwareprojekte so schwer durchzuführen sind.

3.2. Problematiken von Softwareprojekten

An Software werden immer höhere Anforderungen gestellt. Beispiele hierfür sind im Folgenden[4] beschrieben.

Wachsende Komplexität
Es entstehen immer neue Anwendungsgebiete mit immer schwierigeren Aufgaben.
Beispiele:

Jahr	Beispiel	LOC (Lines Of Code)
1982	Space Shuttle	50 Millionen
1994	SAP R/3-Software	7 Millionen
1997	SAP R/3-Software	30 Millionen
1999	SAP R/3-Software	50 Millionen
2000	Verwendete Software bei AT&T	500 Millionen
2000	Verwendete Software bei General Motors	2000 Millionen

Fehlerzahl und Konsequenzen
1979 gab es durchschnittlich 7 - 20 Defekte in 1000 Zeilen Quellcode.
1994 entstanden durchschnittlich 0,2 - 0,5 Defekte in 1000 Zeilen Quellcode.
Praktisch könnten 0,1 Defekte in 1000 Zeilen Quellcode beispielsweise bedeuten:

pro Jahr:	20.000 fehlerhafte Medikamente,
pro Woche:	500 Fehler bei medizinischen Operationen,
pro Tag:	16.000 verlorene Briefe bei der Post und 18 Flugzeugabstütze,
pro Stunde:	22.000 falsch verbuchte Schecks.

[3] Quelle: CHAOS Report 1995, Standish Group International, Inc.
[4] www4.in.tum.de/lehre/vorlesungen/sw_engineering/WS0203/folien/SWT.WS02-03.2seiten.pdf, abgerufen am 17.6.11

Weitere Herausforderungen bei der Erstellung von Software

- Die Produktanforderungen (neue oder geänderte Funktionen, Benutzeroberfläche, usw.) sind schwer zu spezifizieren und ändern sich möglicherweise während des Entwicklungsvorgangs,
- die Hardware und Systemsoftwarekomponenten können sich ändern (neuer Prozessor, andere Peripherie (= Umgebung), anderes Fabrikat, technische Neuerungen usw.),
- die Softwareentwurfsmethoden und Werkzeuge ändern sich.

3.3. Qualitätsanforderungen

Einem Kunden genügt schon lange nicht mehr, dass die Software „irgendwie läuft“. In der folgenden Abbildung und den Definitionen sind die üblichen Qualitätsanforderungen an Software dargestellt:[5]

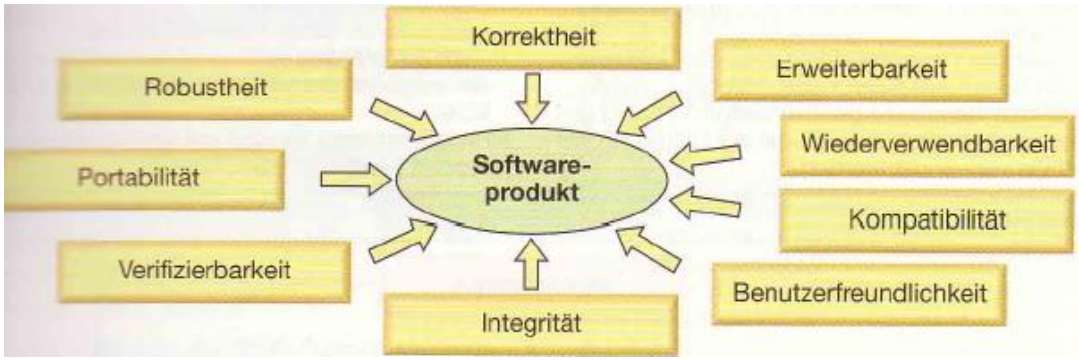


Abbildung: Qualitätskriterien

Qualitätskriterien

- Robustheit
  Vermeidung von Systemabstürzen bei Fehlbedienung, falschen Eingabedaten und Hardwarefehlern.
- Portabilität
  Softwareprodukte werden oft wesentlich länger eingesetzt als die zugrunde liegende Hardware und müssen auf verschiedenen Plattformen einsetzbar sein.
- Verifizierbarkeit
  Aufwand zum Testen und Abnahme der Anwendung.
- Integrität
  Schutz gegen unberechtigten Zugriff.
- Benutzerfreundlichkeit
  Adäquatheit (optimaler Bedienungskomfort),
  Erlernbarkeit (Klarheit und Einfachheit der Benutzerschnittstelle und der Bedienungsanleitung) und Robustheit (siehe oben).
- Kompatibilität
  Einfache Schnittstellen zu anderen Softwareprodukten.
- Wiederverwendbarkeit
  Verwendung des Programmcodes in neuen Anwendungen.
- Erweiterbarkeit
  Einfache Anpassungsfähigkeit bei Spezifikationsänderungen
- Korrektheit
  Das Programm erfüllt die zugrunde gelegte Spezifikation.

[5] Vgl. Hübscher, Heinrich u. a., *IT-Handbuch*. IT- Systemelektroniker/-in, Fachinformatiker/-in, Braunschweig: Westermann Schulbuch Verlag, 20005, 4. Auflage, S. 235.

4. Softwareentwicklungsmodelle

Computer entwickelten sich immer weiter und mit den Computern die Anforderungen, die mit Software zu lösen waren. Zusätzlich steigt, wie im vorherigen Abschnitt beschrieben, die Qualitätsanforderung an Software.

Um diesen Ansprüchen bei der Umsetzung von Softwareprojekten gerecht zu werden, wurden so genannte Vorgehensmodelle entwickelt. Es gibt zahlreiche Modelle, die sich in der Praxis bewährt haben.
Man unterscheidet zwischen Phasenmodellen und agilen Vorgehensmodellen. Die Phasenmodelle sind zusätzlich in sequentielle und iterative unterteilt.
Im Folgenden stellen wir als sequentielle Phasenmodelle das Wasserfallmodell und das V-Modell vor. Beispiele für iterative Modell sind das Spiralmodell und das prototyping-orientierte Life-Cycle-Modell. Agile Modelle sind XP und Scrum.

4.1. Das Wasserfallmodell

Das erste bekannte Softwareentwicklungsmodell war das so genannte Wasserfallmodell, dessen Phasen in zahlreichen Softwareentwicklungsmodellen wieder zu finden sind[6].

Durch die Einteilung in Phasen sollte die Entwicklung neuer Software planbarer und kontrollierbarer werden. Diesen Phasen lassen sich die Begriffe unseres Ausgangssatzes „Analyse, Entwurf, Realisierung und Bereitstellung von Anwendungen“ zuordnen.
Für jede dieser Phasen wird definiert, welche Ergebnisse geliefert werden sollen. Die neue Phase wird in der Regel erst begonnen, wenn die vorherige abgeschlossen ist. In der Praxis hat sich gezeigt, dass eine streng sequentielle Vorgehensweise meist nicht durchführbar ist. Deshalb wurde im Wasserfallmodell die Möglichkeit zu Rückkopplungen zu vorherigen Phasen eingeführt. Allerdings ist so eine Rückkopplung nur für aufeinander folgende Phasen vorgesehen.

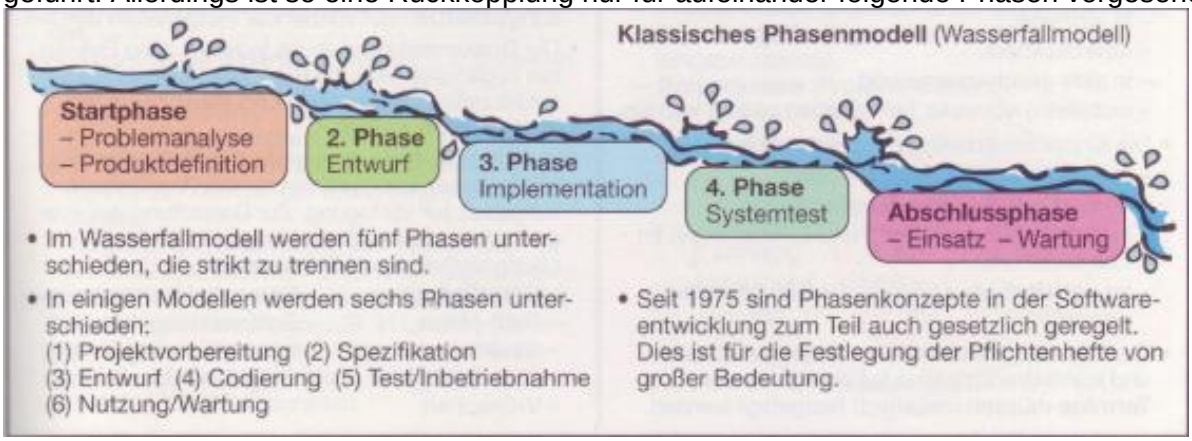


Abbildung Wasserfallmodell

Im folgenden Abschnitt werden Aktivitäten und Ergebnisse der einzelnen Phasen beschrieben.

[6] Vgl. Hübscher, Heinrich u. a., IT-Handbuch. IT- Systemelektroniker/-in, Fachinformatiker/-in, Braunschweig: Westermann Schulbuch Verlag, 20005, 4. Auflage, S. 200 und Balzert, Helmut, Lehrbuch der Softwaretechnik, 2. Auflage, Spektrum Verlag Heidelberg, Berlin, Oxford 2000.

Startphase (Analyse und Definition)

- Analyse
  - ggf. Ist-Zustand erheben
  - Arbeitsschritte für den mit der Software abzubildenden Aufgabenbereich feststellen und dokumentieren
  - existierende Wechselwirkungen erfassen
  - für das Projekt verfügbare personelle, finanzielle und zeitliche Ressourcen erfassen
  - Hauptbedingungen an Funktionalität, Daten, Leistungen und Qualität des Systems festlegen
  - Durchführbarkeit fachlich und wirtschaftlich untersuchen
  - Ergebnisdokumente:
    - Lastenheft (fachliche Beschreibung der Basisanforderungen)
    - Projektkalkulation inklusive eines groben Projektplanes
    - Wirtschaftlichkeitsberechnung

- Definition
  - Erstellung eines Kontraktes zwischen Auftraggeber und dem Softwarehersteller, in dem genau festgelegt wird, was das geplante Softwaresystem leisten soll und welche Prämissen für seine Realisierung gelten
    (das "WAS" der SW – Programmfunktionen).
  - Ergebnisdokumente:
    - Pflichtenheft = verbale Beschreibung der Anforderungen an die Software bezüglich Daten, Funktionen, Benutzeroberfläche, …
    - Konzept für die Benutzeroberfläche
    - Detaillierte Beschreibung der funktionalen Problemlösung aus Auftraggebersicht

Entwurfsphase

- Festlegung der inneren Struktur des Softwaremodells (das „WIE“ des SW- Programmablaufs)
- Ergebnisdokumente:
  - Spezifikation der einzelnen Softwarekomponenten (Module) und deren Zusammenwirkung
  - Entwurf des logischen Datenmodells
  - Entwurf der Systemarchitektur
  - Entwurf von Algorithmen

Implementierung (Codierung und Test):

- Umsetzung der in der Entwurfsphase erzielten Ergebnisse in eine auf dem Rechner ausführbare Form
- Ergebnisse:
  - Verfeinerung der Algorithmen und Umsetzung in eine Programmiersprache (Quellcode) einschließlich Dokumentation
  - Physisches Datenmodell
  - Tests und Testdokumentation

Systemtest

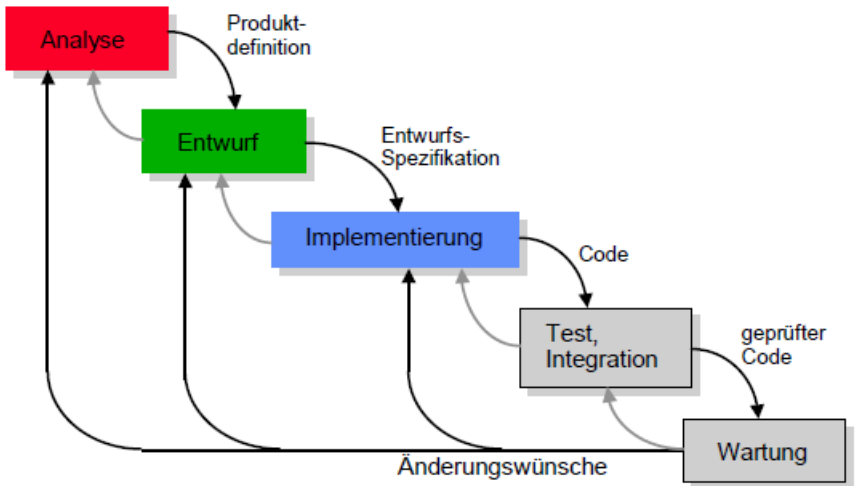
- Die Wechselwirkungen der Systemkomponenten werden unter realen Bedingungen getestet, wobei möglichst viele Fehler aufzudecken sind:
  - Sicherstellung, dass die Systemimplementierung die Systemspezifikation erfüllt.
  - Überprüfung nicht-operativer Anforderungen, wie Effizienz, ….
  - Installation und Test in der vorhandenen Hardwareumgebung des Kunden
  - Abnahmetest durch den Kunden
- Ergebnisse:
  - Testumgebung
  - Tests und Testdokumentation

Abschlussphase (Betrieb und Wartung)

- Nach erfolgreichem Test wird das Softwaresystem für den Betrieb freigegeben.
- Typische Aktivitäten sind:
  - Sicherung von Altdaten
  - Schulungen
  - Support
  - Behebung von Fehlern, die während des Betriebs auftreten
  - Durchführung von Systemerweiterungen

4.2. Das Wasserfallmodell in der Variante von Winston Royce

Die folgende Variante[7] von Winston Royce aus dem Jahr 1970 ermöglich den Rücksprung in die jeweils vorhergehende Phase und das Einbringen von Änderungswünschen während der Wartungsphase.



[7] http://www4.in.tum.de/lehre/vorlesungen/sw_engineering/WS0203/folien/SWT.WS02-03.6seiten.pdf, abgerufen am 20.6.11

4.3. Das V-Modell

Die Bundesrepublik Deutschland hat zur Ausschreibung und Durchführung von IT- Projekten, speziell für das Militär, das so genannte V-Modell[8] entwickelt, was für Projekte in diesem Bereich verbindlich ist. Parallel wird es in vielen Unternehmen angewendet.
Das V-Modell ist eine Erweiterung des Wasserfall-Modells, in dem für jede Phase einheitliche und verbindliche Ergebnisse sowie begleitende Aktivitäten zur Qualitätssicherung vorgegeben werden.

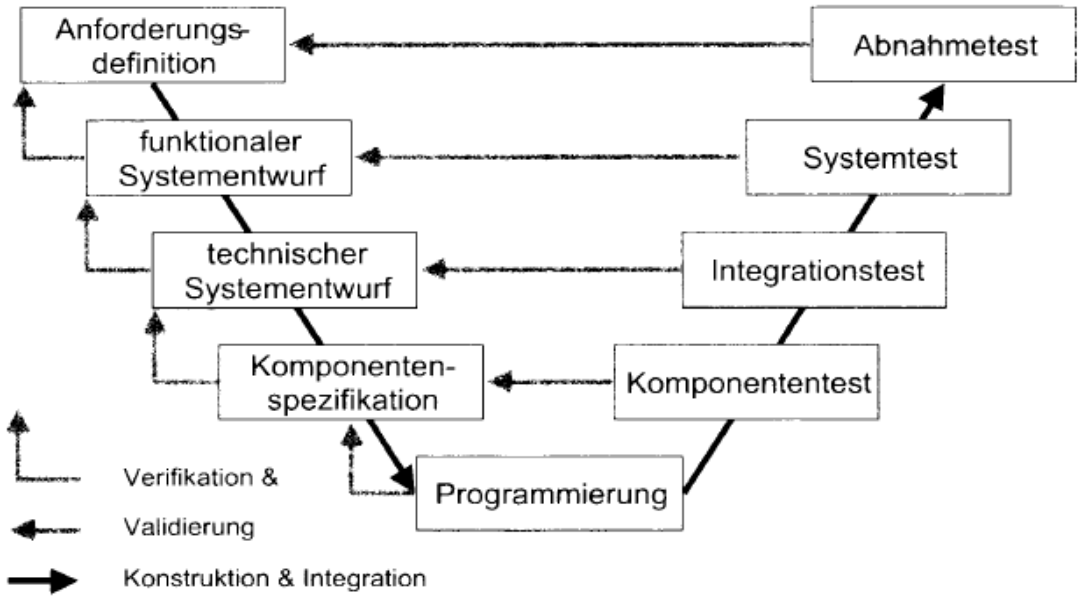


Abbildung: V-Modell

Das V-Modell besteht aus zwei unterschiedlichen, aber korrespondierenden Handlungssträngen. Der erste Strang beschreibt konstruktive Aktivitäten bis zur Programmierung. Auf der Programmierung setzen dann im zweiten Handlungsstrang Testaktivitäten auf. Jede Testaktivität basiert auf einer Planungsaktivität.
Daraus ergibt sich bildlich ein „V“, das dem Modell den Namen gibt.

konstruktiven Aktivitäten:

Anforderungsdefinition	Wünsche und Anforderungen des Auftraggebers werden gesammelt und verbindlich festgehalten. Man spricht auch von der „Spezifikation“ der Funktionen.
Funktionaler Systementwurf	Beschreibung der inhaltlichen Funktionen (beispielsweise der Prozess Auftrag→Lieferung→Rechnung)
Technischer Systementwurf	Entwurf der Systemarchitektur (beispielsweise Server, Clients, technische Umgebung)
Komponentenspezifikation	Beschreibung von Teilaufgaben (beispielsweise einer Lieferung) und ihrer Schnittstellen

[8] Andreas Spillner, Tilo Linz: Basiswissen Softwaretest, 2005

Programmierung:

Implementierung der Teilaufgaben und ihrer Schnittstellen in einer gegebenen Programmiersprache.

Tests:

Komponententest	Prüft, ob jede Komponente (Teilaufgabe) Anforderungsdefinition ihre Vorgaben erfüllt.
Integrationstest	Prüft, ob durch Schnittstellen miteinander verbundene Komponenten gemäß Definition zusammen funktionieren.
Systemtest	Prüft, ob das System als Ganzes funktioniert.
Abnahmetest	Prüft, ob das System aus Auftraggebersicht die vereinbarten Funktionalitäten erfüllt (→Vertrag).

In jeder Teststufe wird also überprüft, ob das Programm diejenigen Anforderungen erfüllt, die in der korrespondierenden Konstruktionsaktivität definiert wurden.
Diese Überprüfung wird „Validieren“ oder „Validierung“ genannt.
Verifikation ist der Beweis der Korrektheit und Vollständigkeit eines Phasenergebnisses relativ zum Input (Spezifikation, Phaseneingangsdokument) der vorherigen Phase.

Die einzelnen Teststufen werden bereits in den Konstruktionsaktivitäten vorgedacht und vorbereitet. Dabei werden beispielsweise Testmethoden, Testwerkzeuge und Testfälle definiert.

Beispiel: der Kunde wünscht, dass auf jedem ausgedruckten Dokument das Firmenlogo erscheint. Diese Forderung würde man in die Anforderungsdefinition schreiben. Dazu wird dann ein Testfall *„Drucke Dokument aus und prüfe, ob es das Firmenlogo beinhaltet.“* formuliert. Dieser Testfall wird im Abnahmetest durchgeführt.

In jeder Phase ist ein Rücksprung in die jeweils vorhergehende Phase möglich.

4.4. Das Spiralmodell

Das Wasserfallmodell setzt voraus, dass alle wesentlichen Anforderungen der späteren Anwender bekannt sind und keine wesentlichen Anforderungsänderungen während des Softwareprojektes entstehen. Dies ist in der Realität bei Softwareprojekten aufgrund der Komplexität sehr selten der Fall, weshalb das klassische Wasserfallmodell einer umfassenden Kritik unterzogen und verfeinert und ergänzt wurde.
Eine dieser Verfeinerungen stellt das von Boehm 1986-88 entwickelte Spiralmodell dar. Zu dieser Zeit war er bei TRW Inc., einem Unternehmen, das sich auf hoch entwickelte Produkte und Dienstleistungen für die Luftfahrtindustrie, Informationssystemhersteller und die Automobilmärkte spezialisiert hat.
Es entstand in einer Zeit, in der man mit großen Softwareprojekten schlechte Erfahrungen machte. Entweder produzierte man vieles doppelt, ging auf die Bedürfnisse der Beteiligten nicht ein, erhielt unzuverlässige Programme oder die Kosten schienen geradezu ins Unermessliche zu wachsen.
Hier wird der Softwareentwicklungsprozess als eine Spirale angesehen.
Jede einzelne „Spiralwindung“ hat das Ziel, ein neues Zwischenprodukt zu entwickeln und zu testen. Vor der Entwicklung müssen die Entwicklungsziele und Entwicklungspfade festgelegt werden, nach der Entwicklung erfolgt ein Soll-Ist-Abgleich mit den ursprünglichen Anforderungen. Im Rahmen dieses Vergleichs können weitere Entwicklungs- und Arbeitsschritte festgelegt werden und eine neue Spirale beginnt.

Aufgrund dieses Vorgehens wird das Spiralmodell auch als iteratives[9] bzw. inkrementelles Vorgehensmodell bezeichnet.
Der Radius der Spirale stellt den Gesamtaufwand bis zu einem bestimmten Zeitpunkt dar und der Winkel bezieht sich auf den Projektfortschritt in den einzelnen Spiralzyklen.
Durch das Spiralmodell wird nach Boehm das Risiko eines Scheiterns bei großen Softwareprojekten entscheidend minimiert, da das Gesamtprojekt in Teilprojekte zerlegt wird. Für jedes Teilprojekt können ein Projektplan erstellt, Ziele und Randbedingungen beschrieben, Alternativen gesucht und bewertet und Risiken abgeschätzt werden. Nach jedem Teilprojekt kann entschieden werden, ob das Projekt fortgesetzt wird oder nicht.[10] Außerdem eignet es sich sehr gut für die Einbettung von Qualitätssicherungsmaßnahmen in den Softwareentwicklungsprozess.

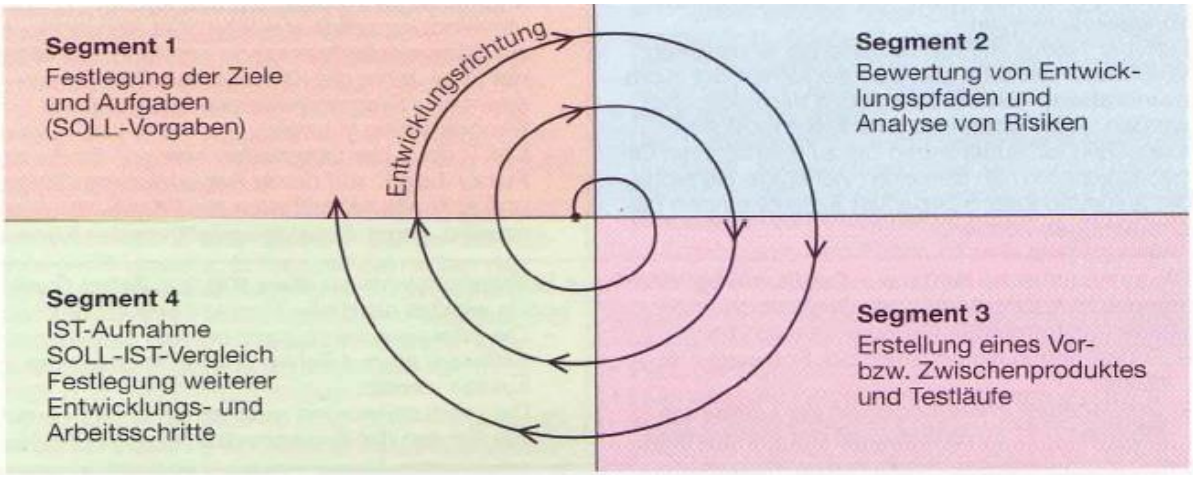


Abbildung 1 Spiralmodell

4.5. Extreme Programming

Extreme Programming (XP), auch Extremprogrammierung, ist ein flexibles Vorgehensmodell in der Softwaretechnik, das sich den Problemen in wiederholten kleinen Schritten unter Verwendung von Rückkoppelungen sowie einer kommunikationsintensiven Herangehensweise zielgerichtet annähert.
Daher wird es auch als agiles, iteratives und inkrementelles Modell bezeichnet.[11]

Um die Grundsätze agiler Softwareentwicklung festzulegen, wurde im Februar 2001 das Agile Manifest[12] formuliert:

1. *Individuen und Interaktionen sind wichtiger als Prozesse und Werkzeuge.*
2. *Funktionierende Programme sind wichtiger als ausführliche Dokumentation.*
3. *Die stetige Abstimmung mit dem Kunden ist wichtiger als die ursprüngliche Leistungsbeschreibung in Verträgen.*
4. *Der Mut und die Offenheit für Änderungen stehen über dem Befolgen eines festgelegten Plans.*

XP wurde von Kent Beck, Ward Cunningham und Ron Jeffries während ihrer Arbeit im Projekt Comprehensive Compensation System bei Chrysler als Vorgehen zur Erstellung von Software

[9] Itero, iterare: wiederholen, etwas zum zweiten Mal tun
[10] Vgl. Hübscher, Heinrich u. a., IT-Handbuch. IT- Systemelektroniker/-in, Fachinformatiker/-in, Braunschweig: Westermann Schulbuch Verlag, 20005, 4. Auflage, S. 200 und Balzert, Helmut, Lehrbuch der Softwaretechnik, 2. Auflage, Spektrum Verlag Heidelberg, Berlin, Oxford 2000.
[11] (Wikipedia (o. J.). <http://de.wikipedia.org/wiki/Extreme Programming>. Rev. 2006-07-12.)
[12] http://de.wikipedia.org/wiki/Agile_Softwareentwicklung, abgerufen am 19.6.11

entwickelt. Das so genannte C3-Projekt dauerte von 1995 bis 2000. Die zu entwickelnde Software wurde im Bereich der Lohnabrechnung eingesetzt.

Extreme Programming wird häufig bei Projekten angewendet, in denen der Kunde die genauen Projektanforderungen noch nicht kennt und somit eine genaue Spezifikation nicht möglich ist.

Charakteristika:

- Klares, strukturiertes Vorgehen
- Kleine Teams von maximal 10 Entwicklern
- Teamarbeit (beispielsweise Aufwandsschätzung im Team, nicht allein durch einen Projektmanager)
- Prioritätenanalyse zur Identifikation der unbedingt umzusetzenden Funktionen und der Reihenfolge der Umsetzung
- Offene und stetige Kommunikation zwischen allen Beteiligten
- User Stories, die die Funktionsanforderungen an ein System aus Sicht eines Akteurs darstellen, dienen als Werkzeug zur Beschreibung der umzusetzenden Funktionalitäten[13]. Die Use Stories werden auf so genannte Story Cards geschrieben und für alle sichtbar auf einem Medium platziert.
  Beispiel: Ein Lagerarbeiter beschreibt, dass er nach dem Verladen der Ware auf den LKW die Lieferpapiere ausdruckt.
- Gleichmäßige Verteilung der Arbeitsbelastung
- Zahlreiche und frühe Modultests
- Permanenter Test und permanente Integration neuer Funktionalitäten, Bereitstellung in Form einer ersten ausführbaren Version (Prototyp) und Durchführung von Akzeptanztests
- Tägliches „Stand-up Meeting“, bei dem jeder Entwickler berichtet, was er am Vortag geleistet hat, was er heute leisten möchte und wo es gegebenenfalls Probleme gab.
- „Pair Programming“, das heißt nach dem Vier-Augenprinzip arbeiten immer zwei Leute an einer Aufgabe und wechseln sich beim Programmieren kontinuierlich ab.

Ziele:

- schnellere Bereitstellung der Software
- höhere Qualität
- mehr Kundenzufriedenheit.

Anteil von XP in Projekten:

- Laut Forrester Research werden circa 14 Prozent aller Projekte mit agilen Methoden durchgeführt.

[13] Eine User Story ist ein Use Case in Kurzform und folgt dem Muster „Als xxx kann ich xxxx, um xxxx'“, wobei das um auch weggelassen werden kann, falls die Absicht klar erkennbar ist. Diese Formulierung dient als Vorlage (Template) der abzuarbeitenden Anforderungen und ist situationsabhängig änderbar. Für eine ausführliche Beschreibung sei auf Teil 2 dieses Skriptes verwiesen.

4.6. Scrum

Scrum[14] (engl.: „Gedränge“) ist ein agiles Vorgehensmodell. Der Grundgedanke und die erste Einführung entstammen der „schlanken Produktion“ (lean production) und wurde zunächst im Rahmen eines Produktionsprozesses von Ikujirō Nonaka und Hirotaka Takeuchi im Jahr 1986 entwickelt.
Als Software-Entwicklungsmodell wurde Scrum 1991 von Ken Schwaber, Jeff Sutherland und Mike Beedle vorgestellt und etabliert. Ken Schwaber ist ein Softwareentwickler, Produktmanager und Industrieberater. Jeff Sutherland ist Mathematiker und Arzt und arbeitete in Führungspositionen bei Softwareherstellern. Mike Beedle ist Coach in der IT-Branche und Programmierer.
Scrum wird inzwischen vielfach eingesetzt, z. B. auch bei Konzernen wie SAP, BASF und Allianz.

Man geht davon aus, dass die Komplexität der Softwareentwicklung so groß ist, dass sie sich vorher weder genau planen noch in detaillierte Arbeitsschritte zerlegen lässt.
Der Grundgedanke bei Scrum ist, dass ein Team in einem festgelegten Rahmen mit festgelegten Rollen selbständig und eigenverantwortlich agiert. Dabei steht die ständige Weiterentwicklung aller am Prozess Beteiligten und der Arbeitsmittel im Vordergrund. Hohe Priorität haben die Interaktionen untereinander und mit dem Kunden, Kreativität und Mut zu Änderungen. Abgelehnt wird ein Management „von oben“.

4.6.1. Rollen

(1) Product Owner
- legt das gemeinsame Ziel (meist mittels User-Storys) fest.
- setzt regelmäßig die Prioritäten der einzelnen Product-Backlog-Elemente (siehe 4.6.3).
- legt fest, welches die wichtigsten Features sind, aus denen das Entwicklungsteam eine Auswahl für den nächsten Sprint (siehe 4.6.2) trifft.
- darf dem Team in der Mitte des Sprints nicht mehr Arbeit zu geben.
- kann nichts an der Sprint Planung bis zum nächsten Sprint Meeting ändern.
- kann den laufenden Sprint jederzeit beenden.
- unterhält eine ständige Kommunikation mit dem Team.
- prüft, welche Maßnahmen den meisten geschäftlichen Nutzen erzeugen.
- liefert das Produkt an den Kunden.

(2) Team
- besteht idealerweise aus 5 bis 9 Personen.
- schätzt die Aufwände der einzelnen Backlog-Elemente (siehe 4.6.3) ab.
- beginnt mit der Implementierung der für den nächsten Sprint machbaren Elemente. Dazu wird vor dem Beginn des Sprints ein weiteres Planungstreffen durchgeführt, bei dem die höchstpriorisierten Elemente des Backlogs und konkrete Aufgaben aufgeteilt werden.
- arbeitet selbstorganisiert im Rahmen einer Time Box (dem Sprint) und hat das Recht (und die Pflicht), selbst zu entscheiden, wie viele Elemente des Backlogs nach dem nächsten Sprint erreicht sein müssen.

[14] http://de.wikipedia.org/wiki/Scrum, abgerufen am 5.6.11 und http://www.youtube.com/watch?v=vmGMpME_phg, abgerufen am 23.6.11

(3) Scrum Master
- überwacht die Aufteilung der Rollen und Rechte.
- sorgt für die Transparenz während der gesamten Entwicklung.
- unterstützt das Team dabei, Verbesserungspotentiale zu erkennen.
- ist nicht Teil des *Teams*,.
- sorgt dafür, dass die Arbeitsbedingungen stimmen.
- sorgt für die ordnungsgemäße Durchführung und Implementierung von *Scrum* im Rahmen des Projektes.
- achtet darauf, dass der Product Owner nicht in den Selbstorganisationsprozess des Teams eingreift.

4.6.2. Der Scrum-Prozess

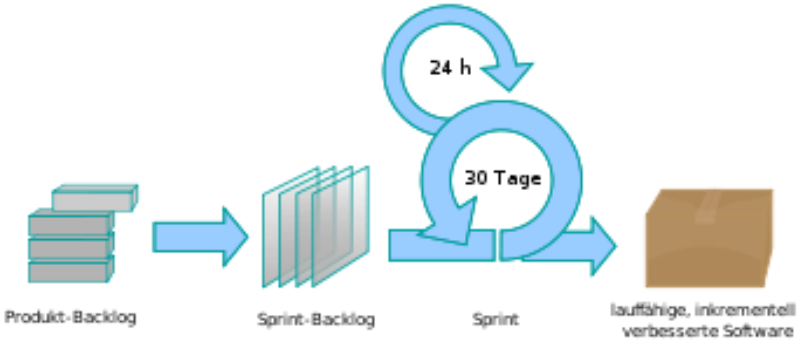


Abbildung: Der Scrum-Prozess

(1) Sprint-Planungstreffen 1
- Der *Product Owner* erklärt dem Team alle einzelnen *Backlog*-Einträge (siehe 4.6.3), die für den nächsten Sprint benötigt werden.
- Der *Product Owner* einigt sich mit dem Scrum-Team auf das Sprint-Ziel (siehe Punkt (3). Dieses bildet nach dem Sprint die Basis für die Abnahme.
- Die höchst priorisierten Einträge des Product Backlog werden entsprechend dem Ergebnis des Treffens in das Selected Backlog (siehe 4.6.3) übernommen.

(2) Sprint-Planungstreffen 2
- wird eigenverantwortlich vom Scrum-Team durchgeführt.
- alle Arbeiten (→ Selected Backlog) werden an die Team-Mitglieder verteilt. Eine Aufgabe sollte ca. einen Tag in Anspruch nehmen, ansonsten wird die Aufgabe weiter aufgeteilt. Aus der Aufgabenverteilung entsteht das Sprint Backlog.

(3) Sprint
- zentrales Element des Entwicklungszyklus von Scrum.
- die Umsetzung einer Iteration mit einer Iterationslänge von ca. 30 Tagen (in der Praxis ca. 1 - 4 Wochen).
- vor dem Sprint wird der Product Backlog erstellt.
- für einen Sprint wird ein Sprint Backlog erstellt.
- am Ende eines Sprints steht immer eine lauffähige, getestete, inkrementell verbesserte Software.

(4) Daily Scrum
- An jedem Tag findet ein kurzes (maximal 15-minütiges) Daily Scrum statt.
- Das Team stellt sich gegenseitig die folgenden Fragen:
  „Welche Aufgaben hast Du seit dem letzten Meeting fertiggestellt?“
  „Welche Aufgaben wirst Du bis zum nächsten Meeting bearbeiten?“
  „Gibt es Probleme, die Dich bei deinen Aufgaben behindern?“
- Falls neue Hindernisse erkannt wurden, müssen diese vom Scrum Master bearbeitet werden. Dazu werden sie in das Impediment Backlog (siehe 4.6.3) eingetragen.
- Im Daily Scrum haben nur die Teammitglieder und der Scrum Master eigenständiges Rederecht.

(5) Review
- Nach einem Sprint wird das Sprint-Ergebnis einem informellen Review durch das Team und durch den Product Owner unterzogen.
- Dazu wird das Ergebnis des Sprints (die laufende Software) vorgeführt, eventuell werden technische Eigenschaften präsentiert.
- Der Product Owner prüft, ob das Sprint-Ergebnis seinen Anforderungen entspricht.
- Eventuelle Änderungen können in Form von Erweiterungen, Umpriorisierungen oder durch das Entfernen von Elementen des Product Backlogs dokumentiert werden.

(6) Retrospektive
- In der Retrospektive wird die zurückliegende Sprint-Phase betrachtet („Was war gut?“, „Was könnte verbessert werden?“).
- Das Verbesserungspotential wird priorisiert und einem Verantwortungsbereich (Team oder Organisation) zugeordnet.
- Alle der Organisation zugeordneten Themen werden vom Scrum Master aufgenommen und in das Impediment Backlog eingetragen.
- Alle teambezogenen Punkte werden in das Product Backlog aufgenommen.
- Ziel: den vergangenen Sprint zu beleuchten und daraus zu lernen.

4.6.3. Dokumente

Product Backlog	enthält die inhaltliche und technische Beschreibung des Produkts. Das Product Backlog muss nicht vollständig sein; es wird laufend fortgeführt.
Selected Backlog	enthält die Einträge aus dem Product Backlog, die notwendig sind, um das Ziel des Sprints zu erfüllen. Es wird von dem Product Owner zusammen mit dem Scrum Team erstellt und gilt somit als Vereinbarung und als Grundlage der späteren Abnahme.
Sprint Backlog	enthält alle Aufgaben und ihre Priorisierung, die notwendig sind, um das Ziel des Sprints zu erfüllen. Eine Aufgabe sollte dabei in nicht länger als einem Tag erledigbar sein. Längere Aufgaben sollten in kurze Teilaufgaben zerlegt werden. Bei der Planung des Sprints werden nur so viele Aufgaben eingeplant, wie das Team in diesem Sprint für realistisch durchführbar hält. Außerdem benennt das Sprint Backlog die für die Aufgaben Verantwortlichen.
Burndown Chart	eine graphische, pro Tag zu erfassende Darstellung des noch zu erbringenden Restaufwands pro Sprint.
Impediment Backlog	enthält Hindernisse des Projekts. Der *Scrum Master* ist dafür zuständig, diese gemeinsam mit dem Team auszuräumen.

4.7. Kanban

Kanban[15] ist ein Vorgehensmodell zur Softwareentwicklung, bei dem die Anzahl paralleler Arbeiten, der Work in Progress (WiP), reduziert und somit schnellere Durchlaufzeiten erreicht und Probleme – insbesondere Engpässe – schnell sichtbar gemacht werden sollen.

Kanban ist eine Technik aus dem Toyota-Produktionssystem, mit der ein gleichmäßiger Fluss (Flow) in der Fertigung hergestellt und so Lagerbestände reduziert werden sollen. Der Fokus liegt dabei auf „one piece flow“, d.h. den optimalen Fluss jedes einzelnen Produktes durch die Fertigung. In der Softwareentwicklung wird zwar der Name Kanban übernommen, die einzelnen Techniken werden jedoch nicht direkt übertragen. Es werden einige grundlegende Prinzipien aus der Lean Production (‚Schlanke Produktion‘), und mehr noch dem Lean Development (auch Lean Product Development), übernommen und ergänzt durch das klassische Risikomanagement. Insbesondere die Ideen Don Reinertsens, der verschiedene Techniken vorstellt, wie sich Produkte in wesentlich kürzerer Zeit entwickeln lassen, spielen eine wichtige Rolle in Kanban. Als „Begründer“ von Kanban in der IT gilt David Anderson, der das Gesamtkonzept erstmals 2007 der Öffentlichkeit vorstellte.

4.7.1. Kanban-Praktiken

David Anderson beschrieb sechs Praktiken, die Unternehmen in ihre Arbeitsweise integrieren, wenn sie Kanban machen:

Visualisiere den Fluss der Arbeit
: Die Wertschöpfungskette mit ihren verschiedenen Prozessschritten (zum Beispiel Anforderungsdefinition, Programmierung, Dokumentation, Test, Inbetriebnahme) wird gut sichtbar für alle Beteiligten visualisiert. Dafür wird ein Kanban-Board (in der Regel ein großes Whiteboard) verwendet, auf dem die unterschiedlichen Stationen als Spalten dargestellt werden. Die einzelnen Anforderungen werden auf Karteikarten oder Haftnotizen festgehalten und durchwandern mit der Zeit als so genannte Tickets das Kanban-Board von links nach rechts.

Begrenze die Menge angefangener Arbeit
: Die Anzahl der Tickets (Work in Progress – WiP), die gleichzeitig an einer Station bearbeitet werden dürfen, wird limitiert. Wenn beispielsweise die Programmierung gerade zwei Tickets bearbeitet, und das Limit für diese Station zwei beträgt, darf sie kein drittes Ticket annehmen, auch wenn die Anforderungsdefinition ein weiteres bereitstellen könnte. Hierdurch entsteht ein Pull-System, bei dem sich jede Station ihre Arbeit bei der Vorgängerstation abholt, anstatt fertige Arbeit einfach an die nächste Station zu übergeben.

Miss und steuere den Fluss
: Die Mitglieder eines Kanban-Prozesses messen typische Größen wie Längen von Warteschlangen, Zykluszeit und Durchsatz, um festzustellen, wie gut die Arbeit organisiert ist, wo man noch etwas verbessern kann und welche Versprechen man an die Partner geben kann, für die man arbeitet. Dadurch wird die Planung erleichtert und die Verlässlichkeit gesteigert.

Mache die Regeln für den Prozess explizit
: Um sicherzustellen, dass alle Beteiligten des Prozesses wissen, unter welchen Annahmen und Gesetzmäßigkeiten man arbeitet, werden möglichst alle Regeln, die es gibt, explizit gemacht. Dazu gehören z.B.
  - eine Definition des Begriffes ‚fertig‘, ähnlich der Definition of Done in Scrum,
  - Bedeutung der einzelnen Spalten,

[15] https://de.wikipedia.org/wiki/Kanban_%28Softwareentwicklung%29, abgerufen am 08.07.2016

- Antworten auf die Fragen: wer zieht, wann zieht man, wie wählt man das nächste zu ziehende Ticket aus der Menge der vorhandenen Tickets aus, usw.

Wichtige Hinweise:

- Fördere Leadership auf allen Ebenen in der Organisation
- Verwende Modelle, um Chancen für kollaborative Verbesserungen zu erkennen

Die Visualisierung und die Begrenzung des WiP sind einfache Mittel, mit denen rasch sichtbar werden, wie schnell die Tickets die verschiedenen Stationen durchlaufen und wo sich Tickets stauen. Die Stellen, vor denen sich Tickets häufen, während an den nachfolgenden Stationen freie Kapazitäten vorhanden sind, werden als Bottlenecks bezeichnet. Durch Analysen des Kanban-Boards können immer wieder Maßnahmen ergriffen werden, um einen möglichst gleichmäßigen Fluss (Flow) zu erreichen. Beispielsweise können die Limits für einzelne Stationen verändert werden, es können Puffer eingeführt werden (insbesondere vor Bottlenecks, die durch nur zeitweise Verfügbarkeit von Ressourcen entstehen), die Anzahl der Mitarbeiter an den verschiedenen Stationen kann verändert werden, technische Probleme werden beseitigt usw. Dieser kontinuierliche Verbesserungsprozess (japanisch: Kaizen) ist wesentlicher Bestandteil von Kanban.

4.7.2. Gemeinsamkeiten von Kanban und Scrum

- schlank ("lean") und agil
- begrenzen den WiP
- setzen auf Transparenz, um den Prozess zu verbessern
- fokussieren darauf, möglichst schnell und möglichst häufig releasefähige Software-Inkremente auszuliefern
- basieren auf selbstorganisierenden Teams
- erfordern, dass Anforderungen in kleine Einheiten heruntergebrochen werden
- In beiden wird der Releaseplan immer wieder optimiert, indem empirische Daten ausgewertet werden (Team-Geschwindigkeit / Durchlaufzeiten)

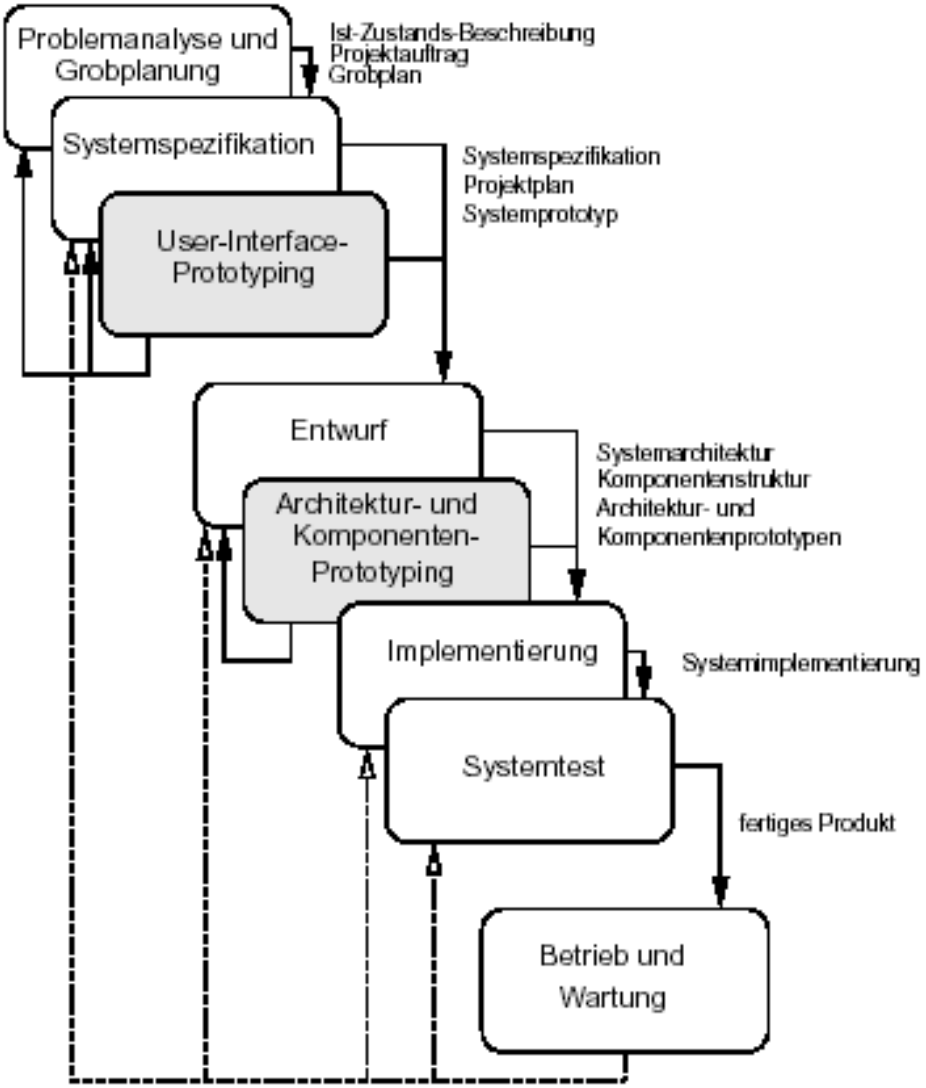
4.7.3. Unterschiede zwischen Kanban und Scrum

Kanban	Scrum
Iterationen sind optional. Es kann unterschiedliche Taktzeiten geben.	Iterationen mit gleichen Längen sind vorgeschrieben.
Vereinbarungen sind optional.	Das Team vereinbart, eine bestimmte Menge an Arbeit während der nächsten Iteration zu erledigen.
Die Durchlaufzeit wird als Basis-Metrik für Planung und Prozessverbesserung verwendet.	Die Team-Geschwindigkeit ist die Basis-Metrik für Planung und Prozessverbesserung.
Cross-funktionale Teams sind optional. Experten-Teams sind erlaubt.	Cross-funktionale Teams sind vorgeschrieben.
Keine Vorschrift bezüglich der Größe von Anforderungen.	Anforderungen müssen so aufgeteilt werden, dass sie sich innerhalb einer Iteration erledigen lassen.
WiP wird direkt limitiert.	WiP wird indirekt limitiert (durch die Menge an Anforderungen, die in einen Sprint „passt“).
Schätzungen sind optional.	Schätzungen sind vorgeschrieben.
Neue Anforderungen können zu jedem Zeitpunkt an das Team gegeben werden, falls Kapazitäten frei sind.	Während eines laufenden Sprints können keine neuen Anforderungen an das Team gegeben werden.
Gibt keine Rollen vor.	Schreibt drei Rollen vor (*Product Owner*, *Scrum Master*, Team).
Ein Kanban-Board kann von mehreren Teams und/oder Einzelpersonen geteilt werden.	Ein Scrum-Board gehört einem einzelnen Team.
Ein Kanban-Board wird kontinuierlich weitergepflegt.	Das Scrum-Board wird nach jedem Sprint gelöscht und neu aufgesetzt.
Priorisierung ist optional.	Schreibt vor, dass alle Einträge im *Backlog* priorisiert sein müssen.

4.8. Das prototyping-orientierte Life-Cycle-Modell

Genauso wie beim Spiralmodell ist die Entwicklung beim prototyping-orientierten Life-Cycle-Modell[16] iterativ.

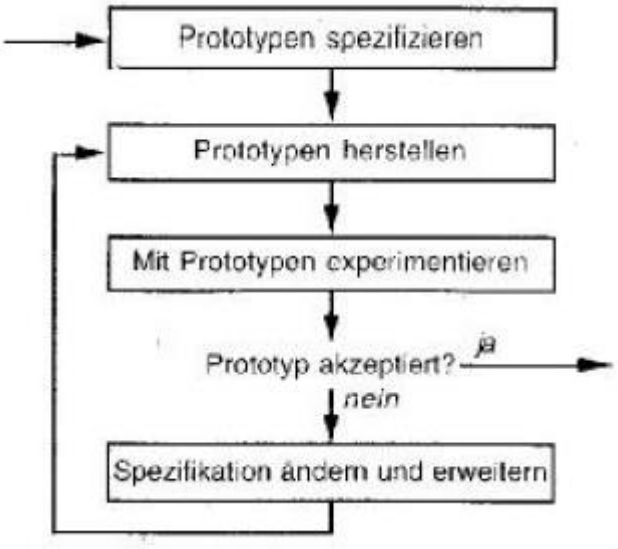
Die Phasen eines Softwareprojekts und die Ordnung ihrer zeitlichen Abfolge bezeichnet man als Software-Life-Cycle.
Ein Prototyp ist ein einfach zu änderndes und zu erweiterndes, ausführbares Modell des geplanten Produkts, das mit wesentlich geringerem Aufwand hergestellt wird. Es besitzt nicht alle Eigenschaften des Zielsystems. Die Anwender müssen aber die wesentlichen Systemeigenschaften vor der Systementwicklung erproben können.
Unter Prototyping versteht man alle Arbeiten und Schritte, die zur Herstellung von Prototypen notwendig sind.



Die Phase Problemanalyse, Spezifikation und User-Interface-Prototyping laufen zeitlich überlappt ab. Genauso verschmelzen die Phasen Entwurf, Implementierung und Test sehr stark ineinander, weshalb man von Aktivitäten anstelle von Phasen spricht.

[16] Wir beziehen uns in diesem Abschnitt auf die Proseminararbeit von Christian Steinbauer und Thomas Aichholzer vom 7.5.2003, Universität Klagenfurt

Zuerst wird ein Prototyp aufgrund von Resultaten vorangegangener Aktivitäten produziert. Anhand dieses System-Modells kann unter realitätsnahen Bedingungen geprüft werden, ob die Anforderungen des Anwenders erfüllt werden. Enthält das Modell Fehler, können Änderungen für die nächste Iterationsphase definiert werden. Damit wird das Risiko einer falschen oder unvollständigen Systemspezifikation vermindert.



Im Gegensatz zum klassischen Phasenmodell wird nicht so spät sondern so früh wie möglich implementiert. Dadurch erhält man ausführbare Prototypen und nicht nur Beschreibungen, und man kann Systemspezifikation und Systemarchitektur anhand eines schrittweise entwickeln. Durch die Überlappung der einzelnen Aktivitäten und durch die Art der Ergebnisse (lauffähige Prototypen) wird das Risiko eine Fehlentscheidung verringert und die durch Experimente erzielten Zwischenergebnisse erleichtern die Qualitätssicherung.

Die häufigsten Gegenargumente des Prototyping sind höhere, nicht planbare Kosten und längere, ebenfalls nicht planbare Entwicklungszeiten.

Die Vorteile des Prototyping sind

- höhere software-ergonomische Qualität des Endprodukts,
- bessere Gestaltung von Arbeitsorganisation und Arbeitsbedingungen (durch frühzeitiges Erkennen solcher Auswirkungen)
- geringerer Wartungs- und Weiterentwicklungsaufwand
- höhere Akzeptanz bei den Anwendern,
- geringere Schulungskosten.

5. Grundlagen Softwaretest

Die Ausführungen zum Thema „Testen“ basieren auf den Ausführungen des Buches „Basiswissen Softwaretest“ von Andreas Spillner und Tilo Linz.[17]

5.1. Motivation

Software ist aus unserem Leben nicht mehr wegzudenken. Neben Standard- und Business-Software spielt sie in zahlreichen Produkten des täglichen Gebrauchs (Autos, …) eine zunehmende Rolle.
Fehler in Software können nicht nur zu Geld-, Zeit- oder Imageverlust sondern sogar zu Personenschäden führen.

Mit dem Testen von Software wird das Ziel verfolgt, möglichst viele Fehler vor der Freigabe des Systems zu beheben. Ein vollständiger Test ist nicht durchführbar.

Der Testaufwand ist immer in Abhängigkeit von dem Risiko und der Gefährdungsbewertung zu setzen. In der Regel ist er zwischen 25 und 50 % des Gesamtaufwandes anzusetzen.

Beim Testen sind folgende Prinzipien zu verfolgen:

1. Testen zeigt die Anwesenheit von Fehlern
2. Vollständiges Testen ist nicht möglich
3. Mit dem Testen frühzeitig beginnen
4. Fehlerzustände sind in der Regel nicht gleichverteilt. Meist finden sich nach dem Nachweisen von Fehlerwirkungen in einem Programmteil noch weitere Fehlerwirkungen. Auf diesen Umstand muss flexibel reagiert werden.
5. Zunehmende Testresistenz (Pesticide paradox) – Tests lassen in ihrer Wirksamkeit nach. Wenn sie wiederholt werden, decken sie keine neuen Fehlerzustände auf. Daher sind Testfälle regelmäßig zu überprüfen und anzupassen.
6. Testen ist abhängig vom Umfeld, das heißt keine Software wird gleich getestet. Ein Hauptkriterium für den Testumfang ist beispielsweise wie sicherheitskritisch der Einsatz der späteren Software ist.
7. Trugschluss: Keine Fehler bedeutet ein brauchbares System
   Nutzer sollten frühzeitig in das Testen einbezogen werden, um sicherzustellen, dass das System deren Erwartungen entspricht.

5.2. Definitionen

Bevor beschrieben wird, wie Software getestet wird, seien folgende Begriffe geklärt:

Fehlerwirkung:

Wirkung eines Fehlerzustandes, der bei Ausführung eines Testobjektes nach „außen“ in Erscheinung tritt bzw. die Abweichung zwischen beobachtetem Ist- und gewünschtem Sollverhalten.

[17] Vgl. German Testing Board e.V. & Swiss Testing Board (Hrsg.): *Certified Tester - Foundation Level Syllabus.* Deutschsprachige Ausgabe, Möhrendorf 2005, International Software Testing Qualifications Board (ISTQB)
und Basiswissen Softwaretest, Spillner, Linz, dpunkt.verlag, 2005.

Defekt bzw. Fehlerzustand

1. Inkorrektes Teilprogramm, inkorrekte Anweisung oder Datendefinition, die Ursache für eine Fehlerwirkung ist.
2. Zustand, der unter bestimmten Bedingungen zu einer Fehlerwirkung führt.

Testen

1. Gesamter Prozess, ein Programm auf systematische Weise auszuführen, um die korrekte Umsetzung der Anforderungen nachzuweisen und um Fehlerwirkungen aufzudecken. Jede einzelne Ausführung des Testobjekts zum Zwecke der Überprüfung.
2. Oberbegriff für alle Tätigkeiten und Stufen im Testprozess.
3. Mit dem Testen verbundene Ziele:
   - Vorbeugung von Fehlerwirkungen durch Analyse des Programms und von Dokumenten
   - gezieltes und systematisches Auffinden von Fehlerwirkungen, die auf Defekte hinweisen
   - Produktqualität bestimmen
   - Vertrauen gewinnen und erhöhen

Defektbehebung oder Debugging

- Tätigkeit des Lokalisierens, Analysierens und Entfernens von Fehlerzuständen in der Software

Fundamentaler Testprozess

- Planung und Steuerung
- Analyse und Design
- Realisierung und Durchführung
- Auswertung und Bericht
- Abschluss

Testfall

- Umfasst die für die Ausführung eines Tests notwendigen Vorbedingungen, die Menge der Eingabewerte und die Menge der erwarteten Sollwerte, die Prüfanweisung (Art, wie Eingaben zu übergeben und Sollwerte abzulesen sind) sowie die erwarteten Nachbedingungen.

Testszenario

- Zusammenfassung von Testfällen, wobei der 1. Testfall den Input für den nächsten Testfall liefert, …

Testlauf

- Ausführung eines oder mehrerer Testfälle mit festgelegten Randbedingungen.

Testtreiber

- Programm, das es ermöglicht, ein Testobjekt ablaufen zu lassen, mit Testdaten zu versorgen und Ausgaben/Reaktionen des Testobjektes entgegenzunehmen.
- Simulation von noch nicht entwickelten Komponenten

5.3. Teststufen

Wann getestet wird ist abhängig vom verwendeten Softwareentwicklungsmodell.
Hier werden die zeitlich aufeinander folgenden Teststufen am Beispiel des V-Modells beschrieben, allerdings finden sich in fast allen Modellen diese Teststufen in abgewandelter Form wieder.
Grundidee des V-Modells ist, dass Entwicklungs- und Testarbeiten zueinander korrespondierende, gleichberechtigte Tätigkeiten sind, was bildlich durch das V symbolisiert wird.[18]

Zur Verdeutlichung ist hier nochmals das V-Modell (siehe auch 4.3) dargestellt:

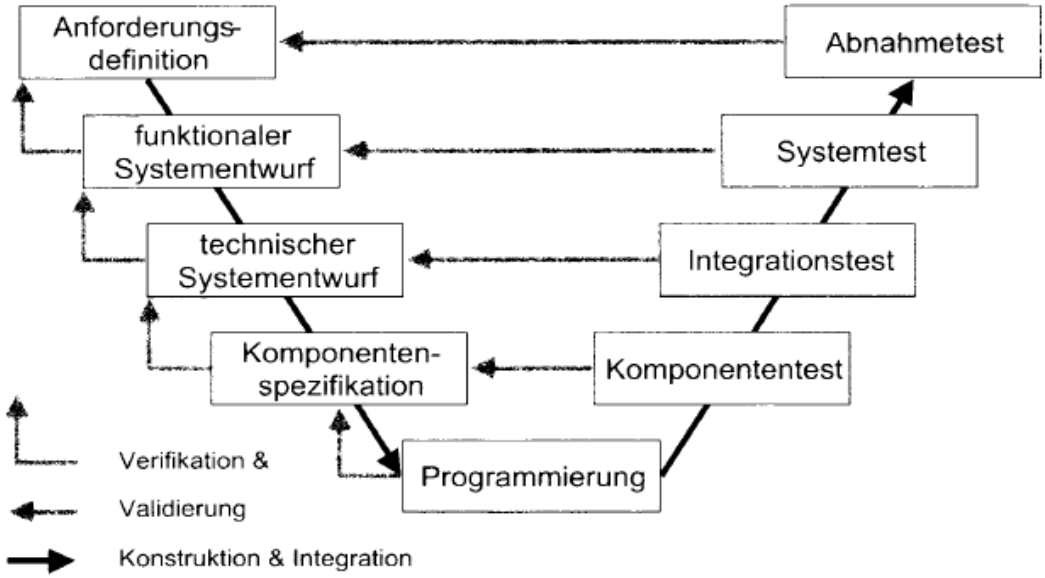


Abbildung: V-Modell

Hier wird in jeder Stufe validiert, worunter man die Prüfung der Entwicklungsergebnisse gegen die ursprünglichen Anforderungen versteht. Man stellt sich die Frage, ob das Produkt im Kontext der beabsichtigten Produktnutzung sinnvoll erstellt wurde.

Darüber hinaus wird verifiziert, um die Korrektheit und Vollständigkeit eines Phasenergebnisses relativ zu seiner direkten Spezifikation nachzuweisen.

Modul – bzw. Komponententest

Beim Komponententest werden einzelne Softwarebausteine, wie Module, Units oder Klassen, geprüft. Deshalb wird er häufig auch Modultest, Unit Test bzw. Klassentest genannt. Vorteil ist, dass sich Fehlerwirkungen direkt den getesteten Komponenten zuordnen lassen.
In der Regel werden die Komponenten “frisch“ von der Festplatte der Entwickler getestet, so dass der Test eine sehr entwicklungsnahe Arbeit ist und oft vom Entwickler selbst durchgeführt wird.

[18] Vgl. hierzu Kapitel 4.3.

Integrationstest

Der Integrationstest setzt voraus, dass die ihm übergebenen Testobjekte (Komponenten bzw. Module) bereits getestet sind und Defekte möglichst korrigiert wurden.
Diese Komponenten werden dann von Entwicklern, Testern oder Integrationsteams zu größeren Teilsystemen verbunden.
Ziel des Integrationstests ist das Finden von Fehlerzuständen in Schnittstellen und im Zusammenspiel von integrierten Komponenten.

Systemtest

Hier wird das integrierte System getestet, um zu prüfen, ob die spezifizierten Anforderungen vom Produkt erfüllt werden. Hierbei wird das System aus der Perspektive des Kunden und späteren Anwenders betrachtet.
Die Tester validieren, ob die Anforderungen vollständig und angemessen umgesetzt wurden.
Hierbei werden viele Funktionen und Systemeigenschaften erst mit dem Gesamtsystem bereitgestellt, so dass sie erst jetzt getestet werden können.
Als Testumgebung sollten die Hard- oder Sofwareprodukte zum Einsatz kommen, die der zukünftigen Produktivumgebung sehr nahe kommen.

Abnahmetest

Die bisherigen Teststufen lagen in der Verantwortung des Softwareherstellers.
Vor Inbetriebnahme der Software erfolgt nun abschließend der Abnahmetest, bei dem die Sicht und das Urteil des Kunden im Vordergrund stehen.

Es gibt folgende Formen von Abnahmetests:

1. vertragliche Akzeptanz
   Als Testkriterien gelten die im Entwicklungsvertrag festgeschriebenen Abnahmekriterien sowie die Erfüllung gesetzlicher Vorschriften. Auf Basis des Ergebnisses des Abnahmetestes entscheidet der Kunde, ob er das bestellte Softwaresystem als mangelfrei betrachtet und die vom Hersteller vertraglich geschuldete Leistung als erfüllt ansieht.
2. Benutzerakzeptanz
   Durchzuführen, wenn Kunde und Anwender verschiedene Personengruppen sind. Anwendergruppen testen das System hinsichtlich der Benutzerakzeptanz. Sollte eine Anwendergruppe das Produkt nicht akzeptieren, kann dies zum Scheitern des gesamten Projektes führen. Daher sollte man die Akzeptanz schon zu einem möglichst frühen Zeitpunkt mit Hilfe eines Prototyps testen.
3. Akzeptanz durch Systembetreiber
   Das neue System muss sich aus Sicht der Systemadministratoren in die neue Systemlandschaft einfügen. Hier ist zu testen, wie Backup-Routinen, der Wiederanlauf nach einer Systemabschaltung, die Benutzerverwaltung und Aspekte zur Datensicherung berücksichtigt werden.
4. Feldtest (Alpha- und Beta-Test)
   Der Feldtest sollte erst nach einem erfolgreichen Systemtest gestartet werden. In der Regel wird der Feldtest bei Standardsoftware durchgeführt.
   Ziel ist, Einflüsse aus nicht vollständig bekannten oder nicht spezifizierten Produktivumgebungen zu erkennen und ggf. zu beheben.
   Der Hersteller liefert das Produkt an ausgewählte Kunden, deren Produktivumgebungen die verschiedenen möglichen Umgebungen gut abdecken. Diese Kunden führen dann ausgewählte Testfälle des Herstellers durch oder setzten das Produkt probehalber unter realistischen Bedingungen ein.
   Unter Alpha- bzw. Beta-Tests versteht man das Testen von Vorabversionen durch repräsentative Kunden. Alpha-Tests finden beim Hersteller, Beta-Tests beim Kunden statt.

5.4. Testmethoden

Statischer Test

Bei statischen Tests wird das Testobjekt nicht mit Testdaten versehen und ausgeführt, sondern einer Analyse unterzogen. Diese kann in Form einer intensiven Betrachtung von Dokumenten und Quellcode durch eine oder mehrere Personen oder durch entsprechende Werkzeuge erfolgen. Ziel ist die Ermittlung von Fehlern und Verstößen gegen vorhandene Spezifikationen. Techniken sind das Review und die statische Analyse.

Unter einem Review versteht man ein bestimmtes Vorgehen bei der Prüfung von Dokumenten zur Sicherung der Qualität. Ziel ist das frühzeitige Erkennen und Beheben von Fehlern. Ein Teil des Reviews kann der so genannte Schreibtischtest sein, bei dem die Ausführung des Programms manuell analysiert wird, i. a. vom Ersteller selbst. Der Schreibtischtest sollte vor jeder weiteren Einzel- oder Gruppenprüfung stattfinden, um die gröbsten Fehler zu eliminieren. Ansonsten findet die Überprüfung in der Regel im Team statt, wodurch die Teilnehmer vom gegenseitigen Wissensaustausch profitieren.

Neben Reviews dient die statische Analyse zum Aufdecken von Fehlern in einem Dokument. Diese findet allerdings werkzeugunterstützt statt.
Compiler führen beispielsweise eine statische Analyse des Programmtextes durch, indem sie die Einhaltung der Syntax der jeweiligen Programmiersprache prüfen.[19]
Weitere Werkzeuge kontrollieren beispielsweise die Einhaltung von Programmierrichtlinien, den Kontrollfluss innerhalb des Programms (ist der Ablauf einfach oder komplex und somit fehleranfällig, …),…

Dynamischer Test

Beim dynamischen Test liegt ein ausführbares Programm bzw. Programmfragment vor, für das Testfälle definiert und durchgeführt werden.

Ein Testfall umfasst die für die Ausführung des Tests notwendigen Vorbedingungen, die Menge der Eingabewerte und die Menge der erwarteten Sollwerte, die Prüfanweisung (wie Eingaben an das Testobjekt übergeben und Sollwerte abzulesen sind) sowie die erwarteten Nachbedingungen.

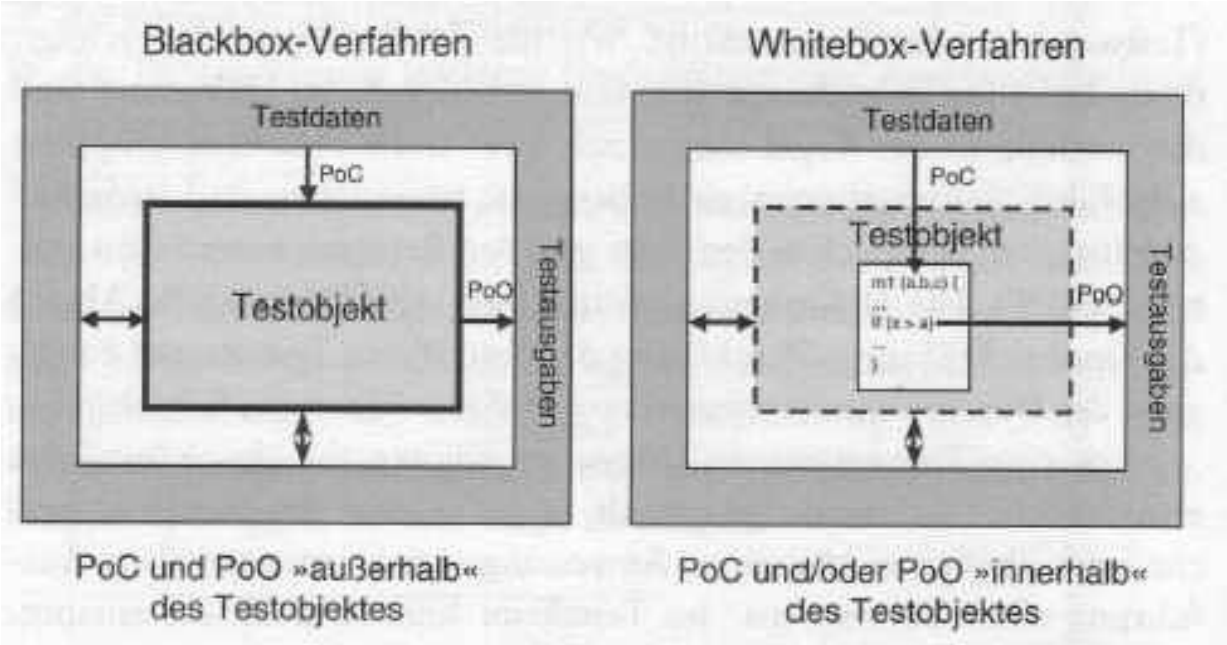
Man unterscheidet zwischen White- und Blackboxtestfallentwurfsverfahren.
Bei den Blackbox-Verfahren wird das Testobjekt als Black Box angesehen, das heißt über den Programmtext und den inneren Aufbau des Programms sind keine Informationen notwendig. Beobachtet wird das Verhalten des Testobjektes von außen, man spricht auch davon, dass der Point of Observation außerhalb des Testobjektes liegt. Die Konstruktion der Testfälle beruht auf der Anforderungsdefinition.

Whitebox-Verfahren greifen auf den Programmtext zurück. Während der Ausführung der Testfälle wird der innere Ablauf im Testobjekt analysiert, somit liegt der Point of Observation innerhalb des Testobjekts. Ein Eingriff in den Ablauf des Testobjekts ist in Ausnahmefällen möglich und Testfälle werden auf Basis der Programmstruktur des Testobjektes gewonnen.

In vielen Fällen lässt sich die Konstruktion von Testfällen nicht eindeutig einem der Verfahren zuordnen und man spricht von Greybox-Verfahren.

[19] Vgl. hierzu auch das AW-Theoriematerial zum Thema IDE und Compiler.

Die folgenden Abbildungen stellen das Black- und Whitebox-Verfahren schematisch dar.[20]



PoC = Point of Control (Schnittstelle zwischen Testdaten und Testobjekt)
PoO = Point of Observation (Schnittstelle zwischen Testausgaben und Testobjekt)

5.5. Testarten

Das Ziel des Testens variiert von Teststufe zu Teststufe.
Es lassen sich folgende grundlegende Testarten unterscheiden:

1. funktional
2. nicht funktional
3. strukturbezogen
4. änderungsbezogen

Funktionaler Test

Basis für den funktionalen Test ist die Anforderungsdefinition. Ziel ist, von außen sichtbares Ein-/und Ausgabeverhalten eines Testobjektes zu prüfen. Als Hilfsmittel dienen hierzu in der Regel nach dem Blackboxverfahren konstruierte Testfälle.

Beispiele für ein Autoverkaufsprogramm:

- Ein Automodell wird ausgewählt, dessen Grundpreis gemäß Verkaufshandbuch angezeigt wird.
- Eine Zusatzausstattung wird ausgewählt. Der Preis erhöht sich um den Preis des Zubehörteils.

[20] Vgl. Basiswissen Softwaretest, S 108, Spillner, Linz, dpunkt.verlag, 2005

Nicht Funktionaler Test

Ziel ist zu testen, wie gut bzw. mit welcher Qualität das System seine Funktionen erbringt, das heißt ob in der ISO 9126 festgeschriebene Qualitätsmerkmale, wie Zuverlässigkeit, Benutzbarkeit, Effizienz (siehe Qualitätsmerkmale in Kapitel 3.2) erfüllt werden.

Während des Systemtests sollten zusätzlich folgende nicht funktionalen Systemeigenschaften überprüft werden.

- Lasttest:
  Messung des Systemverhaltens in Abhängigkeit von der steigenden Systemlast (Anwender arbeiten parallel, Anzahl Transaktionen, …)
- Performanztest:
  Messung von Antwortzeiten für bestimmte Anwendungsfälle, in der Regel bei steigender Last
- Volumen- / Massentest:
  Beobachtung des Systemverhaltens in Abhängigkeit von der Datenmenge (Verarbeitung großer Dateien)
- Stresstest:
  Beobachtung des Systemverhaltens bei Überlastung
- Test der Stabilität / Zuverlässigkeit:
  Zum Beispiel Messung der Ausfälle pro Betriebsstunde
- Test auf Robustheit:
  Prüfung der Systemreaktion bei Fehlbedienung, Hardwareausfall sowie Überprüfung der Fehlerbehandlung
- Test auf Kompatibilität / Datenkonversion:
  Prüfung der Verträglichkeit mit anderen Systemen, wie den Import und Export von Datenbeständen
- Test unterschiedlicher Konfigurationen:
  Test von unterschiedlichen Betriebssystemen, Landessprachen, Hardwareplattformen
- Test auf Benutzungsfreundlichkeit:
  Prüfung von Erlernbarkeit und Bedienung, Verständlichkeit der Systemausgaben, Anwenderbedürfnissen
- Prüfung der Dokumentation:
  Bedienungsanleitung, GUI
- Prüfung auf Änderbarkeit, Wartbarkeit:
  Verständlichkeit und Aktualität der Entwicklungsdokumente, modulare Systemstruktur, ….

Strukturbezogener Test

Strukturbezogenes Testen basiert auf der internen Struktur der Software. Man analysiert den Kontrollfluss innerhalb von Komponenten, die Aufrufhierarchie von Prozeduren oder Menüstrukturen. Testfälle werden nach dem Whitebox-Verfahren konstruiert. Der Test findet häufig in den Teststufen Komponenten- und Integrationstest statt.

Test nach Änderungen

Mit der erstmaligen Auslieferung steht ein Softwareprodukt erst am Anfang seines Lebenszyklus. Oft ist es noch Jahre später im Einsatz und wird korrigiert und geändert. Jedes mal entsteht eine neue Version des ursprünglichen Produktes.
Jedes Update zieht einen Test nach sich. Diese Tests werden **Regressionstests** genannt.
Ziel ist nachzuweisen, dass durch die vorgenommenen Änderungen keine neuen Defekte eingebaut oder bisher maskierte Fehlerzustände freigelegt wurden.
Diese Tests finden in allen Teststufen statt und umfassen funktionale, nicht funktionale und strukturelle Tests.

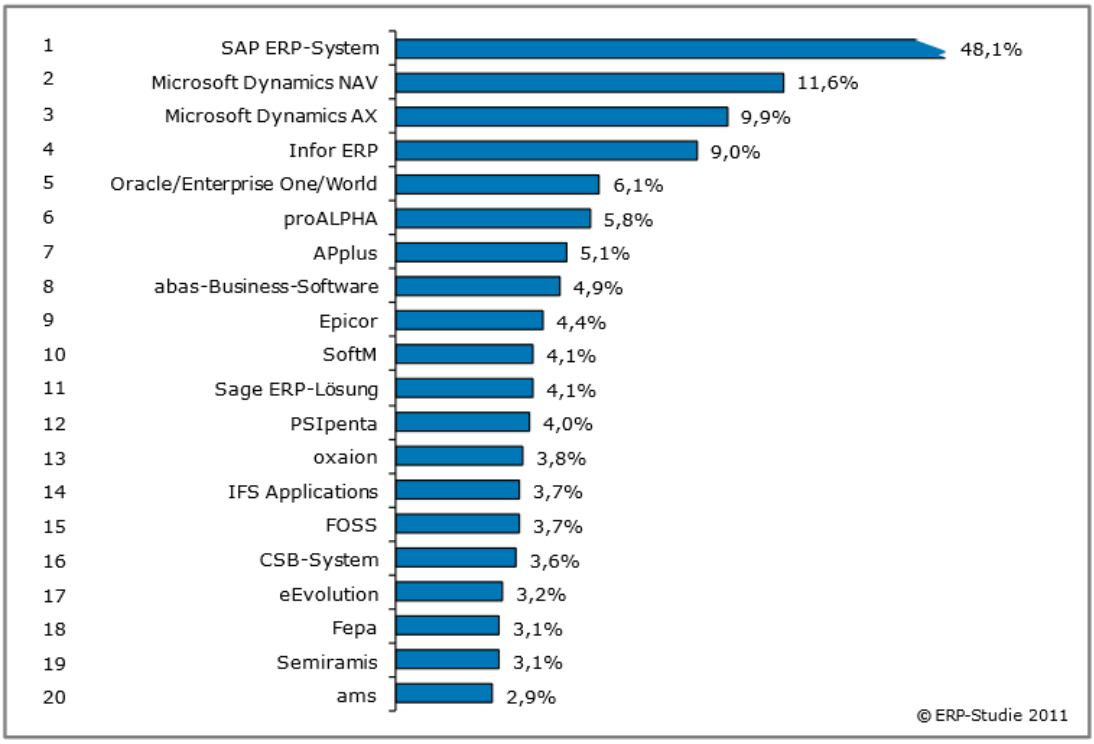
6. Enterprise-Resource-Planning-System (ERP-System)

6.1. Begriff

Im Gabler Wirtschaftslexikon wird der Begriff Enterprise-Resource-Planning-System wie folgt beschrieben:

„Ein Enterprise-Resource-Planning-System oder kurz ERP-System dient der funktionsbereichsübergreifenden Unterstützung sämtlicher in einem Unternehmen ablaufenden Geschäftsprozesse. Entsprechend enthält es Module für die Bereiche Beschaffung/Materialwirtschaft, Produktion, Vertrieb, Forschung und Entwicklung, Anlagenwirtschaft, Personalwesen, Finanz- und Rechnungswesen, Controlling usw., die über eine (in Form einer relationalen Datenbank realisierte) gemeinsame Datenbasis miteinander verbunden sind. Durch die unternehmensweite Konsolidierung der Daten ist eine Unterstützung der Planung über sämtliche Unternehmensebenen hinweg (von der Konzernebene über verschiedene Werke, Sparten und Abteilungen bis hin zu einzelnen Lagerorten) möglich.“[21]

Laut einer Anwenderstudie der Konradin Mediengruppe haben rund 80% der Betriebe heute mindestens ein ERP-Standardsystem im Einsatz.[22]
Die ebenfalls dieser Studie entnommene Grafik zeigt die Verbreitung von ERP-Standardsystemen in Industriebetrieben ab 50 Mitarbeitern.[23]



[21] Vgl. Springer Gabler Verlag (Herausgeber), Gabler Wirtschaftslexikon, Stichwort: Enterprise-Resource-Planning-System, online im Internet:
http://wirtschaftslexikon.gabler.de/Archiv/17984/enterprise-resource-planning-system-v12.html
[22] Vgl. http://www.industrieanzeiger.de/c/document_library/get_file?uuid=89d246e8-6bc1-4923-8721-28fb506a70cc&groupId=12503, Stand 7.7.2015, S. 17
[23] Vgl. http://www.industrieanzeiger.de/c/document_library/get_file?uuid=89d246e8-6bc1-4923-8721-28fb506a70cc&groupId=12503, Stand 7.7.2015, S. 23

6.2. Beispiel

Einen der bekanntesten ERP-Anbieter stellt die 1972 in Walldorf gegründete SAP SE dar. Sie beschreibt ihr Angebot wie folgt:

„Mit über 40 Jahren Erfahrung und fast 50.000 Kunden ist unsere marktführende Software für Enterprise Resource Planning (ERP) ein bewährtes und zuverlässiges Fundament für große internationale Konzerne und kleine und mittelständische Unternehmen in 25 Branchen. Mit SAP ERP profitieren Sie von der rollenbasierten Zugriffssteuerung für Daten, Anwendungen und Analysewerkzeuge und optimieren alle betriebswirtschaftlichen Prozesse von der Beschaffung bis zum Personalwesen.“[24]

Beispiele für die Anwendung finden Sie unter dem Link
http://www.mmbbs.de/index.php?id=488.

Hier werden ausführliche Beispiele für Geschäftsprozesse und deren Unterstützung durch die ERP-Software dargestellt.

[24] http://www.sap.com/germany/pc/bp/erp.html, Stand 23.06.2016

7. Industrie 4.0

Industrie 4.0[25] ist ein Begriff, der auf die Forschungsunion der deutschen Bundesregierung und ein gleichnamiges Projekt in der Hightech-Strategie der Bundesregierung zurückgeht. Er soll die Verzahnung der industriellen Produktion „mit modernster Informations- und Kommunikationstechnik“ bezeichnen. Zentraler Befähiger und wesentlicher Unterschied zu Computer Integrated Manufacturing (demzufolge Industrie 3.0 genannt) ist die Anwendung der Internettechnologien zur Kommunikation zwischen Menschen, Maschinen und Produkten. Technologische Grundlage sind cyber-physische Systeme und das „Internet der Dinge“. Die Ziele sind im Wesentlichen klassische Ziele der produzierenden Industrie wie Qualität, Kosten- und Zeiteffizienz, aber auch Ressourceneffizienz, Flexibilität, Wandlungsfähigkeit sowie Robustheit (oder Resilienz) in volatilen Märkten. Industrie 4.0 zählt zu den Kernthemen der Digitalen Agenda der Bundesregierung.

Mit der Bezeichnung „Industrie 4.0“ soll das Ziel zum Ausdruck gebracht werden, eine vierte industrielle Revolution einzuleiten. Die erste industrielle Revolution bestand in der Mechanisierung mit Wasser- und Dampfkraft, darauf folgte die zweite industrielle Revolution: Massenfertigung mit Hilfe von Fließbändern und elektrischer Energie, daran anschließend die dritte industrielle Revolution oder digitale Revolution mit Einsatz von Elektronik und IT (v. a. die speicherprogrammierbare Steuerung) zur Automatisierung der Produktion.

Der Begriff wurde erstmals 2011 zur Hannovermesse in die Öffentlichkeit getragen. Im Oktober 2012 wurden der Bundesregierung Umsetzungsempfehlungen übergeben. Am 14. April 2013 wurde auf der Hannover-Messe der Abschlussbericht mit dem Titel Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0 des Arbeitskreises Industrie 4.0 vorgelegt. Der Arbeitskreis stand unter dem Vorsitz von Siegfried Dais (Robert Bosch GmbH) und Henning Kagermann (acatech). Die zuständige Promotorengruppe der Forschungsunion blieb auch nach Vorlage des Berichtes noch aktiv, so u. a. in der Arbeitsgruppe Industrie 4.0 der gleichnamigen Plattform Industrie 4.0, einem Zusammenschluss der Branchenverbände Bitkom, VDMA und ZVEI. Die Plattform Industrie 4.0 wurde seitdem weiter ausgebaut und steht inzwischen unter der Leitung der Bundesministerien für Wirtschaft und Energie (BMWi) sowie Bildung und Forschung (BMBF). Ziel der Plattform ist die Weiterentwicklung des Begriffes der Industrie 4.0 im Dialog von Gewerkschaften, Wirtschaftsverbänden, Unternehmen, Wissenschaft und Politik.

Weiter Informationen für Ihre Präsentation finden Sie

- auf der Plattform Industrie 4.0 (http://www.plattform-i40.de) und
- im Kurs „Hands on Industrie 4.0“, OpenHPI, https://mooc.house

[25] https://de.wikipedia.org/wiki/Industrie_4.0, abgerufen am 08.07.2016

8. Big Data

Der englischsprachige Begriff Big Data[26] bezeichnet Datenmengen, welche

- zu groß oder
- zu komplex oder
- zu schnelllebig oder
- zu schwach strukturiert

sind, um sie mit manuellen und klassischen Methoden der Datenverarbeitung auszuwerten. Der traditionellere Begriff im Deutschen ist Massendaten. Big Data ist häufig der Sammelbegriff für digitale Technologien, die in technischer Hinsicht für die neue Ära digitaler Kommunikation und Verarbeitung und in sozialer Hinsicht für den gesellschaftlichen Umbruch verantwortlich gemacht werden. Big Data steht grundsätzlich für große digitale Datenmengen, aber auch für die Analyse und Auswertung.

In der Definition von Big Data bezieht sich das "Big" auf die drei Dimensionen "volume" (Umfang, Datenvolumen), "velocity" (Geschwindigkeit, mit der die Datenmengen generiert und transferiert werden) und "variety" (Bandbreite der Datentypen und -quellen). Erweitert wird diese Definition um die zwei V's "value" und "validity", welche für den unternehmerischen Mehrwert und die Sicherstellung der Datenqualität stehen. Der Begriff „Big Data“ unterliegt als Schlagwort einem kontinuierlichen Wandel; so wird mit Big Data ergänzend auch oft der Komplex der Technologien beschrieben, die zum Sammeln und Auswerten dieser Datenmengen verwendet werden. Die gesammelten Daten können aus verschiedensten Quellen stammen:

- angefangen bei jeglicher elektronischer Kommunikation,
- über von Behörden und Firmen gesammelte Daten,
- bis hin zu den Aufzeichnungen verschiedenster Überwachungssysteme.

Big Data kann auch Bereiche umfassen, die als privat gelten. Der Wunsch der Industrie und bestimmter Behörden, möglichst freien Zugriff auf diese Daten zu erhalten, sie besser analysieren zu können und die gewonnenen Erkenntnisse zu nutzen, gerät dabei unweigerlich in Konflikt mit geschützten Persönlichkeitsrechten des Einzelnen. Ein Ausweg ist allein durch

- Anonymisieren vor dem Ausbeuten, wenn nicht schon durch
- Anonymisieren vor dem Auswerten

zu erreichen. Klassische Anwender von Methoden des Big Data sind die Provider sozialer Netzwerke und von Suchmaschinen. Die Analyse, Erfassung und Verarbeitung von großen Datenmengen ist heute in vielen Bereichen alltäglich.

Big Data generiert Potenziale zur Geschäftsprozessverbesserung in allen Funktionsbereichen des Unternehmens, vor allem aber im Bereich der Technologieentwicklung & IT sowie des Marketings. Datenmengen dienen im Allgemeinen der Umsetzung von Unternehmenszielen oder zur staatlichen Sicherheit. Bisher haben vor allem große Branchen, Unternehmen und Anwendungsbereiche der Wirtschaft, Marktforschung, Vertriebs- und Servicesteuerung, Medizin, Verwaltung und Nachrichtendienste die digitalen Methoden der Datensammlung für sich genutzt. Die erfassten Daten sollen weiterentwickelt und nutzenbringend eingesetzt werden. Die Erhebung der Daten dient meistens für konzernorientierte Geschäftsmodelle, sowie Trendforschung in den sozialen Medien und Werbeanalysen, um zukunftsweisende und gewinnbringende Entwicklungen zu erkennen und in diese Prognosen zu investieren.

Weiter Informationen für Ihre Präsentation finden Sie im Kurs „Hands on Industrie 4.0“, OpenHPI, https://mooc.house (Kurswoche 1, Vortrag von Uwe Weiss (Blue Yonder): „Produktionsprozesse mit Big Data optimieren“).

[26] https://de.wikipedia.org/wiki/Big_Data, abgerufen am 08.07.2016