Informationstechnische Projekte

Kompetenzbereich	**Qualitätsmanagement**
Thema	**Software Qualitätsmanagement**

Version	Datum	Änderung	Ersteller
1.2	24.03.15	Ergänzung des Kapitels „Techniken des SQM“	Bu
1.1	24.03.15	Korrektur der Seitenumbrüche	Bu
1.0	16.03.15	Erstellung	Bu

Inhaltsverzeichnis

Allgemein

Software ist heute in vielen Bereichen des täglichen Lebens im Einsatz und nicht mehr wegzudenken. Oft ist dem Anwender nicht mehr bewusst, wie viel Software zum Funktionieren seiner Geräte erforderlich ist.

Das Verständnis der Auswirkungen von selbst kleinsten Fehlern ist auch bei den Entwicklern nicht immer vorhanden, weil ihnen „die großen Zusammenhänge“ fehlen bzw. sie nicht immer genau wissen, in welchem Umfeld ihr Code eingesetzt wird. Entwickler verwenden zur Steigerung der Produktivität sehr häufig – zum Teil auch sehr umfangreiche – Module, deren gesamte Komplexität und Funktionsweise sie nicht verstehen. Die Beherrschung des eigenen Systems und die Fehlereingrenzung wird somit immer schwieriger.

Der zunehmende wirtschaftliche Druck in der Softwareindustrie führt häufig dazu, dass die Zeit für detaillierte Analysen des Zielsystems reduziert und die Software vor der Veröffentlichung nicht mehr ausreichend getestet wird. Zur Effizienz- und Qualitätssteigerung wurden und werden laufend neue Modelle entwickelt.

Drastische Beispiele für Fehlerfolgen

Die Steuersoftware für das amerikanische Kampfflugzeug F-16 hatte verschiedene schwere Fehler die großteils in Simulationen entdeckt wurden. Einer davon hätte dazu geführt, dass das Flugzeug beim überqueren des Äquators eine um 180° auf den Rücken gedreht wird und zwar mit der für das Flugzeug maximal möglichen Beschleunigung. Die Piloten hätten diese Beschleunigung nicht überlebt (s. http://catless.ncl.ac.uk/Risks/3.44.html#subj1[/url])

Abbildung 1: F-16 "Flip Problem"

Auch in der zivilen Luftfahrt übernehmen mittlerweile Computer viele Aufgaben der Piloten. Eine Maschine der österreichischen Lauda Air verunglückte am 26. Mai 1991 um 23:17 Uhr Ortszeit auf einem Flug zwischen Hongkong und Wien mit Zwischenlandung in Bangkok kurz nach dem Start in Bangkok. Der Grund dafür war, dass im Flug die Schubumkehr aktiviert worden ist.

Abbildung 2: Schubumkehr bei einem Verkehrsflugzeug

Die Überprüfung im Flugsimulator, ob ein Pilotenfehler zu dem Absturz geführt hat, hat gezeigt, dass das Flugzeug unter diesen Umständen nicht mehr steuerbar war.

Einer der am besten dokumentierten Softwarefehler war die Ursache des Absturzes der Ariane 5 Trägerrakete der ESA. Die hohen Beschleunigungswerte der neuen Raketentriebwerke führten zu einem Integer-Überlauf in einem von der Ariane 4 übernommenen SRI (bestimmt die Lage im Raum). Dieses Modul ging daraufhin in einen Modus, in dem es nur Status-Informationen aussendet. Ein baugleiches redundantes Modul verhielt sich genauso. Die Status-Informationen wurden allerdings als Kurs-/Navigationsinformationen interpretiert und das Steuersystem hat somit Kursabweichungen korrigiert, die überhaupt nicht vorhanden waren. Kurz vor der Zerstörung der Rakete durch die anströmende Luft, wurde sie gesprengt um den Schaden am Boden durch zu große Trümmer zum minimieren. Details können hier nachgelesen werden: http://de.wikipedia.org/wiki/Ariane_V88.

Besonders hohe Sicherheitsstandards müssen im medizinischen Bereich angewendet werden. Eine Verkettung unglücklicher Umstände führte dazu, dass das Kombigerät Therac-25 zur radiologischen Untersuchung und Bestrahlung Patienten z.T. mit der 100 fachen Strahlendosis behandelt hat. Drei Patienten starben daran und drei weitere wurden schwer verletzt.

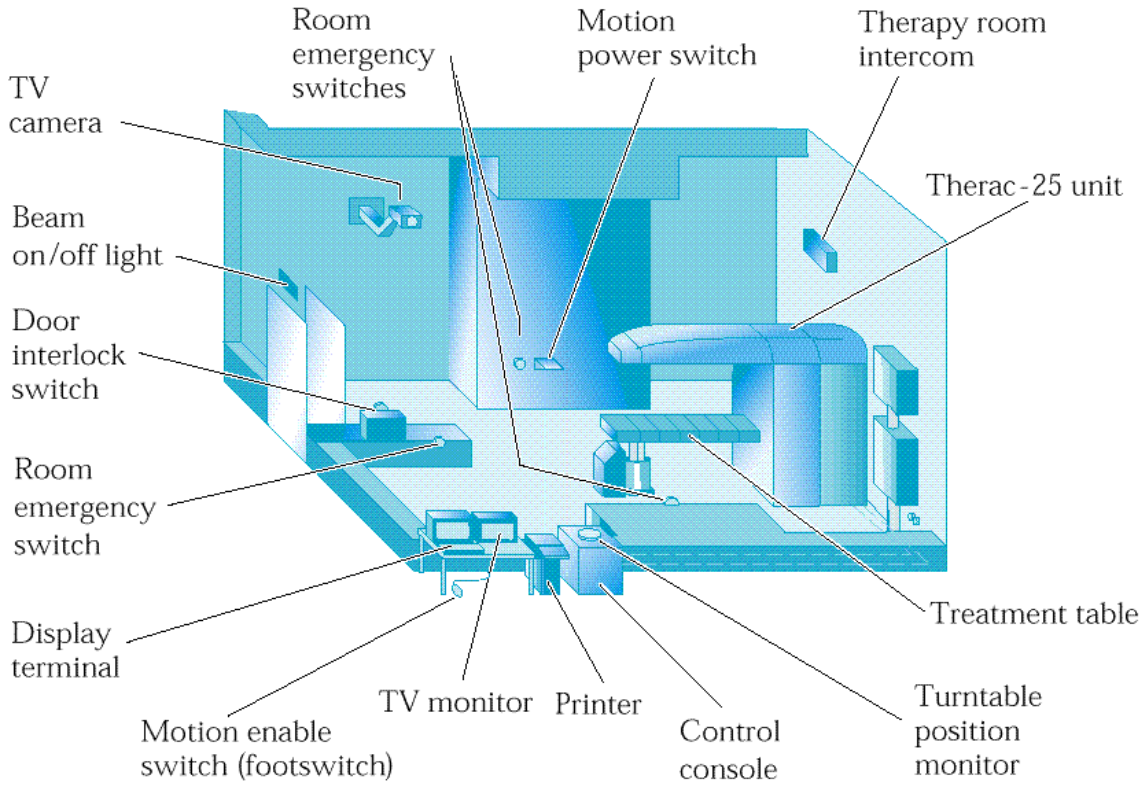


Abbildung 3: Therac-25

Weitere Beispiele

- Segway scooter – wirft seinen Fahrer ab, wenn der Akkuladestand unter einen kritischen Wert sinkt.
- Toyota Prius - Software schaltet Motor ab
- US Airways Tickets um unter 2$
- Europäische Luftraumüberwachung
- Gepäcksortierungssystem in London Heathrow
- Zentrale Brief- und Paketsortieranlage in Wien
- USS Yorktown – durch Eingabe von „0“ direkt in die Datenbank der Steuerung war das Kriegsschiff 2:45 nicht manövrierfähig (s. http://de.wikipedia.org/wiki/USS_Yorktown_%28CG-48%29)

Definition von Software Qualität

Die Definition von Softwarequalität hat zum Ziel, qualitative – oft intuitive – Aussagen über die Qualität von Software durch quantitative, reproduzierbare Aussagen zu ersetzen.

Beispiele

Qualitativ, intuitiv:

- Der Entwickler sagt: „Ich habe mein Softwaremodul ausgetestet“
- „Wir wollen Null-Fehler-Software machen!“

Quantitativ, reproduzierbar:

- „Mein Testwerkzeug zeigt zur Zeit eine Zweigüberdeckungsrate von 57% (70 von 123 Zweigen) an. In unserer Firma gelten Module als ausreichend getestet, wenn die Zweigüberdeckungsrate mindestens 95% beträgt. Ich muss daher noch mindestens 47 Zweige testen und erwarte aufgrund der Erfahrung mit vergleichbaren Modulen, dass ich dafür etwa 1,5 Mitarbeitertage zusätzlichen Aufwand benötige.“
- „Das Restrisiko beträgt 5 FIT[1].“

1 Failure in Time, Anzahl der auftretenden Fehler pro 10^9 Betriebsstunden

Eine Definition für Softwarequalität

„Unter Softwarequalität versteht man die Gesamtheit der Merkmale und Merkmalswerte eines Softwareprodukts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen"[2]

Modelle des Software Qualitätsmanagments

Historische Entwicklung

Mitte der 1960er-Jahre überstiegen erstmalig die Kosten für die Software die Kosten für die Hardware („Software Krise“). In der Folge kam es zu den ersten großen gescheiterten Softwareprojekten. Die bis dahin genutzten Techniken zur Softwareentwicklung konnten mit dem Umfang und der Komplexität der Software nicht mehr Schritt halten.

Auf einer NATO-Tagung 1968 wurde als Reaktion der Begriff des *Software Engineering* geprägt:

„zielorientierte Bereitstellung und systematische Verwendung von Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresystemen.“ (Lit.: Balzert, S.36)

Heute arbeitet eine eigene Gruppe am IEEE an einer Sammlung verschiedenster Teilbereiche des Software Engineering. Die Ergebnisse dieser Arbeit sind im sog. SEBOK (Software Engineering Body of Knowledge) zusammengefasst (http://www.computer.org/web/swebok).

Entwicklungsschritte der Softwareentwicklung

Früher wurden Tests am Ende der Softwareentwicklung durchgeführt. Die Folgen davon sind ähnlich, wie für materielle Güter: man verliert Entwicklungszeit und weitere Ressourcen.

Heute verfolgt man zunehmend sog. konstruktive und prozessbezogene Ansätze. Diese setzen bereits bei der Analyse der Anforderungen an ein Produkt an. Fehler sollen dadurch von vornherein vermieden werden. Die Qualität des Produktes soll durch die Qualität des Herstellungsprozesses garantiert werden. Eine Steigerung der Qualität des Herstellungsprozesses soll somit unmittelbar zur Steigerung der Qualität des Produktes führen.

2 Helmut Balzert', 'Lehrbuch der Softwaretechnik, Teil 2: Softwaremanagement, Software-Qualitaetssicherung, Unternehmensmodellierung, 1998, S. 257, ISBN 3-8274-0065-1

Basis für prozessorientiertes Software-Qualitätsmanagement sind

- ISO 900x
- V-Modell
- Capability Maturity Model (CMM)
- SPICE (ISO 15 504)

Software Lebenszyklus

Die Software Krise führte zur Entwicklung strukturierter Prozessmodelle für die Erstellung von Software. Ein Prozessmodell bietet eine Anleitung für den Entwickler, welche Aktivitäten als nächstes kommen. Der Vorteil einer solchen Herangehensweise ist, dass der Entwicklungsprozess (aus betriebswirtschaftlicher Sicht) besser plan- und kontrollierbar wird.

- Projektplanung
- Kostenkontrolle
- Abnahme von Zwischen- und Endprodukten
- ...

Die ersten Modelle hielten sich sehr eng an den sog. Software Lebenszyklus (s. Abbildung 4)

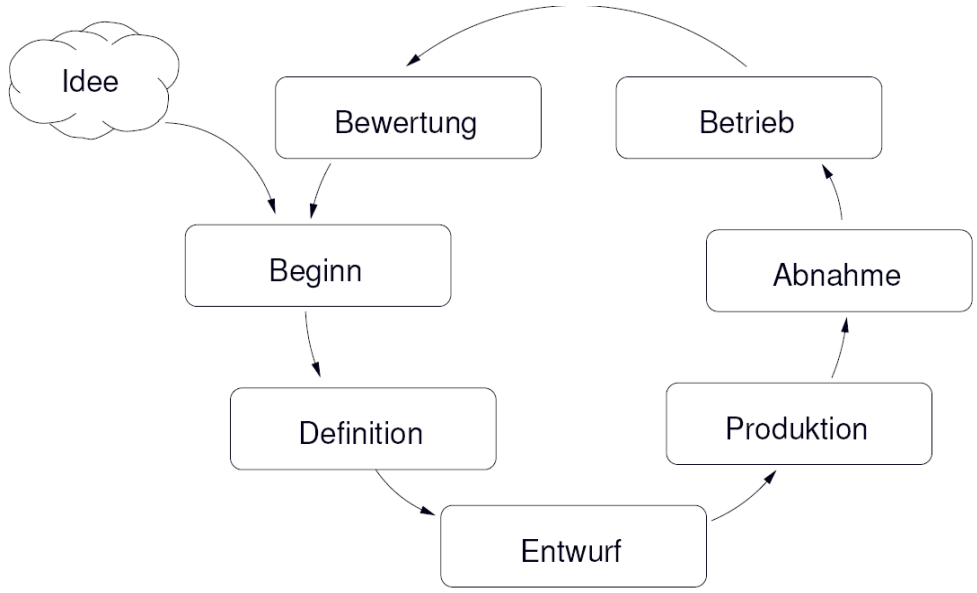


Abbildung 4: Software Lebenszyklus

Wasserfallmodell

Beim Wasserfallmodell wird der Entwicklungsprozess in Phasen zerlegt. Jede Phase hat einen wohldefinierten Anfangs- und Endpunkt, welcher klar den Übergang zur nächsten Phase definiert (s. Abbildung 5). Eine Phase muss abgeschlossen sein, bevor die nächste beginnen kann. Jede Phase produziert zumindest ein Dokument (bei der Softwareentwicklung zählt auch der Programmcode als Dokument).

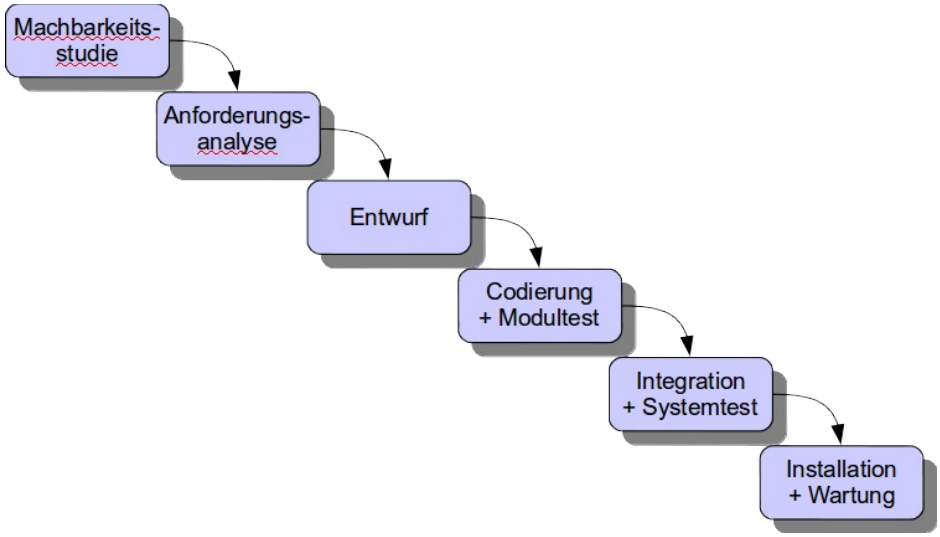


Abbildung 5: Phasen eines Wasserfallmodells

Schwächen dieses Modells

- Am Projektanfang sind nur ungenaue Schätzungen der Kosten, Aufwände und Ressourcen möglich.
- Ein Pflichtenheft, egal wie sorgfältig erstellt, kann nie den Umgang mit dem fertigen System ersetzen. Dies ist für viele Kunden ein Problem.
- Es gibt Kunden, mit denen man keine präzisen Pflichtenhefte erstellen kann, weil sie nicht wissen, was sie wollen.
  Beispiel: In Berlin sollte ein Programm zur Wohnungsvermittlung erstellt werden. Es sollte die Wohnungen nach sozialen Kriterien vergeben, aber auch verfügbare Wohnungen sofort vermitteln.
- Oft werden die endgültigen Anforderungen erst nach einer gewissen Experimentierphase deutlich.
- Die Änderung von Anforderungen wird in diesem Modell überhaupt nicht unterstützt, stattdessen werden die Anforderungen frühzeitig (Phase Anforderungsanalyse) eingefroren.

- Am Ende jeder Phase muss ein Dokument abgeliefert werden. Dies kann zu einer Papierflut und ausufernder Bürokratie führen, ohne die Qualität zu verbessern.

Wasserfallmodell mit Rückkopplung

In der Praxis wird eine strenge Trennung der einzelnen Phasen selten erreichen - dies ist nur der Idealfall um der ad-hoc Entwicklung ohne klare Struktur entgegenzuwirken. Das Wasserfallmodell mit Rückkopplung bietet deshalb die Möglichkeit, in frühere Phasen zurückzukehren, wenn etwa Fehler oder Inkonsistenzen Phase entdeckt werden (s. Abbildung 6).

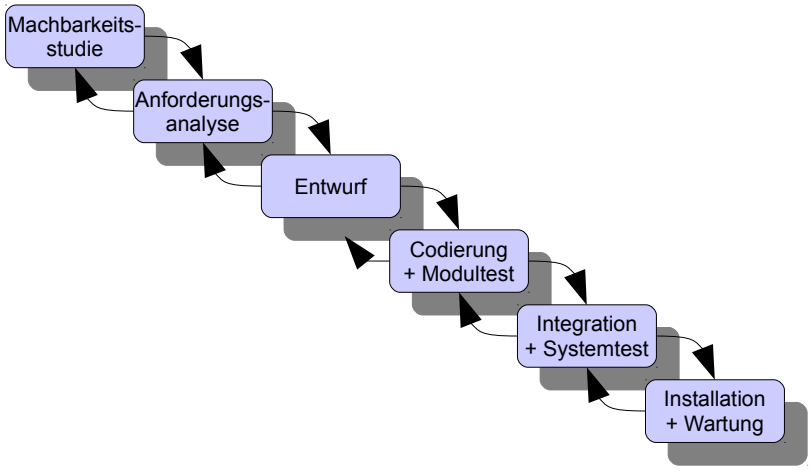


Abbildung 6: Wasserfallmodell mit Rückkopplung

Wenn man z.B. bei Tests erkennt, dass eine wichtige Funktionalität fehlt, kann diese kann nicht einfach „dazuprogrammiert“ werden, sondern muss auch sauber analysiert und konzipiert werden.

Weitere Modelle

Evolutionäre Entwicklung

Evolutionäre Modelle sind eine Alternative zum Wasserfallmodell. Die Realisierung des Produktes geschieht als Folge von Annäherungen an das Zielsystem. Jede neue Approximation entsteht durch Änderung bzw. Erweiterung der Vorgängerversion. Alle Phasen des Software Lebenszyklus können so realisiert werden!

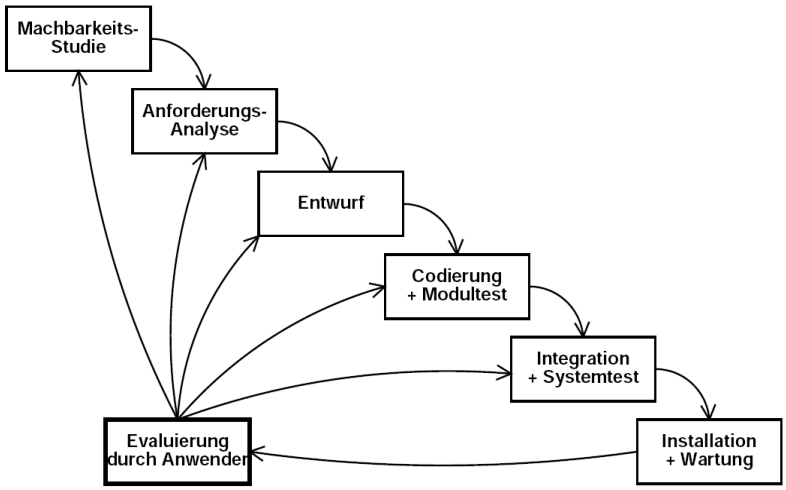


Abbildung 7: Evolutionäres Modell

Prototyping

Ein Prototyp ist ein System mit

- reduziertem Funktionsumfang und/oder
- reduzierter Benutzerschnittstelle und/oder
- reduzierter Leistung

Eine mögliche Unterscheidung von Prototypen bietet das Schichtmodell für Software mit Präsentations-, Logik- und Datenhaltungsschicht. Danach kann man zwei Arten von Prototypen unterscheiden:

Horizontaler Prototyp

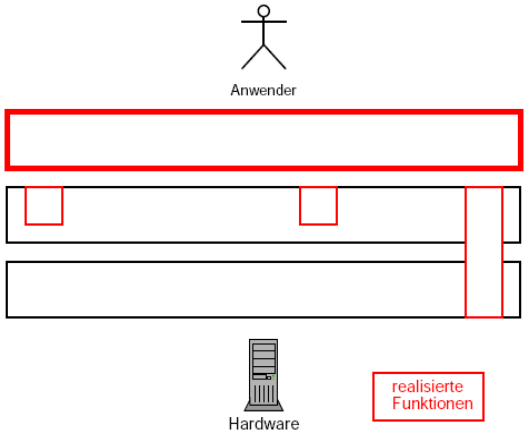


Abbildung 8: Horizontaler Prototyp

Beim Horizontalen Prototyp wird nur eine Systemschicht (s. Abbildung 8) implementiert - z.B. eine hohle Benutzerschnittstelle ohne echte Funktionalität.

So können zukünftige Benutzer frühzeitiges zur Validierung der Spezifikationen eingebunden werden. Ein Nachteil dieses Prototyps ist, dass er nur für Demos benutzbar ist.

Vertikaler Prototyp

Der Vertikale Prototyp hat meist einen eingeschränkten Funktionsumfang, der allerdings durch alle Ebenen des Systems implementiert wird (s. Abbildung 9). So können frühzeitig technische Eigenschaften validiert (z.B. durchgängiger Zugriff auf die Datenhaltungsschicht von der Präsentationsschicht aus) und Teile des Kernsystems an den Kunden ausgeliefert werden. Diese Art Prototyp ist meist sehr teuer und aufwändig.

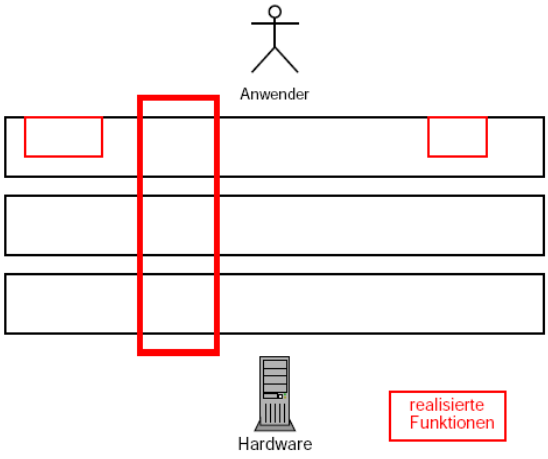


Abbildung 9: Vertikaler Prototyp

Vertikaler Prototyp

Bei „Rapid Prototyping“ werden oft Code-Generatoren oder Skriptsprachen verwendet, um dem Kunden rasch Teilaspekte der Anwendung demonstrieren zu können. So können frühzeitig Validierungen vorgenommen werden.

Nachteile des Rapid Prototyping sind u.U. ineffiziente und schlechte Benutzerschnittstellen und bei Skriptsprachen schlechte Systemarchitektur („Spagetticode“, z.T. keine saubere Modularisierung möglich)

Ohne Anspruch auf Vollständigkeit seien hier noch einige weitere Modelle aufgezählt:

- Spiralmodell
- kombiniert evolutionäre Aspekte mit Risikobewertung
- Agile Methoden (z.B. Extreme Programming (XP))
  ein Prozess zum schnellen Programmerstellen in unsicherem Umfeld
  - Auftraggeber beschreibt die Anforderungen über Stories
  - Pair Programming (Programmieren in Paaren) sorgt für ständiges Gegenlesen durch den Partner

Messung von Software Qualität

Zur Beurteilung von Software Qualität und zu deren Verbesserung müssen geeignete Kriterien und Merkmale gefunden werden, denn die oben angeführte Definition von Software Qualität ist bewusst sehr allgemein gehalten. Es geht darum, qualitative – oft intuitive – Aussagen über Software durch quantitative, reproduzierbare Aussagen zu ersetzen (s. Abbildung 10).

Abbildung 10: Indizien für qualitativ hochwertige Software

Die Qualitätsmerkmale von Software können grob in zwei Gruppen gegliedert werden (s. Abbildung 11):

- äußere Qualitätsmerkmale und
- innere Qualitätsmerkmale

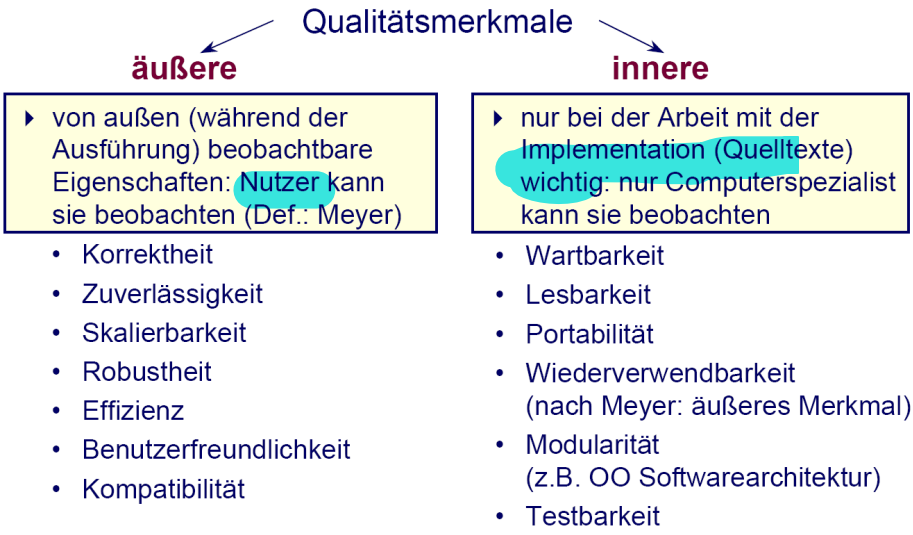
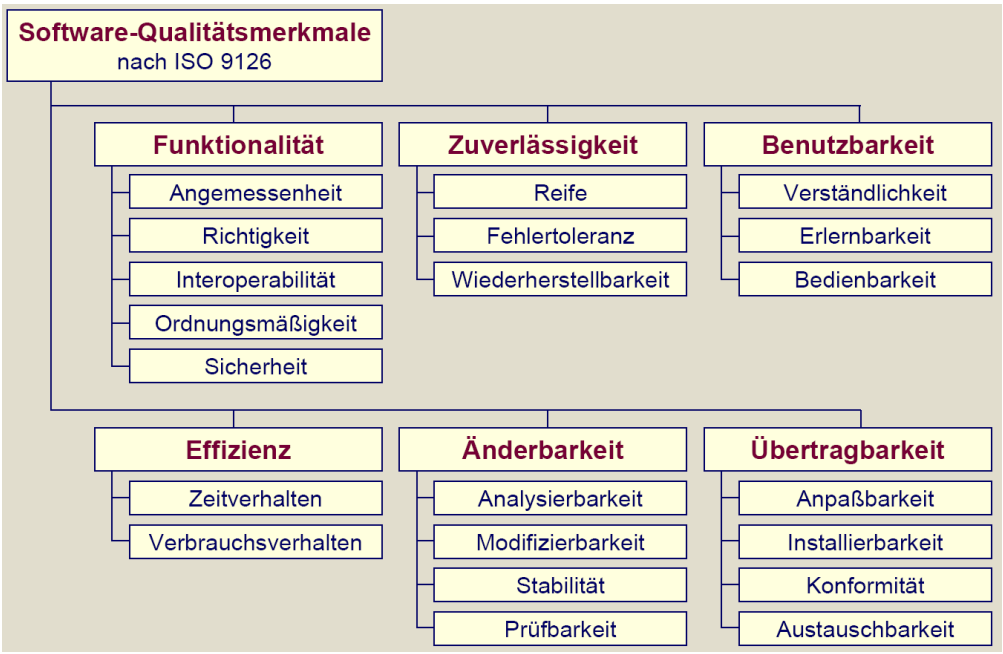


Abbildung 11: äußere und innere Qualitätsmerkmale

Die ISO definiert folgende Software-Qualitätsmerkmale:



Techniken des Software Qualitätsmanagements

ISO 90003:2004

Die ISO 90003:2004 ist die Anwendung von ISO 9000ff auf Software bzw. Softwareentwicklung. Sie soll ein Leitfaden für die Anwendung von ISO 9001 auf die Entwicklung, Lieferung und Wartung von Software sein.

CMM / CMMI

CMM (Capability Maturity Modell) und CMMI (Capability Maturity Modell Integration) sind sogenannte Reifegradmodelle. Das CMM wurde auf Initiative des US-Verteidigungsministeriums vom Software Engineering Institute (SEI) an der Carnegie Mellon University/Pittsburgh (untersteht dem US-Verteidigungsministerium) entwickelt. Ziel war die Entwicklung von Richtlinien zur Beurteilung der

- Leistungsfähigkeit und
- Vertrauenswürdigkeit

von SW-Lieferanten in Angebotsphase.

Allerdings entstanden für jede Disziplin (SW-Entwicklung, SW-Beschaffung, …) jeweils eigene CMM, was zu einem Wildwuchs und Unübersichtlichkeit führte. Deshalb wurden die Arbeiten am CMM eingestellt und das CMMI entwickelt. Es sollte ein einheitliches, modulares und vor allem allgemein verwendbares Modell für die Bereiche

- Systems Engineering
- Software Akquisition und
- Software Engineering

bieten. Es ermöglicht multidisziplinäre Prozessverbesserungen, ist aber wegen seiner vielen Dokumente und Vorschriften ein unhandliches Modell.

CMM bzw. CMMI sind Reifegradmodelle, die 5 Reifegrade (maturity levels) definieren. Sie charakterisieren den Grad der Beherrschung des Entwicklungsprozesses. Der Reifegrade wird aufgrund bestimmter Kriterien bestimmt:

Reifegrad		Eigenschaften	Notwendige Verbesserungen
1	„Initial“	Prozessablauf „chaotisch“ und Leitung „ad hoc“ (wenn's brennt)	Projektführung, Projektplanung, Konfigurationsmanagement, Qualitätssicherung
2	„Managed“	Prozess intuitiv und personenabhängig beherrscht	Ausbildung, technische Praktiken (Reviews, Tests), Konzentration auf Normen und Teams
3	„Defined“	Prozess qualitativ erfasst, institutionalisert und projektspezifisch anpassbar	Analyse und „Messung“ des Prozesses, quantitative Qualitätspläne
4	„Quantitatively Managed“	Prozess quantitativ erfasst und verstanden	Wechsel in der Technologie, Problemanalyse, Vermeidung von Problemen
5	„Optimizing“	Rückwirkung der Verbesserungen auf den Prozess	Organisation der Produktion auf optimierter Ebene

Nach der Feststellung des aktuellen Reifegrades gibt CMMI konkrete Verbesserungs-vorschläge zur Erreichung des nächsthöheren Reifegrades an. In diesem Modell können keine Stufen übersprungen werden!

BOOTSTRAP

Anpassungen von CMM(I) an die Anforderungen von ISO 9000ff, führten zu BOOTSTRAP. Mit BOOTSTRAP sollten

- die Messung der Prozessreife
- konkrete Maßnahmen zur Zertifizierung nach ISO 9000ff und
- Prozessverbesserungen

möglich sein. Die Erhebung des Reifegrades erfolgt anhand von Fragebögen, deren Auswertung empfohlene Sofortmaßnahmen (Umsetzung in „Mini-Projekten“) liefert.

Zusätzlich zu Reifegrad und Aktionsplan wird auch ein Stärken-Schwächen-Profil erstellt. Ist eine ISO 9000ff -Zertifizierung beabsichtigt, kann mit dem BOOTSTRAP-Reifegrad die Chance auf eine erfolgreiche Zertifizierung abgeschätzt werden (> 3 ist Zertifizierung ohne großen Zusatzaufwand möglich).

ISO/IEC 15504 (SPICE)

Die zunehmende Internationalisierung bei der Umsetzung von IT-Projekten führte zur Entwicklung der Software Process Improvement and Capability dEtermination (SPICE). Damit ist es möglich, internationale Projekte nach gemeinsamen Qualitätsnormen abzuwickeln. Es beinhaltet dazu Standards zur skalierbaren Softwareprozessmessung und -verbesserung.

SPICE kennt 3 Modi, um bei Projekten rasch die Qualitätsfähigkeit bestimmen zu können:

- Zustandsbestimmung
- Prozessverbesserung
- Selbstbefragung und -auswertung

Die wesentlichen Vorteile von SPICE sind:

- international einheitliche Qualitätsstandards bei Auswahl von Zulieferern (z.B. bei internationalen Ausschreibungen)
- einheitliches Vorgehensmodell zur Qualitätsbestimmung für Softwareanbieter

	Reifegradstufen	Prozessattribute	Ziel der Bewertung
5	"Optimierend"	▪ Prozessinnovation (PA 5.1) ▪ Kontinuierliche Verbesserung (PA 5.2)	Wird der Prozess kontinuierlich verbessert?
4	"Vorhersagbar"	▪ Prozessmessung (PA 4.1) ▪ Prozesssteuerung (PA 4.2)	Bewegt sich der Prozess in vorgegebenen Grenzen?
3	"Etabliert"	▪ Prozessdefinition (PA 3.1) ▪ Prozessanwendung (PA 3.2)	Existiert ein Standard-prozess?
2	"Geführt"	▪ Management d. Durchführung (PA2.1) ▪ Management d. Arbeitsprodukte (PA 2.2)	Wird der Prozess gesteuert?
1	"Durchgeführt"	▪ Process performance (PA 1.1)	Erfüllt der Prozess seinen Zweck?
0	"Unvollständig"		

Abbildung 12: Reifegradmodell von SPICE