Karlsruher Institut für Technologie

Institut für Technik der Informationsverarbeitung

Vorlesung: Informationstechnik (IT)

Institutsleitung
Prof. Dr.-Ing. K. D. Müller-Glaser
Prof. Dr.-Ing. J. Becker
Prof. Dr. rer. nat. W. Stork

Prof. Dr.-Ing. K.D. Müller-Glaser

Institut für Technik der Informationsverarbeitung (ITIV)

Softwareentwicklung & Qualität

Überblick

Karlsruher Institut für Technologie

- Qualität von Software
- Entwicklungsablauf
- Verifikation & Validierung
- Tests
- Kundenakzeptanz

Software Engineering

Technische Disziplin, die sich mit allen Aspekten der Softwareherstellung beschäftigt:

- Von den frühen Phasen der Systemspezifikation, der Analyse, dem Design, der Implementierung , dem Test,
  bis hin zur Wartung des Systems, nachdem sein Betrieb aufgenommen wurde

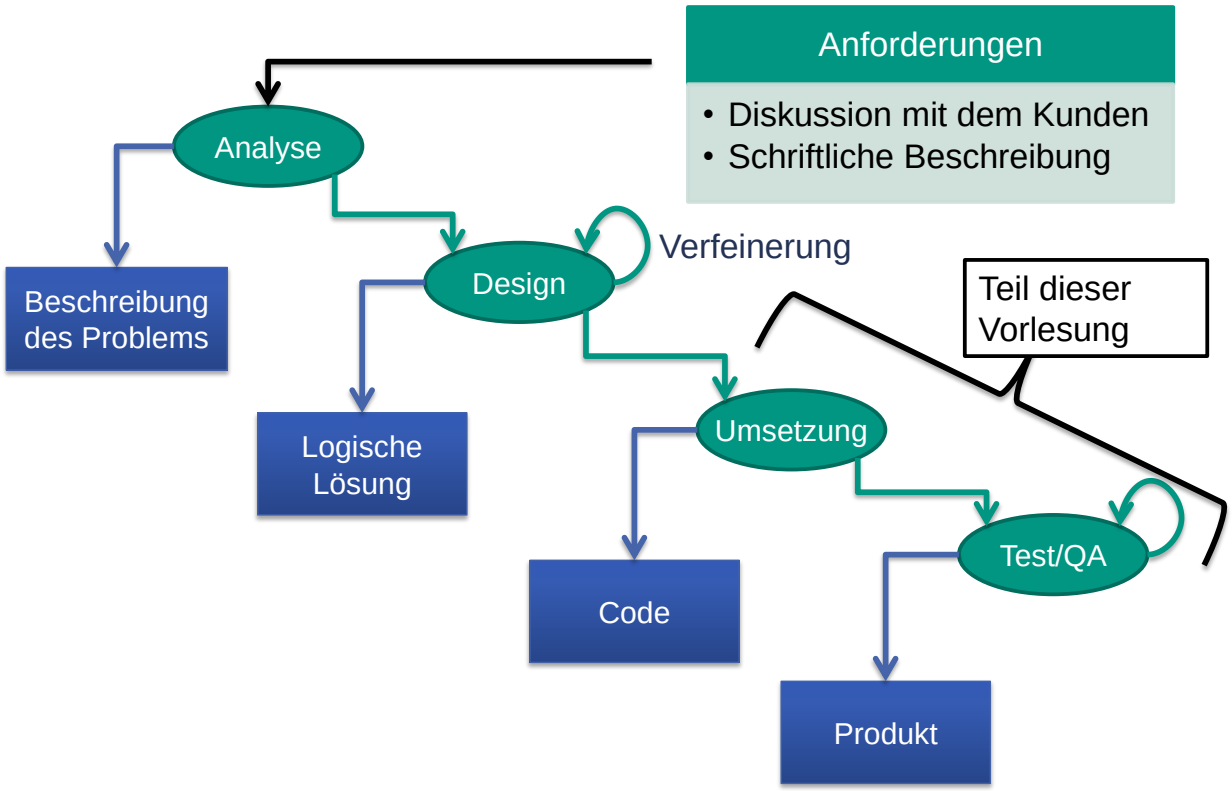
Softwareentwicklung ist mehr als das bloße Schreiben von Programmcode.

Sie beginnt bereits bei der Erfassung des Bedarfs, geht weiter bei der Planung einer Softwarearchitektur, erstreckt sich über mehrere Entwicklungszyklen und endet nicht bei der Dokumentation und Installation beim Kunden.

Wesentliche Merkmale guter Software

Zuverlässigkeit	• Verlässlichkeit • Zugriffsschutz • Betriebssicherheit, keine körperlichen, seelischen oder wirtschaftlichen Schäden
Wartungsfreundlichkeit	• Softwareveränderung • Weiterentwicklung • Diagnostizierbarkeit
Effizienz	• keine Verschwendung von Systemressourcen wie Speicher oder Prozessorkapazität • Kosteneffizienz
Benutzerfreundlichkeit	• nutzbar ohne unangemessene Anstrengung • Benutzeroberfläche • Dokumentation
Nützlichkeit	• Erfüllung der Benutzeranforderungen

Software-Entwicklungsprozesse



Anforderungen erfassen

- Es finden Kundengespräche statt
- Der Bedarf wird festgestellt
  - Lastenheft
- Festhalten von (schriftlich! da Vertragsgrundlage):
  - Pflichtenheft
  - Zielvereinbarungen
- Kunde erhält Einschätzung über Umfang und Form der Lösung

Der potentielle Kunde hat meist zu Anfang zwar eine Vorstellung, welches Problemfeld er gelöst haben möchte, kann aber nicht unbedingt daraus konkrete Anforderungen an die dafür notwendige Software ableiten.

Deshalb muss zunächst soweit möglich festgestellt werden, was die Anforderungen des Kunden sind (Erstellung eines Lastenheftes), was man als Auftragnehmer (Entwicklungsunternehmen) überhaupt leisten kann, sollte und ob man dies alleine schaffen kann.

Kunde und Verkaufsberater einigen sich dann auf ein Pflichtenheft, dass festhält, was alles geleistet werden soll.

Nicht zu unterschätzen ist auch der Aufwand, der in eine Einschätzung über die Dauer des Projekts gesteckt werden muss.

Analyse

- Warum braucht der Kunde diese Lösung?
- Warum von uns?

- Wo liegt eigentlich das Problem bei der Umsetzung?
  (Systematische Erfassung und Analyse der Anforderungen siehe Vorlesung Systems and Software Engineering)

- Welche Ressourcen benötigt man?

⇨Beschreibung des Problems

Weiß man, was der Kunde verlangt, muss noch einmal intern geklärt werden, welche Bedürfnisse den Anforderungen beim Kunden zugrunde liegen. Dies hilft später bei der Auslegung einzelner Features, dem Kunden auf Anhieb das zu liefern, was ihm den optimalen Nutzen beschert.

Für die eigene Unternehmensstrategie ist auch relevant zu erfahren, weshalb man für den Kunden der Anbieter der Wahl ist, um seine Marktnische besser ausfüllen zu können und Schwachstellen ausbessern zu können.

Dann wird untersucht, was eigentlich gelöst werden muss: Viele Aspekte der Kundenanforderung sind zwar mit „Standardlösungen“ abzudecken, trotzdem muss erfasst werden, in welchen Schritten diese angewandt werden. Dabei wird dann auch klar, wo es noch echten Entwicklungsbedarf gibt, und wo man bestimmte Ressourcen (Personal, Rechner, Entwicklungstools, Fahrzeuge, Schulungen etc.) braucht.

Resultat: Eine Beschreibung des Entwicklungsproblems

Design

- Wie lässt sich das Problem gliedern?
- Wie sehen die Schnittstellen aus
  - zu bestehender Software beim Kunden
  - zu Hardware
  - zu anderen Programmteilen (untereinander)
- Modularisierung, Entwurf für und mit Wiederverwendung
- Welche Bibliotheken können eingesetzt werden
- Was lässt sich kostengünstiger/besser extern einkaufen
- Passt das Design zum Problem? → Verfeinerung

⇨ **Architektur**

Umsetzung

- Programmierer setzen den ihnen zugeteilten Systemteil um
- Aufgabe ergibt sich aus Design, Beschreibung des Problems
- Verifikation, dass Design eingehalten wird
- Unter Umständen: Anpassung des Designs

⇨ Code

Die Anforderungen müssen jetzt nach der Analyse und dem Software Design in Programmcode implementiert werden.

Da die Architektur schon eine Aufteilung des Problems und logische Schnittstellen vorsieht, ist eine Aufteilung des Codes auf mehrere Teammitglieder möglich.

Das Design gibt dabei sowohl die Gliederung als auch die Schnittstellen der einzelnen Module vor. Die Funktionalität ergibt sich implizit aus dem Design, oder aus der Problemstellung des Kunden.

Damit die Teams gut zusammenarbeiten wird unter ihnen eine Regel-Kommunikation und zugehöriger Informationsaustausch etabliert.

Codedateien werden in einem Versionierungssystem abgelegt, so dass später Entwicklungsschritte nachvollzogen werden, verschiedene Entwicklungsrichtungen zusammengeführt und Konflikte gelöst werden können.

Bei der Implementierung stellen sich häufig Inkonsistenzen im Design heraus, oder dass Teilaufgaben mit der vorgesehenen Architektur nicht elegant oder gar nicht gelöst werden können. In diesem Fall fließen die Erkenntnisse aus der Implementierung zurück ins Design und dieses wird angepasst (Entwurfs-Iterationen).

Testen

- Fehler (Bugs) können sich immer einschleichen:
  - menschliches Versagen
  - widersprüchliche Anforderungen / Design
  - nicht spezifikationsgemäßes Verhalten von Software aus Bibliotheken, Hardware, anderen Komponenten
- Notwendig für Stabilität
- Erheblicher Zeitaufwand

Das Testen ist ein begleitender Prozess bei der Softwareentwicklung. Jeder Entwurfsschritt wird gefolgt von einem Verifikationsschritt.

Schon beim Kundengespräch wird festgelegt, wie sich das Gesamtsystem verhalten soll; es werden Abnahmetests (Validierung) vereinbart.

Aufgabe des Testens ist es, die Qualität der ausgelieferten Software sicherzustellen.

Tests haben also sowohl die Funktion, Fehler in der Umsetzung festzustellen, als auch den Fortschritt zu dokumentieren (Reifegrad).

Bei der Analyse wird erkannt, wie sich das System informationstechnisch verhalten muss.

In der Designphase wird schon festgelegt, in welcher Art welche Komponenten (Module und ihre Schnittstellen, Hierarchie System, Subsystem, Module, …) implementiert werden.

Dies gibt auch die hierarchische Vorgehensweise beim Testen vor: Es wird bei der Software Implementierung Bottom-Up geprüft, da man ja wohldefinierte Schnittstellen zwischen den einzelnen Modulen hat.

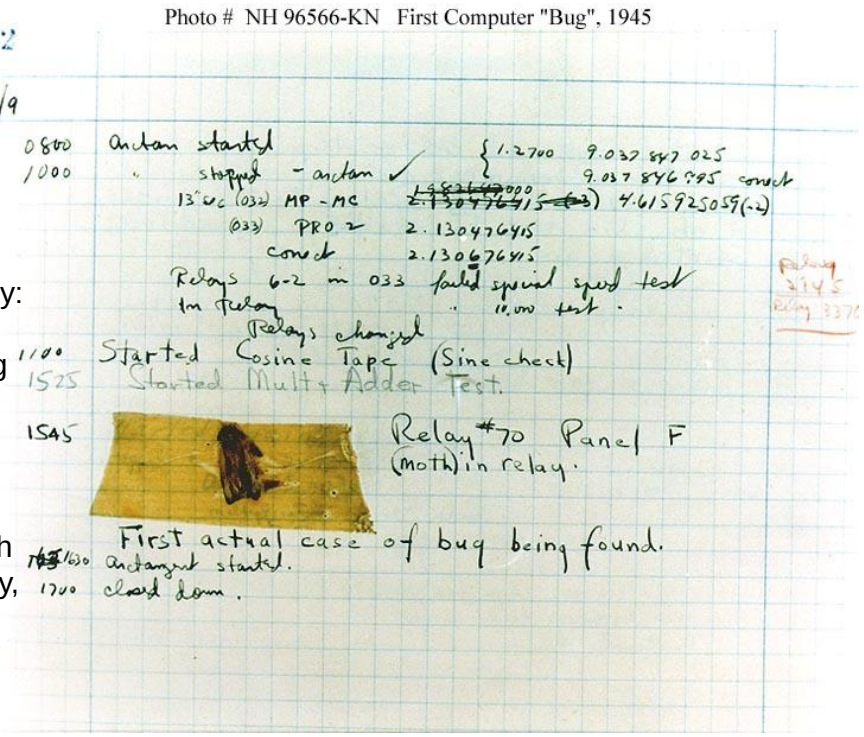
So kann jedes Modul einzeln auf Korrektheit geprüft werden, bevor es in übergeordnete Module integrierte und dort erneut geprüft wird bis das Gesamtsystems (Software, zuletzt Hardware und Software) integriert und geprüft wird.

Testen

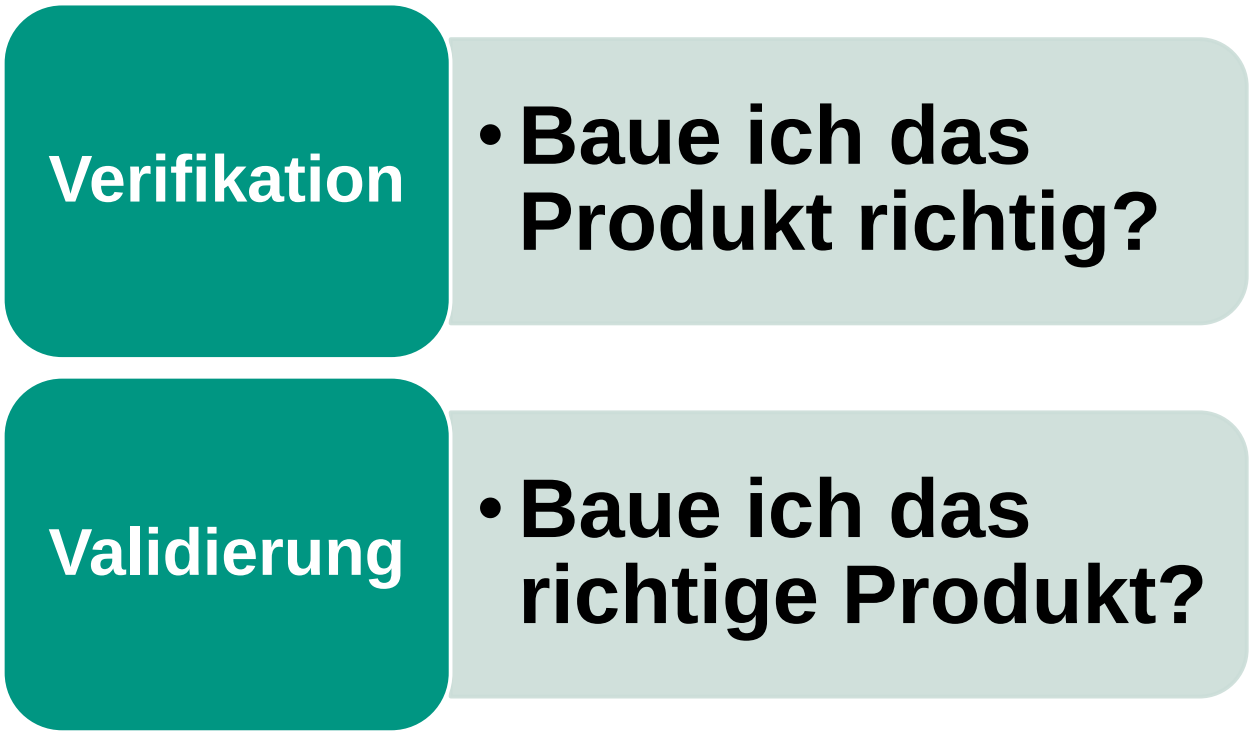
Debuggen:

Moth found trapped between points at Relay # 70, Panel F, of the Mark II Aiken Relay Calculator while it was being tested at Harvard University, 9 September 1945. The operators affixed the moth to the computer log, with the entry:

"First actual case of bug being found". They put out the word that they had "debugged" the machine, thus introducing the term "debugging a computer program". In 1988, the log, with the moth still taped by the entry, was in the Naval Surface Warfare Center Computer Museum at Dahlgren, Virginia.



Verifikation und Validierung



Da es darum geht, eine Aufgabenstellung des Kunden zu lösen, geht Softwarequalitätssicherung über das Schreiben guter und getesteter Software hinaus: Es muss die so Software ausgelegt sein, dass der Kunde diese zur Zufriedenheit einsetzen kann.

Daher muss der Kunde stetig in den Entwicklungsprozess eingebunden werden.

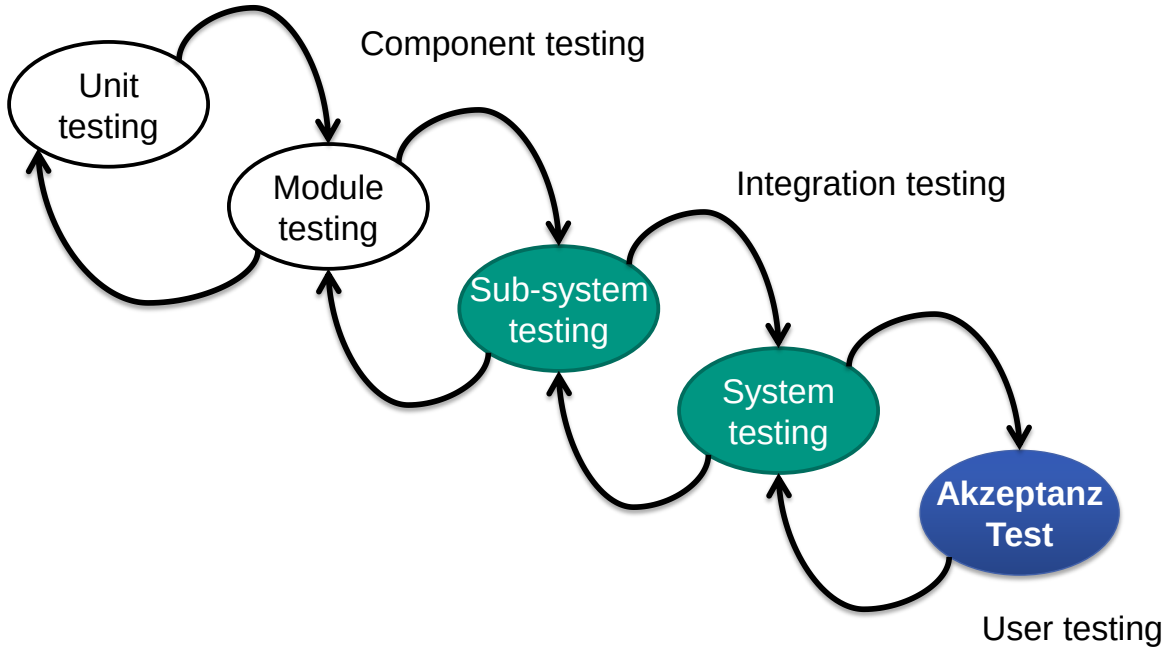
Die Sicherstellung, dass der Kunde nicht nur ein Produkt, das funktioniert, bekommt, sondern das er auch nach seinen Anforderungen gebrauchen kann, nennt man Validierung.

Bei Produkten, bei denen es um für den Benutzer zu bedienende Oberflächen geht, werden häufig zunächst User Interface Dummies erstellt, an denen der Kunde anmerken kann, was er sich konkret anders vorstellt.

Rapid Prototyping stellt eine Technik dar, bei der zunächst ein Teil der Qualitätssicherungsstandards keine Anwendung findet, um dem Kunden in kurzer Zeit einen Prototypen demonstrieren zu können, von dessen Akzeptanz ausgehend dann das endgültige Software-System entworfen wird.

Im Software Engineering wird unterschieden zwischen Validierung und Verifikation.

Software Test – Prozess



Der Testprozess fängt damit an, dass man aus dem Pflichtenheft heraus Test Cases definiert: Routinen, um Funktionen, Module, Kommunikation etc. zu testen.

Dabei werden sowohl elementare Tests für einzelne kritische Programmfunktionen geschrieben, als auch Anforderungen, die halbautomatisch oder manuell überprüft werden müssen.

Das Automatisieren von Tests hat den Vorteil, dass der Entwickler mit nur einem Aufruf überprüfen kann, ob das von ihm veränderte Programm spezifikationsgemäß funktioniert (Regressionstests).

Dabei kann sich der Entwickler zunutze machen, dass er weiß, wie das Programm intern funktioniert. Er wird also Tests schreiben, die nach Möglichkeit alle hypothetisch möglichen Abläufe austesten (White Box Tests). Interessant ist auch die Reaktion des Systems auf Grenzfälle oder ungültige Eingaben.

Während das Testen einzelner Funktionseinheiten sich mittels sog. Unit-Tests noch sehr gut automatisieren lässt, und leistungsfähige Tools bereitstehen, um dem Programmierer das Erkennen von Grenzfällen, Standardwerten, oder unsinniger Werte abzunehmen, ist mit zunehmender Komplexität des zu testenden Subsystems häufig nur noch ein manuelles Definieren von Tests möglich. Da das System hinterher als Ganzes abgenommen werden muss, wird es auch als solches getestet.

Im letzten Schritt wird der Kunde mit einbezogen, und so die Chance gegeben, vor Abgabe des Produkts noch eine Bewertung einfließen zu lassen.

Bei jeder Teststufe können aufgetretene Unstimmigkeiten dazu führen, dass Tests in den tiefer liegenden Systemschichten verändert bzw. ergänzt werden, um auch die Ursache von Fehlfunktionen im übergeordneten Zusammenspiel ausfindig zu machen.

Software Test - Begriffe

- *Unit Testing*
  - *Einzelne Komponenten*
- *Module Testing*
  - Komponenten-Zusammenspiel
- *Sub-system Testing*
  - *Stimmigkeit von Schnittstellen*
- *System Testing*
  - *Gesamtsystem wird untersucht*

Unit Testing:

Einzelne Komponenten werden isoliert auf spezifikationsgerechte Funktionsweise geprüft (z.B. einzelne Funktionen).

Module Testing;

Verbundene Komponenten werden in ihrem internen Zusammenspiel geprüft (z.B. Klassen).

Sub-system Testing:

Verknüpfte Module werden auf Interoperabilität und Funktionalität geprüft. Hierbei treten häufig Probleme wegen nicht übereinstimmender Schnittstellen zutage.

System Testing:

Subsysteme werden integriert und machen nun das Gesamtsystem aus. Es geht darum, Fehler im Zusammenspiel zwischen (abstraktem) Gesamtsystem und seinen Subsystemen zu finden;

außerdem wird validiert, ob das System die Anforderungen des Kunden erfüllt.

Akzeptanztest

Abschließende Phase:

Wird das Produkt abgenommen?

Alpha Test

- **Daten aus dem reellen Einsatzfeld**

Beta Test

- **Lieferung an potentielle Kunden**
- **Es gibt keinen Fehler, der so absurd ist, dass er nicht gemacht würde**

Akzeptanztest:

Abschließende Phase der Tests, bevor das System in Produktiveinsatz geht.

Alpha Test:

Das System wird mit echten Daten aus dem Kundenumfeld getestet, anstatt mit simulierten Daten.

Deckt häufig Fehler und Unzulänglichkeiten der spezifizierten Anforderungen auf, weil beispielsweise das System sich mit echten Daten anders als erwartet verhält.

Ziel: Das System erfüllt die Anforderungen des Kunden und leistet, was es soll.

Beta Test:

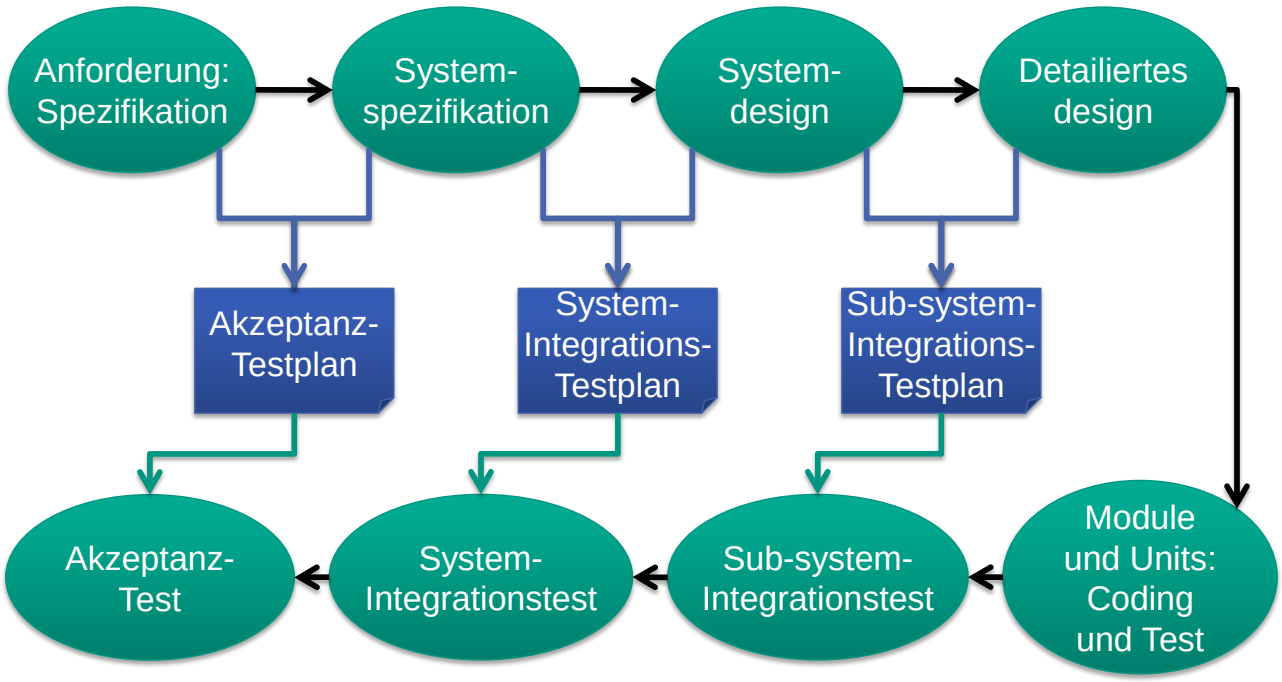
Auslieferung des Systems an einen Kreis potentieller Kunden, die in eine Nutzung des Testsystems einwilligen (Friendly User).

Berichte der Nutzer an den Entwickler, die dafür meist kostenlosen Support leisten.

Stellt Benutzbarkeit unter Beweis und findet im besten Fall die Fehler, die den Entwicklern entgangen sind.

Ziel: Das System ist genau so, wie der Kunde es will!

Software Test – Test Planung



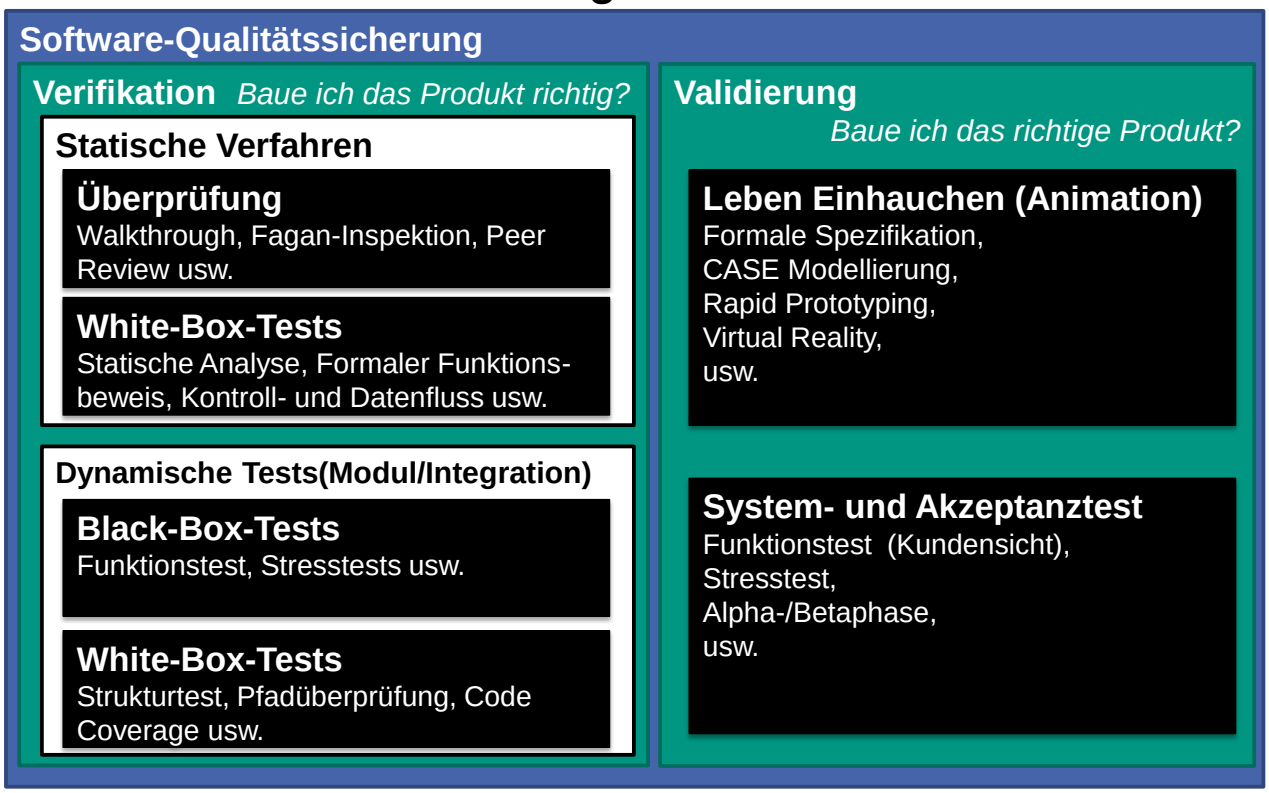
Test von Soft- und Hardware ist zeitaufwendig und teuer (für Echtzeit-Systeme mit umfangreichen Anforderungen 50 – 70% des Entwicklungsaufwands).

Daher sind nur effektive und effiziente Tests akzeptabel.

Da die Tests schon früh vorbereitet werden müssen, werden Testpläne erstellt, nach Möglichkeit sobald man entsprechende Anforderungen gefunden hat.

Positiver Nebeneffekt: Definiert man den Test, bevor man sich an die Implementierung gemacht hat, so reduziert man „Leichtsinnsfehler“, weil man weder in seiner Implementierung gefangen ist noch unbewusst einzelne Teile mit weniger Tests versieht.

Verifikation und Validierung



Fazit

- Softwareentwicklung ist ein vielschichtiger Prozess
- Nur ausgiebige Kommunikation mit dem Kunden sichert Erfolg
- Das Zerlegen der Anforderung gliedert das Problem hierarchisch
- Tests:
  - Planung von oben nach unten
  - Durchführung von unten nach oben