# Einführung in die Informatik

Klaus Knopper

(C) Jan. 2005 knopper@bw.fh-kl.de

# Software-Fehler

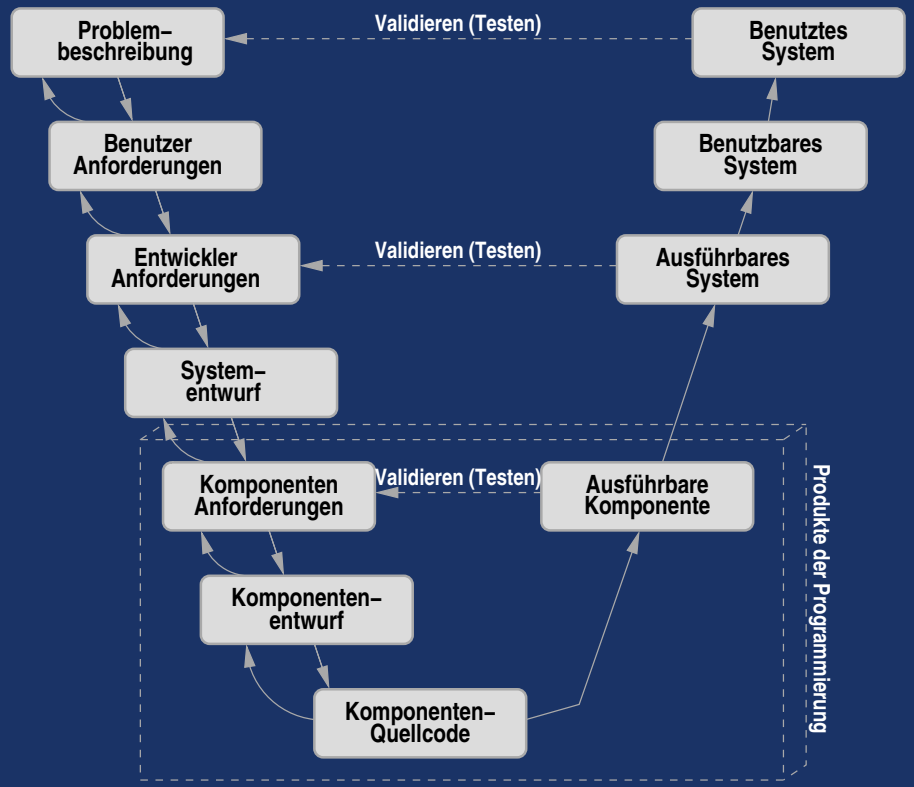
Verifizierung: Wurde die Software richtig gebaut?
Validierung: Wurde die richtige Software gebaut?

Die Validierung kann erst am (fast) fertigen Produkt anhand der Spezifikation durchgeführt werden.

- Die meisten Programme lassen sich (in endlicher Zeit) nicht mit mathematisch exakten Methoden auf Fehlerfreiheit prüfen.
- Die Kosten, um Fehler zu beheben, steigen von der Entwicklungsphase über die Implementation bis zur Anwendung hin stark an.

# Validieren und Verifizieren von Software

Das klassische Verfahren des Software-Engineering ist das sogenannte V-Modell, bei dem in jedem Schritt, wenn möglich, kontrolliert wird, ob das Design den Spezifikationen entspricht, und die Implementation korrekt arbeitet.



# Whitebox-Tests

Testen, ob alle in der Implementation vorhandenen Kontrollstrukturen und Anweisungen im Programm korrekt arbeiten.
Probleme:

- Nicht-erreichbare Anweisungen,
- Anzahl benötigter Testfälle, Eingabedaten.

# Blackbox-Tests

Testen, ob das Programm seine Spezifikation erfüllt, ohne die konkrete Implementierung zu berücksichtigen.
Probleme:

- Anzahl benötigter Testfälle, Rechenzeit,
- Sonderfälle können schlecht erkannt werden, wenn die Implementierung nicht bekannt ist.

# Arbeiten im Team

Problem: Eine umfangreiche Software soll/muss von mehreren Personen im Team entwickelt werden.

Lösungsansatz: Entwicklung in Teilprojekte zerlegen (Module, Objektorientierung)

Neues Problem: Wie koordiniert man die Programmierer, und sorgt dafür, dass es kein „Versions-Chaos“ bei den Quelltexten gibt?

# Software-Design, Spezifikationen, APIs

Mit Hilfe der bereits angesprochenen Hilfsmittel des Software-Engineering (OOA, OOD, Design Patterns, UML) wird das Problem zunächst auf abstrakter Ebene analysiert, dann eine Spezifikation entworfen, und zur Implementierung eine API (Application Programming Interface) festgelegt, an die sich alle Programmierer halten müssen, wobei jeder an der API für seinen Teilbereich mitarbeitet.

Ohne eine genaue Spezifikation und den Programmierern bekannte API ist es kaum möglich, komplexe Software zu schreiben.

# Quellcode-Verwaltung

Um die Arbeit während der Implementationsphase zu erleichtern, sind Systeme entwickelt worden, die verschiedene Versionen von Quelltextdateien verwalten, und Änderungen dokumentieren helfen. Dies sind beispielsweise RCS (Revision Control System für Einzelrechner) und CVS (Concurrent Versions System, ermöglichst es, dass Entwickler das komplette Projekt von einem Server auschecken, und ihre Änderungen wieder einpflegen können, ohne sich gegenseitig zu behindern).
Die meisten Open Source Projekte stellen einen CVS-Server für ihre aktuellen Quelltexte und Dokumentationen im Internet zur Verfügung, mit dem man auch die Fortschritte gut verfolgen kann.

# Entwicklungsumgebungen

Entwicklungsumgebungen integrieren einen Editor mit *Syntax-Highlighting*, Dokumentations/API-*Browser*, *Debugger* und optimalerweise auch eine netzwerkfähige *Versionsverwaltung* für das Arbeiten im Team unter einer graphischen Oberfläche.

Obwohl man auch mit den einzelnen Komponenten sehr gut entwickeln kann, erleichtern solche integrierten Umgebungen mit entsprechenden Plugins die Entwicklungsarbeit im Team oft sehr. Ein Beispiel hierfür ist die in Java geschriebene Entwicklungsumgebung **`eclipse`**, die von mehreren großen Softwarefirmen gemeinsam entwickelt, und unter einer Open Source-Lizenz vertrieben wird.