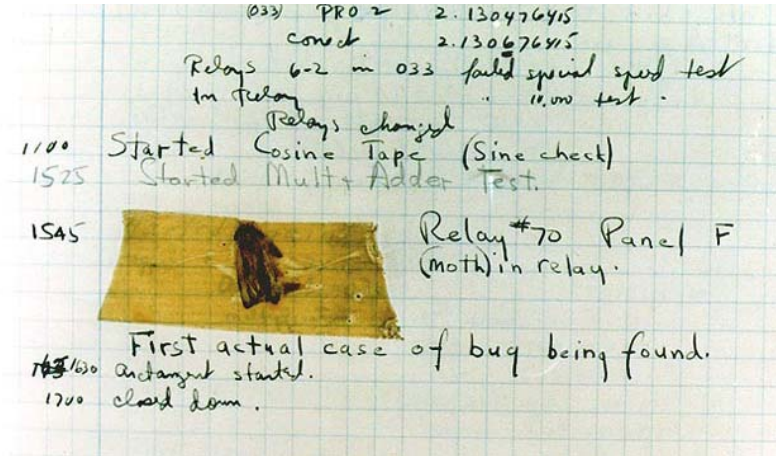
Fehler



Keine Software ist fehlerfrei!

Dies ist eine Tatsache – aber keine Ausrede, und dies schützt Sie auch in keiner Weise und nimmt Ihnen nicht die Verantwortung für Ihr Produkt ab!

Wo kann die Ursache für ein fehlerhaftes Ablaufen eines Programms liegen?

- Hardware (z.B. im Prozessor)
- Betriebssystem
- Entwicklungsumgebung (Compiler, Linker, Bibliotheken)
- Quelltext

Schwerpunktmäßig werden wir uns nun mit Programmierfehlern, also Fehlern im Quelltext, beschäftigen.

Um Sie, die Programmierer, vor einer fatalistischen Einstellung zu bewahren („Fehler gibt‘s überall, das ist nun mal so, da kann man nichts machen“) und ein sorgfältiges Arbeiten zu motivieren, wollen wir uns zunächst einige der bekanntesten Softwarefehler mit fatalen Folgen ansehen:

1982

Im Falklandkrieg 1982 wurde der britische Zerstörer Sheffield mit Exocet-Raketen versenkt, wobei 20 Menschen ertranken. Das Abwehrsystem funktionierte nicht, da nicht vorgesehen war, dass Raketen eigenen Typs gegen eigene Schiffe anfliegen.

1983

Im Jahre 1983 übten Jagdbomber vom Typ F18 mit neuer Bordsoftware. Aufgrund eines Vorzeichenfehlers drehten sie sich nach dem Überfliegen des Äquators auf den Kopf.

1999

Der Mars Climate Orbiter zerschellte 1999 auf dem Mars, statt in eine Umlaufbahn einzutreten. Grund war eine fehlerhafte Programmierung verschiedener Entwicklungsteams an verschiedenen Orten innerhalb der USA. Man vermischte Maßeinheiten (Meter – Yard; kg – Pfund etc.). Dadurch wurde ein Winkel beim Anflug zum Mars falsch berechnet. Problem war die mangelhafte Kommunikation zwischen den Entwicklungsteams und die unvollständige Spezifikation der Software.

[Quelle: http://www.iwi.uni-hannover.de/cms/images/stories/upload/lv/wisem0809/SQM/SWG.pdf]

„Kalter Kaffee“? Von wegen…

1. Strafe und Rückzahlungen kosten Finanzkonzern 240 Millionen Dollar
25 Millionen Dollar Strafe verhängte die US-Finanzaufsichtsbehörde Securities and Exchange Commission (SEC) einem internationalen Finanzdienstleister. Dieser hatte zuvor einen Fehler in einer Softwareanwendung eines Investmentfonds vertuscht. Gleichzeitig musste das Unternehmen den geprellten Anlegern den entstandenen Schaden zurückerstatten: 217 Millionen Dollar (über 160 Millionen Euro).

2. Hunderttausende Gehaltszahlungen verzögert
Landesweit fielen in Japan durch einen Softwarefehler bei einer der größten Banken rund 5.600 Geldautomaten für 24 Stunden aus. Um die Systemwiederherstellung zu beschleunigen, mussten alle 38.000 Geldautomaten vom Netz genommen werden. Mehrere Tage lang war kein Online-Banking möglich. Erst nach einer zehntägigen Verzögerung konnte die Bank alle Lohnüberweisungen mit einem Gesamtvolumen von 1,5 Milliarden Dollar (über eine Milliarde Euro) bearbeiten.

3. Bankautomaten verschenken Geld an Kunden
Im australischen Sydney, Melbourne und Brisbane konnten Kunden über fünfeinhalb Stunden uneingeschränkt Geld an 40 defekten Geldautomaten abheben. Möglich machte den überraschenden Geldsegen eine Störung in der Datenbanksoftware, die die Automaten in den Stand-by-Modus versetzte: Die Geräte erkannten weder die Grenze des Tageslimits, noch wussten sie, ob das Konto genügend gedeckt war.

[Quelle: http://www.channelpartner.de/smb/2576871/] … 2011!

Aber auch bei vergleichsweise „harmlosen“ Programmen können Fehler weitreichende Konsequenzen haben!

Ein Beispiel: Sie erstellen Software zur Berechnung der Volumina von Deponien. Ein Fuhrunternehmen kauft Ihr Programm, berechnet damit das Volumen einer Bauschuttdeponie, kalkuliert so den Aufwand für Abtragung / Abfuhr / Entsorgung, bewirbt sich damit auf eine Ausschreibung und gewinnt den Auftrag.

Nun stellt sich heraus, dass berechnete Volumen nur etwa halb so groß ist wie das tatsächliche. Das Fuhrunternehmen muss aber den Auftrag erfüllen und geht dabei in Konkurs.

Wer haftet für den Schaden?

Das oberste Gebot bei der Softwareentwicklung:
Arbeiten Sie planvoll und sorgfältig!

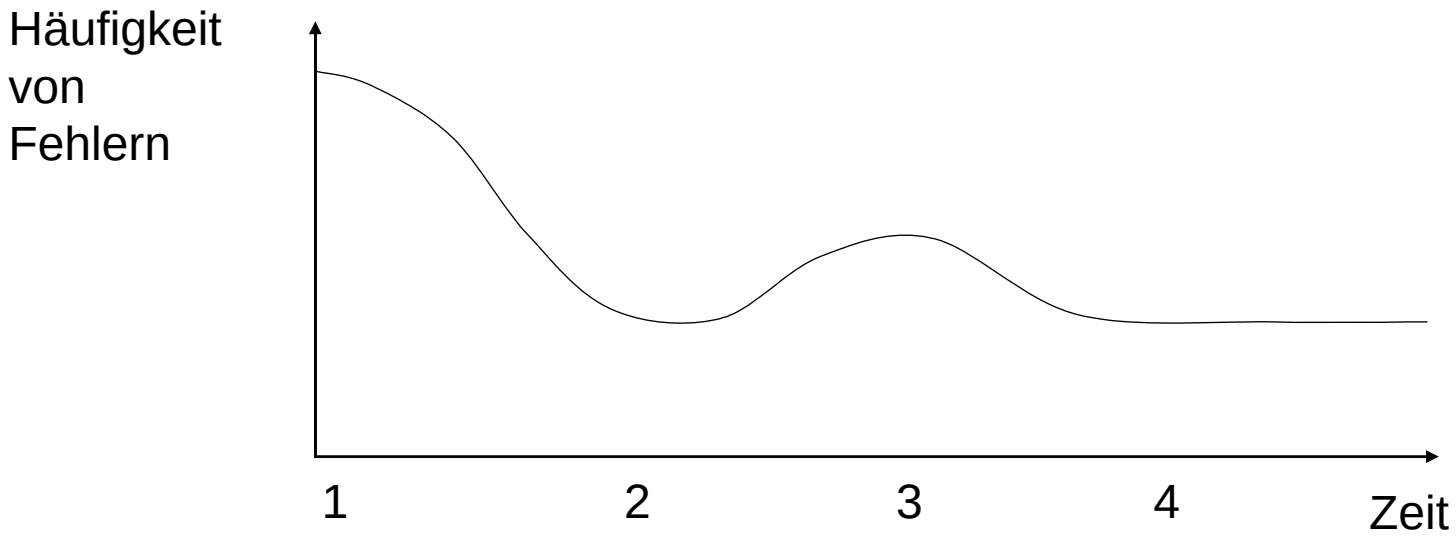
C++ Softwareentwickler/-in

- Sorgfältiges Design statt "Quick & Dirty"
- Teamarbeit statt Einzelkämpfertum
- Dauerhafte Produktentwicklung statt eilige Projektarbeit

Wir glauben, dass gute Software auf diesem Weg entwickelt wird. (…) ist ein junges, innovatives High-Tech Unternehmen mit ca. 120 Mitarbeitern. Wir entwickeln optische 3D-Messsysteme und mess-technische Software für industrielle Anwendungen. Neben den großen internationalen Automobil-, Flugzeug- und Konsumgüter-herstellern sind auch viele klein- und mittelständische Firmen unsere Kunden. Für die Aufstockung unserer ca. 25-köpfigen Softwareabteilung suchen wir …

(Aus einer Stellenanzeige, Frühjahr 2009)

Typische „Fehlerkurve“ von Programmierern



1: Anfänger, unsicher, viele Fehler
2: Zunehmende Sicherheit, weniger Fehler
3: Zunehmende Schlampigkeit, mehr Fehler
4: Reifestadium

Einige Arten von / Beispiele für Programmierfehler:

- Falscher bzw. ungeeigneter Algorithmus
- Ungeeigneter Variablentyp (z.B. integer statt real)
- Unzulässige Eingaben nicht abgefangen (z.B. negative Zahlen bei sqrt)
- Ausnahmen nicht abgefangen (z.B. Null im Nenner)
- Fehlerhafte Schnittstellen zu Unterprogrammen
- Fehlende Synchronisation bei Parallelprozessen
- Speicherüberlauf bei Feldern
- Tippfehler im Quelltext (z.B. falsches Vorzeichen, falscher Exponent)

Das Tückische an diesen Fehlern: Sie können nicht vom Compiler erkannt werden und wirken sich u.U. auch nur sehr selten aus. Aber – es gibt sie! Eine alte Faustregel besagt, dass bei komplexeren Programmen je 1000 Zeilen Quelltext ein bis drei Fehler versteckt sind.

Weitere Fehler: Das Programm ist viel zu langsam, die Bedienung ist nicht oder kaum intuitiv möglich, Online-Hilfe fehlt oder ist unvollständig, …

Wie kann man Fehler möglichst vermeiden?

Vor allem bei komplexeren Programmen wirken sich selbst „kleinere“ Änderungen oft an allen möglichen Stellen aus. Deshalb gilt folgender Grundsatz:

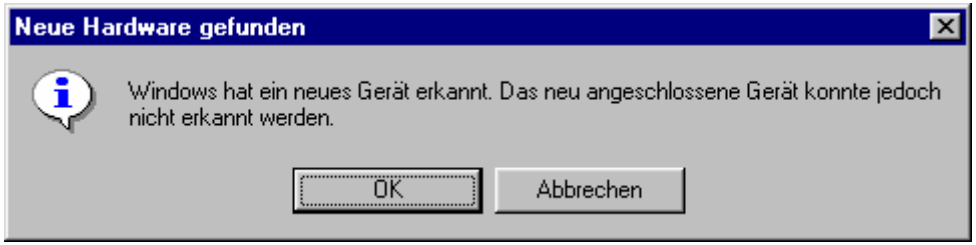
„Mal eben“ (irgend etwas ändern, erweitern usw.) ist tödlich!

Beispiel: Da wird *mal eben* in einem Unterprogramm eine Erweiterung vorgenommen, die zu einem zusätzlichen Parameter in der Schnittstelle führt. Da aber dieses Unterprogramm an allen möglichen Stellen, insgesamt etwa 20 mal, aufgerufen wird, vergisst man garantiert irgendwo, die Schnittstelle anzupassen!

Gegenmittel u.a.: Eine exakte Dokumentation aller Programmteile.

Selbst bei sorgfältiger Programmierung kann es Fälle geben, wo das Programm nicht korrekt arbeiten kann, insbesondere bei fehlerhaften Eingabedaten.

Wichtig: Das Programm darf niemals „abstürzen“ oder „sich aufhängen“! Treten Fehler auf, muss eine sinnvolle (für den Nutzer verständliche) Fehlermeldung ausgegeben werden, z.B. „Fehler beim Lesen der Eingabedatei – bitte Format kontrollieren“.



[Quelle: http://www.gregor-jonas.de/witze/bugs.htm]

Ein Kunde meldet einen Fehler…

Geben Sie dem Kunden das Gefühl, dass Sie ihn und das Problem ernst nehmen! Sorgen Sie umgehend für eine Lösung.

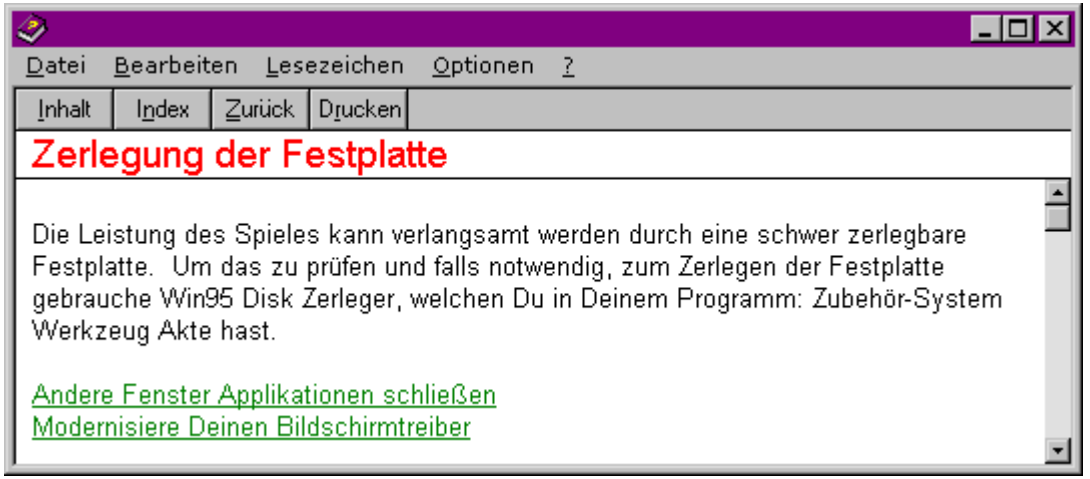
Auch dem Kunden ist bekannt, dass kein Programm fehlerfrei ist. Wenn er aber weiß, dass Sie sich umgehend um eine Lösung kümmern, wird er ein zufriedener Kunde bleiben!

Ein unverzichtbares Mittel, um die Anzahl von Fehlern zu minimieren bzw. Fehler aufzuspüren, sind Tests.

Und im Notfall:

[Quelle wie vorige Seite]

(vermutlich gemeint: Die Festplatte sollte mal defragmentiert werden…)



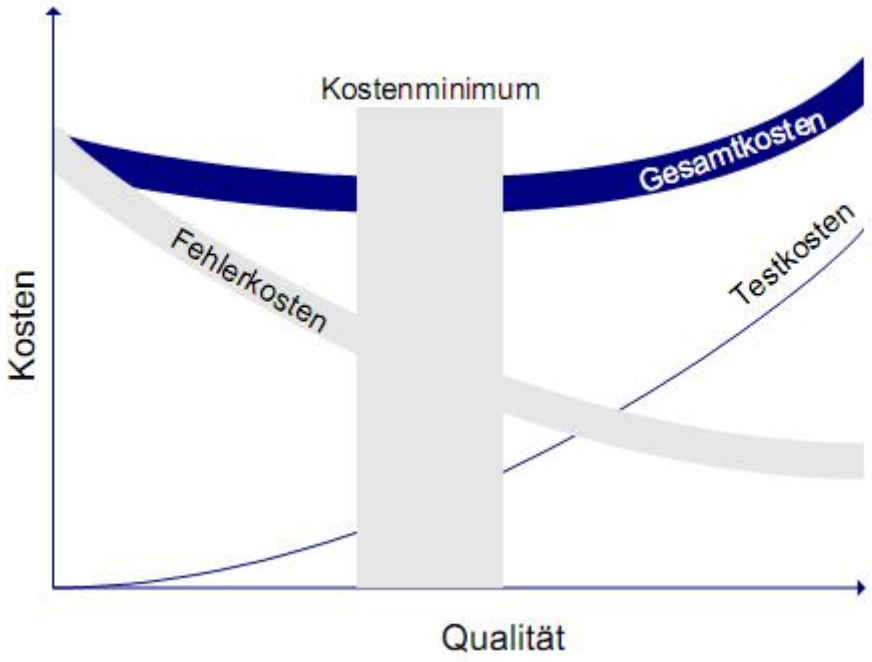
Softwaretests

Vorbemerkungen

- Tests können Fehler finden, aber nicht die Fehlerlosigkeit eines Programms beweisen! Ein fehlerfrei durchlaufener Test bedeutet nicht, dass das Programm fehlerfrei ist!

- Soweit möglich, sollten Tests nicht vom Programmentwickler selbst, sondern von einer unabhängigen Person durchgeführt werden (Stichwort Betriebsblindheit). Hierzu müssen wir jedoch einen Blick auf die einzelnen Teststufen werfen.

- Tests kosten ebenso Zeit und Geld wie die Programmentwicklung. Es ist daher ein sinnvolles / ausgewogenes Verhältnis von Aufwand und Ertrag anzustreben.



Fehlerkosten versus Testkosten
[Quelle: Skript der TU Wien, link s.u.]

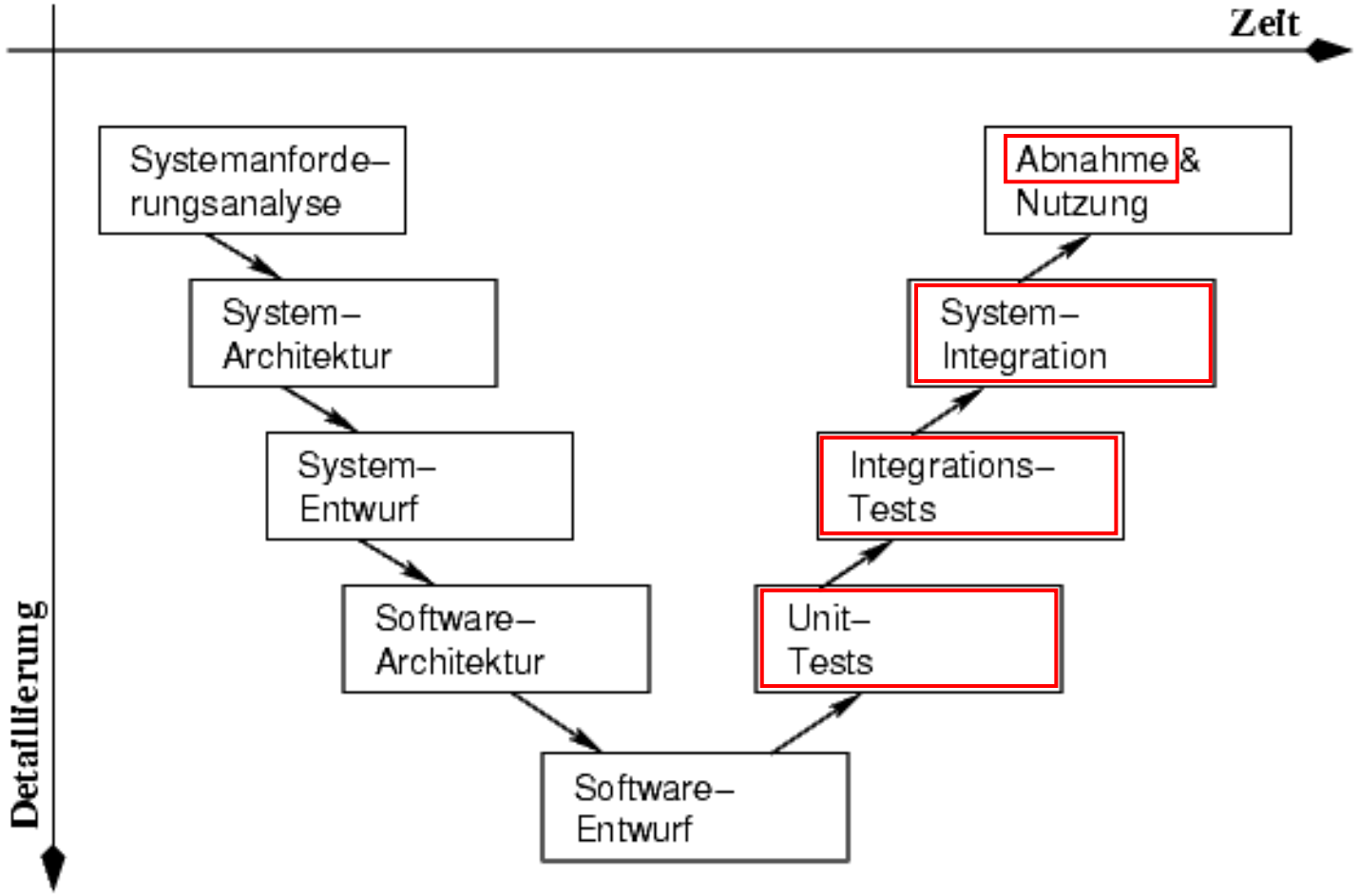
Teststufen

Komponententest (Modultest, Unit-Test)
… wird auf der untersten Ebene durchgeführt. Gestestet werden die einzelnen Komponenten (Unterprogramme / Module / Units / Methoden), hier meist noch vom Entwickler selbst.

Integrationstest
… testet die Zusammenarbeit von unabhängigen Modulen. In dieser Stufe kommen z.B. die Schnittstellen ins Spiel.

Systemtest
… testet das gesamte Programm, beispielsweise in Bezug auf das Pflichtenheft. Wird wie die vorigen Tests vom Hersteller durchgeführt, z.B. in einer eigenen Testabteilung („Warenausgangskontrolle“)

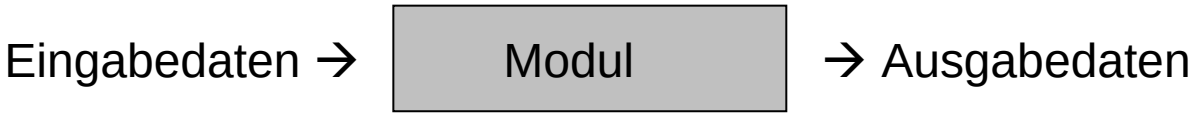
Abnahmetest
… durch den Kunden, mit beliebigen realen Daten. Die Software wird hierbei als „black box“ betrachtet.



Beispiel: Tests im „V-Modell“

Box-Modelle

black box (Funktionstest): Erfolgt ohne Kenntnis über den inneren Aufbau bzw. den Quelltext des entsprechenden Moduls.

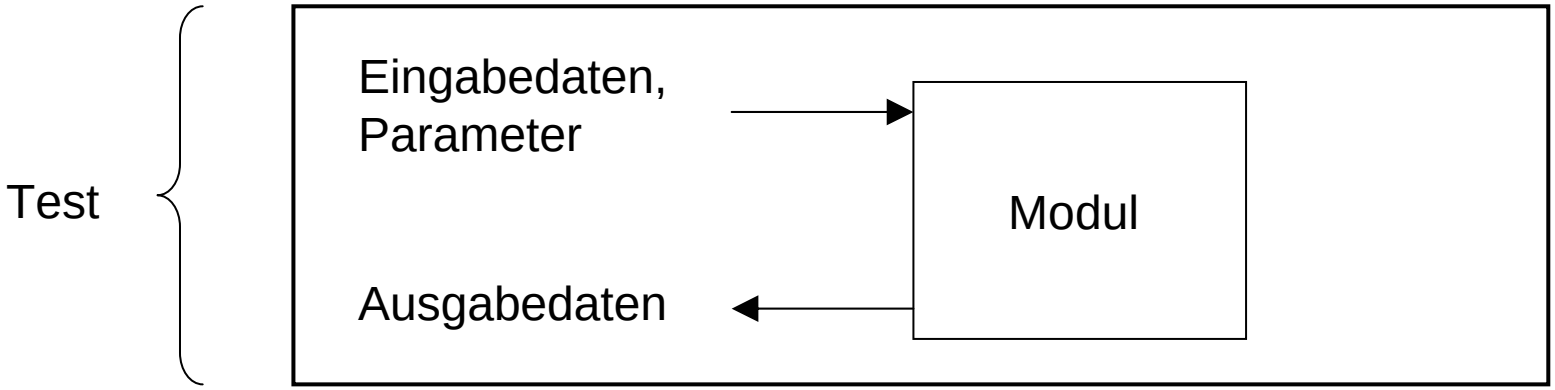


white box (Strukturtest): Wird vom Programmierer durchgeführt unter genauer Kenntnis des Aufbaus bzw. des Quelltextes.

grey box: Test werden vom Programmierer entwickelt vor Fertigstellung des zu testenden Moduls → testgetriebene Entwicklung.

Zum Weiterlesen:
http://fbim.fh-regensburg.de/~hab39652/SE/Kapitel_9_Test_01_Motivation.pdf
http://qse.ifs.tuwien.ac.at/courses/skriptum/download/06_testen_wid_20031017.pdf

Testgetriebene Entwicklung („test-driven development“, TDD)



1) Entwicklung des Tests
2) Entwicklung des Moduls
3) Austesten des Moduls, bis dieses fehlerfrei läuft
4) Optimieren des Codes („refactoring“), wiederum testen

Eine solche Vorgehensweise ist u.U. interessant auf der Stufe der Komponententests (unit tests) und ein Beispiel für das grey-box-Modell: Der Tester kennt zwar den Quelltext, testet aber das Modul „wie es ist“ ausschließlich auf seine Funktionalität.

Beispiel:

Berechnung des größten gemeinsamen Teilers von i und j nach dem Verfahren von Euklid.

Das Modul *ggt* wird programmiert und fortan als black box betrachtet.

Vorab (!) wurde der folgende Test geschrieben:

```
integer function ggt(i,j)

integer*4   i,j,a,b,c,rest

a = abs(max(i,j))
b = abs(min(i,j))
if (b == 0) then
  ggt = -99
  return
end if

do while (rest /= 0)
  c = a/b
  rest = a-c*b
  ggt  = b
  a = b
  b = rest
end do

end
```

Division durch Null abfangen

Test für das Modul *ggt*:

```
program test

integer*4   i,j,ggt,ergebnis

read(5,*)i,j
ergebnis = ggt(i,j)
print*,'ggt = ',ergebnis

end
```

Bemerkung:

Solange ausschließlich ganze Zahlen, beide ungleich Null, eingegeben werden, wird *ggt* erwartungsgemäß funktionieren! Bei Eingabe einer oder zweier Nullen würde das Programm abstürzen. Dieser Fall ist in *ggt* bereits abfangen.

Ebenfalls sicherstellen: Eingabewerte nur ganze Zahlen.

Option: Eingabe der Zahlen nicht von Hand, sondern per Zufallsgenerator (vgl. auch unser Beispiel zur Matrix-Invertierung).