

Softwarequalität

Qualitätsbegriff



Was ist Qualität?

Qualitätsbegriff



In der Norm ISO 8402 (Quality management and quality assurance) heißt es:
„Qualität ist die Gesamtheit von Merkmalen einer Einheit bezüglich ihrer Eignung, festgelegte und **vorausgesetzte** Erfordernisse zu erfüllen.“

Qualität aus Anwendersicht (**anwenderbezogene Ansatz**)

- Das Produkt tut genau das, was man von ihm **intuitiv** erwartet
- Das Produkt **fühlt sich gut an**
- Bei der Benutzung stellt sich von selbst **Zufriedenheit** ein

Qualität aus Herstellersicht:

- Das Produkt erwirtschaftet Gewinn und erzeugt innerbetrieblich **keinen Stress**
- hohe Lebensdauer bzw. langer Produktzyklus
- erzeugt bei Mehrheit der Beteiligten am Produktionsprozess **Zufriedenheit** und evtl. sogar **Stolz**
- keine oder sehr geringe Ausschuss- und Nachbearbeitungskosten



Qualitätskosten

Fehlerverhütungskosten: Kosten der Qualitätsplanung, -lenkung, -management, -audits usw., Tests

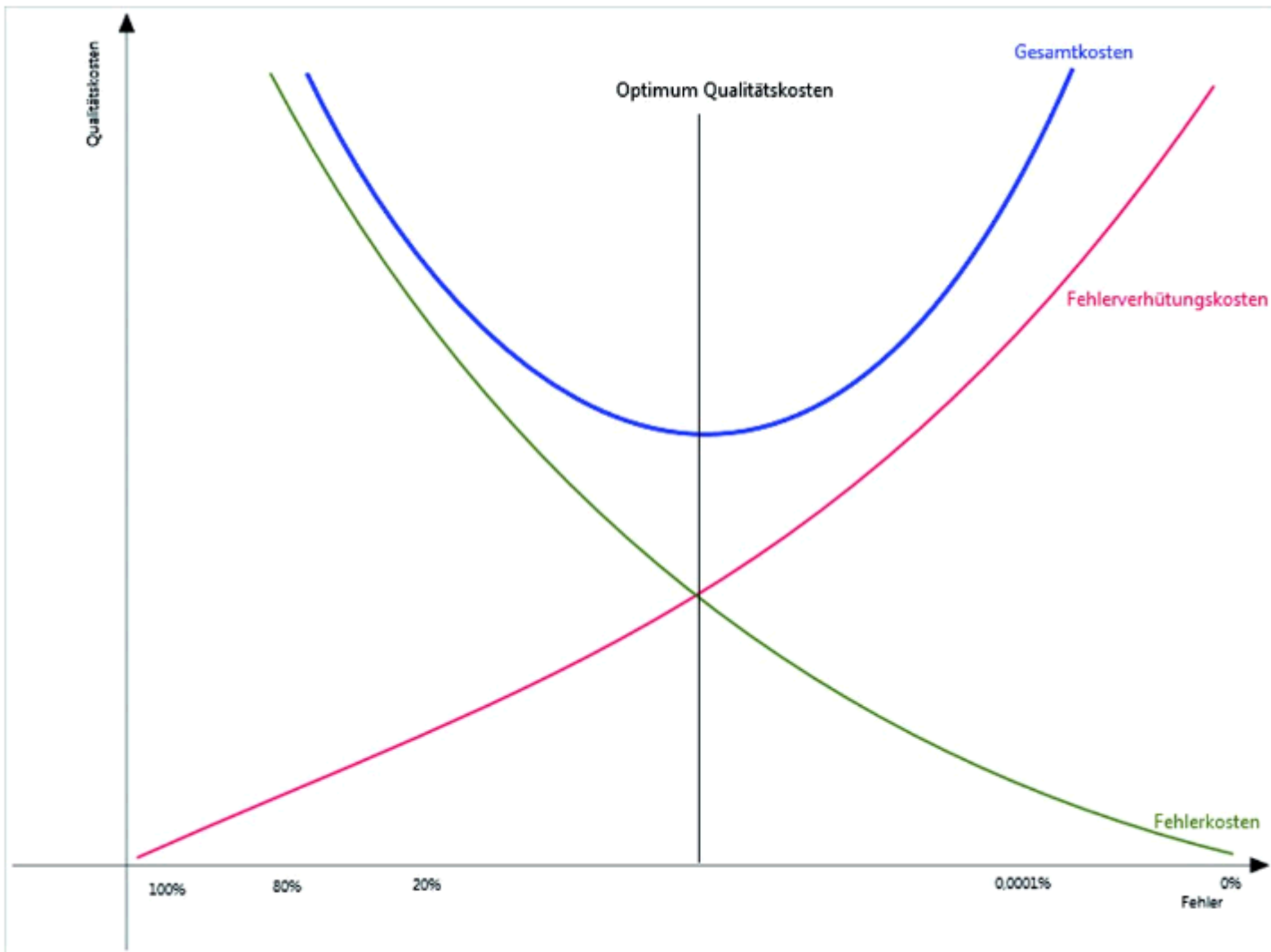


Fehlerkosten: durch Nacharbeit, kostenlose Garantieleistung, Rückrufaktionen, Vertragsstrafen, Auftragsverlust an Konkurrenz, Imageschaden

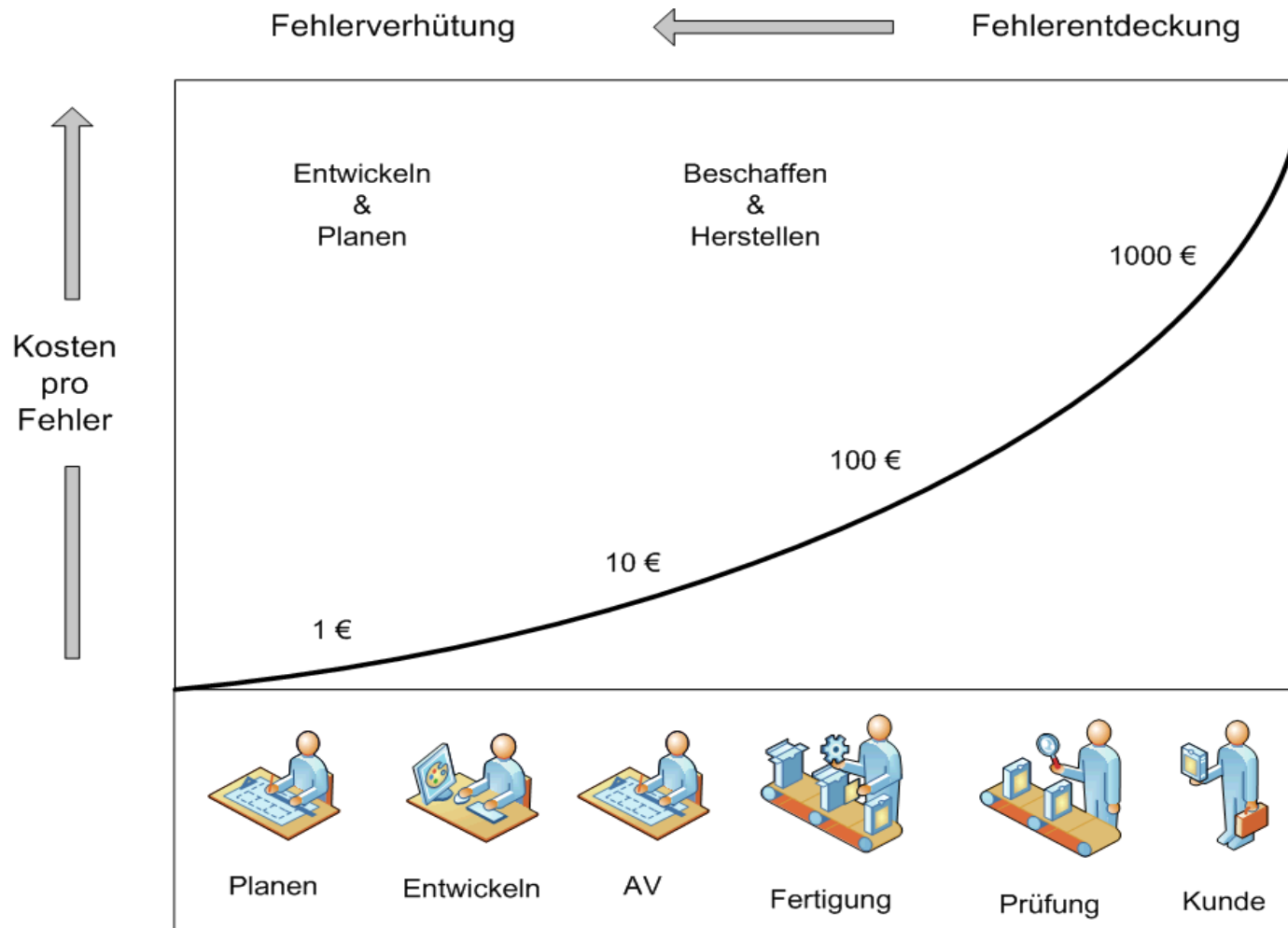


Die **gesamten Qualitätskosten** bewegen sich erfahrungsgemäß zwischen zehn und dreißig Prozent des Umsatzes (variabel je nach Projekt) und können damit relativ schnell den Gewinn aus dem Projekt aufzehren.

Qualitätskosten



rule of ten



Vergleich Branchen

Lebensmittel- Verarbeitung

seit 9000 Jahren

„Beste Vorgehensweise“
aus Erfahrung

geprüftes Personal
(Bäckermeister)

formale Methoden
(Rezept)

einfache
Aufgabenformulierung

einfache Bewertbarkeit
objektiv
(Sinnesorgane)

Bauwesen

seit 4000 Jahren

„Beste Vorgehensweise“
aus Erfahrung

geprüftes Personal
(Architekt/Polier/
Bauingenieur)

formale Methoden
(Entwürfe und Pläne)

mittlere
Aufgabenformulierung

mittlere objektive
Bewertbarkeit (Wasserwaage, Laser)

Softwareentwicklung

seit 50 Jahren

momentan noch keine
„Beste Vorgehensweise“

ungeprüftes Personal
(ab wann ist man ein
Programmierer?)

viele konkurrierende
formale Methoden

komplexe
Aufgabenformulierung

schwierige Bewertbarkeit
meist nur subjektiv

Qualität



Zufriedenheit

(rationale Ebene)

Qualität
+
Mehrwert



Begeisterung

(emotionale Ebene)

Softwarefehler und Folgen

92

9/9

0800 Antan started
 1000 " stopped - antan ✓
 1300 (032) MP-MC ~~2.130476415~~ { 1.2700 9.032 847 025
 (033) PRO-2 2.130476415 9.032 846 795 correct
 correct 2.130476415 4.615925059(-2)

Relays 6-2 in 033 failed special speed test
 in relay 10,000 test.

Relay
 2145
 Relay 3370

1100 Relays changed
 Started Cosine Tape (Sine check)
 1525 Started Multi-Adder Test.

1545



Relay #70 Panel F
 (moth) in relay.

First actual case of bug being found.

1650 Antan started.
 1700 closed down.

Abbildung 1. Der „erste Computerfehler“ 1945 [1]

Ariane 5 Jungfernflug — 4.6.1996

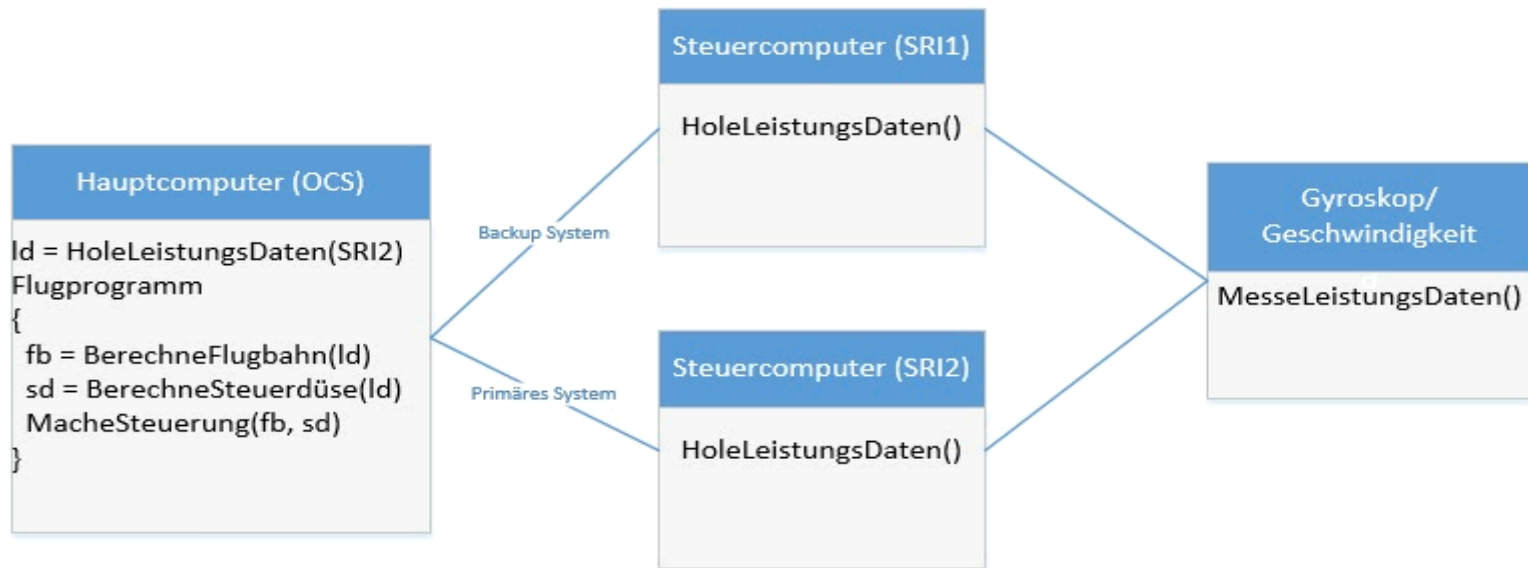
Wegen eines Überlaufs bei einer Zahlkonvertierung im Lageregelungsmodul geriet die europäische Rakete kurz nach dem Start in eine Schräglage und musste gesprengt werden. Durch die Explosion entstand ein Schaden von 1,7 Milliarden DM. Verzögerung des gesamten Raumfahrtprogramms um 3 Jahre.

Ada-Programm des Trägheits-Navigationssystems (Ausschnitt):

```
...
declare
  vertical_veloc_sensor: float;
  horizontal_veloc_sensor: float;
  vertical_veloc_bias: integer;
  horizontal_veloc_bias: integer;
  ...
begin
  declare
    pragma suppress(numeric_error, horizontal_veloc_bias);
  begin
    sensor_get(vertical_veloc_sensor);
    sensor_get(horizontal_veloc_sensor);
    vertical_veloc_bias := integer(vertical_veloc_sensor);
    horizontal_veloc_bias := integer(horizontal_veloc_sensor);
    ...
  exception
    when numeric_error => calculate_vertical_veloc();|
    when others => use_irs1();
  end;
end irs2;
```



Quelle: Mathias Riedl Seminar 2012 „ARIANE 5 Absturz des Flugs 501“



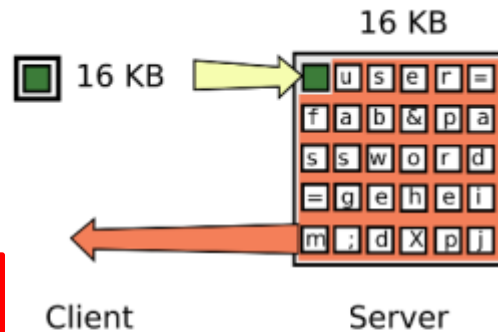
- Ziel: Hardware-Auslastung der SRI-Rechner(Trägheitsnavigationssystem) unter 80%
- Analyse Code im SRI: Gefahr von numerischen Überlauf bei 7 Variablen
- Entscheidung:
 - 4 Variablen wurden im Code gegen Überlauf geschützt
 - 3 Variablen wurden nicht geschützt da Überlauf nicht erwartet (Vertraglich festgehalten)
- Problem: Vergleichsdaten zur Beurteilung des Überlaufverhaltens von Ariane 4
- Konvertierung eines 64Bit Gleitkommazahl in eine der 3 ungeschützten 16Bit Integer-Variablen
- Wert größer als 65535(16Bit) daraufhin folgte ein Operandenfehler in der Lageberechnung – Fehlerbit abgesetzt - Shutdown des SRI
- Hauptcomputer versuchte auf BACKUP-SRI umzuschalten, das war vorher aufgrund des gleichen Fehlers ausgefallen

Open SSL-Heartbleed-Exploit

Februar 2014

Der Angriff

Konkret funktioniert ein Ausnutzen der Lücke so: Der Angreifer schickt dem Server eine Heartbeat-Payload von einem Byte Größe, behauptet aber, sie sei beispielsweise 16 KByte groß. Der Server schreibt das Byte des Angreifers in seinen Speicher in einen Puffer namens `pl`. Da die eigentliche Größe der Payload nicht verglichen wird, geht der Server beim Zurücksenden der Payload von der vom Angreifer angegebenen Größe aus (`payload`). Er reserviert also 16 KByte Speicher (plus ein bisschen Platz für Verwaltungsinformationen)



Wenn der Client über die Länge lügt, füllt der Server das Antwortpaket großzügig mit eigenen Daten auf. ☹

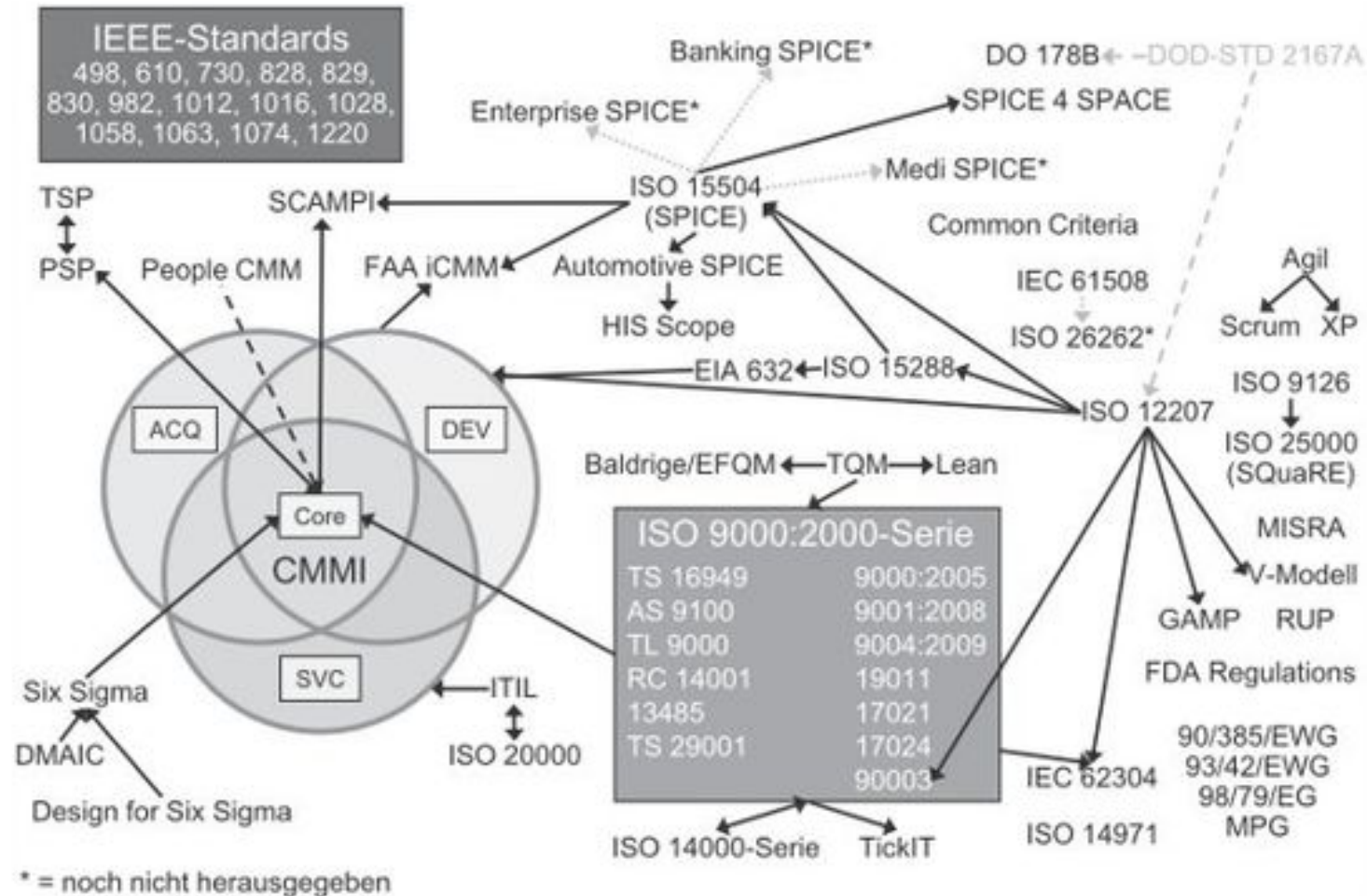
```
buffer = OPENSSL_malloc(1 + 2 + payload + padding);  
bp = buffer;
```

und kopiert dann für die Antwort die vom Angreifer angegebene Payload-Größe (mehrere KByte) an diese Stelle:

```
memcpy(bp, pl, payload);
```

Normen, Standards, Institutionen

Bild 1 gibt einen Überblick über viele – aber nicht alle – für die Software-Entwicklung im Jahr 2011 relevanten Best-Practice-Modelle und Standards. Bild 1 ist aus einem europäischen Blickwinkel erstellt und enthält nur wenige militärische Normen.



- Quelle: <https://www.qz-online.de/qualitaets-management/qm-basics/software-qualitaet/software-testnormen/artikel/software-standards-normen-und-modelle-257987.html>

ISO (Internationale Organisation für Normung)

162 Länder sind Mitglied der ISO. Jedes Land ist mit seinem lokalen staatlichen Normungsgremium mit einem Sitz vertreten

(DIN für Deutschland seit 1951) bekannteste Norm: ISO 9001



ANSI (American Standard Institute)

staatliche Normungsinstitution der USA, repräsentiert 125,000

US-Firmen, vertritt als FullBody Member die USA in der ISO.

bekannteste Norm: C89/C90 ANSI-C



IEEE (Institute of Electrical and Electronics Engineers)

weltweiter nichtstaatlicher Berufsverband von Ingenieuren

Hauptsächlich aus den Bereichen Elektrotechnik und Informationstechnik

430.000 Mitglieder, Gründung 1962, 38 Societies u.a. IEEE Computer Society und IEEE Robotics and Automation Society, bekanntester Standard IEEE802 LAN



Seit 2008 existiert ein PSTO (partner standards development organization)

zwischen ISO und IEEE zur Kooperation bei der Normengebung

IEC (International Electrotechnical Commission)

Gründung 1906 in London. In der IEC sind mehr als 70 Länder vertreten, organisiert in 93 technischen Kommissionen, 80 Unterkommissionen und rund 700 Arbeitsgruppen (Stand 2008)
Arbeitet zum Teil mit ISO zusammen (ISO/IEC 12207)



CENELEC (europäische Komitee für elektrotechnische Normung)
zuständig für die europäische Normung im Bereich Elektrotechnik
CENELEC-Mitglieder sind die nationalen elektrotechnischen Komitees



ETSI (europäische Institut für Telekommunikationsnormen)
gemeinnützige Organisation, 700 Mitglieder aus über 60 Ländern,
darunter Netzbetreiber, Verwaltungen, Anwender und Hersteller



CEN (Europäische Komitee für Normung)

33 Nationalen Mitglieder, drei Mitglieder der Europäischen Freihandelsvereinigung (EFTA), alle Bereiche außer ET und IT



weltweit



EFTA

EU



national



Deutschland

DKE(Deutsche Kommission Elektrotechnik
Elektronik Informationstechnik) im DIN und
VDE



Österreich

Österreichische Elektrotechnische
Komitee (ÖEK) innerhalb des OVE

Verordnungen

erlassen von staatl.
Verwaltungsbehörden
ArbStättV, BildscharbV

Normen

verbindliche Standards von allg.
anerkannten int. Organisation

Prozessnormen ISO/IEC 90003
Produktnormen ISO/IEC 9126

Standards

erarbeitet von
nichtstaatlichen Gremien,
Empfehlungen

IEEE , OPC Foundation

Zertifizierungen

ausgestellt von staatl.
zugelassenen Institutionen
zum Nachweis aus Einhaltung
von Normen

für DIN ISO 9001
für DIN ISO 27001

Bildschirmarbeitsverordnung (BildscharbV)

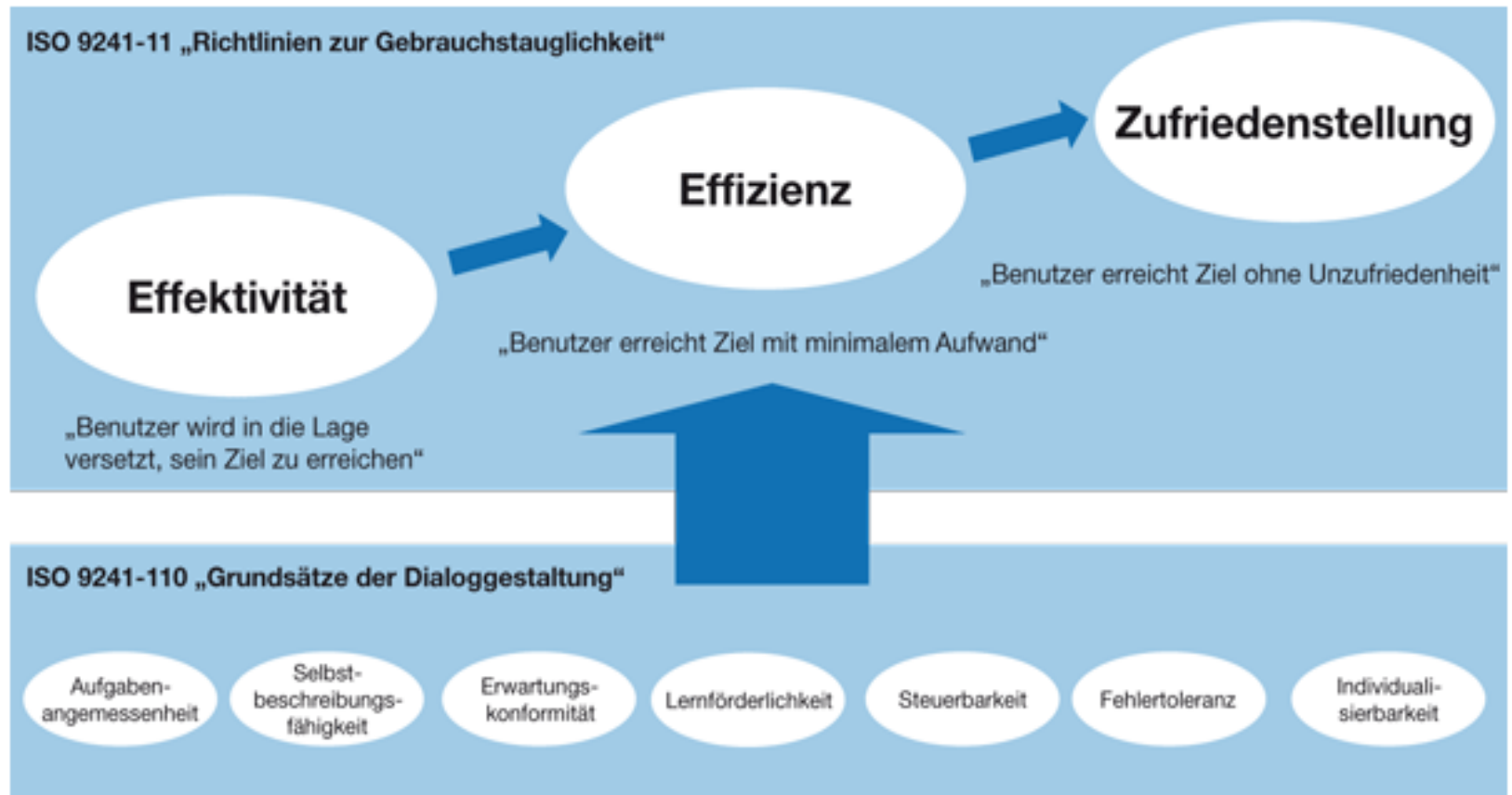
- seit Dezember 2016 nicht mehr eigenständig gültig, sondern nun Teil der ArbStättV
- bezieht sich auf alle Arbeitsplätze, bei denen **dauerhaft** an Bildschirmgeräten gearbeitet wird (lt. stat. Bundesamt 18.5 Millionen überwiegend genutzte Bildschirmarbeitsplätze im Jahr 2014)
- Bedienplätze von Maschinen oder Fahrzeugen mit Bildschirmen und Bildschirmplätze innerhalb von Verkehrsmitteln sind ausgenommen
- Der Arbeitgeber ist auch im Sinne der Bildschirmarbeitsverordnung zur Gefährdungsbeurteilung verpflichtet
- Ziel: Verhinderung von Gesundheitsgefährdungen am Büroarbeitsplatz durch Strahlen, Wärme, ungünstige Bewegungen und Körperhaltung
- **Ergonomie auch für Software ausdrücklich gefordert**
- die Normenreihe DIN EN ISO 9241 Teil 110 enthält konkrete Anforderungen an die ergonomische Gestaltung von Software.

DIN EN ISO 9241-110

Grundsätze der Dialoggestaltung

Usability zu deutsch: Gebrauchstauglichkeit

Gruppe: Software-Ergonomie mit 50 Empfehlungen



IEEE-Standards:

SQAP – Software Quality Assurance Plan IEEE 730

SCMP – Software Configuration Management Plan IEEE 828

STD – Software Test Documentation IEEE 829

SRS – Software Requirements Specification IEEE 830

SVVP – Software Validation & Verification Plan IEEE 1012

SDD – Software Design Description IEEE 1016

SPMP – Software Project Management Plan IEEE 1058

Software Reviews and Audits IEEE 1028

Software Life Cycle Processes IEEE 1074

Systems and Software Engineering – Systems Life Cycle Processes
IEEE 15288

Prozessnormen

Qualitätsmanagement (-System)

ISO 9000:2005 Quality management systems – Fundamentals and vocabulary

ISO 9001:2008 Quality management systems – Requirements

ISO 9004:2009 Managing for the sustained success of an organization – A quality management approach

ISO/IEC 90003:2004 Software engineering – Guidelines for the application of ISO 9001:2000 to computer software

Software-Prozesse und Vorgehensmodelle

ISO/IEC 12207:2008 Systems and software engineering – Software life cycle process

V-Modell XT, Vorgehensmodell zum Planen und Durchführen von Systementwicklungsprojekten des Bundes (Deutschland)

HERMES – Die schweizerische Projektführungsmethode, um Projekte der Informations- und Kommunikationstechnik (IKT) einheitlich und strukturiert durchzuführen

IT-BVM, Vorgehensmodell für die Entwicklung von IT-Systemen des Bundes (Österreich)

Produktnormen

Unter Produktnormen fallen Standards, die einheitliche Kriterien für die Beurteilung von Produktqualität zur Verfügung stellen. Nachfolgend einige Beispiele von Produktnormen:

ISO/IEC 9126 Information Technology – Software Product Evaluation – Quality Characteristics and Guidelines for their Use

ISO/IEC 9126-1 Software Engineering – Product Quality – Part 1: Quality Model

ISO/IEC 25051:2006 Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Requirements for quality of Commercial Off-The-Shelf (COTS) software products and instructions for testing

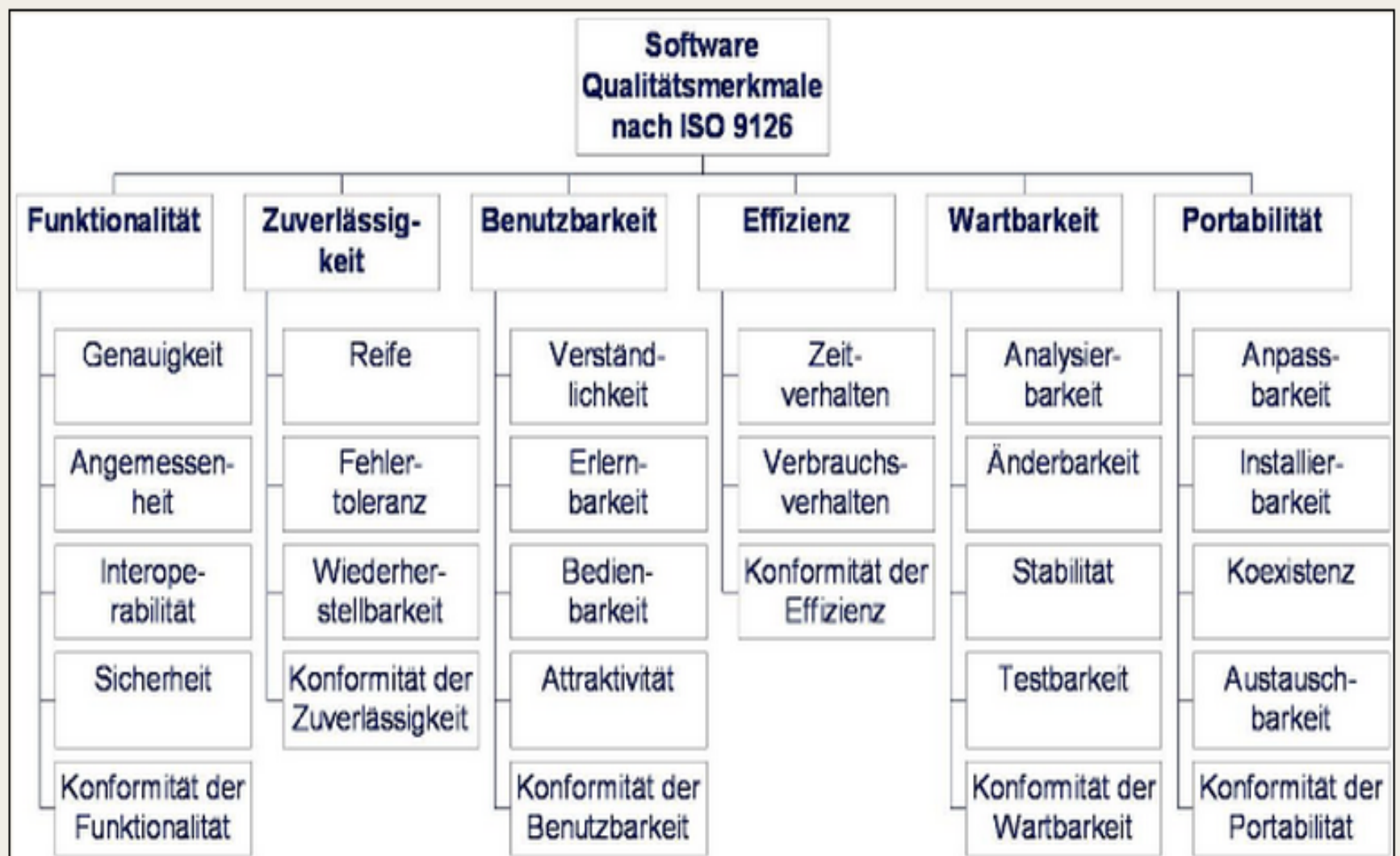


Abb. 2.: Qualitätsmodell für externe und interne Qualität nach ISO/IEC 9126, Teil 1

Kriterien für die Beurteilung der Software Qualität nach ISO 9126

Funktionalität: Sind alle im Pflichtenheft geforderten Funktionen vorhanden und ausführbar?

Zuverlässigkeit: Zu welchem Grad (z. B. Prozent der Arbeitszeit) erfüllt die Software dauerhaft die geforderten Funktionen? Werden alle Funktionen richtig ausgeführt (Korrektheit)?

Benutzbarkeit: Wie schnell lässt sich der Umgang mit der Software vom Benutzer erlernen (Erlernbarkeit)? Wie einfach lässt sich die Software durch den Benutzer handhaben (Bedienbarkeit)?

Effizienz: Welches zeitliche Verhalten (Antwortzeit im Dialogbetrieb, Laufzeit im Stapelbetrieb) und welchen Ressourcenverbrauch zeigt die Software unter den gegebenen Systemvoraussetzungen (Hardware, Betriebssystem, Kommunikationseinrichtungen)?

Wartbarkeit: Mit welchem Aufwand bzw. In welcher Zeit lassen sich Änderungen durchführen! Wie lässt sich der Aufwand für Fehlererkennung und Behebung minimieren ?

Portabilität: Mit welchem Aufwand lässt sich die Software (insbesondere Standardsoftware) an individuelle funktionale betriebliche Gegebenheiten anpassen (Anpaßbarkeit)? Lässt sich die Software ohne größeren Aufwand in anderen Systemumgebungen zum Einsatz bringen (Portabilität)? Kann die Software beim Austausch des Rechners (z. B. leistungsfähiger Prozessor) unverändert eingesetzt werden (Skalierbarkeit, Aufwärtskompatibilität)

IEEE- und ISO-Standards zur Messung der Softwarequalität

Gemäß IEEE-Standard 610 gibt es folgende Definitionen zur Messung der Softwarequalität:

1. Softwarequalität ist der Grad, in dem das System die gestellten Anforderungen erfüllt.
2. Softwarequalität ist der Grad, zu dem ein Softwaresystem die Bedürfnisse und Erwartungen der Benutzer erfüllt (IEEE99).

ISO/IEC 25000

Die internationale Norm ISO/IEC 25000 Software engineering – **Software product Quality Requirements and Evaluation (SQuaRE)** – Guide to SQuaRE **ersetzt seit 2005 die Norm ISO/IEC 9126** und wurde von dem Normungsgremium ISO/IEC JTC 1/SC 07 Software -and Systems Engineering erstellt. Die deutsche Version DIN ISO/IEC 25000 Software-Engineering – Qualitätskriterien und Bewertung von Softwareprodukten (SQuaRE) – Leitfaden für SQuaRE wird seit 2010 durch den NA 043 Normenausschuss Informationstechnik und Anwendungen (NIA) des Deutschen Instituts für Normung vorbereitet.

NORM-ENTWURF

DIN ISO/IEC 25000:2013-04

Titel (deutsch): Software-Engineering - Qualitätskriterien und Bewertung von Softwareprodukten (SQuaRE) - Leitfaden für SQuaRE (ISO/IEC 25000:2005)



DOWNLOAD

VERSAND

ABO **


Sprache: **Deutsch**

☐  119,80 EUR

☐ 126,40 EUR

☐

** Erfahren Sie mehr zu » [Abonnements-Lösungen für Normen](#)

 **IN DEN WARENKORB**

Titel (englisch): Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Guide to SQuaRE (ISO/IEC 25000:2005)

Dokumentart: Norm-Entwurf

Ausgabedatum: 2013-04

Erscheinungsdatum: 2013-05-06

weitere Normen

- IEC 61511: Funktionale Sicherheit. Sicherheitstechnische Systeme für die Prozessindustrie. Teil 1: Allgemeines, Begriffe, Anforderungen an Systeme, Software und Hardware.
- IEC 62061: Sicherheit von Maschinen. Funktionale Sicherheit sicherheitsbezogener elektrischer, elektronischer und programmierbarer elektronischer Steuerungssysteme.
- IEC 60601: Medizinische elektrische Geräte. Allgemeine Festlegungen für die Sicherheit.
- EN 60601-1-4: Medizinische elektrische Geräte. Teil 1-4: Allgemeine Festlegungen für die Sicherheit; Ergänzungsnorm: Programmierbare elektrische medizinische Systeme.
- DIN EN 62304: Medizingeräte-Software. Software-Lebenszyklus-Prozesse
- IEC 61508 : Entwicklung sicherheitskritischer, programmierbarer elektronischer Systeme
- FDIS 26262: Road Vehicles. Functional Safety (löst die IEC 61508 für die Automobilindustrie ab).

Zertifizierungen

Beispiel: ISO/IEC 27001:2013

Anforderungen für Herstellung, Einführung, Betrieb, Überwachung, Wartung und Verbesserung eines dokumentierten Informations-Sicherheits-Managementsystems (ISMS)

Zielgruppen:

Rechenzentren, Verwaltung, Banken, TK-Anbieter, Energieversorger (Vorschrift von BNetzA)

Auditor:

Geprüfte Zertifizierungsstelle z.B. TÜV
Externe Berater für KMU



Fazit:

die Norm für den täglichen
Programmieralltag ist

ISO/IEC 25010:2011

(vormals ISO/IEC 9126)



Qualität erzeugen

Ausbildung

Erfahrung

Fester Ablauf (Workflow)

Tools

Zeit

Machbarkeit

Gelassenheit

Spaß

Professionalität

Qualität simulieren

Schnell – Viel-Machen

Wochenendarbeit

Kamikaze-Aktionen

egal wie

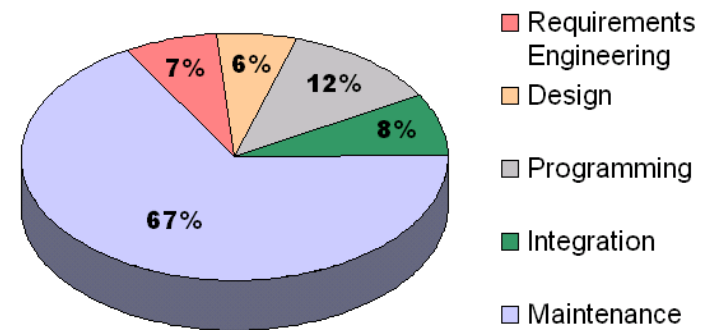
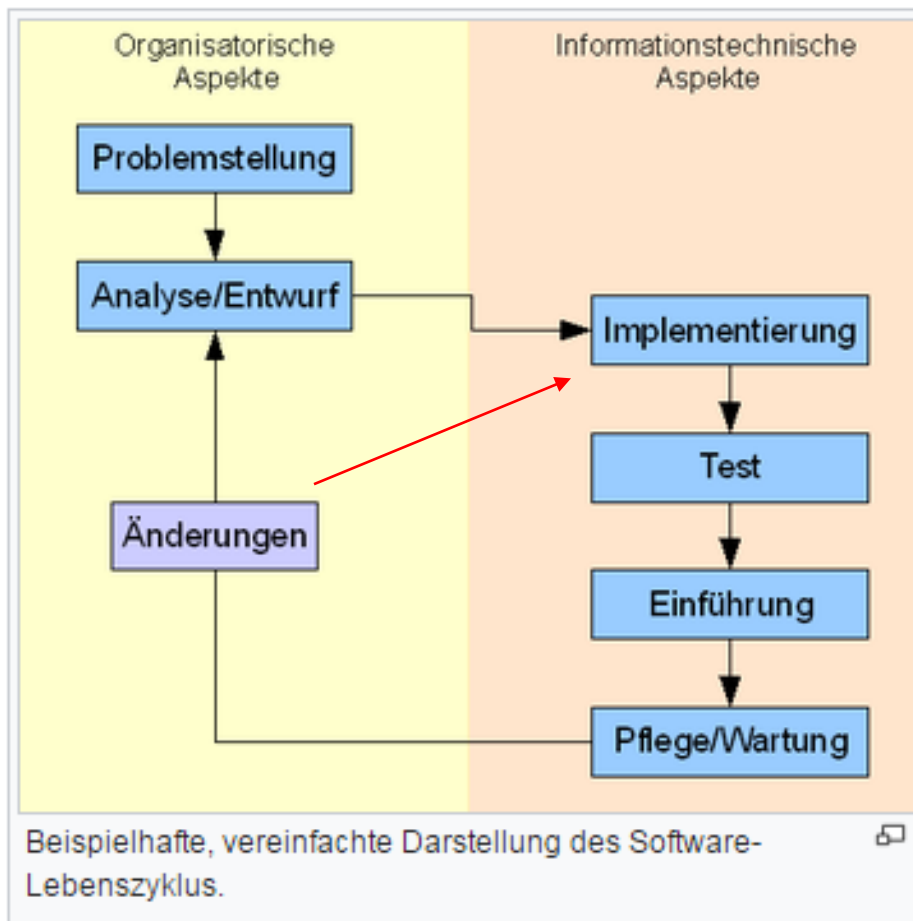
Kosten

Stress

Unzufriedenheit

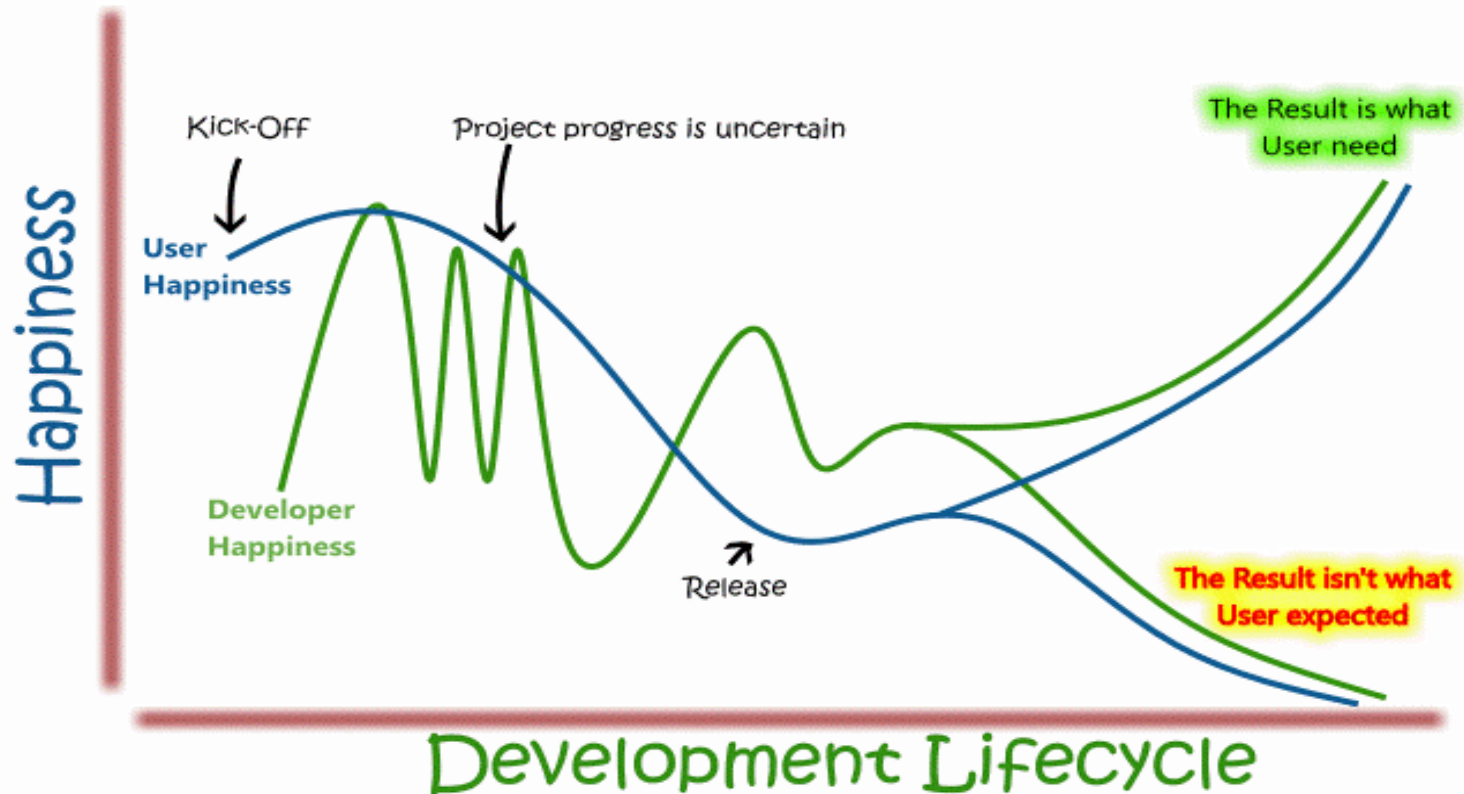
Softwarelebenszyklusmanagement

Norm ISO/IEC 12207



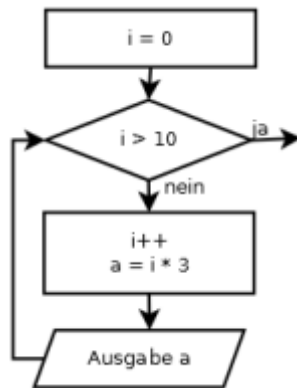
Approximate relative costs of the phases of the software life cycle [Schach 1999]

Happiness in Software Development Lifecycle

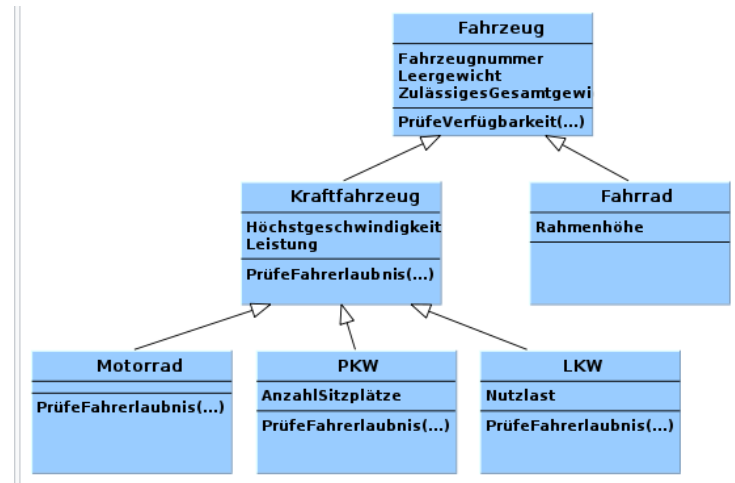


Software-Entwurfsphase

PAP



UML



Merksätze:

Software is not for its creators, it is for its users.

Software wird 1mal geschrieben und n+1 mal gelesen.

quantitativen Bewertung von Softwarequalität

a) direkte Messgrößen

- Zuverlässigkeit: Ausfallzeiten
- Korrektheit: Anzahl Fehler pro Zeiteinheit
- Bedienbarkeit: Anzahl Aufrufe pro Vorgang, Anzahl Klicks oder Touches
- Effizienz: Antwortzeit (Durchschnitt, Spitze) pro Transaktion
- Wertbarkeit: Zeitaufwand je Fehlerbehebung

b) indirekte Messgrößen

- Programmgröße: Anzahl Programmzeilen (LOC = Line of Code)
- Programmstruktur: Anzahl Hierarchieebenen (Schachtelungstiefe), Anzahl Strukturblocke, Anzahl Module
- Programmkomplexität: Anzahl unterschiedlicher Steuerkonstrukte
- Kommentarumfang: Anzahl Kommentarzeilen absolut und relativ im Verhältnis zur Anzahl Programmzeilen

Wer führt Quantitative Bewertung der Software durch?

- Projektverantwortliche sind keine Softwareexperten?
- keine „Beste Beweismethodik“ für Fehlfunktionen

Lösung:

Pair-Programming

Code-Reviews

Tests

Verfügbarkeit eines Systems

Dazu werden die verfügbaren Zeiten den Ausfallzeiten gegenübergestellt. Beispielsweise wird bei Serversystemen oft eine Verfügbarkeit von 0,99 vereinbart. Fällt das System innerhalb eines Testzeitraums von 24 Stunden auch nur 0,5 Stunden aus, so ist die Bedingung verletzt ($23,5 / 24 = 0,979 < 0,99$).

Pair-Programming



Microsoft hat für Windows 7 bestimmte Techniken des **Ex Programming** angewandt, u.a. wohl auch Pair Programming. von <http://de.wikipedia.org/wiki/Paarprogrammierung>
"Paarprogrammierung bedeutet, dass bei der Erstellung des Quellcodes jeweils zwei Programmierer an einem Rechner arbeiten. Ein Programmierer schreibt den Code, während der andere über die Problemstellungen nachdenkt, den geschriebenen Code kontrolliert sowie Probleme, die ihm dabei auffallen, sofort anspricht. Diese können dann sofort (im Gespräch zu zweit) gelöst werden. Die beiden Programmierer sollten sich bezüglich dieser beiden Rollen abwechseln. Auch die Zusammensetzung der Paare sollte sich häufig ändern."

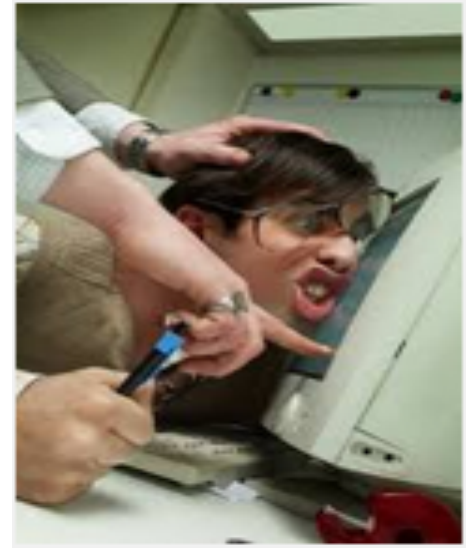
Code Review

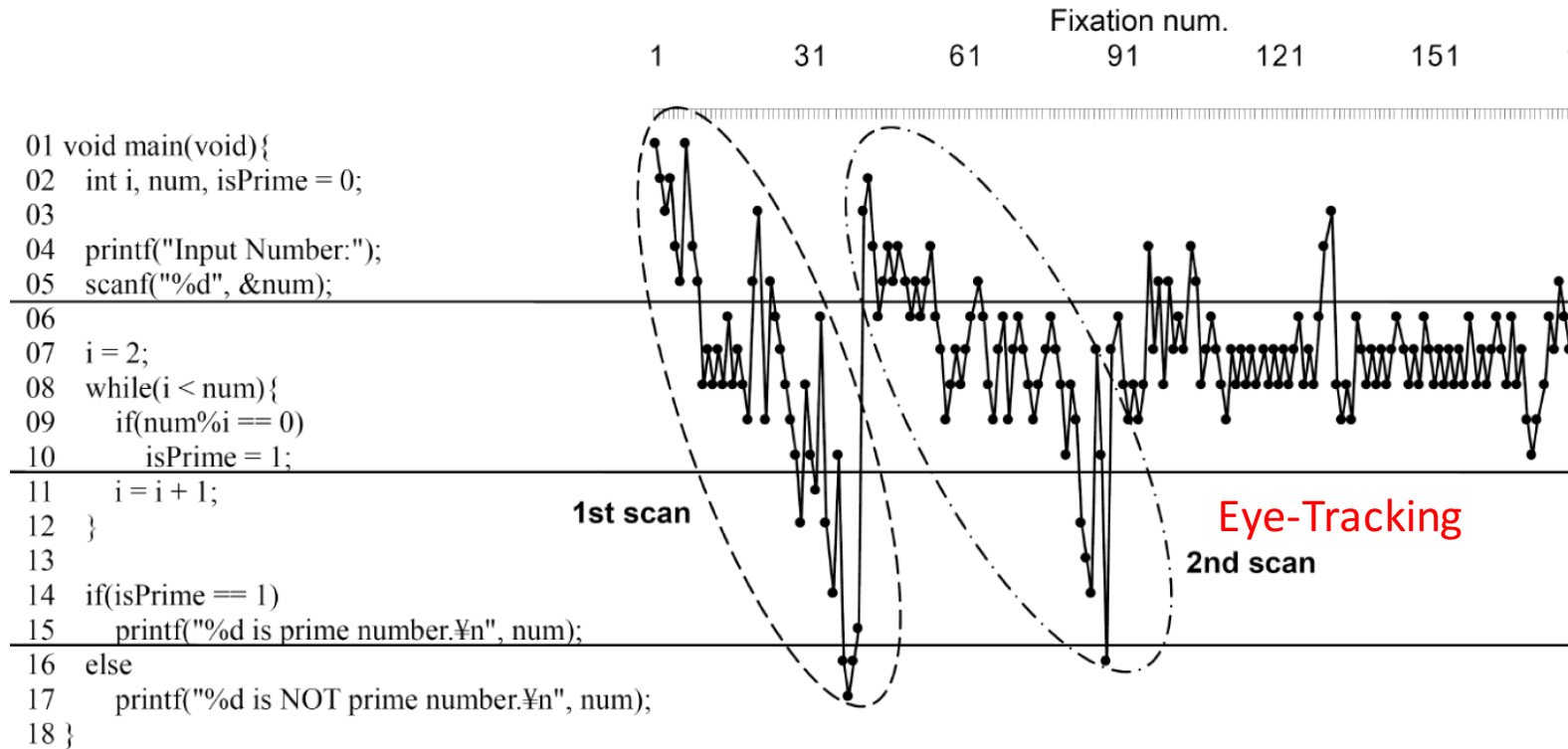
Nach IEE Std610 („Glossary of Software Engineering Terminology“):

Ein Review ist ein mehr oder weniger formal geplanter und strukturierter Analyse- und Bewertungsprozess, indem Produktergebnisse einem Team von Gutachtern präsentiert und von diesen kommentiert und **genehmigt** werden.

Vorteil: unmittelbare Qualitätsverbesserung,
Feedback an Neu-Programmierer

Nachteil: Zusatzbelastung der Leistungsträger im Team





4.03.2015

JavaLand 2015 / Code Reviews (Rabea Gransberger @rgransberger)

Reviewer können nur ungefähr 100 Codezeilen pro Stunde analysieren, so dass schon für die Überprüfung einer mittelgroßen Anwendung mit 20.000 Zeilen rund 200 Stunden benötigt werden, und das ohne Dokumentation, Korrektur etc. Daher geht der Trend stark zu Tool-gestützten, formalisierten Reviews, die sich wiederholt auf große Codemengen anwenden lassen.

Quelle: Computerwoche "Zehn Tipps für den Code Review" 15.02.2013 von Dr. Bruce Sams

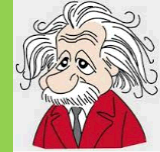
Projektleiter:

- beauftragt Reviews
- achtet auf die Disziplin im Team (muss Sanktionsmöglichkeiten haben)



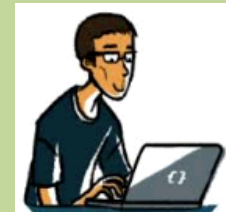
Level A Entwickler (Entwickler):

- arbeitet nicht am Projekt
- führt Reviews für LEVEL B und C durch
- hat Vollzugriff auf alle Softwareobjekte – ist Verwalter der CODEBASIS des Unternehmens
- genehmigt oder verwirft beantragte Änderungen an Modulen/Objekten
- programmiert projekt-übergreifende Funktionen und Objekte - macht die Architektur



Level B-Programmierer (Programmierer):

- arbeitet am Projekt
- darf Reviews für LEVEL C durchführen, wird selbst von LEVEL –A reviewed
- programmiert die Änderungen an Modulen/Objekten im Auftrag vom LEVEL A- Entwickler
- schreibt „schnelle“-Lösungen für sich und Level-C-Programmierer
- die „schnellen Lösungen“ werden nur nach Review in die CODEBASIS zurückgepflegt



Level C-Programmierer (Inbetriebnehmer):

- arbeitet am Projekt
- verschaltet fertige Module - macht die Logik und Parametrierung
- wird nur hin und wieder reviewed – hier Ablaufftest als Feedback
- beantragt Änderungen an Modulen/Objekten beim Level-B-Programmierer
- seine eigenen Module werden niemals in die CODEBASIS zurückgepflegt (Wildwuchs)



Für unerfahrene Entwickler bietet der Codereview durch einen erfahrenen Programmierer ideale Möglichkeiten, sich schnell und praxisorientiert weiterzubilden – Aufstieg in höheres Level als Motivation.

Code review

CHECKLIST

Use this form to help you perform a code review.

About the code

Module name: _____
Version reviewed: _____
Code author: _____

Reviewed by: _____
Date: _____
Language: _____
Number of files: _____

Automated inspection

- ☐ The code compiles without errors
- ☐ The code compiles without warnings
- ☐ There are unit tests
 - ☐ They are sufficient (include all boundary cases, etc.)
 - ☐ The code passes them

- ☐ The code is kept under source control
- ☐ The code has been tested with inspection tools

Tool name	Results
_____	_____
_____	_____
_____	_____

☐ Continue to next section ☐ Stop review here

Design

- ☐ The code is complete (against its specification)
- ☐ There is a good choice of algorithms
- ☐ Optimizations are necessary and appropriate
- ☐ Any missing functionality is marked clearly in the code

General observations about the code's design

- ☐ The code is well structured
- ☐ There is design documentation
- ☐ The code matches the documentation

☐ Continue to next section ☐ Stop review here

General code comments

Style

- ☐ The code layout is clear
- ☐ It follows project style guidelines
- ☐ There is a good (unambiguous) public API
- ☐ There is a good choice of names

Defensive programming

- ☐ Array access is guarded and safe (C/C++)
- ☐ There is a correct choice of types
- ☐ All input is validated
- ☐ There is no use of compiler-specific features

General comments

General comments about the quality of the written code

Error handling

- ☐ Error conditions are routinely handled
- ☐ Assertions are used to validate logic
- ☐ The code is exception safe
- ☐ Errors are propagated, not hidden
- ☐ There are no resource leaks

The code uses multiple threads

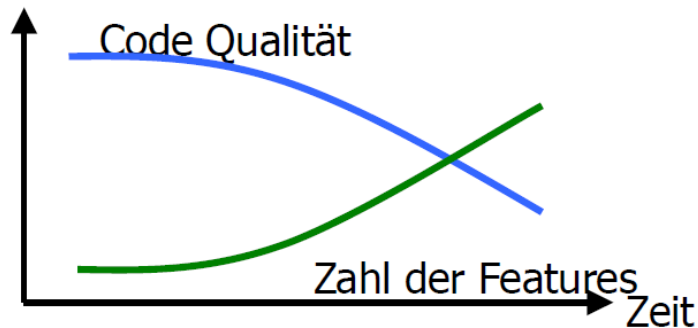
- ☐ It is thread safe
- ☐ There isn't potential for deadlock

Structure

- ☐ There is no redundant code
- ☐ There is no cut-and-paste programming

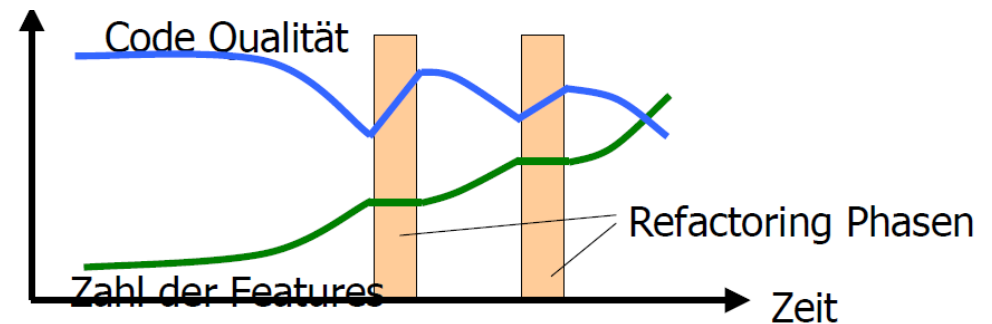
☐ Continue to next section ☐ Stop review here

Refactoring



**Verbesserung des Codes ohne
Änderung des Verhaltens.**

**Kleine Schritte
Wartung von Code**



- Durch das Refactoring wird also eine bestehende Software generell oder auch in Teilen lesbarer, in ihren Strukturen verständlicher, besser modifizierbar, besser testbar und es werden Redundanzen vermieden.

Refactoring

- Gruppen:
 - die Unzufriedenen: Level C-Programmierer als Anwender im Unternehmen - bester Indikator
 - der Stolz: Level-A-Entwickler/Codeverwalter stolz auf „seinem“ Code
 - der Sparsame: Chef „Geht doch - warum nochmal Aufwand investieren - keine Zeit?“
- Problem: wer entscheidet, das es **jetzt** Zeit ist für ein Refactoring?
- Meist erst, wenn die Fehlerkosten schon zu hoch sind und der Druck nicht mehr nur von innen sondern von außen (Kunde) kommt dann haben sich schon viele Refactoring-Schritte angesammelt und wenig Zeit zum Testen!
- Irgendwann: Ach komm wir machen Alles neu.....!?
- Refactoring Vorschläge und Unterstützung in Visual Studio und Eclipse



Oft genannte Vorteile der OO-Programmierung:

- für **Designer**: der Entwurfsprozeß wird einfacher, klarer und handhabbarer.
- für **Programmierer**: klares Objektmodell, mächtige Programmierwerkzeuge, nützliche Bibliotheken.
- für **Manager**: Entwicklung und Wartung von Software wird schneller und billiger durch gesteigerte Produktivität.

Allerdings:

- OOP muss man lernen.
- umso schwerer je tiefer die „prozeduralen Wurzeln“ sind
- Gutes Design für Objekte ist nicht leicht.

OOP-Ansätze für nicht OOP-Systeme

Problem: keine Methoden an Datentypen bindbar (keine Kapselung)

Ansatz: OOP-ähnliche Abstraktion für Datenbehälter (z.B. Job, Teil, Platz)
Datenbehälter als komplexer Datentype (Structs)
Reduzierung globaler Variablen und Funktionen
Aufrufhierarchien und Aufruftiefe maximal 3
Instanzierung nutzen falls vorhanden
grafische Programmierung zumindest in der Aufrufebene nutzen

ABB-RAPID:

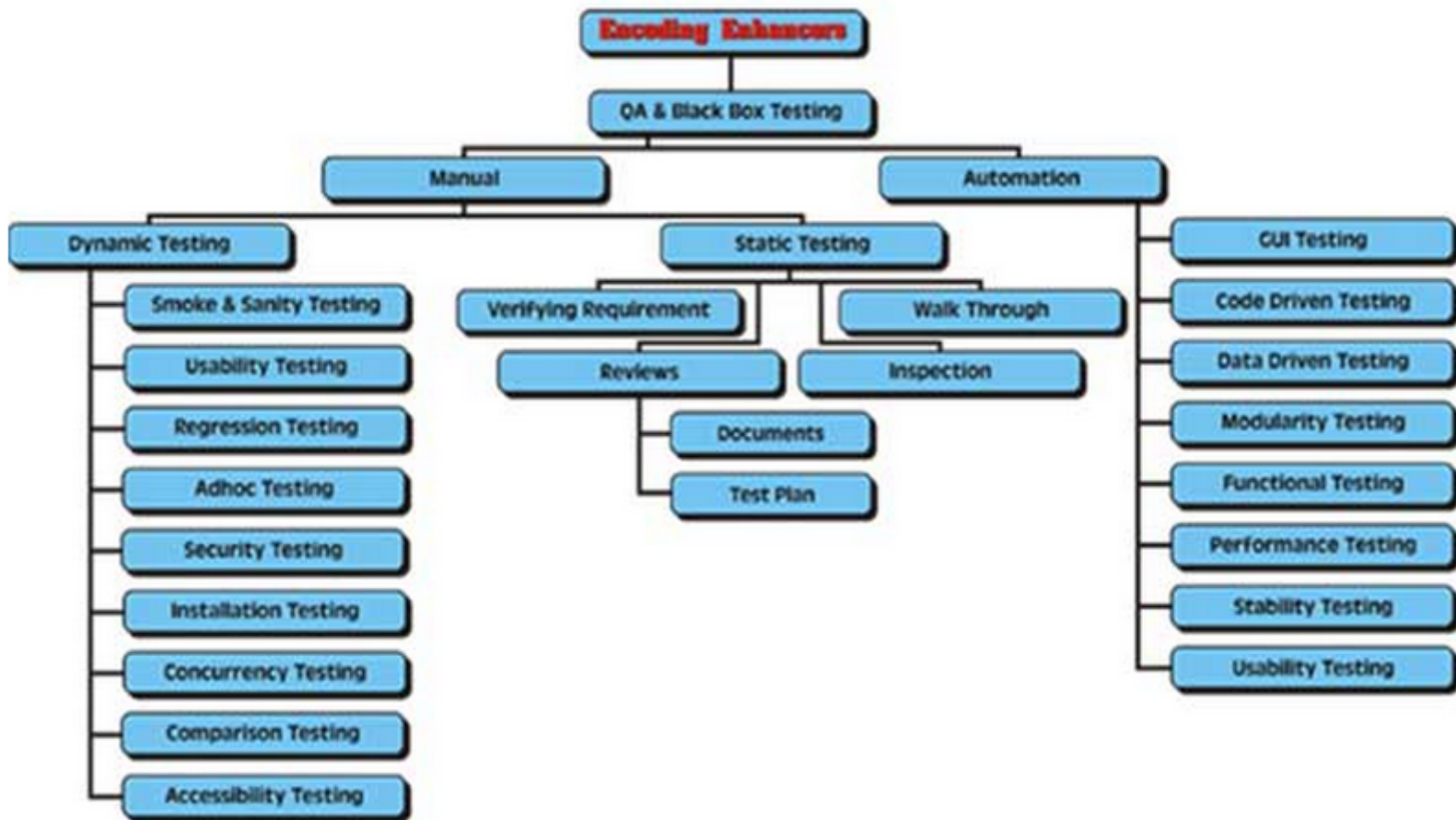
```
! Typ zur Konfiguration eines Jobs
RECORD typeJob
  num No;
  num XMasz;
  num YMasz;
  num ZMasz;
  num SPNo;
  robtarget SPos;
  pos Vto_S1;
  pos Vto_S2;
  pos Vfrom_S1;
  pos Vfrom_S2;
  num Speed;
  num Acc;
  num LoadType;
  loaddata JLoad;
ENDRECORD
```

Beckhoff TWINCAT:

Beispiel für eine Strukturdefinition mit Namen ST_ALIGN_SAMPLE:

```
TYPE ST_ALIGN_SAMPLE:
  STRUCT
    _diField1 : DINT;
    _byField1 : BYTE;
    _iField   : INT;
    _byField2 : BYTE;
    _diField2 : DINT;
    _pField   : POINTER TO BYTE;
  END_STRUCT
END_TYPE
```

Tests



Der Markt für Software Qualität hat inzwischen nur im Bereich des **Software Testens** ein Volumen weltweit von 33,4 Mrd USD (Nelson Hall 2012)

Test Driven Development

klassischer Vorgehensweise:

Erst Programmieren, dann Alles testen

testgetriebenen Entwicklung - UNIT Tests:

Schreibe Tests für das erwünschte fehlerfreie Verhalten, für schon bekannte Fehlschläge oder für das nächste Teilstück an Funktionalität, das neu implementiert werden soll.

DANACH!!!!

Ändere/schreibe den Programmcode mit möglichst wenig Aufwand, bis nach dem anschließend angestoßenen Testdurchlauf alle Tests bestanden werden. Räume dann im Code auf : Entferne Wiederholungen, abstrahiere wo nötig, richte ihn nach den verbindlichen Code-Konventionen aus etc.

Natürlich wieder mit abschließendem Testen. Ziel des Aufräumens ist es, den Code *schlicht* und *verständlich* zu machen.

Quelle: http://de.wikipedia.org/wiki/Testgetriebene_Entwicklung

Weiterbildungen



<https://www.isqi.org/>

Sitz in Potsdam, gegründet , Anfänge 1996

iSQI® Agile Essentials | ISTQB® Certified Tester | IREB® Certified Professional for Requirements Engineering | iSQI® Certified Professional for Model Based Testing | iSQI® Certified Professional for Project Management | iSQI® Certified Model Based Tester | usw.



Schulungen mit iSAQB Zertifizierung

- ▶ iSAQB CPSA-F
- ▶ iSAQB CPSA-A: Soft Skills für Softwarearchitekten
- ▶ iSAQB CPSA-A: Service Oriented Architecture – Technisch (SOA-T)
- ▶ iSAQB CPSA-A: Architekturdokumentation
- ▶ iSAQB CPSA-A: Architekturbewertung
- ▶ iSAQB CPSA-A: Enterprise Architecture Management





ASQF – Kompetenznetzwerk mit über 1.200 Mitgliedern

Der Arbeitskreis Software-Qualität und -Fortbildung e.V. (ASQF) ist das Kompetenznetzwerk für Software-Qualität im gesamten deutschsprachigen Raum.

<https://www.asqf.de/>

Diverse Fachgruppen, Veranstaltungen und Publikationen

Spezielle Probleme der Automatisierungstechnik

- Software nur ein Teil des Gesamtprojektes daher kein Softwareprojektmanagement
- Führungspersonal nicht ausgebildet in Softwaretechnologie
- keine formulierten Qualitätsvorgaben für die Software
- unvollständige Aufgabenbeschreibung(Requirements)
- selten Teamstrukturen - wenn dann ohne Hierarchien
- selten interdisziplinären Teams
- interner Wissenstransfer unzureichend
- selten Wissenstransfer von extern
- Keine Code-Reviews, keine Code-Tests
- Tools in AT noch unzureichend

