Kap. 7:
Entwicklung von Anwendungen
(a) Programmiersprachen

7. Entwicklung von Anwendungen

Konzentration auf

- Sicherheitskritische Anwendungen
- Echtzeitanwendungen

Gliederung

(a)

1. Einführung
2. IEC EN 61508
3. Programmiersprachen (MISRA C/C++, Ada)

(b)

4. Modellierung (SysML, UML MARTE, …)

(c)

5. Validierung
6. Systematisches Testen

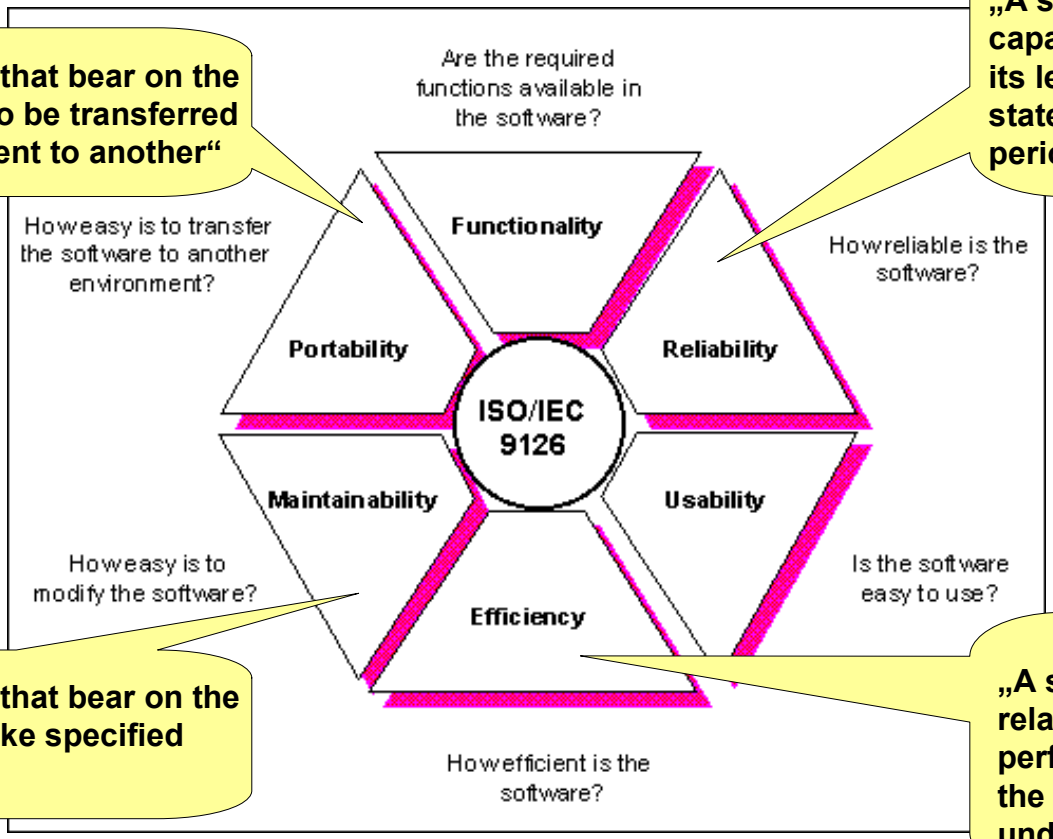
7.1. Einführung

ISO/IEC 9126:1991 → ISO/IEC 25010 (ab 2011)

- **Kriterien zur Evaluation von Softwarequalität**
- **Nicht prozessbezogen**

„A set of attributes that bear on the ability of software to be transferred from one environment to another“

„A set of attributes that bear on the capability of software to maintain its level of performance under stated conditions for a stated period of time“

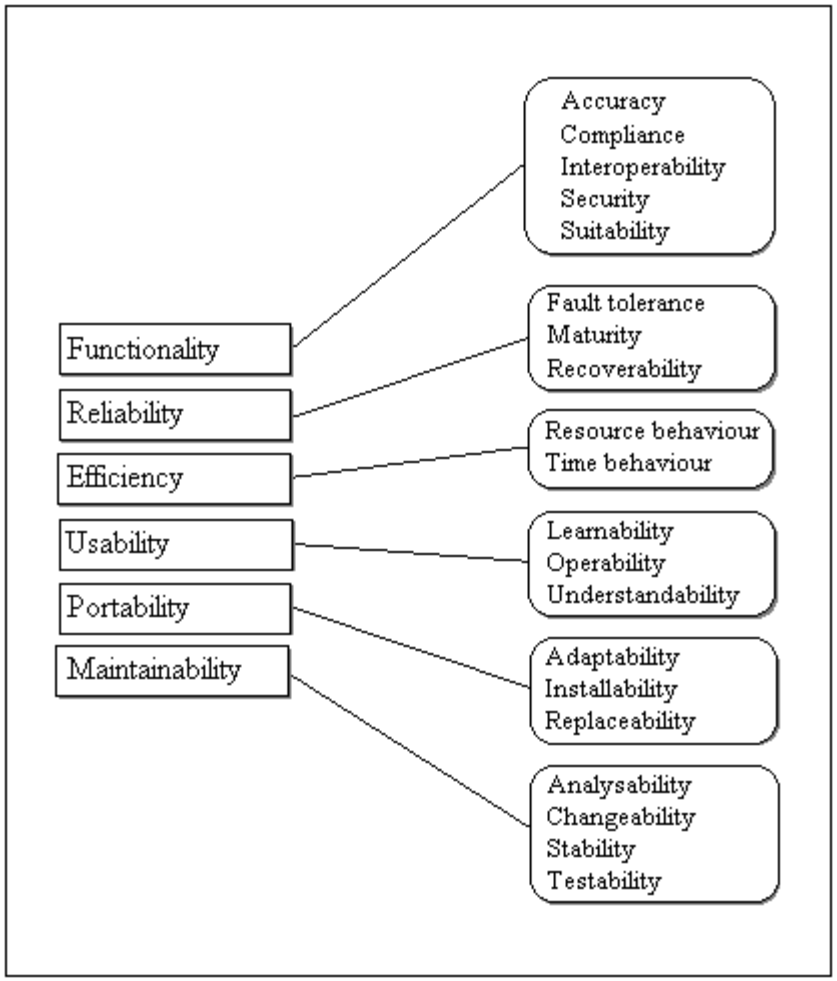


„A set of attributes that bear on the effort needed to make specified modifications“

„A set of attributes that bear on the relationship between the level of performance of the software and the amount of resources used, under stated conditions“

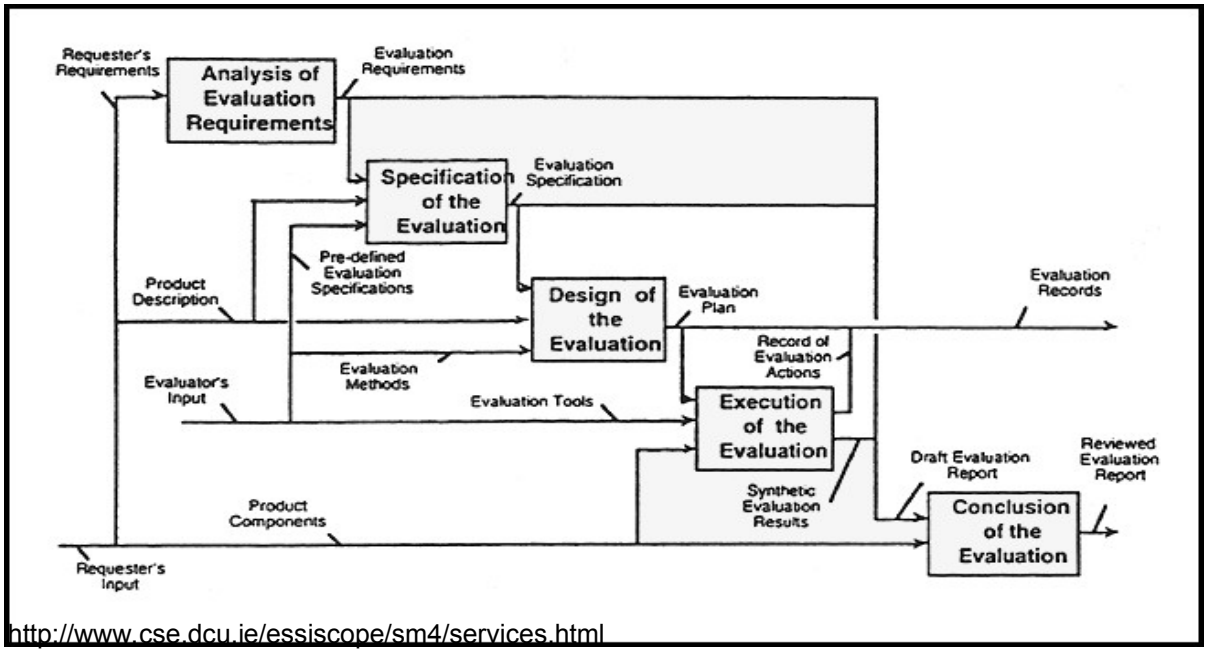
http://www.cse.dcu.ie/essiscope/sm2/9126ref.html

ISO/IEC 9126 Subattribute



ISO/IEC 14598 (aktuell ISO/IEC 25040:2011)

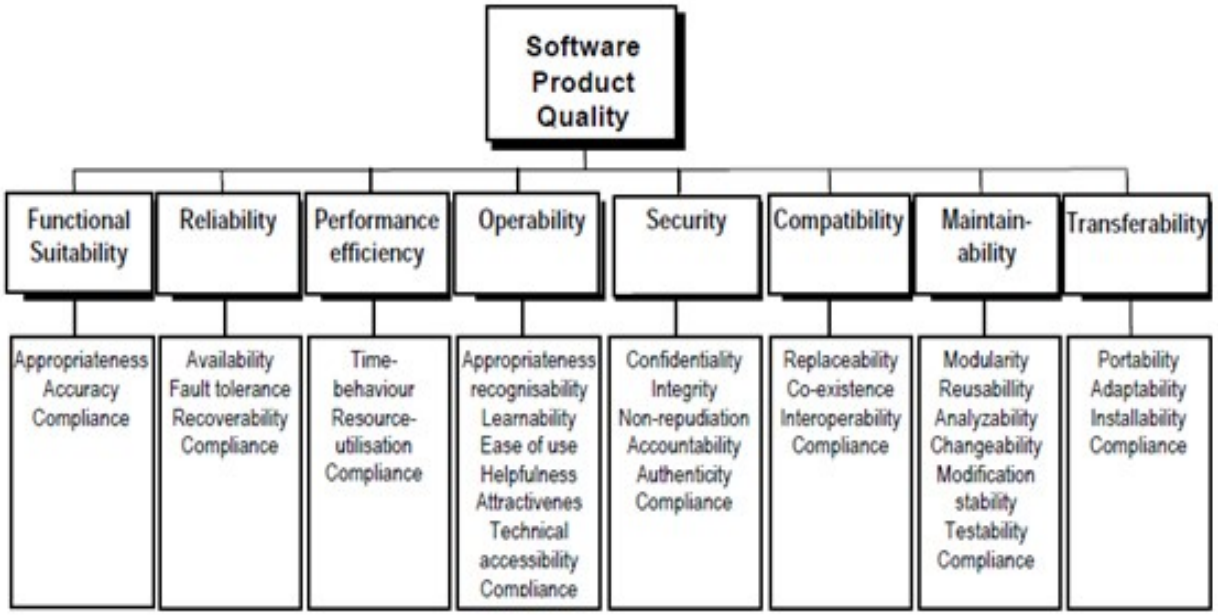
- Betrachtet Prozess zur SW-Produkt-Bewertung
- Charakteristiken:
  - Repeatability
  - Reproducibility
  - Impartiality, Objectivity



http://www.cse.dcu.ie/essiscope/sm4/services.html

ISO/IEC 25010 (ab 2011)

- Weiterentwicklung und Ersatz für 9126 und 14598
- ISO/IEC 25010:2011 Software product Quality Requirements and Evaluation (SQuaRE)
- 8 Qualitätskriterien (statt 6) mit 31 Subkategorien
- Ziel: Spec für nicht-funktionale Eigenschaften



http://http://www.xbosoft.com/english/definition-software-quality/

7.2. IEC EN 61508

- **Titel: Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer elektronischer Systeme**
  - **Brachenübergreifende generische Richtlinie für sicherheitsgerichtete Systeme**
  - **Definiert Sicherheits-Integritätslevel**
  - **Beurteilung elektrischer/elektronischer/programmierbar elektronischer (E/E/PE)-Systeme in Bezug auf die Zuverlässigkeit von Sicherheitsfunktionen**
  - **SIL 1-4: "Vier wohlunterschiedene Stufen zur Spezifizierung der Anforderung für die Sicherheitsintegrität von Sicherheitsfunktionen, die dem E/E/PE-sicherheitsbezogenen System zugeordnet werden, wobei der Sicherheits-Integritätslevel 4 die höchste Stufe der Sicherheitsintegrität und der Sicherheits-Integritätslevel 1 die niedrigste darstellt."**
- **Ziel: Schutz der Gesundheit von Menschen, der Umwelt und von Gütern**

Safety Integrity Levels

- Die Sicherheitsanforderungsstufe stellt ein Maß für die Zuverlässigkeit des Systems in Abhängigkeit von der Gefährdung dar.
- Höherer Aufwand je höher die Gefährdung
- Beispiel: Antiblockiersystem
- Für Informatiker relevant: Sicherheitsfunktionen, die durch ein Programm erbracht werden einschl. der für die Erstellung verwendeten Werkzeuge (z.B. Compiler)

Anwendungs- / Geltungsbereiche

- Anwendung noch freiwillig, aber Druck durch Produkthaftungsgesetz (ProdHaftG)
- Speziell in Deutschland gilt nach §4 ProdHaftG, dass der Hersteller des Endproduktes sowohl bei der Haftung (wie auch bei eventuellen Imageschäden) selbst dann in Mithaftung genommen wird, wenn der Verursacher ausschließlich ein Unterlieferant war.
- IEC 61508 ist als „Sicherheits-Grundnorm“
  - IEC 61511: Funktionale Sicherheit — Sicherheitstechnische Systeme für die Prozessindustrie
  - IEC 61513: Kernkraftwerke — Leittechnik für Systeme mit sicherheitstechnischer Bedeutung – Allgemeine Systemanforderungen
  - EN 50128: Bahnanwendungen — Telekommunikationstechnik, Signaltechnik und Datenverarbeitungssysteme - Sicherheitsrelevante elektronische Systeme für Signaltechnik
  - IEC 62061: Sicherheit von Maschinen — Funktionale Sicherheit sicherheitsbezogener elektrischer, elektronischer und programmierbarer elektronischer Steuerungssysteme
  - IEC 60601: Medizinische elektrische Geräte - Allgemeine Festlegungen für die Sicherheit
  - ISO 26262: Road vehicles – Functional safety

- **Risikoklassen I-IV:**
  - **Kombination aus Kombination der Variablen Schwere des Unfalls und wahrscheinlicher Häufigkeit.**
  - **I: Inakzeptables Risiko**
  - **II: Unerwünscht, bei nicht reduzierbarem Risiko oder Unverhältnismäßigkeit des Aufwands akzeptabel**
  - **III: Tolerierbares Risiko**
  - **IV: Unbedeutendes Risiko**

	Bedeutung/ Schwere			
Häufigkeit	**Katastrophal**	**Kritisch**	**Gering**	**Unbedeutend**
Häufig	I	I	I	II
Wahrscheinlich	I	I	II	III
Gelegentlich	I	II	III	IV
Zukünftig	II	III	III	IV
Unwahrscheinl.	III	III	IV	IV
Unglaublich	IV	IV	IV	IV

- Korrekte Ausführung der Aktionen, um den sicheren Zustand einer Einrichtung zu erreichen oder zu erhalten
- Gewährleistung aller Maßnahmen zur Vermeidung zufälliger oder systematischer Fehler
- Unterscheidung bei der Ermittlung der Ausfallwahrscheinlichkeiten nach Betriebsarten:
  - Niedrige Anforderungsrate → Ausfallwahrscheinlichkeit PFD (Probability of Failure on Demand)
  - Hohe oder kontinuierliche Anforderung → Ausfallwahrscheinlichkeit PFH (Probability of Dangerous Failure per Hour)

SIL-Ausfallwahrscheinlichkeiten

- **Niedrige Anforderungsrate**
  - **PFD (Prob. Of Failure on Demand, Wahrscheinlichkeit des Versagens bei Anforderung)**
  - **RRF (Risk Reduction Factor)**

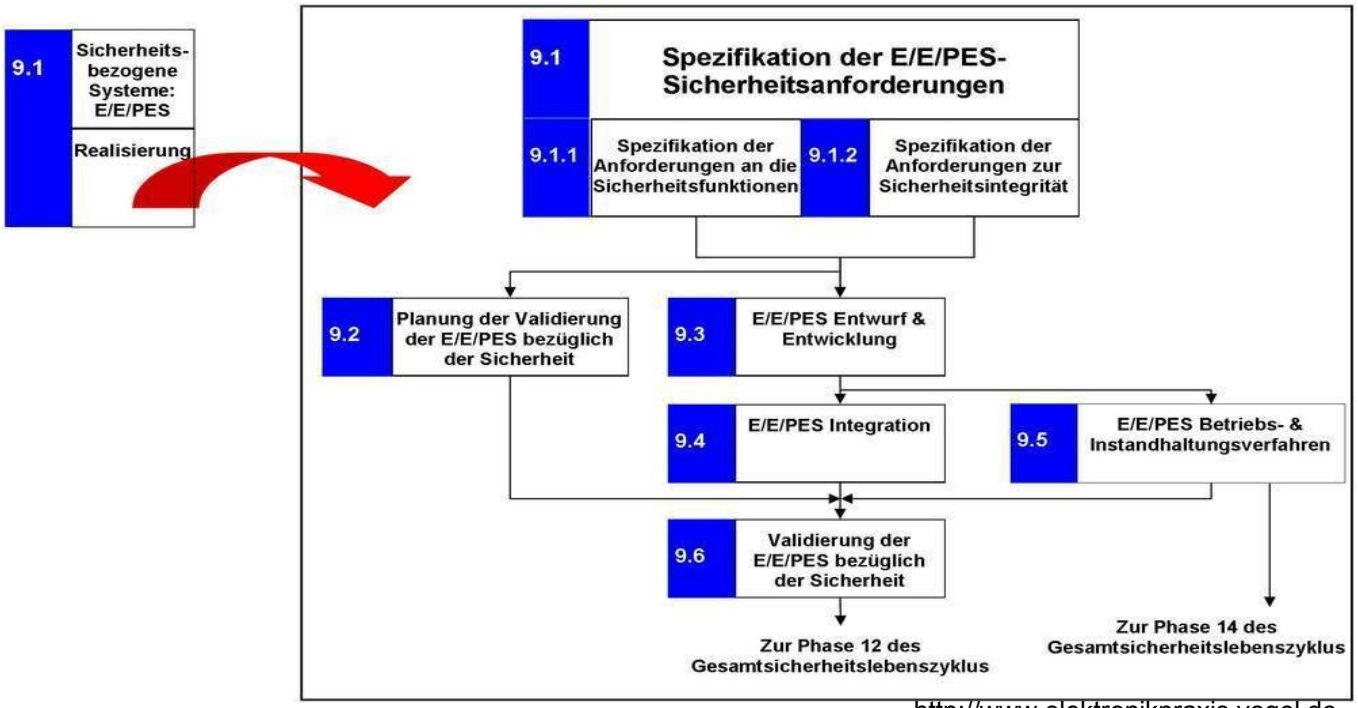
SIL	PFD	PFD (power)	RRF
1	0.1-0.01	10^{-1} - 10^{-2}	10-100
2	0.01-0.001	10^{-2} - 10^{-3}	100-1000
3	0.001-0.0001	10^{-3} - 10^{-4}	1000-10.000
4	0.0001-0.00001	10^{-4} - 10^{-5}	10.000-100.000

SIL-Ausfallwahrscheinlichkeiten (2)

- **Continuous Operation (Dauerbetrieb)**
  - **PFH (Prob. of Failure per Hour, Wahrscheinlichkeit des Versagens pro Stunde)**
  - **RRF (Risk Reduction Factor)**

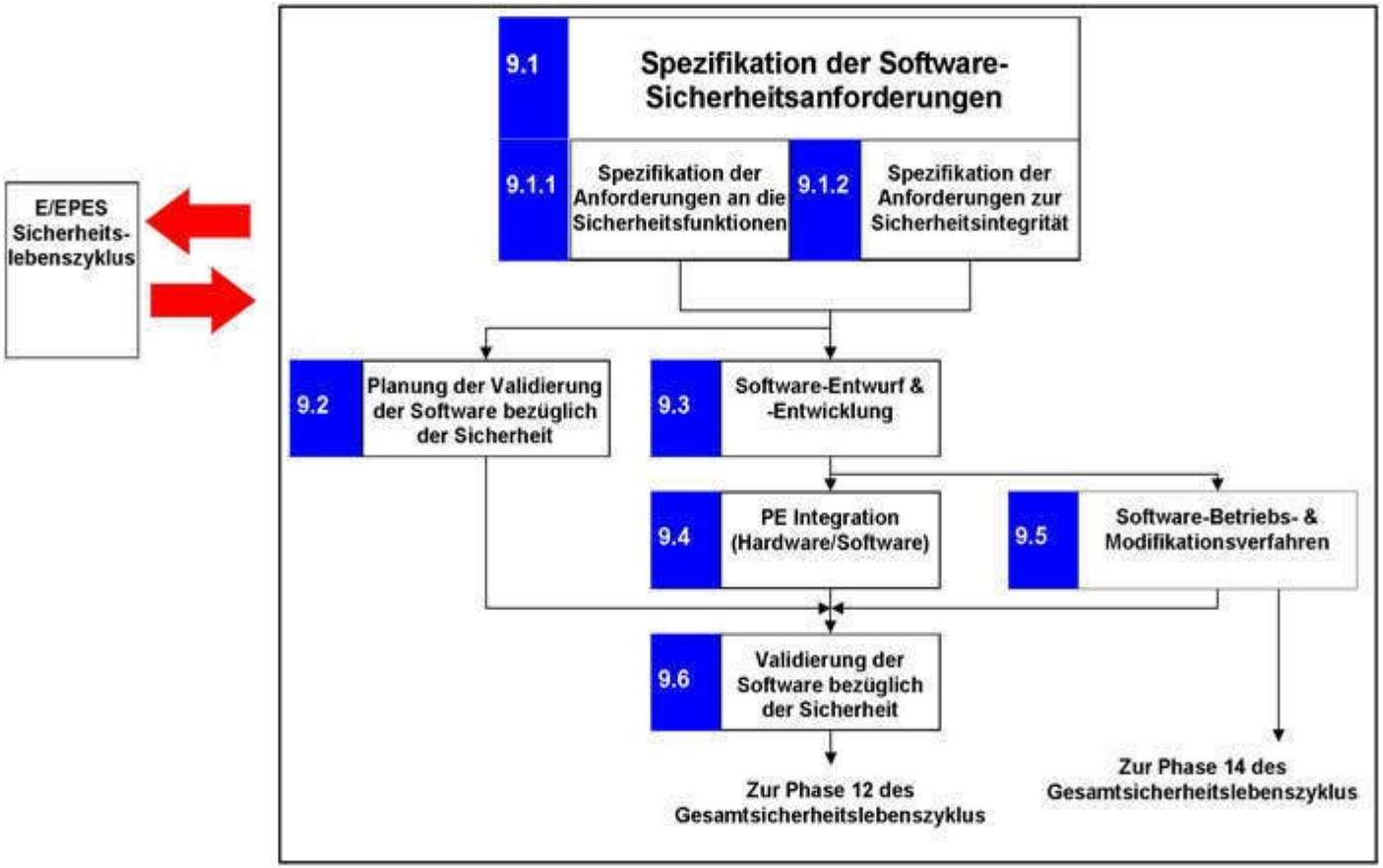
SIL	PFD (power)	RRF
1	10^{-5} - 10^{-6}	100.000-1.000.000
2	10^{-6} - 10^{-7}	1.000.000-10.000.000
3	10^{-7} - 10^{-8}	10.000.000-100.000.000
4	10^{-8} - 10^{-9}	100.000.000-1.000.000.000

- **Gesamtlebenszyklus**
  - **In allen Phasen des E/E/PES-Lebenszyklus müssen Maßnahmen zum Management der funktionalen Sicherheit, der Verifikation und der Beurteilung der funktionalen Sicherheit geplant, durchgeführt und dokumentiert werden.**



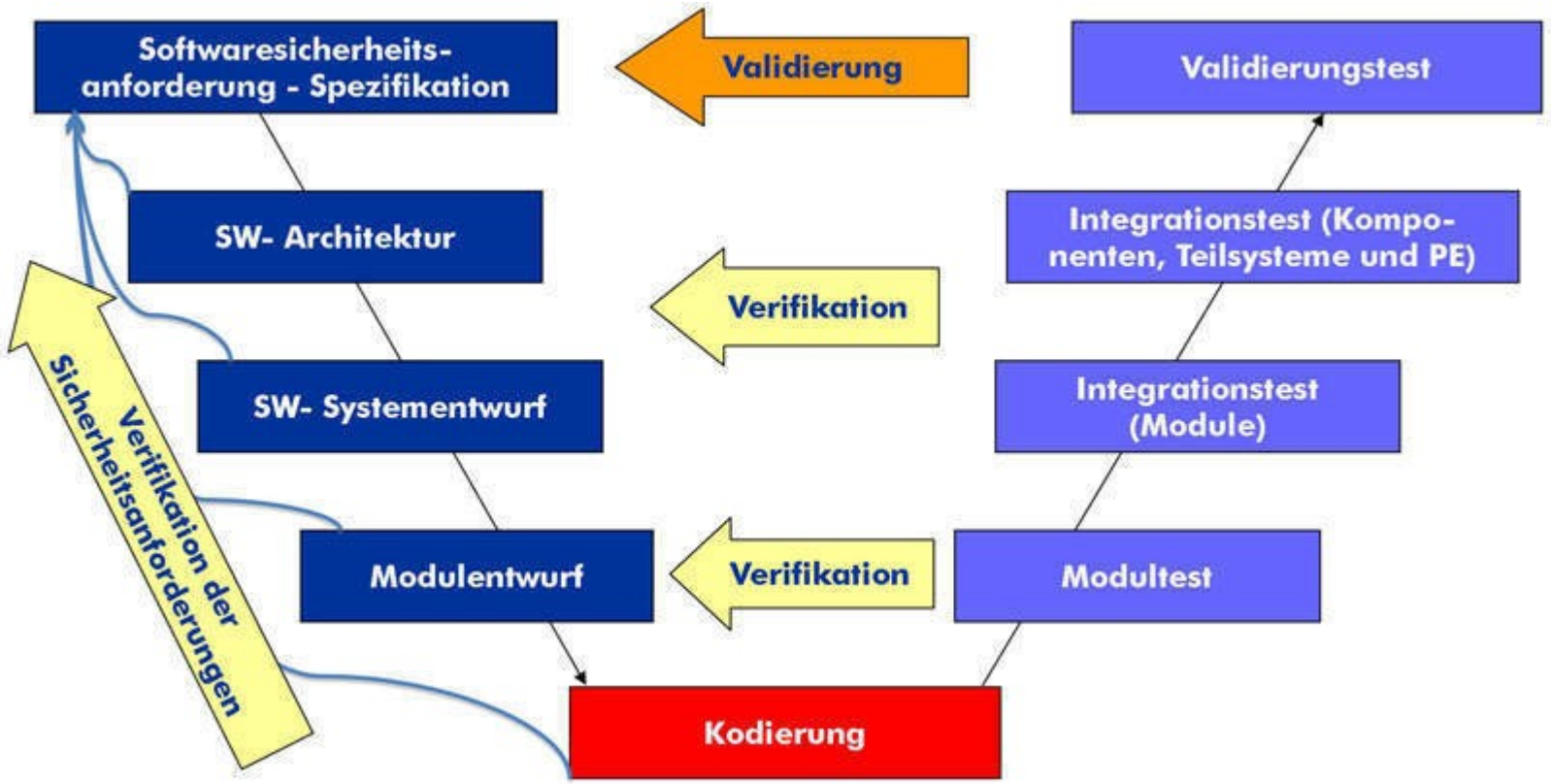
http://www.elektronikpraxis.vogel.de

- **Lebenszyklus von Software (61508 Teil 3)**



http://www.elektronikpraxis.vogel.de

- **V-Modell für SW-Sicherheitsfunktionen (61508 Teil 3)**



http://www.elektronikpraxis.vogel.de

Software-Fehler

- **Entwurfs- und Implementierungsfehler (systematische Fehler)**
- **Sicherheitskritische Software sind Programmteile, die unmittelbar oder mittelbar die Sicherheit eines Systems durch Fehlfunktionen beeinflussen können.**
- **Restfehler in Software:**

Applikation	Umfang der SW	Zahl der Fehler	Zahl der Restfehler	Schwerwiegende Restfehler
Autopilot einer Rakete	30 000	1 500	60	6
Navigationssystem des Space Shuttle	500 000	25 000	1 000	100
Flugkontrollsoftware für Europa und USA	1 000 000	50 000	2 000	200
SW zur Steuerung eines Kernkraftwerkes	1 500 000	75 000	3 000	300

http://www.elektronikpraxis.vogel.de

Anforderungen an SW-Entwicklungsprozess 7.2

- Definition von Techniken und Verfahren, wie sicherheitsgerichtete SW entworfen, implementiert, getestet und dokumentiert werden soll
- Je nach SIL sind dabei bestimmte Methoden der Beschreibung gefordert (z.B. semi- formale oder formale Methoden)
- IEC 61508 Teil-7 enthält Beschreibung der vorgeschlagenen Methoden, äquivalente Methoden und Verfahren sind erlaubt – z.B. UML 2.0
- Entwurf
  - Trennung zwischen sicherheitsgerichteten und anderen Programmteilen
  - Einsatz semi-formeller Methoden: Klassendiagramme, Block-Diagramme, Sequenz-Diagramme, Programmablaufpläne und/oder Zustandsautomaten

http://www.elektronikpraxis.vogel.de

Anforderungen an SW-Entwicklungsprozess 7.2

- **Implementierung**
  - **Nutzung bestimmter Programmiersprachen**
  - **Code Reviews, Walkthroughs**
  - **Statische Code-Analyse: z.B. Lint**

http://www.elektronikpraxis.vogel.de

Anforderungen an SW-Entwicklungsprozess 7.2

Table C.1 — Recommendations for specific programming languages

Programming language		SIL1	SIL2	SIL3	SIL4
1	Ada	HR	HR	R	R
2	Ada with subset	HR	HR	HR	HR
3	MODULA-2	HR	HR	R	R
4	MODULA -2 with subset	HR	HR	HR	HR
5	PASCAL	HR	HR	R	R
6	PASCAL with subset	HR	HR	HR	HR
7	FORTRAN 77	R	R	R	R
8	FORTRAN 77 with subset	HR	HR	HR	HR
9	C	R	---	NR	NR
10	C with subset and coding standard, and use of static analysis tools	HR	HR	HR	HR
11	PL/M	R	---	NR	NR
12	PLM with subset and coding standard	HR	R	R	R
13	Assembler	R	R	---	---
14	Assembler with subset and coding standard	R	R	R	R
15	Ladder Diagrams	R	R	R	R
16	Ladder Diagram with defined subset of language	HR	HR	HR	HR
17	Functional Block Diagram	R	R	R	R
18	Function Block Diagram with defined subset of language	HR	HR	HR	HR
19	Structured Text	R	R	R	R
20	Structured Text with defined subset of language	HR	HR	HR	HR
21	Sequential Function Chart	R	R	R	R
22	Sequential Function Chart with defined subset of language	HR	HR	HR	HR
23	Instruction List	R	---	NR	NR
24	Instruction List with defined subset of language	HR	R	R	R

HR Highly recommended if this technique or measure is not used, the reasons for not using it should be detailed during safety planning.

R Recommended as a lower measure to an HR recommendation.

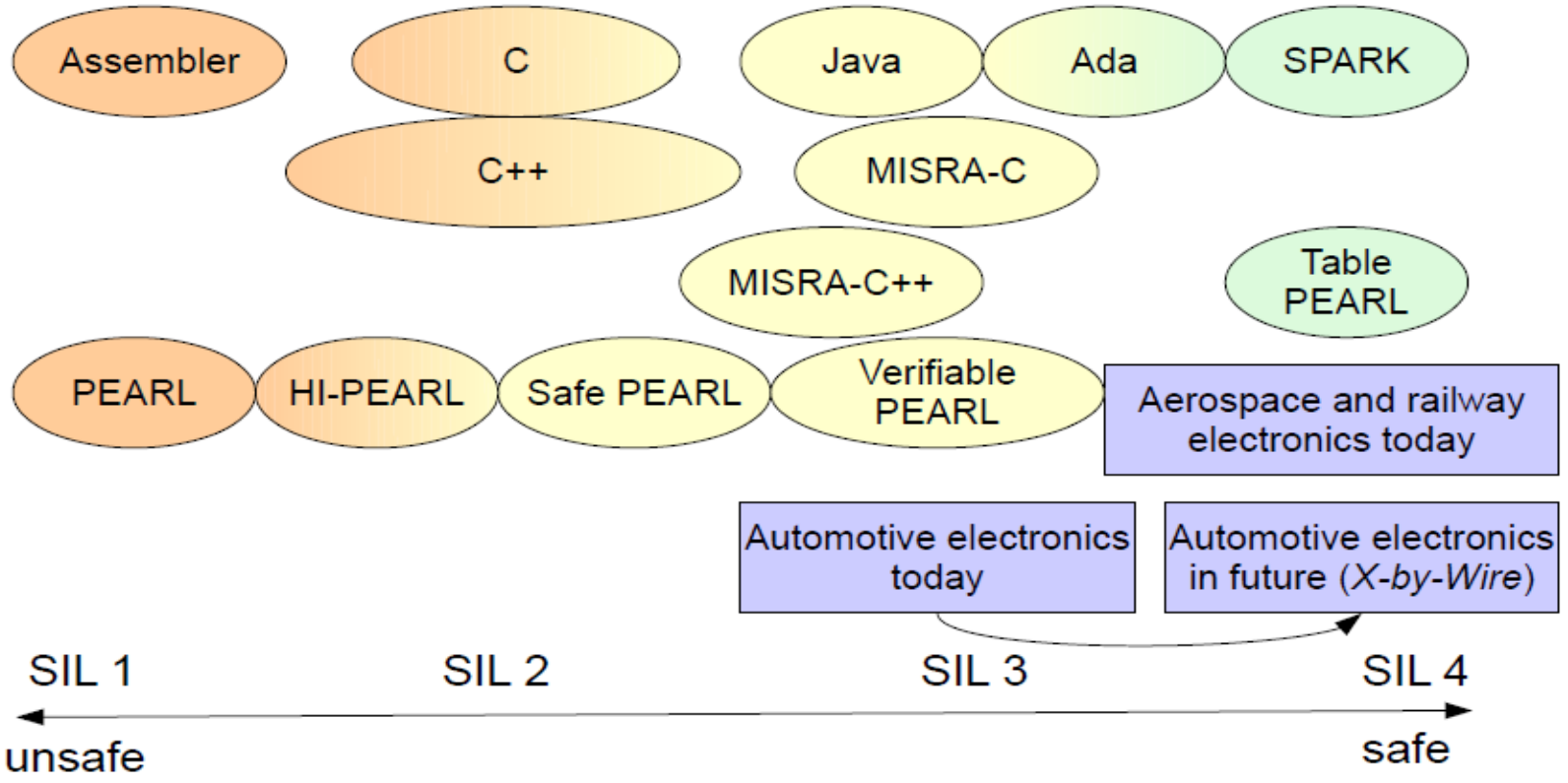
--- The technique has **no recommendation for or against** its use.

NR The technique or measure is positively **not recommended** for this safety integrity level – if it is used, the reasons for using it should be detailed during safety planning.

source: “IEC 61508-7: Functional safety of electrical/electronic/programmable electronic safety-related systems Part 7: Overview of techniques and measures“
Aus Folien Prof. Müller (Chemnitz)

Anforderungen an SW-Entwicklungsprozess 7.2

- **Zusammenfassung aus Folien Prof. Müller (Chemnitz)**



- **Zusammenfassung**
  - **C und C++ für sicherheitskritische Software nur als wohldefinierte Teilmengen mit Regeln und Guidelines und nur bis SIL 3**
  - **Dominant in der Automobilindustrie sind MISRA-C (2004) und MISRA-C++ (2008) mit SIL 3**
  - **Alternative Ada nur auf bestimmte Anwendungsgebiete beschränkt**
  - **Ansätze mit formaler Nachweisführung notwendig für SIL 4**

Anforderungen an SW-Entwicklungsprozess 7.2

- **Test**
  - **Modultests**
  - **Interface- und Datenfluss-Tests**
  - **Integrationstests mit vordefinierten Testfällen/-daten (Funktions- , Black Box-, White Box-Test)**
  - **Stress-, Reaktions-, Performance-Test**
  - **Alle Tests werden entsprechend Testplan durchgeführt**
- **Regressionstests nach Modifikationen**
  - **Beurteilung der Beeinflussung der Sicherheitsfunktion**
  - **Evtl. vollständiger neuer Nachweis**
- **Validierung und Verifikation**
  - **Testplanung und Testfälle aus Spezifikation abgeleitet**
  - **Simulation**
- **Dokumentation, Versionierung und Rückverfolgbarkeit der Ergebnisse**

http://www.elektronikpraxis.vogel.de

7.3 Programmiersprachen

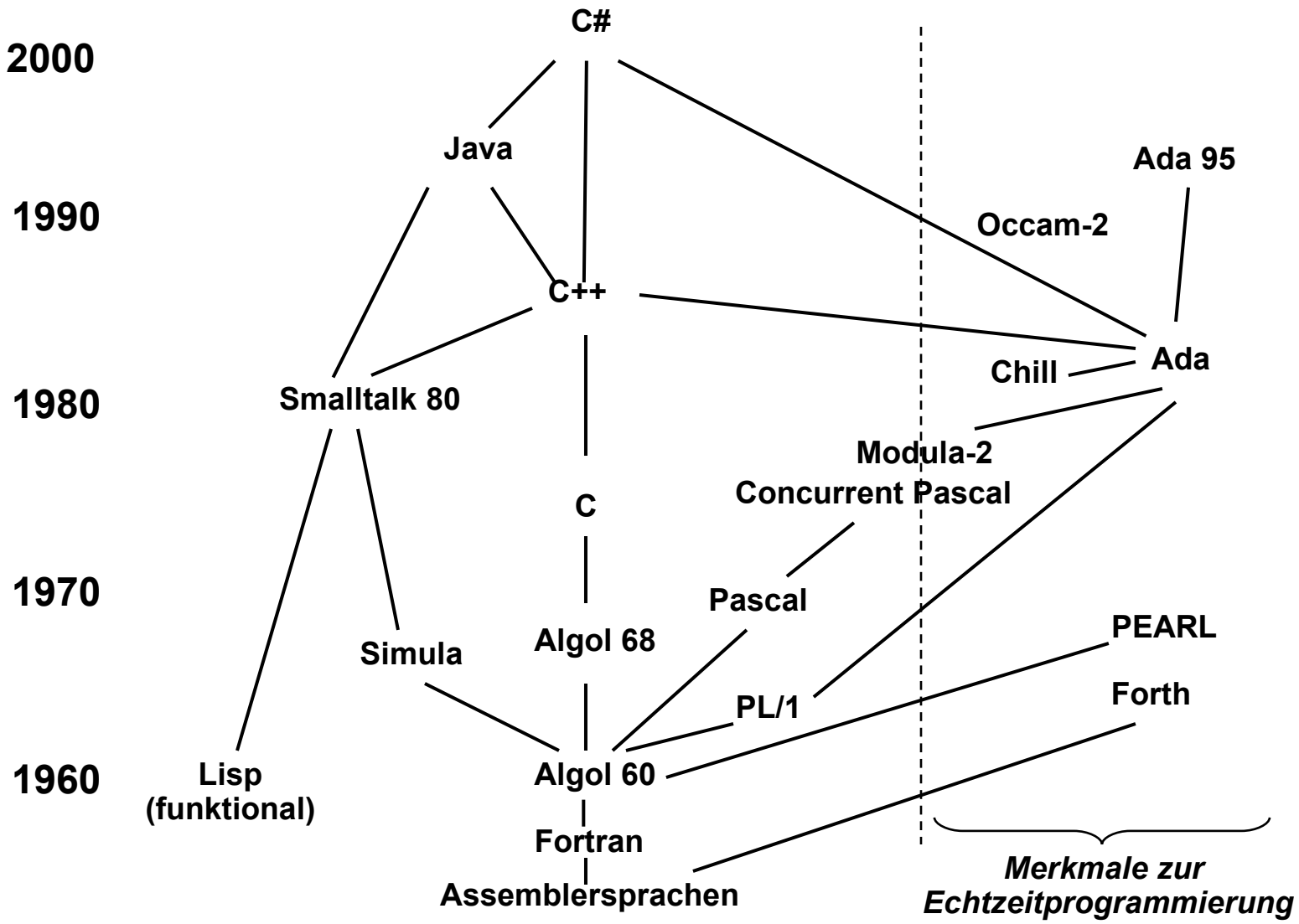
Gliederung

1. Einführung
2. MISRA-C / C++
3. ADA

7.3.1. Einführung

- Imperative Programmierung im Laufe der Zeit
  - '60er: prozedurale Abstraktion
Algol 60
  - '70er: strukturierte Programmierung
Simula (67), Algol 68, C, Pascal
  - '80er: Modularisierung
Modula, Ada
  - '90er: Objektorientierung
C++, Java
  - '00er: Web- und Plattformorientierung
PHP, Ruby, C#, VB.NET
(alle eigentlich schon späte '90er)

Historie (Ausschnitt)



Merkmale zur Echtzeitprogrammierung 7.3.1

- Zeitbedingungen
  - Formulierbarkeit
  - Überwachung der Einhaltung
- Parallelität und Synchronisation
  - Definition und Erzeugung nebenläufiger Prozesse (sporadisch, periodisch)
  - Mechanismen zur Interprozesskommunikation und Synchronisation
  - Einwirkung auf das Scheduling (z.B. durch Prioritäten)
- Anpassungsfähigkeit an spezielle E/A
- Prüfung auf Plausibilität
  - strenge Typprüfung
  - umfassende Fehlererkennung
- Unterstützung des Programmierens im Großen

Entwicklung der "Echtzeitsprachen" 7.3.1

- bis Mitte '70er: Meinung: Echtzeitanwendungen müssen in Assembler implementiert werden !
- Dann: Zwei Entwicklungstendenzen:
  - Verbreitete Wirtssprache + neue Konstrukte für Echtzeitprogrammierung
    - » Beispiele: Real-Time Fortran, Concurrent Pascal
    - » Problem: Wirtssprache behindert Entfaltung der echtzeitbezogenen Sprachmittel
  - Neue Programmiersprache für die Bedürfnisse der Echtzeitprogrammierung
    - » Beispiele: Forth, PEARL
    - » Problem: Nische = keine breite Akzeptanz
- Idee: Neue Universalsprache mit Merkmalen der Echtzeitprogrammierung in den Grundkonzepten ⇒ Ada

Entwicklung der "Echtzeitsprachen" (2) 7.3.1

- Aber: Starke Hinwendung zu C/C++ auch für Echtzeitanwendungen / Kritische Anwendungen
  - obwohl keine Unterstützung dafür in der Sprache!
  - Gründe:
    - » komfortable Entwicklungsumgebungen
    - » erfahrene Entwickler verfügbar
    - » Hinwendung zu offenen Systemen
- Ada wird nur dort angewendet, wo dies vom Auftraggeber vorgeschrieben wird:
  - Militärische und zivile Luft- und Raumfahrt
- Selbst dort C++ möglich:
  - Joint Strike Fighter Air Vehicle (Lockheed Martin)
  - Vgl. JSF C++ Coding Standards (2005)

Bedeutung von C / C++ 7.3.1

- C und C++ gehören zu den derzeit gebräuchlichsten Programmiersprachen
- C (ab 1972)
  - Imperativ, strukturiert
  - Geringe Anzahl an Schlüsselwörtern
  - Portabel, effizient (Bedeutung für Embedded Systems)
  - Für Systemprogrammierung sehr gut geeignet
- Probleme
  - Wenig geeignet für sicherheitskritische Anwendungen
  - z.B. kann fremder Speicher überschrieben werden
  - Viele Anfängerfehler (== / =, && / &, Setzen ;, Pointer, Pointer-Arithmetik)
  - Nicht behoben in C90 oder C99

Kritische Probleme mit C

- *Nicht-spezifiziertes* Verhalten
  - Auswertungsfolge in Ausdrücken
  - Wandlung von int in enumeration
  - Bereichsverletzung
- *Undefiniertes* Verhalten
  - Aktueller Parameter entspricht nicht vorgesehenem Typ
- *Implementierungsabhängiges* Verhalten
  - Probleme bei Portierung
  - Länge des signifikanten Teils eines Bezeichners
  - Ist char signed oder unsigned?
- *Internationalisierung*
  - Länderabhängige Spezifika (z.B. Zeichensatz)

C++ (ab 1983)

- **Pros:**
  - **Höhere Abstraktionsebene und konstruktive Mächtigkeit als C**
  - **Objektorientierung und Generizität (Templates)**
  - **Portierbarkeit (breite Verfügbarkeit von guten Compilern)**
  - **Trotzdem hohe Effizienz des Codes**
  - **Ausgebildete Entwickler, Werkzeugunterstützung, Solide Erfahrungen**
- **Cons:**
  - **Komplex, hoher Lernaufwand**
  - **Kompromisse durch Kompatibilität zu C**
  - **Viele ähnliche Probleme**

Auf C und C++ aufbauende Sprachen 7.3.1

- Java (1995)
  - Bessere Portabilität durch JVM
  - Vermeidung von Zeigerarithmetik
  - Garbage Collection
    - » Vereinfacht Programmierung
    - » Nicht Echtzeit-fähig da nicht deterministisch
- Embedded C++ / EC++ (1996, Rev.3 1999)
  - Einschränkungen für C++
    - » Keine Mehrfachvererbung, keine Exceptions, keine virtuellen Basisklassen, keine dynamischen Typen, keine *_cast-Operatoren, keine Namensräume, keine Templates, …
  - Geringe Verbreitung, primär Japan
  - Starke Kritik von Stroustrup (besser: Coding Standards verwenden!)

Auf C und C++ aufbauende Sprachen (2) 7.3.1

- C# (2001, Rev. 4.0 in 2010, ISO/IEC 23270)
  - Microsoft-orientiert (Lock-In-Gefahr)
  - Konzepte aus vielen anderen Sprachen
  - Objektorientierung
  - Starke statische Typisierung
  - Zeiger nur in „unsicherem Code“ erlaubt
  - Auswertung von Metadaten (z.B. für Methoden) zur Laufzeit
  - Trennung von .NET z.T. schwierig

Auf C und C++ aufbauende Sprachen (3) 7.3.1

- D (2007, www.dlang.org)
  - Erfinder: Walter Bright
  - Einflüsse aus C, C++, Java, C#, Python
  - Imperativ, objektorientiert, funktional (innerhalb imperativer Sprache), parallel, generisch, modular
  - Starke statische Typisierung
  - Zeiger möglichst vermieden, aber Nutzung möglich
  - Vermeidung von Zeiger-Arithmetik
  - Überladbare Operatoren
  - Module, Design by contract
  - Templates
  - Garbage Collection, aber global oder klassenbezogen deaktivierbar und durch eigene Speicherverwaltung ersetzbar
  - Maschinennahe Programmierung möglich

Richtlinien-Ansatz

- Verbreitung von C/C++ förderte andere Sichtweise für Programmierung von sicherheitskritischen und Echtzeit-Anwendungen:
  - Versuche das Problem auf der Engineering-Ebene zu lösen, nicht auf der Programmiersprachen-Ebene
  - Guter Programmierstil durch Beschränkungen statt neuer Sprache
- Ziele:
  - Effizienz
  - Geringer Speicherbedarf
  - Portabilität
  - Echtzeitfähigkeit
  - Weiterverwendung existierender Werkzeuge wie Compiler, …
  - Überprüfbarkeit der Regelmengen (z.B. lint)

7.3.2. MISRA-C / MISRA-C++

http://www.misra.org.uk/

- **MISRA-C am weitesten verbreiteter Standard für C in sicherheitskritischen Anwendungen**
- **Erste Version 1998 veröffentlicht, ursprünglich auf Automobilindustrie beschränkt**
  - **„Guidelines for the use of the C language in Vehicle Based Software“**
- **MISRA-C: 2012:**
  - **Basis C99**
  - **„Guidelines for the use of the C language in critical systems“, (MIRA Limited)**
  - **„safe“ subset von C mit 143 verpflichtenden und 16 empfohlenen Regeln**
  - **Dokumentierte Ausnahmen vom Standard möglich**

MISRA-C / MISRA-C++ (2)

- Einfluss von MISRA-C auf JSF C++
- MISRA C++ (2008):
  - Guidelines for the use of the C++ language in critical systems

Beispiele für MISRA-C Regeln

1 All code shall conform to ISO 9899 standard C, with no extensions permitted

9 Comments shall not be nested

10 Sections of code should not be commented out

11 Identifiers (internal and external) shall not rely on significance of more then 31 characters. Furthermore the compiler/linker shall be checked to ensure that 31 character significance and case sensitivity are supported for external identifiers

13 The basic type of char, int, short, long, float and double should not be used, but specificlength equivalents should be typedef'd for the specific compiler, and these type names used in the code

14 The type char shall always be declared as unsigned char or signed char

19 Octal constants (other than zero) shall not be used

20 All object and function identifiers shall be declared before use

21 Identifiers in an inner scope shall not use the same name as an identifier in an outer scope and therefore hide that identifier

30 All automatic variables shall have been assigned a value before being used

45 Type casting from any type to or from pointers shall not be used

50 Floating point variables shall not be tested for exact equality or inequality

52 There shall be no unreachable code

Beispiele für MISRA-C Regeln (2) 7.3.2

53 All non-null statements shall have a side-effect

54 A null statement shall only occur on a line by itself, and shall not have any other text on the same line

55 Labels should not be used, except in switch statements

56 The goto statement shall not be used

57 The continue statement shall not be used

58 The break statement shall not be used (except to terminate the case of a switch statement)

59 The statements forming the body of an if, else if, else, while, do... while or for statement shall always be enclosed in braces

60 All if, else if constructs should contain a final else clause

61 Every non-empty case clause in a switch statement shall be terminated with a break statement

62 All switch statements should contain a final default clause

63 A switch expression should not represent a Boolean value

64 Every switch statement shall have at least one case

65 Floating point variables shall not be used as loop counters

69 Functions with variable numbers of arguments shall not be used

70 Functions shall not call themselves, either directly or indirectly

Richtlinien für C++

- MISRA -C++
  - Echte Teilmenge von C++, als geeignet für sicherheitskritische Anwendungen
  - 228 Regeln, kaum Beschränkung der charakteristischen Leistungsmerkmale von C++ (Exceptions, Templates erlaubt)
  - Nutzt viele Regeln von MISRA-C
- Beispiele
  - Präprozessor
    - » Nur für include-guards
    - » Defines für Konstanten oder Makros verboten (→ const, enumerations, inline functions)
  - Variablen müssen initialisiert sein
  - Keine unbenutzten Variablen

Alternative Richtlinien für C++

- JSF C++
  - Nutzt z.T. Regeln von MISRA-C
  - Keine Nutzung von Exceptions
  - Nutzung von Software-Metriken zur statischen Code-Beurteilung
  - …
- Ref: Joint Strike Fighter Air Vehicle C++ Coding Standards for the System Development and Demonstration Program, Lockheed Martin Corporation, Dec. 2005

Fazit

- MISRA-C, MISRA-C++ bilden Ansatz, durch Richtlinien die verbreiteten Sprachen C und C++ sicherer zu machen
- MISRA-C sehr etabliert, MISRA-C++ noch nicht
- MISRA-C, MISRA-C++ erlauben die vertretbare Nutzung von C/C++ bis SIL Level 3
  (für SIL 4 formalerer Ansatz notwendig)

7.3.3. Ada

- **Historie**
  - **Auftraggeber: US DoD (Department of Defense)**
  - **Anforderungsbeschreibungen: Strawman (1975), Woodenman (1975), Tinman (1976), Ironman (1977), Steelman(1978)**
  - **Sprachvorschläge verschiedener Institutionen**
  - **Auswahl von 4 Vorschlägen und deren vollständige parallele Entwicklung (nach Farben benannt)**
  - **Auswahl des Vorschlag „Green“ (CII Honeywell Bull, Frankreich, Leitung J. Ichbiah), 1979**
  - **Preliminary Language Reference Manual (PreLRM), 1980**
  - **Standard Language Reference Manual (LRM), 1983, Basis des Ada83-Standards (ANSI/MIL-STD 1815A-1983)**

Ada (2)

- Mehrjähriges Ada9X-Projekt führt 1994/95 zu Ada95 (ISO865)
  - Erweiterungen für Objektorientierung
  - Distributed Programming
- Version Ada 2005
  - macht Anleihen aus Java:
    Interfaces (mit Mehrfachvererbung)
  - Positive Erfahrung durch Lockheed Martin zu Flugsicherungssystem vor Liefertermin fertig und unter geplantem Budget
- Aktuelle Version 2012

Ada (3)

- Ada Ravenscar Tasking Profile
  - `pragma profile(Ravenscar)`
  - Sichere Nebenläufigkeit vonTasks
  - Garantie für
    - » Determinismus
    - » Möglichkeit zur statischen Analyse
    - » Task-Einplanbarkeit
    - » Speicherbeschränktheit
  - Verbietet u.a. Rendezvous-Nutzung für Tasks
  - Zertifizierung bis SIL Level 4 möglich

Ada (4)

- SPARK Ada
  - Strikte Untermenge von Ada mit Annotationen
  - Versionen für Ada 83 und Ada 95
  - Math. Korrektheitsbeweise von Programmen durch statische Analyse mit entsprechenden Werkzeugen möglich
  - RavenSPARK enthält Ravenscar-Profil
  - Von vielen Autoren als sehr gut geeignet für SIL-4-Anwendungen angesehen
- Aktuell SPARK 2014, basiert auf Ada 2012
  - Schwerpunkt Korrektheitsbeweise
  - Sprachvereinfachungen zur besseren Lesbarkeit
  - Verbesserungen bzgl. Concurrency für Multicore-Prozessoren
  - Vgl. Ada Information Clearinghouse: www.adaic.org

Historie

- Namensursprung
  - Ada Augusta Byron,
    Countess of Lovelace (1816-56)
  - Mitarbeiterin von Charles Babbage
  - "the world's first programmer"
- Sprache ist frei verfügbar
  - z.B. im Gegensatz zu Java
  - Jeder Compiler, der sich Ada Compiler nennen will, muss umfangreiche Validierung bestehen
  - Beispiel: GNU Ada Translator (GNAT) frei verfügbar
- Ada Programming Support Environment (APSE)
  - angepasste Entwicklungswerkzeuge
  - Ada Runtime Environment
  - Bibliotheken

Aus Wikipedia

Allgemeine Merkmale von Ada 83

- Objektbasiertheit (abstrakte Datentypen):
  - abgegrenzte Einheiten mit wohldefinierten äußeren und inneren Eigenschaften
  - Wertemenge, Operationenmenge
  - keine Vererbung
- Typ-Konzept und -Erweiterungskonzept
- Trennung von Spezifikationsteilen und zugehörigen Rümpfen (information hiding, Impl.-Unabhängigkeit)
- Exception Handling
- Unterstützung für Programmierung im Großen
  - Module
  - getrennt übersetzbare Einheiten
  - generische Module, Unterprogramme (ähnlich Templates)

Elementarbeispiel


```
with Basis_EA;                    -- Benutzung einer EA-Bibliothek
use  Basis_EA;
procedure GGT is
  subtype MyInteger is Integer range -100 .. 100;
  x,y : MyInteger;
begin
  Schreibe("Bitte den 1.Paramater eingeben: ");
  Lies(x);
  -- Eingabe y
  while x /= y loop
    if x > y then x := x-y;
             else y := y-x;
    end if;
  end loop;
  -- Ausgabe
exception
  when Constraint_Error =>
       -- Fehlerbehandlung für Bereichsverletzung
  when others =>
       -- Fehlerbehandlung für sonstige Fehler
end GGT;
```

Echtzeitmerkmale in Ada 83

- **Zeitbedingungen**
  - **Zugriff auf Echtzeituhr**
  - **Zeitdauern: Typ `duration`**
  - **Zeitpunkte: Typ `time`**
  - **Zeitfunktionen: `package calendar` einschließlich Rechnen + und - mit diesen Typen**
  - **Verzögerungs-Anweisung**
    - » **`delay <zeitraum>`**
    - » **`delay until <zeitpunkt>` (in Ada 95 zusätzlich)**
  - **insgesamt nur schwache Festlegungen in Ada 83**
- **Prüfung auf Plausibilität**
  - **Sehr strenge Typprüfung**
  - **möglichst vollst. Prüfung zur Compile-Zeit**

Echtzeitmerkmale in Ada 83 (2)

- **Parallelität und Synchronisation**
  - **nebenläufige Prozesse durch hierarchisches `task`-Konzept auf Sprachebene mit Verankerung im Ada Runtime-System**
  - **Unterstützung für statische Task-Prioritäten auf Ebene von `pragmas` als Subtyp von `integer`**
  - **gemeinsamer Mechanismus zur Interprozesskommunikation und -Synchronisation durch *Rendezvous*-Konzept**

Beispiel: Prozesstypen, Prozesse


```
with Basis_EA;
use  Basis_EA;
procedure Produzenten is
  task type Producer (ausgabe : character;
                      anzahl  : natural := 10);

  task_A : Producer('A');     -- Mit default Anzahl 10
  task_B : Producer('B', 5); -- Mit Anzahl 5

  task body Producer is
    begin
    for i in 1 .. anzahl loop
      Schreibe(ausgabe);
    end loop;
  end Producer;

begin
  -- ausfuehrbarer Teil der umgebenden Prozedur
end Produzenten;
```

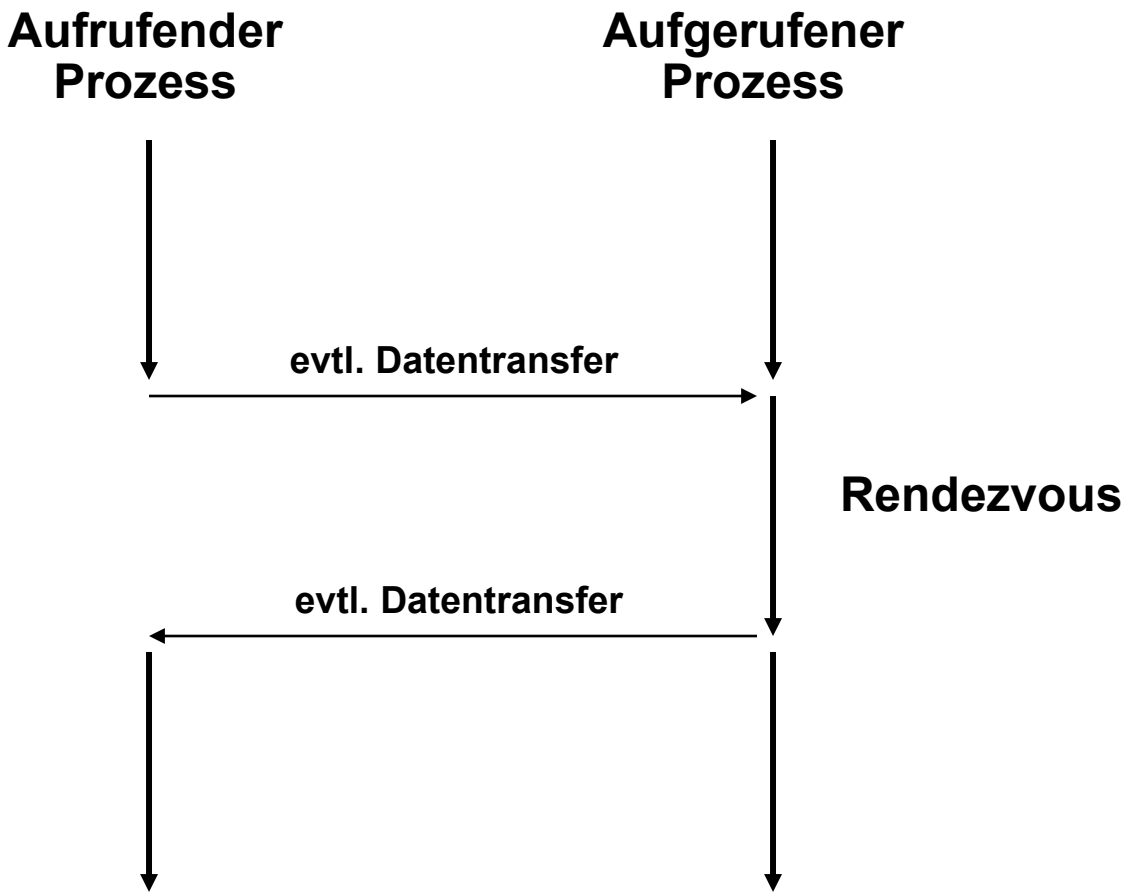
Beispiel: dynamische Prozesserzeugung 7.3.3

```
procedure system is
  -- Deklaration des Prozesstyps Producer

  type ProducerRef is access Producer;  -- typisierter Zeiger-Typ

  myTaskRefs : array (1..10) of ProducerRef;

begin
  ...
  myTaskRefs(4) := new Producer('C', 100);
       -- task startet unmittelbar jetzt
       -- Lebensdauer nicht an umgebenden Block gebunden

end system;
```

Rendezvous-Konzept

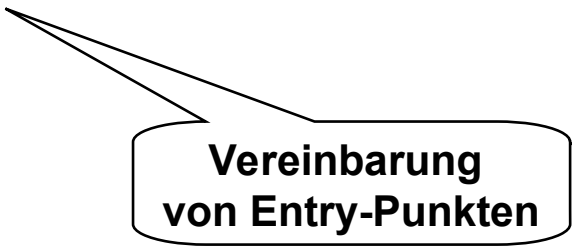
- synchron
  - keine Pufferung
  - wechselseitiges Warten auf den Partner
  - Austausch von typisierten Parametern
- asymmetrisch
  - Aufrufer setzt Entry-Call auf einen bestimmten Eingang des Aufgerufenen/Empfängers ab
  - Aufgerufener signalisiert Bereitschaft zum Rendezvous durch Annahme oder Auswahl-Anweisung (`accept` bzw. `select`-Statement)
- einseitig-anonym
  - Aufgerufener weiß nicht, wer ihn aufruft
- evtl. Anhebung der Priorität des nieder prioren Prozesses eines Rendezvous

Rendezvous-Konzept (2)



Beispiel: Task Entry-Punkte


```
task type buffer is
  entry inbuffer(c : in character);
  entry outbuffer(c : out character);
end buffer;
```



Beispiel: Task Entry-Punkte (2)


```
task body buffer is
  N : constant integer := 1000;
  b : integer range 0..N;
  i,j : integer range 1..N;
  speicher : array (1..N) of character;
begin
  b := 0;   i := j := 1;
  loop
    select
      when b<0 =>  -- Speicher nicht voll
        accept inbuffer(c : in character) do
          speicher(i):=c;
        end inbuffer;
        b := b+1;
        i := (i mod N) + 1;
    or
      when b>0 =>  -- Speicher nicht leer
        accept outbuffer(c : out character) do
          c:=speicher(j);
        end outbuffer;
        b := b-1;
        j := (j mod N) + 1;
    end select;
  end loop;
end buffer;
```

Rendezvous

folgende Verarbeitung

Beispiel: Task Entry-Punkte (2)


```
task body buffer is
  N : constant integer := 1000;
  b : integer range 0..N;
  i,j : integer range 1..N;
  speicher : array (1..N) of character;
begin
  b := 0;   i := j := 1;
  loop
    select
      when b<0 =>  -- Speicher nicht voll
        accept inbuffer(c : in character) do
          speicher(i):=c;
        end inbuffer;
        b := b+1;
        i := (i mod N) + 1;
    or
      when b>0 =>  -- Speicher nicht leer
        accept outbuffer(c : out character) do
          c:=speicher(j);
        end outbuffer;
        b := b-1;
        j := (j mod N) + 1;
    end select;
  end loop;
end buffer;
```

Rendezvous

folgende Verarbeitung

Erweiterte allgemeine Merkmale in Ada 95 7.3.3

- **Objektorientierung durch *tagged types*, von denen (einfach) abgeleitet werden kann und die dabei erweitert werden können.**
- **Typklassen (Vereinigung aus einem Typ T und allen Typen, die aus T direkt oder indirekt abgeleitet sind) mit polymorphen Typen und impliziten Konvertierungen**
- **Abstrakte Unterprogramme, die in abgeleiteten Klassen definiert werden müssen**

Erweiterte Echtzeitmerkmale in Ada 95 7.3.3

- Interrupt-Behandlung auf Basis von geschützten Prozeduren
- Pakete zum Identifizieren von Tasks und zur Verwaltung von Task-Attributen
- Unterbrechbares Task-Scheduling mit Basis-Prioritäten und aktiven Prioritäten
- Prioritäts-gesteuerte Warteschlangen mit der Möglichkeit system- oder benutzerdefinierter Alternativ-Strategien
- Priority Inheritance Protocol bei allen geschützten Operationen
- Dynamische Prioritäten durch spezielles Paket

Erweiterte Echtzeitmerkmale in Ada 95 (2) 7.3.3

- **Protected Types zum koordinierten Zugriff auf gemeinsam benutzte Daten**
  - **spezielle Zugriffsfunktionen (protected operations)**
  - **nur lesender Zugriff ohne Einschränkungen**
  - **beliebiger Zugriff mit wechselseitigem Ausschluss**
  - **Entry-Calls eines Rendezvous mit Blockierung, bis eine Bedingung (Condition) wahr wird**
- **Funktionalität war auch in Ada 83 mit dem vorhandenen Rendezvous-Konzept erreichbar, jetzt natürlicher und komfortabler.**

- Weiterer Standardisierungsprozess (bis 2007)
  - Java-ähnliche Interfaces (inheritance of interfaces) statt echte Mehrfachvererbung wie in C++
  - Real-Time and High-Integrity Support
    - » task termination handlers,
    - » timing events, execution-time clocks,
    - » alternative task dispatching policies,
    - » standardization of the Ravenscar restricted tasking profile
  - ...