Software Qualität

- Einführung
- Software Fehler
- Konstruktive Qualitätssicherung
- Software Test
- Statische Analyse

Konstruktive Qualitätssicherung

- Software Richtlinien
- Typisierung
- Vertragsbasierte Programmierung
- Fehlertolerante Programmierung
- Portabilität
- Dokumentation

Konstruktive Qualitätssicherung

- Software Richtlinien
- Typisierung
- Vertragsbasierte Programmierung
- Fehlertolerante Programmierung
- Portabilität
- Dokumentation

Software Richtlinien

Richtlinien regeln den Gebrauch einer Programmiersprache über die eigenen syntaktischen und semantischen Regeln hinaus.

Motivation:

- Vereinheitlichung
- Fehlerreduktion

Software Richtlinien

- Notationskonventionen
- Sprachkonventionen

Software Richtlinien

- Notationskonventionen
- Sprachkonventionen

Notationskonventionen

Notationskonventionen werden auf verschiedenen Ebenen definiert (Projekt, Sprache, Betriebssystem,..)

Typischerweise betroffen:

- Auswahl und Schreibweise von Bezeichnern
- Einrückungen, Verwendung von Leerzeichen
- Aufbau von Kontrollstrukturen
- Dokumentation

Notationsstile

- **Pascal Case:** Bezeichner, sowie jedes enthaltene Wort startet mit Großbuchstaben.
- **Camel Case:** Bezeichner startet mit Kleinbuchstaben, jedes weitere Wort groß.
- **Uppercase:** Komplett in Großbuchstaben
- **Lowercase:** Komplett in Kleinbuchstaben

Ungarische Notation

Jeder Variablenname besteht aus zwei Teilen:

- **Präfix:** abkürzende Schreibweise für den Datentyp
- **Qualifier:** frei gewählter Name

Bsp:

```
Button butOK;
ListBox lbColorSelector;
CheckBox cbRemindMe;
```

Problem: Verstoß gegen das Single Source Prinzip

Zitat

Linus Torvalds:

> Encoding the type of a function into the name (so called Hungarian notation) is brain-damaged – the compiler knows the types anyway and can check those., and it only confuses the programmer. No wonder Microsoft makes buggy programs.

Coding Style

Die folgenden Coding Styles enthalten Notationskonventionen, gehen aber deutlich darüber hinaus.

C# Coding Style:

- siehe https://msdn.microsoft.com/de-de/library/ff926074.aspx

Java Coding Style: verschiedene Varianten: z.B.

- http://google-styleguide.googlecode.com/svn/trunk/javaguide.html
- http://www.ambysoft.com/essays/javaCodingStandards.html
- Siehe auch: *Michael Inden, Der Weg zum Java Profi, Seite 1160*

GNU Coding Standards for C:

- *https://www.gnu.org/prep/standards/html_node/Writing-C.html#Writing-C*

Software Richtlinien

- Notationskonventionen
- Sprachkonventionen

Sprachkonventionen

Bisher: Layout und Syntax,

Jetzt: Semantische Besonderheiten einer Sprache

Bsp: MISRA-C: Programmierstandard ➔ http://www.misra-c.com

(MISRA: Motor **I**ndustry **S**oftware **R**eliability **A**ssociation)

Der MISRA-C-Programmierstandard definiert eine Untermenge des Sprachumfangs von C, d.h. er umfasst Richtlinien die zu einer Qualitätssteigerung (insbesondere der Softwarequalitätsaspekte der Zuverlässigkeit und Wartbarkeit) in der Software-Entwicklung führen sollen.

Hintergrund von MISRA

Weite Verbreitung von C gerade auch in sicherheitskritischen Bereichen,

ABER

- **Verhalten teils undefiniert.**
- **C macht es leicht, die Sprache falsch zu gebrauchen.**
- **C erlaubt schwer verständliche Konstrukte.**
- **C überlässt dem Entwickler die Fehlerbehandlung zur Laufzeit.**

➔ MISRA unterstützt Entwickler bei der Entwicklung speziell sicherheitskritischer Systeme

Vision von MISRA

The MISRA C Guidelines define a subset of the C language in which the opportunity to make mistakes is either removed or reduced.

Umfang von MISRA

1. **Empfehlungen zu**

   - Tool Selection
   - Projekt Aktivitäten
   - Implementierung der MISRA Compliance

2. **Guidelines (Directives and Rules)**

Beispiele für MISRA Richtlinien

- Konstanten in einem vorzeichenlosen Kontext müssen mit einem U-Suffix versehen werden.
- Variablen vom Typ float (Gleitkommazahlen) sollen nicht mit den Vergleichsoperatoren == oder != getestet werden.
- goto soll nicht verwendet werden.
- magic numbers vermeiden und stattdessen sinnvoll benannte Konstanten verwenden: #define MAXSIZE 12.
- Division durch null verhindern: if (b!=0) a/=b;
- Compilerunabhängigkeit sicherstellen, z. B. shiften neg. Zahlen: -3<<4 ==> -3*(1<<4)
- Operatorrangfolgen sind nicht trivial, daher Klammern verwenden: (a && b || c) ==> ((a && b) || c).
- Rekursion darf in keiner Form auftreten (weder indirekt noch direkt).

MISRA Regelkategorien

Regeln	Kategorie	Regeln	Kategorie
(1.1) - (1.5)	Übersetzungsumgebung	(12.1) - (12.13)	Ausdrücke
(2.1) - (2.4)	Spracherweiterungen	(13.1) - (13.7)	Kontrollstrukturen
(3.1) - (3.6)	Dokumentation	(14.1) - (14.10)	Kontrollfluss
(4.1) - (4.2)	Zeichensatz	(15.1) - (15.5)	Switch-Konstrukt
(5.1) - (5.7)	Bezeichner	(16.1) - (16.10)	Funktionen
(6.1) - (6.5)	Datentypen	(17.1) - (17.6)	Pointer und Arrays
(7.1)	Konstanten	(18.1) - (18.4)	Struct und Union
(8.1) - (8.12)	Deklarationen und Definitionen	(19.1) - (19.17)	Präprozessor
(9.1) - (9.3)	Initialisierung	(20.1) - (20.12)	Standardbibliotheken
(10.1) - (10.6)	Typkonversion (Arithmetik)	(21.1)	Laufzeitfehler
(11.1) - (11.5)	Typkonversion (Pointer)		

Auszug aus dem MISRA Regelsatz

Regel	Beschreibung
(1.4)	"The compiler/linker shall be checked to ensure that 31 character significance and case sensitivity are supported for external identifiers."
(2.2)	"Source code shall only use `/* ... */` style comments."
(3.2)	"The character set and the corresponding encoding shall be documented."
(4.2)	"Trigraphs shall not be used."
(5.1)	"Identifiers (internal or external) shall not rely on the significance of more than 31 characters."
(6.4)	"Bit fields shall only be defined to be of type `unsigned int` or `signed int`."
(7.1)	"Octal constants (other than zero) and octal escape sequences shall not be used."
(8.5)	"There shall be no definitions of objects or functions in a header file."
(9.1)	"All automatic variables shall have been assigned a value before being used."
(10.6)	"A U-suffix shall be applied to all constants of unsigned type."
(11.3)	"A cast should not be performed between a pointer type and an integral type."
(12.3)	"The `sizeof`-Operator shall not be used on expressions that contain side effects."
(13.3)	"Floating-point expressions shall not be tested for equality or inequality."
(14.1)	"There shall be no unreachable code."
(15.3)	"The final clause of a `switch` statement shall be the `default` class."
(16.1)	"Functions shall not be defined with variable numbers of arguments."
(17.4)	"Array indexing shall be the only allowed form of pointer arithmetic."
(18.4)	"Unions shall not be used."
(19.6)	"`#undef` shall not be used."
(20.12)	"The time handling functions of library `<time.h>` shall not be used."
(21.1)	"Minimisation of run-time failures shall be ensured by the use of at least one of: a) static analysis tools/techniques; b) dynamic analysis tools/techhniques; c) explicit coding of checks to handle run-time faults."

Software Richtlinien

- Sprachkonventionen in Java Programmen
  - In Firmen
  - In Projekten

- Beispiel. Google Java Style:
  - http://google-styleguide.googlecode.com/svn/trunk/javaguide.html

Empfehlung: keine neuen Styles erfinden, sondern bestehende verwenden.

Gefahren

Soziale Aspekte bei der Durchsetzung von Konventionen durch Codereviews

- Diejenigen, die die Konventionen am besten kennen, sind die Buhmänner.
- Der Gereviewte fühlt sich persönlich angegriffen.
- Der Reviewer will deswegen nichts mehr anmerken.

➔ Empfehlungen zum Vorgehen siehe nächste Seite

Durchsetzung von Konventionen

- Alle Beteiligten an der Erarbeitung der Konventionen mitarbeiten lassen.
- Konventionen gemeinsam weiterentwickeln.
- Begründete Ausnahmen akzeptieren (und dokumentieren)
- Toolgestützt prüfen (sowohl am Arbeitsplatz als auch beim Build Prozess)
- Regeln *zu Beginn* vereinbaren.
- Bestehende Regeln nur mit Bedacht ändern.