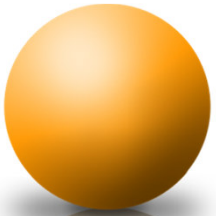


# Etablierung von Clean Code im sicherheitskritischen Umfeld

Ein Erfahrungsbericht von

Jan Haulsen

Senior Consultant,
BeOne Frankfurt GmbH

BeOne

# Expertise

- Das Thema Clean Code interessiert und motiviert mich seit vielen Jahren
- Langjährige Praxis als Software-Entwickler in drei Geräteentwicklungs-Projekten (Laufzeit > 5 Jahre, Teams > 15 Personen)
- Erfahrung als Coach für das Thema bei mehreren Projekten

# Senior Consultant Jan Haulsen (20 Jahre Berufserfahrung)

Studium: Elektrotechnik

Sprachen:
Deutsch, Englisch

**Kenntnisse**

- Softwareprojekte in technischer Produktentwicklung
- Objektorientierte Programmierung C++
- Embedded Software, auch sicherheitskritisch
- Zertifizierter Systems Engineer (CSEP)
- Zertifizierter Testmanager ISTQB Foundation Level
- Zertifizierter SCRUM Master
- Coaching durch Pair Programming  und Reviews
- Modellbasierte Entwicklung mit UML
- Rapid Prototyping mit Test-GUIs (Qt Framework)
- Software Engineering: Clean Code, Refactoring
- Konfigurationsmanagement
- Prozessverbesserung: CMM

**Werkzeuge**

- Visual Studio, Visual Basic
- Eclipse mit Qt
- ClearCase, Subversion
- Rhapsody C++
- Betriebssyteme MS Windows, Linux, VxWorks

**Branchen**

- Medizintechnik
- Professionelle Fernseh- und Videotechnik (Mechatronik und Signalverarbeitung),

# Was nun besprochen werden soll

- Es ist ein Erfahrungsbericht aus einem laufenden Projekt im sicherheitskritischen Umfeld
- Es wurde in der aktuellen Projektphase erreicht, dass der Clean Code Aspekt in neu entstehenden Code einfließt – dies war vorher kaum ein Thema
- Der Weg zu diesem Erfolgt führte über
  - ✓ den Einsatz von statischer Codeanalyse
  - ✓ die intensive Diskussion mit den Entwicklern

# Entwicklungsumfeld - Projektvorgaben

- Medizintechnik Geräteentwicklung
- Sicherheitskritische Software
  *Klasse C gemäß IEC 62304: Tod oder schwere Verletzung möglich*
- Mehrere Scrum Teams an mehreren Standorten

- Hohe Anforderungen an die Verifikation:
  - ✓ Verfolgbarkeit von Anforderungen zur Umsetzung
  - ✓ Änderungen nur nach Beauftragung (Change Request)
    → **Starke Einschränkung für spontanes Refactoring**
  - ✓ Durchführung von Prüffällen nach jeder Änderung, inklusive Regression
    → **Starke Einschränkung auch für geplantes Refactoring**

# Entwicklungsumfeld - Infrastruktur

- Einsatz von statischer Codeanalyse (PC-Lint + Axivion)
  - ✓ MISRA Regeln
  - ✓ Architektur
  - ✓ Namensregeln
  - ✓ Zyklische Funktionsaufrufe (‚Cycles‘)
  - ✓ Duplikate
  - ✓ Metriken

  Duplikate und Metriken haben den stärksten Bezug zu Clean Code Aspekten

- Regelmäßiger Software-Build:
  Mehrere Durchläufe pro Tag mit Ergebnissen der Codeanalyse

- Strenge Einhaltung von Prozessen, u. a. Durchführung von Codereviews mit Hilfe eines skriptbasierten Review-Assistenten

# Clean Code Verbesserung durch Reviews (1)

- Durch die toolbasierte **statische Codeanalyse** sind viele Aspekte eines Codereviews bereits überprüft
- Erweiterung des Fokus bei den Codereviews auf **nichtfunktionale Aspekte**
- Es bieten sich viele Gelegenheiten, **mit dem Autor** die gewählte Implementierung aus der Clean Code Sichtweise **zu diskutieren** und dabei jeweils passende Vorschläge zu machen
- Wichtig:
  Es geht dabei oft nur um die Diskussion der Vorschläge!
  Am Ende der Diskussion steht häufig die Entscheidung, den Code nicht zu ändern, weil
  - ✓ die gewählte Lösung völlig akzeptabel ist
  - ✓ der Testaufwand einer Änderung zu hoch ist
  - ✓ der Implementierungsaufwand zu hoch ist und nicht beauftragt wird
  - ✓ Sicherheits- oder Performanceaspekte gegen die Änderung sprechen

# Clean Code Verbesserung durch Reviews (2)

- Diskutieren, nicht belehren!
- Eine gute Diskussion führt zu sehr positiven Effekten:
  - ✓ das Wissen über Clean Code Techniken kommt in die Entwicklungsgruppe
  - ✓ die Akzeptanz für angeregte Codeänderungen steigt
  - ✓ neu geschriebener Code hat eine bessere Qualität
- Wenn diese Reviews von nur wenigen Personen durchgeführt werden, bekommen diese Personen einen guten Überblick über die unterschiedlichen Programmierstile. Mit diesem Wissen
  - ✓ können Best Practices einfach erkannt und kommuniziert werden
  - ✓ können gezielt Schulungsmaßnahmen geplant werden
- Dies ist jedoch eine zusätzliche Rolle, die die üblichen Peer Reviews nur ergänzt!

# Was funktioniert hat

- Social Engineering:
  - ✓ Den Entwickler in Ruhe den Code erklären lassen und erstmal nur zuhören
  - ✓ Positives Feedback, wenn Clean Code Anwendung erkannt wurde
  - ✓ Argumente gegen Clean Code in konkreten Fällen gelten lassen
- Software Engineering:
  - ✓ Eine einführende Schulungsmaßnahme, die die wesentlichen Elemente von Clean Code vorstellt und auf das geplante Vorgehen bei den Reviews hinweist
  - ✓ Schaffung einer Rolle „Teamübergreifender Reviewer“
  - ✓ Betrachtung der Metriken
  - ✓ Skriptbasierter Review-Assistent

# Was NICHT funktioniert hat

- Der esoterische Ansatz von clean-code-developer.de
- Die Diskussion über Clean Code ausschließlich über einen Vortrag zu führen
- Hoher Aufwand für die automatische Überprüfung von Namen
- Das Verkennen von Sachverhalten, die gegen Clean Code sprechen

# MISRA Regeln

- Ein Regelwerk für die Vermeidung von problematischen Software-Konstrukten.
- Ursprünglich 1998 von der **M**otor **I**ndustry **S**oftware **R**eliability **A**ssociation für die Sprache C entwickelt.
- Seit 2008 zwar auch für C++ erweitert, aber der Schwerpunkt der Regeln bezieht sich auf die strukturierte Programmierung.
- Diese Regeln können von als Konfiguration für Tools zur statischen Codeanalyse verwendet werden.
- Eine Auswahl von Tools:

✓ PC-Lint
✓ CppCheck
✓ Axivion
✓ Klocwork
✓ Eclipse Plugins
✓ …

# Beispiele für MISRA Meldungen

```
146                 // Beim Wiedereinlesen sind die Einträge alt, werden aber für das Fortsetzen
147                 // erstmal mit in die aktive Liste übernommen
148                 if (Time.getTimestamp() < static_cast< time_t >(SystemDate_Object.getDate_s_1() - 10))
149                 {
150                     ActiveDataList .setValue(DatabaseId, DataLength, Buffer, Time);
```

| | |
|---|---|
| Entity | record |
| Message | Violates MISRA C++ 2008 Required Rule 5-0-8, Cast of integer cvalue expression to larger type |

```
457             AddressDb = pDataBaseObject->getAddress();
458
459             while (Bytes-- != 0)
460             {
461                 if (*AddressDb != Drivers::NOVRAM_ReadByte(AddressNovram))
462                 {
```

| | |
|---|---|
| Entity | check |
| Message | Possible use of null pointer 'AddressDb' in argument to operator 'unary *' [Reference: file source/ line 539; file source/ Class.cpp: line 457] |

```
360                 }
361         } while (sizeof(SoundSchema_Class::Sound_Enum) == BytesRead);
362
```

| | |
|---|---|
| Entity | threadImplementation |
| Message | Violates MISRA C++ 2008 Required Rule 5-0-4, Implicit conversion changes signedness |

```
498                     if (Data::QualificationTest_Class::getTestSwitch() == 4712UL)
499                     {
500                         const sint32 Latency = Timer.sleepUntilNextInterval(20000L); // [usec]
501                     }
502                     else if (Data::QualificationTest_Class::getTestSwitch() == 4713UL)
503                     {
```

| | |
|---|---|
| Error number | 438 |
| Entity | threadImplementation |
| Message | Last value assigned to variable 'Latency' (defined at line 500) not used [MISRA C++ Rule 0-1-6] |

```
514
515                 } while ((CurrentIndex < PlaybufferSize) && (false == Stop_.get()));
516
517                 if (true == Stop_.get())
```

| | |
|---|---|
| Entity | threadImplementation |
| Message | Violates MISRA C++ 2008 Required Rule 5-14-1, side effects on right hand of logical operator: '&&' |

# Metriken – Clean Code ist messbar

- Ziel der Erfassung von Metriken ist die Erhöhung der Software-Güte.
- Eine einheitliche Definition von Software-Güte existiert nicht.
- Ansatz ist es, Softwarestrukturen mittels Metriken zu analysieren.

„Was wir nicht messen, können wir auch nicht steuern“

- Metriken helfen dabei
  - ✓die Software objektiv zu bewerten
  - ✓Ansatzpunkte für sinnvolle Restrukturierungen zu finden

Niedrige Messwerte deuten auf Clean Code hin.

# Metriken – Beispiele für messbare Werte

- **Data Abstraction Coupling:** Anzahl von klassenbasierten Membervariablen, ein Maß für die Komplexität einer Klasse
- **Depth Inheritance Tree:** zählt alle Klassen, von der eine Klasse erbt. Ein großer Wert erhöht das Risiko, dass der Überblick über den Funktionsumfang verloren geht.
- **Lines of Code:** Ein Indiz für die Änderungsfreundlichkeit
- **McCabe Complexity:** Die zyklomatische Komplexität ist ein Maß für die Komplexität eines Moduls. Der Wert entspricht Anzahl minimal nötiger Testfälle für vollständige Abdeckung. Nicht geeignet für objektorientierte Strukturen.

# Metriken – Komplex bleibt komplex

- Es ist nicht möglich, alle Metrikwerte klein zu halten.
- Z. B. bedeuten kleine Klassen auch viele Klassen, eine geringe McCabe Complexity bedeutet mehr Methoden.
- Abweichungen vom vereinbarten Grenzwert sind daher auch wirklich nur Abweichungen und keine Verstöße.
- Beim Versuch, die Metriken zu optimieren, kann es oft zu Verschlimmbesserungen kommen.



# Metriken – Beispiel Data Abstraction Coupling

DAC ist ein Maß für die Komplexität einer Klasse.

Es ist die Anzahl von klassenbasierten Membervariablen.

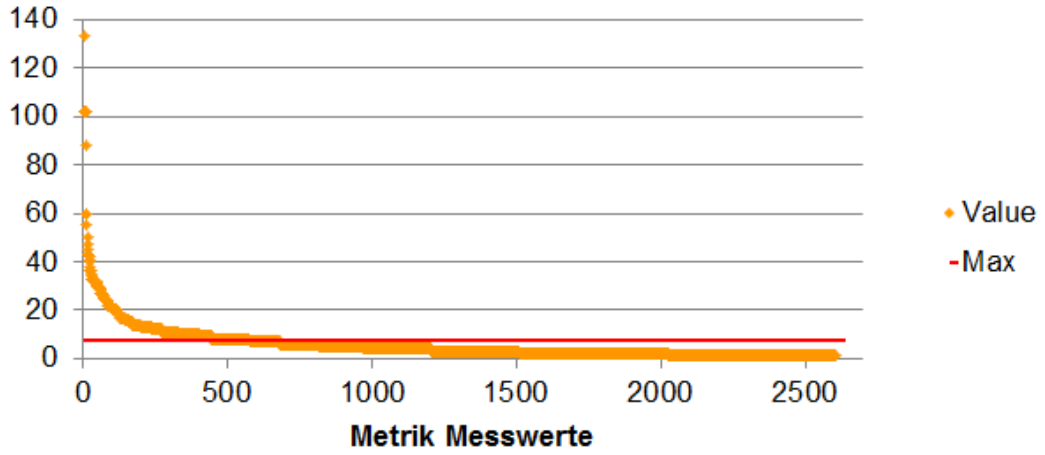
In der Annahme, dass diese Membervariablen mehr oder weniger miteinander zu tun haben müssen, steigt die Komplexität einer Klasse exponentiell mit diesem Wert.

Wenn die o.g. Annahme nicht zutrifft, ist die Klasse nicht gut entworfen.

Metriken im Projekt:

Min 0, Max 7

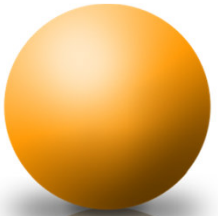
23% überschreiten Max



# Metriken – Die richtige Auswahl

# Fazit

- Im vorgestellten Projekt hat die Mischung gestimmt
  - ✓ Funktionierende Prozesse
  - ✓ Toolbasierte statische Codeanalyse
  - ✓ „Social Engineering“: Diskussionen beim Review
- Bei sicherheitskritischer Software sind die Möglichkeiten des Refactoring begrenzt. Der Weg führt hier stärker über die Einsicht des Entwicklers, bei neuen Implementierungen besser zu werden.
- Richtig konfigurierte Statische Codeanalyse lohnt sich
- Die Erfassung und Auswertung von Metriken ist dabei das stärkste Element in Bezug auf Clean Code



# Fragen / Diskussion

# Ihr Ansprechpartner


**Jan Haulsen**

**BeOne Frankfurt GmbH**
Liebigstraße 19
60323 Frankfurt

Tel.: +49 (0) 69 24 70 39 60
Mobil: +49 (0) 151 11 35 50 70

jan.haulsen@beone-group.com
**www.beone-group.com**