Wie Clean Code zum Teamsport wird

Von der Vision zur Mission

Hi

Halina

[Senior Software Entwickler]

Clean Code ➡ Team ?

Was ist eigentlich Clean Code?

Ist das nicht subjektiv?

“simple to read
and
painless to maintain“

lesbar
verständlich
wartbar
testbar

Was sagen Entwickler?

Was verstehst Du unter Clean Code?

Batman: “- SoC
- KISS
- YAGNI
- SLA
- SRP
- DEMETER
- DRY

Robin: “ Hmmm, gleiche Einrückung?

… Also Leerzeilen und Umbrüche sowas?“

namen…
Tests.
“

- boy scout rule

…
lufthol …

Woher kommen die unterschiedlichen Sichten auf Clean Code?

individuelles Fachwissen
&
individuelle Erfahrung

Dreyfus Model

Brüder Stuart und Hubert Dreyfus, 1980

Dreyfus Model

5 levels of skill aquisition

Novice ➡ braucht Regeln

Advanced Beginner ➡ testet Regeln

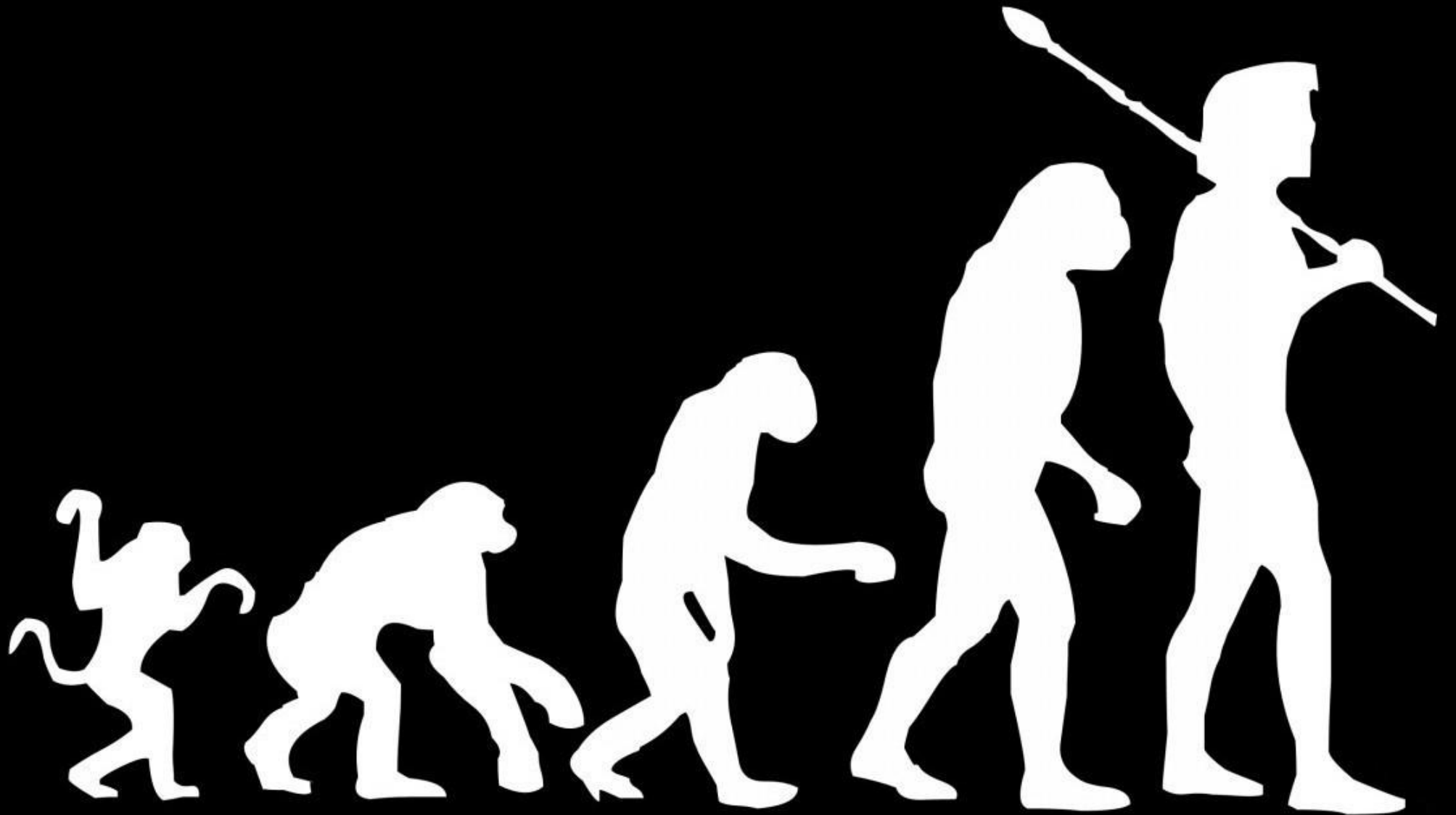
Competent ➡ wendet Regeln an

Proficient ➡ greift auf Regeln zurück

Expert ➡ weit über Regeln hinaus

Dan North, GOTO 2017

Woran jetzt jeder denkt:

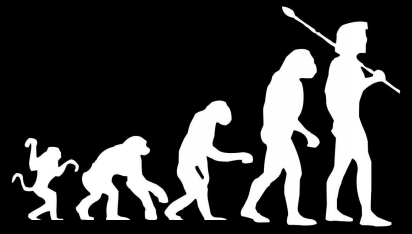


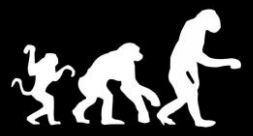
Moment!

Hier geht es um
einzelne Fähigkeiten

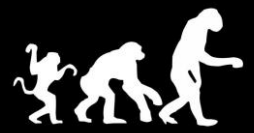
Robin

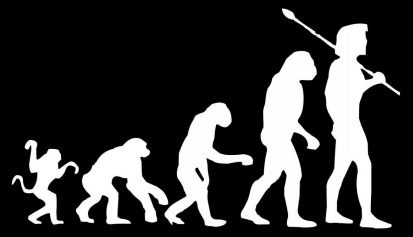
C 

Java 

JS 

Ruby 

PHP 

C++ 

Dreyfus Squared

Dan North, GOTO 2017

	Novice	Advanced Beginner	Competent	Proficient	Expert
Novice			✗		✗
Advanced Beginner		✗			
Competent					
Proficient					✗
Expert					

Berücksichtigen bei

- Onboardings / Einführungen
- Trainings / Workshops
- Diskussionen / Meetings
- Fragen / Antworten
- Code Reviews
- Teamzusammenstellung / Pairings

Ok.

Also je nach Level geht jeder anders mit Regeln um oder kennt sie noch gar nicht.

Betrachten wir wieder unser Team …

Welche Regeln?

Coding Guidelines!

Eigene Guidelines einfach festlegen?

Nur sinnvoll, wenn das Team die Guidelines ernst nimmt

Schritt 1

Sicherstellen, dass jeder im Team versteht, warum Guidelines so wichtig sind

Schritt 2

Von Anfang an jeden im Team bei der Erstellung der Guidelines involvieren

Vorteile

- Code Ownership
- Man kann Dinge nachschlagen
- Keine Streitereien zwischen Einzelnen
- Neuzugang kann sich einfach einlesen
- Lesbarer Code, als hätte man ihn selbst geschrieben
- Positiver Effekt auf Wartbarkeit, wenn Guidelines eingehalten werden ➡ weniger Kosten
- Weniger erfahrene Entwickler können sich leichter orientieren und besser dazulernen

Was legt man da denn überhaupt fest?

Was sind Coding Guidelines?

- Formatierung
- Kommentare
- Umfang von Klassen & Funktionen
- Testing
- Datei Organisation
- Naming Conventions
- Architektur Guidelines

... und, und, und

Ist das nicht viel?

Wie machen das andere?

Google Java Style Guide

Table of Contents

1 Introduction
- 1.1 Terminology notes
- 1.2 Guide notes

2 Source file basics
- 2.1 File name
- 2.2 File encoding: UTF-8
- 2.3 Special characters

3 Source file structure
- 3.1 License or copyright information, if present
- 3.2 Package statement
- 3.3 Import statements
- 3.4 Class declaration

4 Formatting
- 4.1 Braces
- 4.2 Block indentation: +2 spaces
- 4.3 One statement per line
- 4.4 Column limit: 100
- 4.5 Line-wrapping
- 4.6 Whitespace
- 4.7 Grouping parentheses: recommended
- 4.8 Specific constructs

5 Naming
- 5.1 Rules common to all identifiers
- 5.2 Rules by identifier type
- 5.3 Camel case: defined

6 Programming Practices
- 6.1 @Override: always used
- 6.2 Caught exceptions: not ignored
- 6.3 Static members: qualified using class
- 6.4 Finalizers: not used

7 Javadoc
- 7.1 Formatting
- 7.2 The summary fragment
- 7.3 Where Javadoc is used

1 Introduction

This document serves as the **complete** definition of Google's coding standards for source code in the Java™ Programming Language. A Java source file is described as being *in Google Style* if and only if it adheres to the rules herein.

ASP.NET Reference

CodeDOM Quick Reference

Configuration File Schema

Design Guidelines for Developing Class Libraries

- Guidelines for Names
- Type Design Guidelines
- Member Design Guidelines
- Designing for Extensibility
- Design Guidelines for Exceptions
- Usage Guidelines

MSBuild Reference

Performance Counters

Unmanaged API Reference

Additional Managed Reference Topics

Design Guidelines for Developing Class Libraries

Visual Studio 2010 | Other Versions

The design guidelines for developing class libraries are for library development that extends and interacts with the .NET Framework. The goal of the .NET Framework design guidelines is to help library designers ensure that their users reap the benefits of API consistency and ease of use by providing a unified programming model that is independent of the programming language used for development. It is strongly recommended that you follow these design guidelines when developing classes and components that extend the .NET Framework. Inconsistent library design adversely affects developer productivity and discourages adoption.

These guidelines are intended to help class library designers understand the trade-offs between different solutions. There might be situations where good library design requires that you violate these design guidelines. Such cases should be rare, and it is important that you have a clear and compelling reason for your decision.

For more information on design guidelines, see the "Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries" book by Krzysztof Cwalina and Brad Abrams, published by Addison-Wesley, 2005.

In This Section

Guidelines for Names
Describes guidelines for naming types and members in class libraries.

IN THIS ART

In This Secti

See Also

Coding style

This document attempts to explain the basic styles and patterns used in the Mozilla codebase. New code should try to conform to these standards, so it is as easy to maintain as existing code. There are exceptions, but it's still important to know the rules!

This article is particularly for those new to the Mozilla codebase, and in the process of getting their code reviewed. Before requesting a review, please read over this document, making sure that your code conforms to recommendations.

Article navigation

1. Naming_and_formatting_code.
2. General practices.
3. C/C++ practices.
4. JavaScript practices.
5. Java practices.

Coding Guidelines festlegen

pro Sprache / Projekt

- Bestehende Guidelines als Basis auswählen

  ➡ Vorlage suchen (Sprache/Framework)
- Nur festhalten, wo das Team davon abweichen will
- Den ersten Entwurf im Team präsentieren

  ➡ per Mail, Confluence und/oder Meeting
- Offene Diskussion & Zeit für Erklärungen
- Commitment von allen einholen
- Gemeinsam (!) “Strafen” bei Regelverstoß festlegen

Moment, Strafen?

Wieder zurück...

Coding Guidelines festlegen

pro Sprache / Projekt

- Bestehende Guidelines als Basis festlegen

  ➡ Vorlage suchen (Sprache/Framework)
- Nur festhalten, wo das Team davon abweichen will
- Den ersten Entwurf im Team präsentieren

  ➡ per Mail, Confluence und/oder Meeting
- Offene Diskussion & Zeit für Erklärungen
- Commitment von allen einholen
- Germeinsam(!) “Strafen” bei Regelverstoß festlegen
- regelmäßig prüfen/aktualisieren/erweitern

Wie trifft man solche Entscheidungen im Team?

Methoden

Gerald M. Weinberg's - "Becoming a Technical Leader".

- **Voting**
  ohne große Diskussion abstimmen
- **Consensus**
  Team diskutiert, bis sich alle auf eine Lösung einigen
- **The Strong Leader**
  Diskussion im Team, Entscheidung durch Leader.
  Leader-Rolle kann wechseln

Voting

Gerald M. Weinberg's – "Becoming a Technical Leader"

Pro

- Schnelle Ergebnisse
- Gefühl der Gleichberechtigung
- Hilfreich, wenn keiner Verantwortung übernehmen will
- Gut, wenn es keinen Spezialisten zu dem Thema gibt
- Entscheidungen selten total schlecht

Kontra

- Oft kein optimales Ergebnis
- Ohne Diskussion kein Lerneffekt

Consensus

Gerald M. Weinberg's – "Becoming a Technical Leader"

Pro

- Großartige Ergebnisse
- Hoher Lerneffekt durch Diskussion

Kontra

- Sehr zeitintensiv
- Wird schnell anstrengend, wenn Meinungen zu weit auseinander liegen

Strong Leader

Gerald M. Weinberg's – "Becoming a Technical Leader"

Pro

- Schnelle Ergebnisse
- Großartige Ergebnisse, wenn der Leader tiefes Fachwissen hat

Kontra

- Schlechte Ergebnisse, wenn der Leader keine guten Fachkenntnisse hat und er andere Meinungen nicht berücksichtigt
- Keine Gleichberechtigung führt vielleicht zu dem Gefühl, weniger wichtig zu sein

Es muss nicht immer ein Meeting sein:

- Slack
- Confluence
- Handzeichen
- Entscheidungs-Log

etc.

Tipps für Team-Entscheidungen

- Wenn man auf Widerstand stößt, Moderator-Rolle an den „Rebell“ abgeben
- Stillere Teammitglieder einbeziehen und nach ihrer Meinung fragen
- Wenn man gemeinsam entscheidet, ist das ganze Team dafür verantwortlich

  ➡ keine Blame Kultur
- Wenn über ein komplexes Thema entschieden werden muss, vorab ein Zeitkontingent zur Einarbeitung festlegen
- Meinungen entschärfen:

  „Wie seht ihr das?“

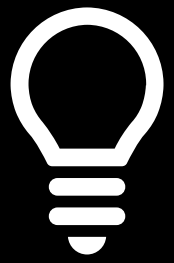
  „Lasst uns den anderen Ansatz zusammen durchspielen!“

Gut

Guidelines festgelegt

Und wie läuft das jetzt jeden Tag?

Wir müssen doch Features bauen!?



Automatisieren, was geht!

Statische Codeanalyse

- Tools bereits in der Entwicklungsumgebung einrichten
- Statische Codeanalyse der Testsuite vorschalten

- ✓ Spart Arbeit
- ✓ Frühes Feedback
- ✓ Die Tools kritisieren den Code, nicht andere Teammitglieder
- ✓ Deckt Probleme auf, ein Mensch vielleicht nicht wahrnimmt
- ✓ Lerneffekt

Und was ist damit, das die statische Codeanalyse nicht finden kann?

Code Reviews!

Warum?

Vorteile

- Konsistenter Code
- Code Ownership
- Wissen verteilen
- Teamstandards erhöhen
- Entwickler geben sich mehr Mühe, wenn Reviews anstehen
- Hands on learning: dein Code, dein Anwendungsfall
- Weniger Security Issues
- Weniger Bugs

Wie?

Review Guidelines

Review Checkliste

➡ im Team erarbeiten und updaten

- Allgemein (Coding Standards)
- Performance
- Security
- Dokumentation
- Testing (ausreichend und sinnvoll)

Grundregeln

- Wirklich konstruktiv sein
- Zeit nehmen für Erklärungen
- Immer etwas Gutes finden
- Harte Kritik entschärfen:
  - ➡ „ … ist nicht so gut, was meinst Du?“
  - ➡ „Ich glaube, es wäre besser, weil …“
- Reviews vor dem Mergen abschließen (Dod)
- Anderen für Kritik danken
- Lob / Begeisterung für gute Ansätze

Wie kann man sonst noch Wissen im Team verteilen?

Wege, Wissen im Team zu verteilen

- Code Reviews ✓
- Pair-Programming
- Workshops
- (e-learning) Kurse / Clean Code Videos
- Brown Bag Lunch
- Buch Club
- Programmierbier
- Dokumentation

Findet einen Weg, der zu eurem Team passt!

Zusammengefasst:

- Bewusstsein für verschiedenen Lernstufen oder Erfahrungsstufen schärfen
- Coding Guidelines etablieren
- Entscheidungsprozess im Team optimieren
- Code Reviews einführen
- Soviel automatisieren wie möglich
- Austausch fördern

Organisiert Euch!

Danke für‘s Zuhören

Ressources:

Dan North, GOTO 2017 - https://www.youtube.com/watch?v=lvs7VEsQzKY
Brüder Stuart und Hubert Dreyfus, 1980 - Dreyfus Model
Gerald M. Weinberg’s book “Becoming a Technical Leader“
https://team-coder.com/boosting-the-quality-of-team-decisions/

Bilder:

https://www.publicdomainpictures.net/en/view-image.php?image=129416&picture=darwin-evolution
https://creativecommons.org/publicdomain/zero/1.0/
https://www.buildabear.de/mode/haarreif-mit-hasenohren/a-22348/
https://pixabay.com/de/service/terms/#usage
https://pixabay.com/de/gugelhupf-kuchen-gebacken-2842827/
https://pixabay.com/de/marke-kreuz-falsch-nr-abstimmung-39951/