WHITEPAPER
CLEAN CODE.

SNELLER WERKEN,
MINDER FRUSTRATIE.

WHITEPAPER
CLEAN CODE.

SNELLER WERKEN EN VEEL MINDER FRUSTRATIE

Clean Code zou je kunnen definiëren als code die gemakkelijk te begrijpen is. Zowel voor het systeem als voor de software developer. Klinkt logisch? Toch hebben developers in de praktijk vaak te maken met code die niet clean is. Hoe dat komt lees je in dit artikel.

WAT IS CLEAN CODE?

Clean Code is geen exacte wetenschap. In zijn boek "Clean Code" geeft schrijver Robert C. Martin een set regels, maar in deze regels zit veel ruimte voor eigen interpretatie. Martin raadt aan om de regels in gedachten te houden bij het schrijven van code en ze te gebruiken op een manier die voor jou of binnen jouw bedrijf als logisch wordt gezien.

Code die niet clean is, is makkelijker te definiëren. 'Spaghetti' noemen we dat bij Team Rockstars IT: Je past links iets aan en rechts gaat er iets kapot.

Het is belangrijk om bij het programmeren in gedachten te houden dat je code niet alleen schrijft voor jezelf, maar ook voor degene die jou op termijn opvolgt. Die persoon zou vrij snel (al dan niet direct) moeten begrijpen wat jij met een bepaald stuk code hebt bedoeld. Dat is de gedachtegang van Clean Code.

'BIJ CLEAN CODE IS HET DE BEDOELING DAT HET ZO SNEL MOGELIJK DUIDELIJK IS WAT EEN BEPAALD STUK CODE DOET. CLEAN CODE IS CODE DIE GESCHREVEN IS DOOR IEMAND DIE ER ECHT OM GAF, DIE ER MOEITE VOOR HEEFT GEDAAN HET MOOI NEER TE ZETTEN.'

Rick Worms
IT Rockstar

DUIDELIJK

DIRECT

BETEKENISVOL

LEESBAAR

ELEGANT

SIMPEL

EFFICIENT

OPTIMAAL

EXPLICIET

MINIMAAL

EENVOUDIG

DE VOORDELEN VAN CLEAN CODE

SNELHEID

Bij Clean Code is het als software developer snel duidelijk wat een bepaalde code doet, wat de bedoeling is geweest van degene die het heeft geschreven. Nieuwe code toevoegen of bijvoorbeeld een bugfix gaan daardoor veel sneller.

EFFICIËNTIE

Een software developer die nieuw binnenkomt op een project is veel sneller productief; hij of zij snapt al snel hoe een code is geschreven en kan hier makkelijk mee verder gaan.

GEEN FRUSTRATIE

Clean Code scheelt heel veel frustratie. Als jij als software developer ergens iets aan wil passen en er ontstaat een enorme bug omdat de code allerlei dingen doet die het niet zou moeten doen, dan levert dat heel veel frustratie op. En dan hebben we het nog niet gehad over de tijd (en het geld) dat het kost om die bugs te fixen. Iets dat weer frustratie voor de hele business oplevert.

"IK BEN EENS LETTERLIJK DRIE DAGEN BEZIG GEWEEST OM EEN BUG TE FIXEN IN EEN BESTAAND STUK CODE DIE NIET CLEAN WAS. DE CODE DIE IK UITEINDELIJK NA DIE DRIE DAGEN HEB OPGELEVERD WAS ÉÉN REGEL."

Rick Worms
IT Rockstar

TEAM ROCKSTARS IT

'SOFTWARE DIE AF IS, IS HETZELFDE ALS EEN GRASVELD DAT NET GEMAAID IS'

Jim Benson

Ook als je een stuk software schrijft met Clean Code moet je over een tijd aan onderhoud – refactoring – doen. In het begin ziet de software er mooi uit, maar al snel komt er een bugfix tussendoor die meteen opgelost moet worden, waardoor er geen tijd is om dat met Clean Code te doen. Dan volgt er nog een bug, die op dezelfde – niet clean – manier wordt opgelost en voor je het weet is het geen mooi gemaaid grasveld meer, maar moet het weer onderhouden worden.

EEN AANTAL PRINCIPES UITGELEGD

Maak alles duidelijk binnen de code. Dat betekent dat je zo min mogelijk commentaar gebruikt. Als je commentaar moet schrijven is de code niet duidelijk. Bovendien zegt het commentaar meestal wat er verderop ook in de code staat, dus wat is dan de zin van het commentaar? Het gevaar van commentaar dat het niet wordt aangepast als de code op termijn wel verandert. Dan krijg je dus code die iets anders zegt dan het commentaar. Soms is commentaar overigens wél relevant. Bijvoorbeeld voor het geven van achtergrondinformatie, bij bepaalde wet- of regelgeving of bij hele specifieke keuzes die gemaakt zijn bij het schrijven van de code.

SINGLE RESPONSIBILITY PRINCIPLE

Een stuk code heeft maar één verantwoordelijkheid. Het doet maar één ding. Zodra een stuk code verantwoordelijk is voor meer acties, wordt het vanzelf verwarrend.

BOY SCOUTING RULE

Als je aan het werk bent binnen een bestaand stuk code, laat het dan schoner achter dan hoe je het hebt gevonden.

GA OOK AAN DE SLAG MET CLEAN CODE

De voordelen van Clean Code zijn duidelijk. Het klinkt zo logisch om vanaf vandaag Clean Code te gaan schrijven. We raden je aan om dat zeker ook te gaan doen bij nieuwe code. Schrijf code zodat het direct duidelijk is wat het doet. Voor het project, maar ook voor jouw mogelijke opvolger.

Alle bestaande code refactoren, dat is bijna niet te doen. Maar op het moment dat je met een stuk geschreven code aan de slag gaat, bijvoorbeeld in het geval van een bugfix, kun je deze wel (deels) refactoren. In combinatie met de Boy Scouting Rule krijg je dan langzaam maar zeker steeds meer Clean Code. Met alle positieve gevolgen van dien!

WIL JE MEER WETEN OVER CLEAN CODE?

Ben je geïnspireerd geraakt door dit artikel en wil je meer weten over Clean Code?
Lees dan het boek “Clean Code” van Robert C. Martin, of bekijk de talks van Peter Hilton, Kevlin Henney en Venkat Subramaniam hieronder. Je kunt ook contact met Team Rockstars IT opnemen om met een van onze developers verder te sparren over Clean Code.

Kevlin Henney
Seven Ineffective Coding Habits of Many Programmers

Peter Hilton
How to write maintainable code

Venkat Subramaniam
12 Ways to Make Code Suck Less