

Week 4

Kwaliteitsproblemen

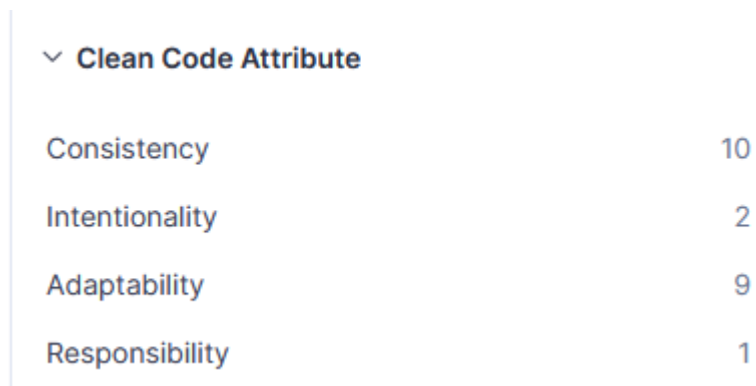
Inhoudsopgave

Inleiding	3
Clean code problemen	4
Software quality problemen	5
Verbeteringen	6
Gevonden bugs door SonarQube	8

Inleiding

Het programma SonarQube heeft de volgende problemen gevonden. Dit valt onder de categorie Clean Code en Softwarekwaliteit. In deze pdf worden de gevonden bugs onderverdeeld in deze 2 categorieën en verder toegelicht. Verbeteringen om de code bruikbaar te maken komen in het hoofdstuk daarna aan bod.

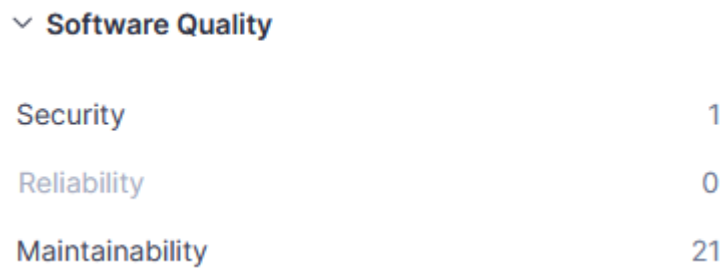
Figuur 1

A screenshot of a SonarQube interface showing a table of Clean Code attributes. The table has a header 'Clean Code Attribute' with a dropdown arrow. Below it are four rows: 'Consistency' with a value of 10, 'Intentionality' with a value of 2, 'Adaptability' with a value of 9, and 'Responsibility' with a value of 1.

▼ Clean Code Attribute	
Consistency	10
Intentionality	2
Adaptability	9
Responsibility	1

Clean code

Figuur 2

A screenshot of a SonarQube interface showing a table of Software Quality attributes. The table has a header 'Software Quality' with a dropdown arrow. Below it are three rows: 'Security' with a value of 1, 'Reliability' with a value of 0, and 'Maintainability' with a value of 21.

▼ Software Quality	
Security	1
Reliability	0
Maintainability	21

Software Quality

Clean code problemen

Clean code richt zich op het schrijven van leesbare, begrijpelijke en onderhoudbare code. De gevonden problemen hebben volgens SonarQube voornamelijk te maken dus met consistency, intentionality, adaptability en responsibility.

Consistency:

Variabele- en functienamen:

Weinig consistentie in het benoemen van variabelen en functies leidt tot onduidelijkheid.

Coding style:

Een inconsistent gebruik van inspringing en spaties kan het moeilijk maken om de code te lezen.

Intentionality:

Commentaar:

Onvoldoende of slecht commentaar kan de onduidelijkheid van de code vergroten.

Magische getallen en strings:

Gebruik van hardcoded getallen en strings zonder duidelijke uitleg vergroot ook onduidelijkheid van de gegeven code.

Adaptability:

Hard-coded waarden:

Het rechtstreeks invoeren van waarden in de code maakt het moeilijk om wijzigingen aan te brengen.

Overmatige afhankelijkheden:

Te veel afhankelijkheden tussen verschillende delen van de code maken het moeilijker om aanpassingen te doen.

Responsability:

Single Responsibility Principle (SRP): Functies en klassen moeten een duidelijk afgebakende verantwoordelijkheid hebben.

Ongepaste koppeling: Modules die te erg van elkaar afhankelijk zijn, kunnen problemen veroorzaken bij het begrijpen en wijzigen van de code.

Software quality problemen

Onder software quality heeft Jenkins onderscheid gemaakt in security en maintainability als de twee meest relevante eigenschappen om te focussen om de code te verbeteren. De volgende zaken waren van het grootste belang:

Security:

Onvoldoende authenticatie en autorisatie:

Gebrekkige implementatie van gebruikers authenticatie en autorisatie kan toegang tot gegevens mogelijk maken, waardoor beveiligingsrisico's ontstaan.

Onvoldoende gegevensencryptie:

Het ontbreken van encryptie in bijvoorbeeld database.php kan leiden tot datalekken en andere security kwetsbaarheden.

Onvoldoende logging en monitoring:

Een gebrek aan logging en monitoring maakt het moeilijk om beveiliging te waarborgen

Maintainability:

Slechte codekwaliteit:

Onduidelijke en rommelige code maakt het moeilijk voor ontwikkelaars om wijzigingen aan te brengen en bugs op te lossen.

Gebrek aan modulaire structuur:

Een gebrek aan scheiding maakt het lastig om specifieke functies te isoleren en te onderhouden.

Onvoldoende testdekking:

Het ontbreken van uitgebreide tests kan leiden tot onvoorziene bugs en introduceert risico's bij het aanbrengen van wijzigingen.

Oplossing:

Het naleven van best practices voor schone code bevordert de onderhoudbaarheid. Het toepassen van modulaair ontwerp vergemakkelijkt het onderhoudsproces. Een duidelijke set van tests maakt het onderhoud en de verdere ontwikkelingen van de software een stuk makkelijker.

Verbeteringen

MVC

Om de gehele code beter begrijpbaar te maken is het verstandig om een designpattern te gebruiken. Design patterns zorgen voor betere herbruikbaarheid van code en maken het makkelijker voor andere developers om code te begrijpen en door te kunnen ontwikkelen. Leesbaarheid en flexibiliteit zijn hier dan ook grote voordelen. Er is voor MVC gekozen voor de hive applicatie omdat dit een aantal voordelen heeft. Ten eerste wordt MVC veelal gebruikt in de web wereld waar onze hive applicatie ook in functioneert. Het wordt standaard toegepast waarin de Model de back-end is, de Controller de API en de View de front-end van een applicatie. Voor de hive applicatie is het MVC patroon ook op deze manier toegepast. Dit zorgt dan voor de bovengenoemde voordelen van herbruikbaarheid, leesbaarheid en flexibiliteit. Verdere voordelen van het gebruik van MVC zijn:

Modulaire ontwikkeling: Met MVC kunnen ontwikkelaars tegelijkertijd aan verschillende delen van de applicatie werken zonder elkaars werk te hinderen. Front-end ontwikkelaars kunnen zich bijvoorbeeld concentreren op het ontwerpen en implementeren van views, terwijl back-end-ontwikkelaars kunnen werken aan de business logic en datamanipulatie zonder zich zorgen te hoeven maken over hoe het eruit ziet.

Herbruikbaarheid van code: Vanwege de scheiding van verschillende lagen kunnen componenten in MVC worden hergebruikt in verschillende delen van de applicatie of zelfs in verschillende applicaties. Een model dat gebruikersgegevens representeert, kan bijvoorbeeld zonder aanpassingen in verschillende delen van de applicatie worden hergebruikt.

Testbaarheid: MVC vergemakkelijkt het testen van unit en integratietesten. Omdat elke component een goed gedefinieerde rol heeft en losjes gekoppeld is aan andere componenten, is het eenvoudiger om geautomatiseerde tests voor individuele componenten te schrijven. Dit maakt het eenvoudiger om bugs te identificeren en op te lossen en zorgt ervoor dat wijzigingen in één deel van de applicatie niet onbedoeld andere delen beïnvloeden.

Consistency

Dit kan worden gewaarborgd en toegepast in de verbeterde code. Door de naamgeving zinvol te maken door middel van het MVC patroon en de functies bijpassende beschrijvende maar korte namen te geven wordt ervoor gezorgd dat de leesbaarheid is vergroot en de logica duidelijk gemaakt. Ook wordt er door Jenkins aangegeven dat een closing tag `</>` overbodig is. Deze worden ook verwijderd in de verbeterde code.

Intentionality

Goede documentatie en commentaren zijn essentieel om anderen en jezelf te helpen begrijpen waarom bepaalde beslissingen zijn genomen. Het is beter om constanten of variabelen duidelijker te benoemen. Dit valt ook weer terug op het gebruik van een design pattern waarin de naamgeving logisch wordt gemaakt door middel van het MVC model. Zo heten de verschillende klassen `gameController`, `hiveView` en `hiveModel`. Dit geeft meteen aan wat de logica in deze klassen zijn en de werking is ook meteen duidelijk. Verder geeft

Jenkins ook aan dat er nested loops gevonden zijn zonder curly braces. Deze braces worden ook toegevoegd aan de verbeterde code met juiste indentatie.

Adaptability

Het is beter om configuraties en instellingen apart te houden om de aanpasbaarheid te verbeteren. Het gebruik van interfaces en het minimaliseren van koppelingen kan helpen de aanpasbaarheid te vergroten. Door het gebruik van MVC wordt het programma meer modulair en kunnen de verschillende klassen afzonderlijk van elkaar werken en aangepast worden. Verder gaat er een style.css file gemaakt worden om hierin de css van de frontend apart te houden van de back-end.

Responsability

Te veel verantwoordelijkheden in één entiteit maken de code complex en moeilijk te begrijpen.

Het toepassen van het Dependency Inversion Principle (DIP) kan de verantwoordelijkheid beter verdelen. Hierom is het MVC patroon ook toegepast. Dit zorgt voor een juiste verdeling van verantwoordelijkheden over meerdere klassen heen. Dit maakt de code overzichtelijk en begrijpbaar ook voor andere programmeurs die het MVC patroon kennen om er sneller en beter mee te kunnen werken. Het MVC gaat zo worden toegepast binnen de hive applicatie dat er meerdere controllers zijn, die elk een eigen stuk van verantwoordelijkheid van de back-end op zich nemen. De verantwoordelijkheden blijven beperkt per klasse op deze manier. Het doel is hiervan om ervoor te zorgen dat elke klasse hooguit één verantwoordelijkheid heeft.

Security:

Het is belangrijk om sterke encryptie protocollen te gebruiken. Regelmatige updates en monitoring van bibliotheken zijn ook belangrijk. Goede logging is belangrijk voor het opsporen van potentiële bedreigingen.

Maintainability:

Het naleven van best practices voor schone code bevordert de onderhoudbaarheid. Het toepassen van modulair ontwerp vergemakkelijkt het onderhoudsproces. Een duidelijke set van tests maakt het onderhoud en de verdere ontwikkelingen van de software een stuk makkelijker. MVC zorgt hier ook voor de modulariteit van het programma. Elke klasse kan apart getest worden. En de functies worden klein en begrijpbaar gehouden om overzichtelijkheid en aanpasbaarheid te bevorderen.

Andere veranderingen voor overzichtelijkheid en aanpasbaarheid zijn dat alle bestanden nu in een src map komen te staan. Hierin komen de eerder genoemde style.css file en de index.php. De index.php is de entry point van het programma. Deze roept de klassen en functies aan uit het MVC mapje dat zich ook in de src bevindt. Buiten de src map is er een ander mapje waarin code staat en dat is het mapje tests. Deze wordt gebruikt om functionaliteit en dan met name de code uit het MVC mapje te testen op juistheid. Hierin bevinden zich unit tests en in deze map wordt ook gebruikt voor test-driven development.

Gevonden bugs door SonarQube

Database.php:

Consistency issue

[Add a new line at the end of this file.](#)

Consistency issue

[Rename function "get_state" to match the regular expression `^\[a-z\]\[a-zA-Z0-9\]*\$`.](#)

Responsibility issue

[Add password protection to this database.](#)

Consistency issue

[Remove this closing tag `"?>"`.](#)

move.php

Consistency issue

[Add a new line at the end of this file.](#)

Adaptability issue

[Replace "include_once" with namespace import mechanism through the "use" keyword.](#)

Intentionality issue

[Add curly braces around the nested statement\(s\).](#)

Adaptability issue

[Replace "include" with namespace import mechanism through the "use" keyword.](#)

Consistency issue

[Replace "include" with "include_once".](#)

Consistency issue

[Split this 137 characters long line \(which is greater than 120 authorized\)](#)

Consistency issue

[Remove this closing tag `"?>"`](#)

pass.php

Consistency issue

[Add a new line at the end of this file.](#)

Adaptability issue

[Replace "include" with namespace import mechanism through the "use" keyword.](#)

Consistency issue

[Replace "include" with "include_once".](#)

Consistency issue

[Split this 135 characters long line \(which is greater than 120 authorized\).](#)

Consistency issue

[Remove this closing tag ">".](#)

play.php

Consistency issue

[Add a new line at the end of this file.](#)

Adaptability issue

[Replace "include_once" with namespace import mechanism through the "use" keyword.](#)

Intentionality issue

[Add curly braces around the nested statement\(s\).](#)

Adaptability issue

[Replace "include" with namespace import mechanism through the "use" keyword.](#)

Consistency issue

[Replace "include" with "include_once".](#)

Consistency issue

[Split this 133 characters long line \(which is greater than 120 authorized\).](#)

Adaptability issue

[Replace "include_once" with namespace import mechanism through the "use" keyword.](#)

Consistency issue

[Remove this closing tag ">".](#)

restart.php

Consistency issue

[Add a new line at the end of this file.](#)

Consistency issue

[Split this 135 characters long line \(which is greater than 120 authorized\).](#)

Adaptability issue

[Replace "include_once" with namespace import mechanism through the "use" keyword.](#)

Consistency issue

[Remove this closing tag ">".](#)

undo.php

Adaptability issue

[Replace "include" with namespace import mechanism through the "use" keyword.](#)

Consistency issue

[Replace "include" with "include_once".](#)

Consistency issue

[Remove this closing tag ">".](#)

util.php

Consistency issue

[Remove this closing tag ">".](#)

Intentionality issue

[This function has 4 returns, which is more than the 3 allowed.](#)

Intentionality issue

[Add curly braces around the nested statement\(s\).](#)

Consistency issue

[Remove this closing tag ">".](#)