Zestaw 3

Testowanie Gier

Karol Stuła, Wojciech Adamiec

13 kwietnia 2021

Spis treści

1. CCCC

C and C++ Code Counter

Project Summary

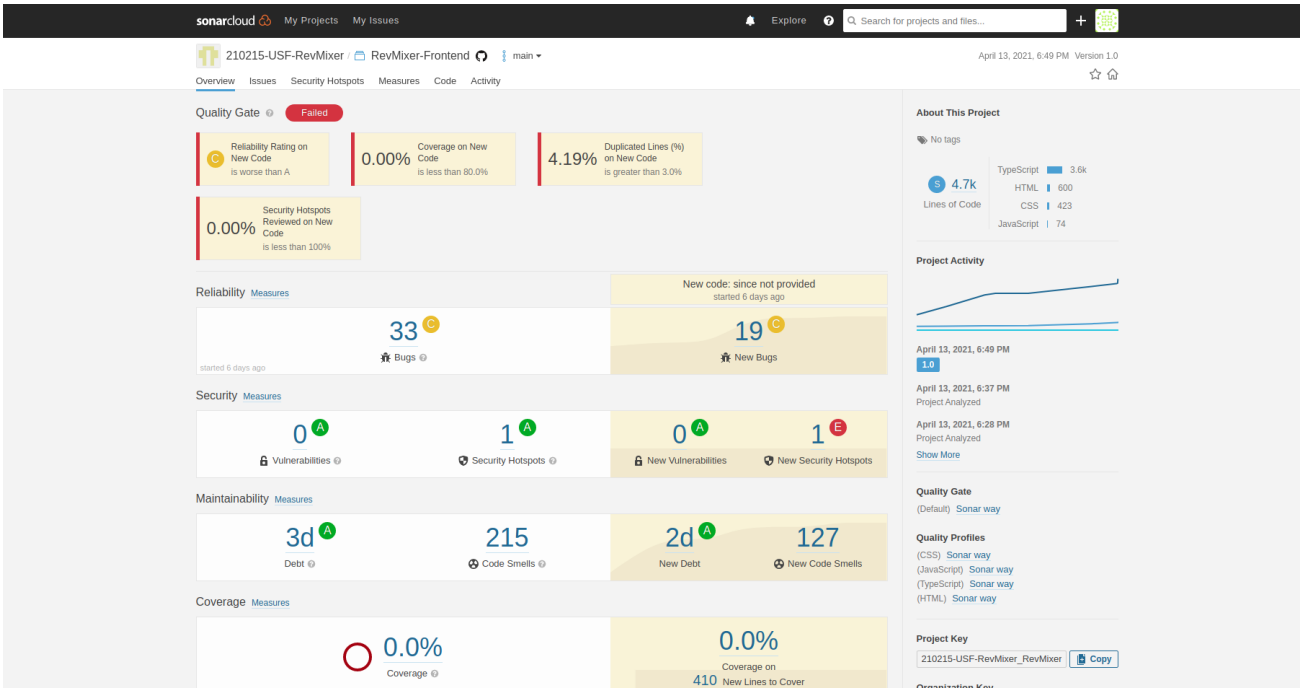
This table shows measures over the project as a whole.

- NOM = Number of modules
  Number of non-trivial modules identified by the analyser. Non-trivial modules include all classes, and any other module for which member functions are identified.
- LOC = Lines of Code
  Number of non-blank, non-comment lines of source code counted by the analyser.
- COM = Lines of Comments
  Number of lines of comment identified by the analyser
- MVG = McCabe's Cyclomatic Complexity
  A measure of the decision complexity of the functions which make up the program.The strict definition of this measure is that it is the number of linearly independent routes through a directed acyclic graph which maps the flow of control of a subprogram. The analyser counts this by recording the number of distinct decision outcomes contained within each function, which yields a good approximation to the formally defined version of the measure.
- L_C = Lines of code per line of comment
  Indicates density of comments with respect to textual size of program
- M_C = Cyclomatic Complexity per line of comment
  Indicates density of comments with respect to logical complexity of program
- IF4 = Information Flow measure
  Measure of information flow between modules suggested by Henry and Kafura. The analyser makes an approximate count of this by counting inter-module couplings identified in the module interfaces.

Two variants on the information flow measure IF4 are also presented, one (IF4v) calculated using only relationships in the visible part of the module interface, and the other (IF4c) calculated using only those relationships which imply that changes to the client must be recompiled of the supplier's definition changes.

Metric	Tag	Overall	Per Module
Number of modules	NOM	1	
Lines of Code	LOC	127	127.000
McCabe's Cyclomatic Number	MVG	28	28.000
Lines of Comment	COM	0	0.000
LOC/COM	L_C	******	
MVG/COM	M_C	******	
Information Flow measure (inclusive)	IF4	0	0.000
Information Flow measure (visible)	IF4v	0	0.000
Information Flow measure (concrete)	IF4c	0	0.000
Lines of Code rejected by parser	REJ	0	

2. SonarCloud



3. Metryki oprogramowania

3.1. Złożoność cyklomatyczna

Metryka oprogramowania opracowana przez Thomasa J. McCabe'a w 1976, używana do pomiaru stopnia skomplikowania programu. Podstawą do wyliczeń jest liczba dróg w schemacie blokowym danego programu.

Każda pętla, każda instrukcja warunkowa i każdy skok ma istotny wpływ na złożoność cyklomatyczną - im więcej zazębiających się instrukcji tego typu tym większa złożoność.

Dokładniej rzecz biorąc przy wyliczaniu złożoności cyklomatycznej chcemy operować na grafie przepływu sterowania. Na stworzonym na bazie kodu źródłowego grafie przeprowadzamy obliczenia:

$$M = E - N + 2P$$

gdzie:

- E: liczba krawędzi w grafie
- N: Liczba wierzchołków w grafie
- P: liczba spójnych składowych w grafie

Wyniki tej metryki czytamy w następujący sposób:

- od 1 do 10 – kod dość prosty stwarzający nieznaczne ryzyko
- od 11 do 20 – kod złożony powodujący ryzyko na średnim poziomie
- od 21 do 50 – kod bardzo złożony związany z wysokim ryzykiem
- powyżej 50 – kod niestabilny grożący bardzo wysokim poziomem ryzyka

3.2. Liczba zapachów kodu

Z ang. *Code Smell* – Cecha kodu źródłowego mówiąca o złym sposobie implementacji. Metryką jest ilość występujących zapachów.

Przykłady:

- Długa metoda - istnieją bardzo długie metody
- Powielony kod - ten sam fragment kodu powtarza się w kilku miejscach
- Zazdrość o kod - istnieją metody intensywnie korzystające z danych innej klasy
- Zbyt mała intymność - istnieją klasy, których działanie jest zależne od implementacji innych klas

3.3. Stosunek długu technicznego

Z czasem kiedy kod staje się coraz większy i bardziej skomplikowany jest go również ciężej modyfikować lub usprawniać. Każde niedociągnięcie spowodowane chęcią szybkiego postępu może spowodować potencjalne problemy w przyszłości.

Dług techniczny jest metryką pozwalającą na kwantyfikacje tego problemu i jest on mierzony jako stosunek kosztu naprawy danego oprogramowania do kosztu jego wytworzenia:

$$TDR = Remediationcost/DevelopmentCost$$

Koszt wytworzenia można liczyć np. za pomocą iloczynu ilości linii kodu oraz kosztu jednej linii:

$$DevelopmentCost = CPL \cdot LOC$$

Z kolei koszt naprawy można liczyć np. za pomocą godzin w postaci funkcji liniowo zależnej od dowolnej innej metryki np. złożoności cyklomatycznej.

Im niższa wartość tej metryki tym lepiej dla całego projektu. W zależności od sposobu liczenia zespół może ustalić sobie graniczny próg tej metryki np. w postaci 5%.