Izveštaj primene alata za verifikaciju u okviru samostalnog praktičnog projekta na kursu Verifikacija Softvera

Matematički fakultet

Nikola Mićić, 1086/2022
nikolamicic065@gmail.com
Mentor: Ivan Ristović

Decembar 2022.

Sažetak

Ovaj rad će sadržati detaljan opis analize projekta sa spiskom naredbi i alata koji su korišćene i zaključcima koji su napravljeni.

Projekat nad kojim će biti primenjeni alati se nalazi na github adresi: https://gitlab.com/matf-bg-ac-rs/course-rs/projects-2021-2022/22-Panzerkrieg

Autori ovog projekta su moje kolege Jovan Marković (69/2018), Tamara Miković (82/2017), Sara Stanković (57/2018), Tina Mladenović (116/2018), Miodrag Todorović (100/2018) i projekat je rađen u okviru kursa Razvoj softvera na osnovnim studijama Matematičkog fakulteta, smer informatika.

Primena alata će biti izvršena na master grani, nad komitom čiji je hash code sledeći: 9eb938b8d4c4fc2ecd2c22c9cae26c964d5371d8

Sadržaj

1 Spisak primenjenih alata

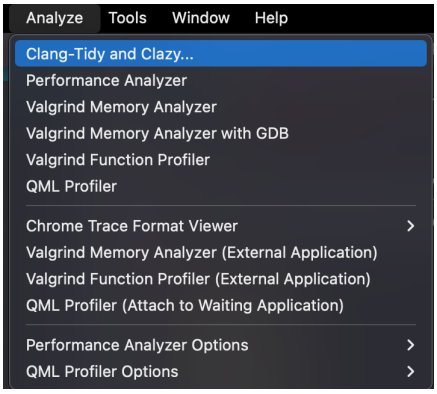
Spisak alata za verifikaciju koji su primenjeni nad projektom su:

1. Clang-tidy i Clazy

1.1 Clang-tidy i Clazy

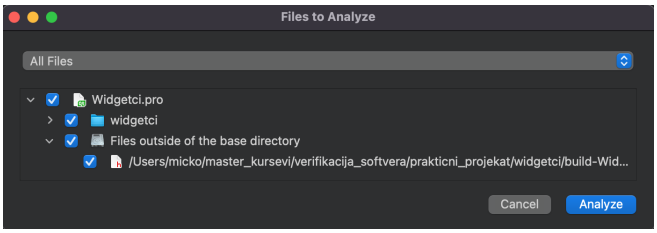
Clang-tidy i Clazy su alati koji su pokrenuti preko QtCreator-a. Alat je pokrenut preko Debug moda i sastoji se iz narednih koraka koji se mogu videti na slikama ispod.

Prvi korak je biranje alata Clang-Tidy i Clazy u okviru padajućeg menija Analyze.



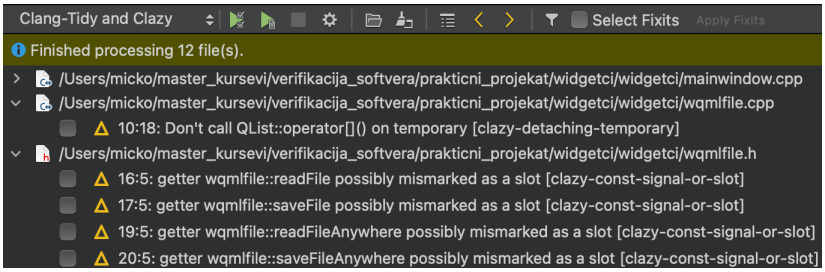
Slika 1: Prvi korak

Drugi korak se sastoji iz biranja fajlova koji će biti analizirani. U ovom slučaju sam birao sve fajlove uključujući glavni folder *code* u okviru kojeg se nalaze podfolderi *headers* i *src*. Nakon toga kliknuti dugme *Analyze*.



Slika 2: Drugi korak

Deo rezultata ovih alata je predstavljen na slici 3 ispod. Može se videti upozorenje da postoji neinicijalizovano polje na kraju poziva konstruktora, sa objašnjenjima. Ceo izlaz pozvanih alata se može naći u okviru repozitorijuma za analizu datog projekta, u okviru foldera Clang-tidy i Clazy.



Slika 3: Deo rezultata pozvanih alata

1.2 Cppcheck

Cppcheck je alat koji sam pokretao preko terminala. Za instalaciju alata na macOS-u sam koristio komandu *brew install cppcheck*.

Alat predstavlja statičku analizu koda i proverava sve .cpp fajlove na prosleđenoj putanji i vraća uočene greške i upozorenja.

Pozivanje komande i deo rezultata je izgledao kao na slici ispod. Celokupan izlaz pozivanja alata se može naći u okviru repozitorijuma za analizu datog projekta, u okviru foldera Cppcheck.

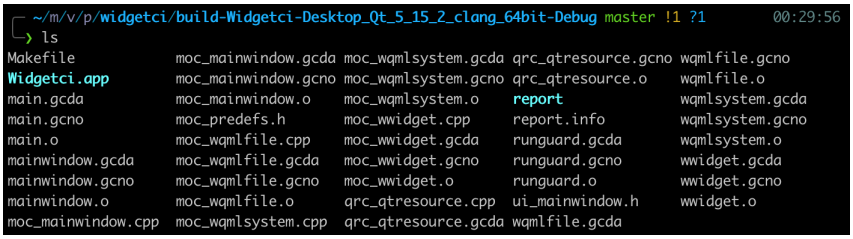
Slika 4: Način pokretanja i rezultati alata cppcheck

1.3 GCov

Primena alata GCov je urađena iz sledećih koraka. Prvo su u *Widgetci.pro* fajlu dodata dva flega:

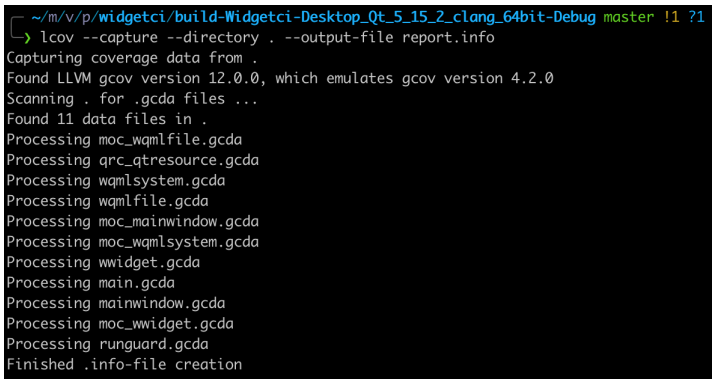
```
QMAKE_CXXFLAGS += -coverage
QMAKE_LFLAGS += -coverage
```

Izgradnja projekta sa novim flagovima je generisala naredne fajlove, među kojima su i dodatno generisani fajlovi koji imaju ekstenziju '.gcno' i '.gcda'. Svi fajlovi se mogu videti na slici 5



Slika 5: Generisani fajlovi

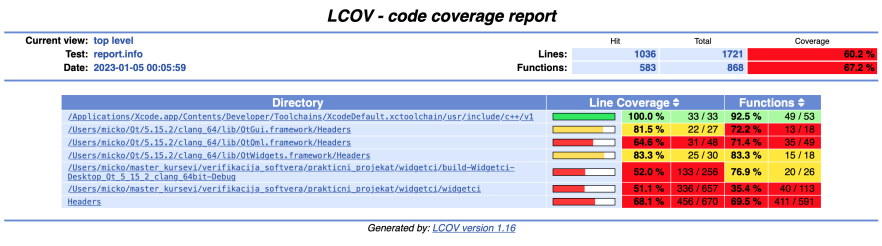
Zatim je pokrenuta naredba koja kreira traženi izveštaj u datoteci *report.info*, a rezultat naredbe možemo videti na sledećoj slici 6



Slika 6: rezultat poziva lcov naredbe

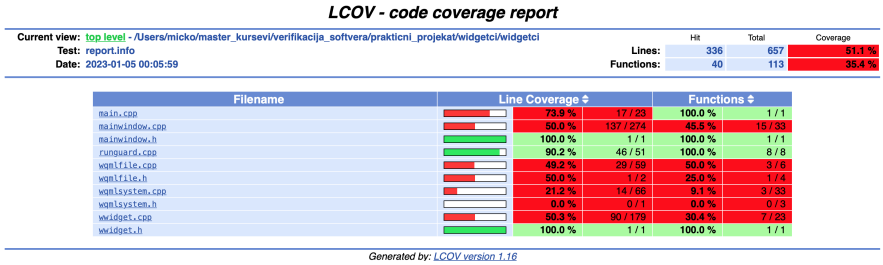
Komandom **genhtml -o report report.info** dobijamo html izveštaj preko podataka iz *report.info* datoteke, a kompletni izlazi prethodne dve komande se mogu naći u okviru repozitorijuma za analizu datog projekta, u okviru foldera GCov.

U folderu report unutar build foldera se nalazi ceo izveštaj, a html verziju izveštaja možemo videti otvaranjem index.html fajla i ona izgleda kao na slici 7 ispod.



Slika 7: html verzija izveštaja pokrivenosti koda

Detaljniji prikaz pokrivenosti koda nad fajlovima u okviru projekta možemo videti klikom na neki od ponuđenih putanja. Klikom na folder i podfolder *widgetci*, otvara se analiza pokrivenosti koda izlistanih fajlova koji se mogu videti na slici 8 ispod.



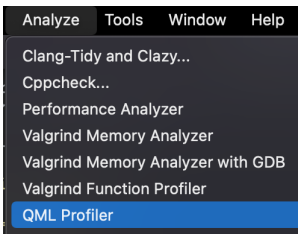
Slika 8: detaljniji prikaz izveštaja pokrivenosti koda

1.4 QML Profiler

QML Profiler je alat koji omogućava da se dobiju neophodne dijagnostičke informacije, omogućujuću da se analizira kod aplikacije kako bi se otkrili problemi sa performansama. Primer problema sa performansama predstavljaju previše JavaScript-a u određenim frejmovima, C++ funkcija koje traju predugo ili troše previše memorije. QML Profiler predstalvja deo i Qt Creator-a i Qt Design Studio-a.

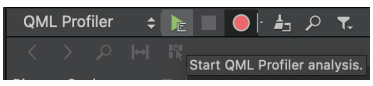
Kroz korake će biti predstavljen način rada ovog alata i rezultati prikazani.

Pokretanje alata preko Qt Creator-a se vrši preko padajućeg menija Analyze i biranje alata QML Profiler, kao na slici 9 ispod.



Slika 9: Biranje alata QML Profiler-a

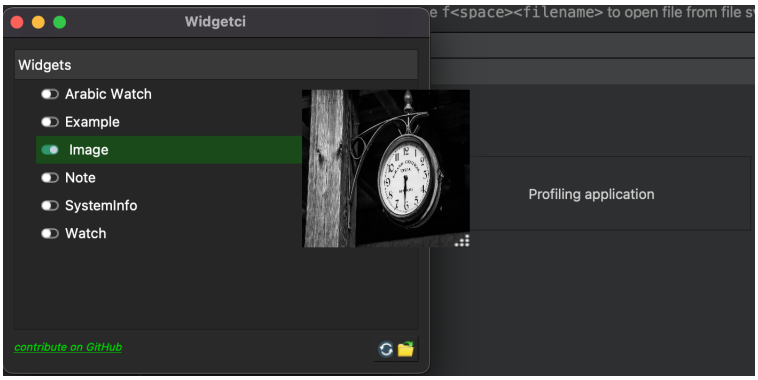
Zatim se na zeleno dugme pokreće alat, kao što se može videti na slici 10 ispod.



Slika 10: Pokretanje QML Profiler-a

Analiza QML Profiler-a je odrađena pri pokretanju widget-a aplikacije koja predstavlja prikazivanje slike sata koja može da se povećava i

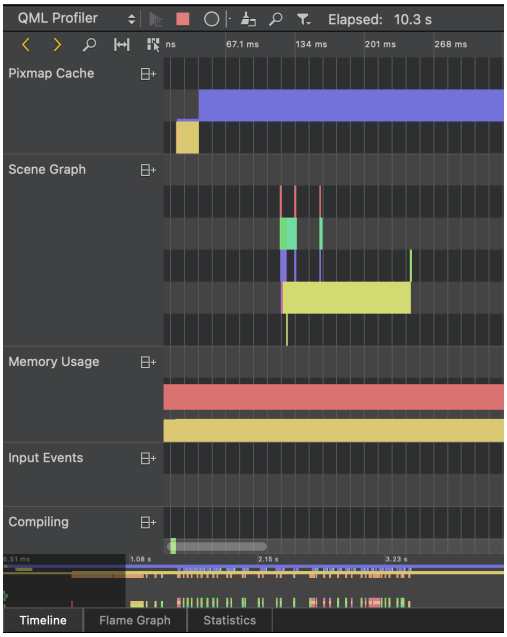
smanjuje pomeranjem donje desne ivice kao što se može videti na slici 11 ispod.



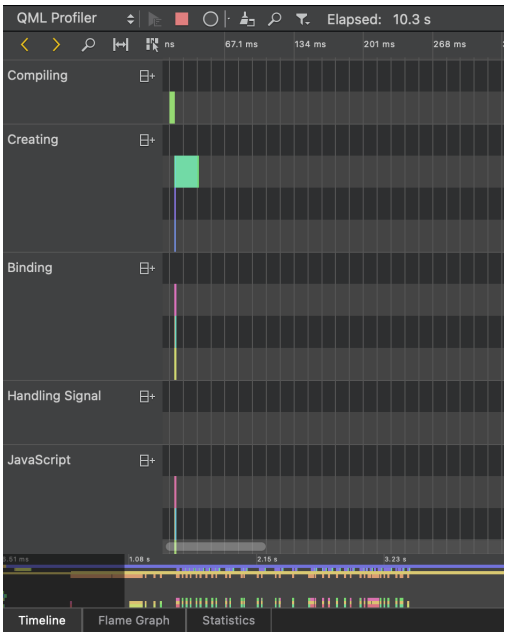
Slika 11: Interakcija sa aplikacijom tokom analiziranja QML Profiler-om

Rezultati QML Profiler-a predstavljaju analizu nakon 10 sekundi prikazivanja sata i povećavanja i smanjivanja njegovih dimenzija pomeranjem donje desne ivice.

Timeline QML Profiler-a izdeljen po sekcijama u prvoj sekundi izvršavanja izgleda kao na slikama 12,13 (duže od jedne sekunde nije moglo biti prikazano na screenshot-u)

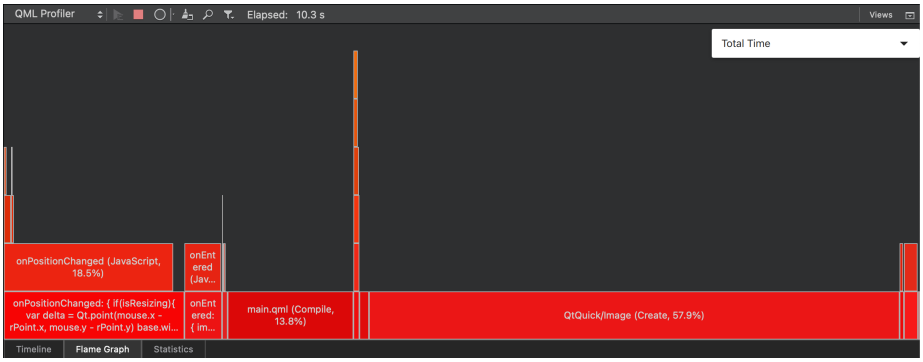


Slika 12: Prvi deo Timeline-a QML Profiler-a



Slika 13: Drugi deo Timeline-a QML Profiler-a

Rezultat analize prikazan kroz Flame Graph, za sekciju Total Time izgleda ovako 14.

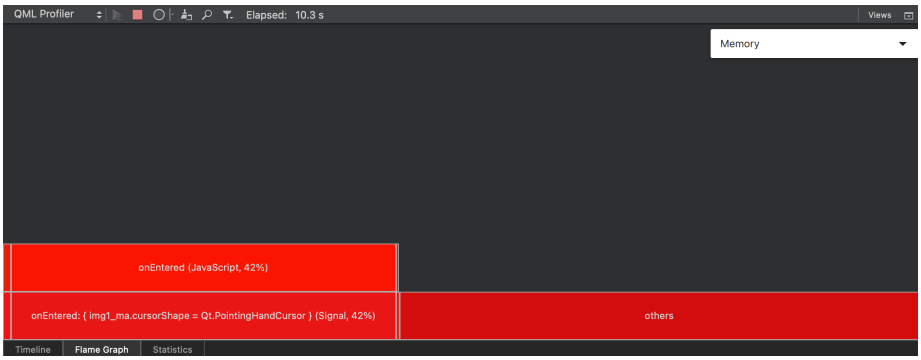


Slika 14: Flame Graph - Total Time

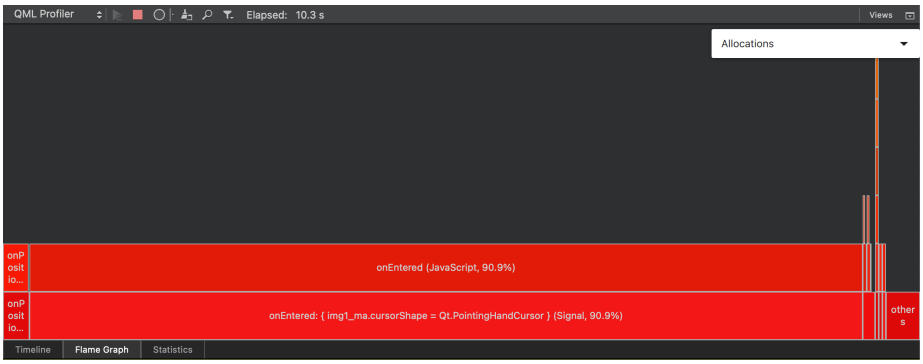
Rezultat analize prikazan kroz Flame Graph, za sekciju memorijskog utroška izgleda ovako 15.

Rezultat analize prikazan kroz Flame Graph, za sekciju alokacija memorije izgleda ovako 16.

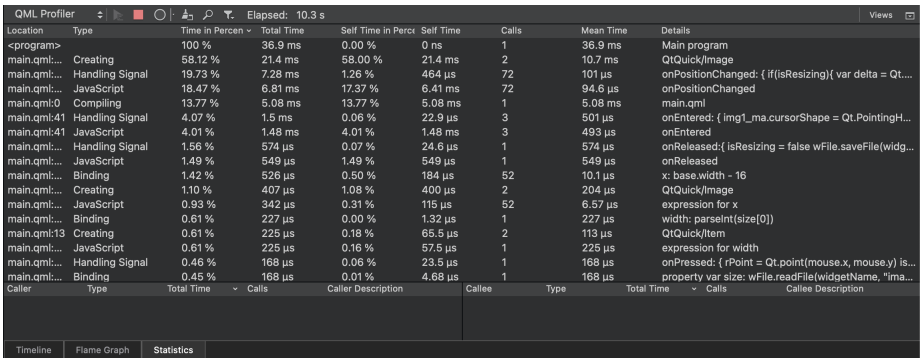
Na kraju se može videti statistički prikaz analize nad svim komponentama ove aplikacije 17.



Slika 15: Flame Graph - Memory



Slika 16: Flame Graph - Memory



Slika 17: Flame Graph - Memory