

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	2
1.1	Clang-tidy i Clazy	2
1.2	Cppcheck	3
1.3	GCov	3
1.4	QML Profiler	5

1 Spisak primenjenih alata

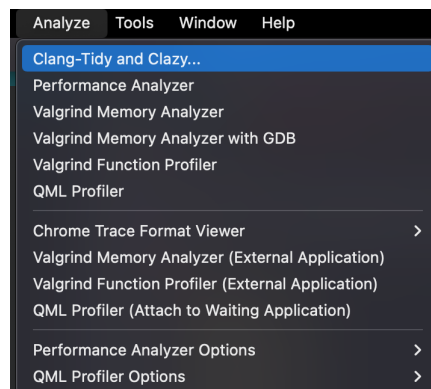
Spisak alata za verifikaciju koji su primenjeni nad projektom su:

1. Clang-tidy i Clazy

1.1 Clang-tidy i Clazy

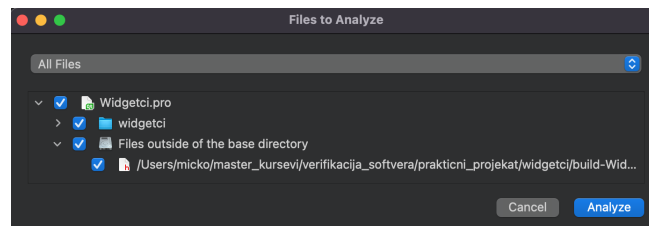
Clang-tidy i Clazy su alati koji su pokrenuti preko QtCreator-a. Alat je pokrenut preko Debug moda i sastoji se iz narednih koraka koji se mogu videti na slikama ispod.

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



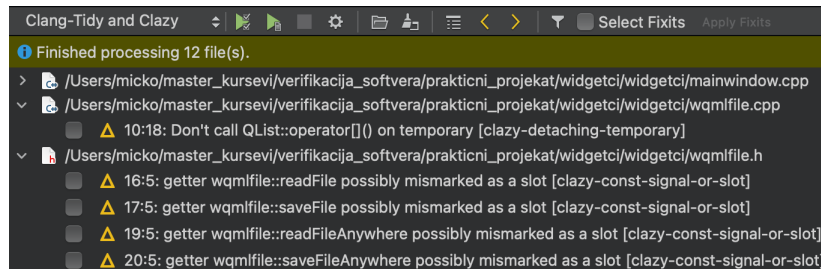
Slika 1: Prvi korak

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.



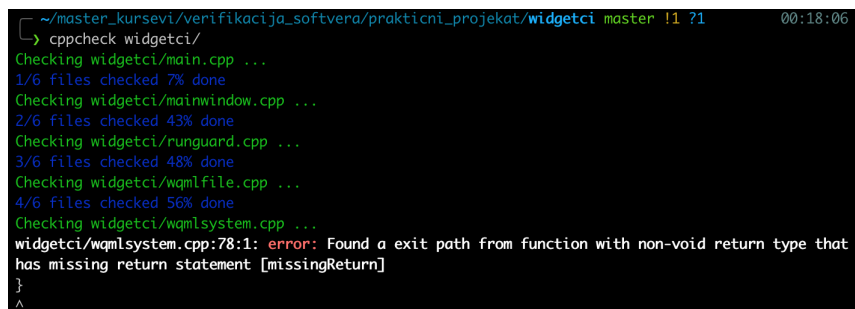
Slika 3: Deo rezultata pozvanih alata

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 prosledenoj 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

```

~/m/v/p/widgetci/build-Widgetci-Desktop_Qt_5_15_2_clang_64bit-Debug master !1 ?1 00:29:56
└─> ls
Makefile          moc_mainwindow.gcda moc_wqmlsystem.gcda qrc_qtresource.gcno wqmlfile.gcno
Widgetci.app      moc_mainwindow.gcnoc moc_wqmlsystem.gcnoc qrc_qtresource.o    wqmlfile.o
main.gcda         moc_mainwindow.o    moc_wqmlsystem.o    report              wqmlsystem.gcda
main.gcnoc        moc_predefs.h       moc_wwidget.cpp     report.info         wqmlsystem.gcnoc
main.o           moc_wqmlfile.cpp    moc_wwidget.gcda    runguard.gcda      wqmlsystem.o
mainwindow.gcda  moc_wqmlfile.gcda   moc_wwidget.gcnoc   runguard.gcnoc     wwidget.gcda
mainwindow.gcnoc moc_wqmlfile.gcnoc   moc_wwidget.o       runguard.o          wwidget.gcnoc
mainwindow.o      moc_wqmlfile.o      qrc_qtresource.cpp  ui_mainwindow.h    wwidget.o
moc_mainwindow.cpp moc_wqmlsystem.cpp  qrc_qtresource.gcda wqmlfile.gcda

```

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

```

~/m/v/p/widgetci/build-Widgetci-Desktop_Qt_5_15_2_clang_64bit-Debug master !1 ?1
└─> lcov --capture --directory . --output-file report.info
Capturing coverage data from .
Found LLVM gcov version 12.0.0, which emulates gcov version 4.2.0
Scanning . for .gcda files ...
Found 11 data files in .
Processing moc_wqmlfile.gcda
Processing qrc_qtresource.gcda
Processing wqmlsystem.gcda
Processing wqmlfile.gcda
Processing moc_mainwindow.gcda
Processing moc_wqmlsystem.gcda
Processing wwidget.gcda
Processing main.gcda
Processing mainwindow.gcda
Processing moc_wwidget.gcda
Processing runguard.gcda
Finished .info-file creation

```

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.

LCOV - code coverage report				
Current view: top level		Hit	Total	Coverage
Test: report.info		Lines: 1036	1721	60.2 %
Date: 2023-01-05 00:05:59		Functions: 583	868	67.2 %
Directory	Line Coverage	Functions		
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/include/c++/v1	100.0 % 33 / 33	92.5 % 49 / 53		
/Users/nicko/Desktop/5.15.2/clang_64/lib/objc.framework/Headers	81.8 % 22 / 27	72.2 % 18 / 18		
/Users/nicko/Desktop/5.15.2/clang_64/lib/objc.framework/Headers	84.8 % 31 / 36	71.4 % 35 / 49		
/Users/nicko/Desktop/5.15.2/clang_64/lib/objc.framework/Headers	83.3 % 25 / 30	83.3 % 15 / 18		
/Users/nicko/master/kurseji/verifikacija_softvera/prakticni_projekat/widgetci/build-Widgetci-Desktop_Qt_5_15_2_clang_64bit-Debug	52.0 % 133 / 256	76.9 % 20 / 26		
/Users/nicko/master/kurseji/verifikacija_softvera/prakticni_projekat/widgetci/widgetci	51.1 % 336 / 657	35.4 % 40 / 113		
Headers	68.1 % 456 / 670	69.5 % 411 / 591		

Generated by: LCOV version 1.16

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.

LCOV - code coverage report

Current view: top level - /Users/micko/master_kursevi/verifikacija_softvera/prakticni_projekat/widgetci/widgetci		Hit	Total	Coverage
Test: report.info		Lines: 336	657	51.1 %
Date: 2023-01-05 00:05:59		Functions: 40	113	35.4 %

Filename	Line Coverage %	Line Coverage #	Functions %	Functions #
main.cpp	73.9 %	17 / 23	100.0 %	1 / 1
mainwindow.cpp	58.0 %	137 / 234	45.5 %	15 / 33
mainwindow.h	100.0 %	1 / 1	100.0 %	1 / 1
runguard.cpp	90.2 %	46 / 51	100.0 %	8 / 8
widgetfile.cpp	49.2 %	29 / 59	50.0 %	3 / 6
widgetfile.h	50.0 %	1 / 2	25.0 %	1 / 4
widgetsystem.cpp	21.2 %	14 / 66	8.1 %	3 / 38
widgetsystem.h	0.0 %	0 / 1	0.0 %	0 / 5
widgetset.cpp	50.3 %	90 / 179	30.4 %	7 / 23
widgetset.h	100.0 %	1 / 1	100.0 %	1 / 1

Generated by: LCOV version 1.16

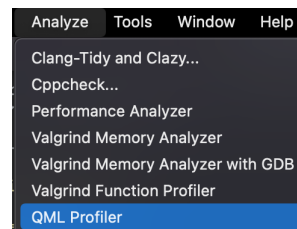
Slika 8: detaljniji prikaz izveštaja pokrivenosti koda

1.4 QML Profiler

QML Profiler je alat koji omogućava da se dobiju neophodne dijagnostičke informacije, omogućujući 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 predstavlja 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

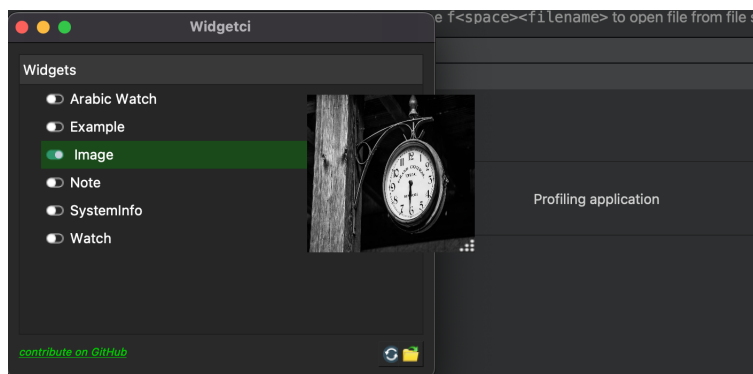
Zatim se na zeleno dugme pokreće alat, kao što se može videti na slici 10 ispod.



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

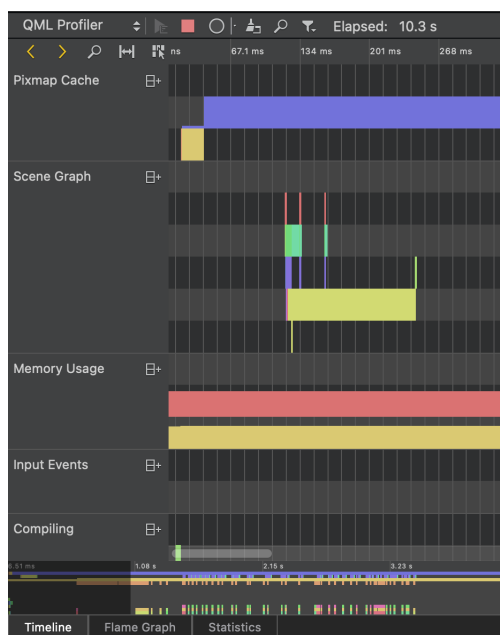
smanjuje pomeranjem donje desne ivice kao što se može videti na slici 11 ispod.



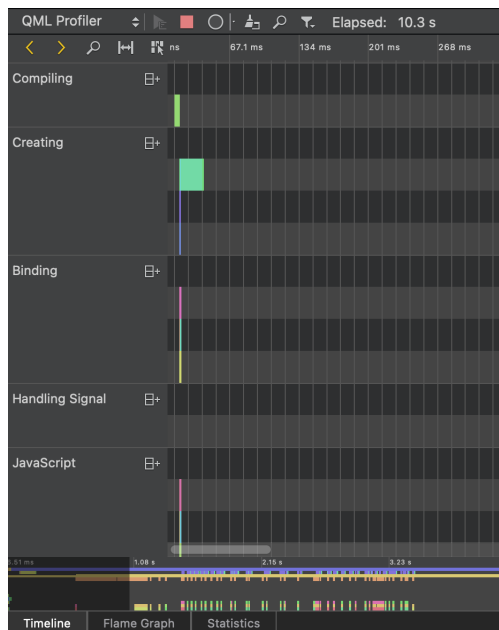
Slika 11: Interakcija sa aplikacijom tokom analiziranja QML Profiler-om

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

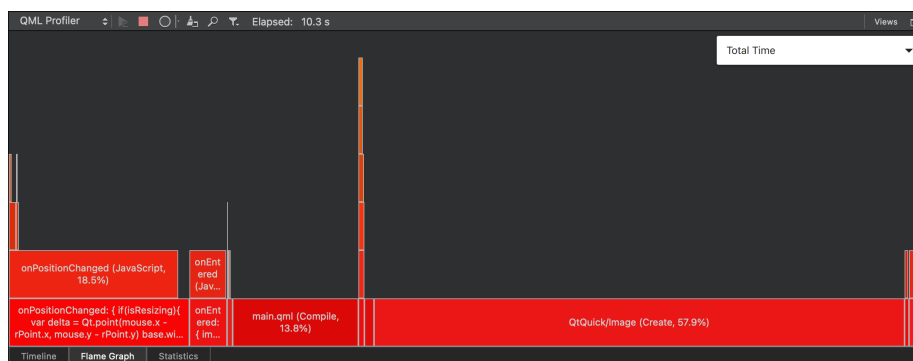


Slika 12: Prvi deo Timeline-a QML Profiler-a



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

Rezultat analize prikazan kroz Flame Graph, za sekciju Total Time izgleda ovako [14](#).

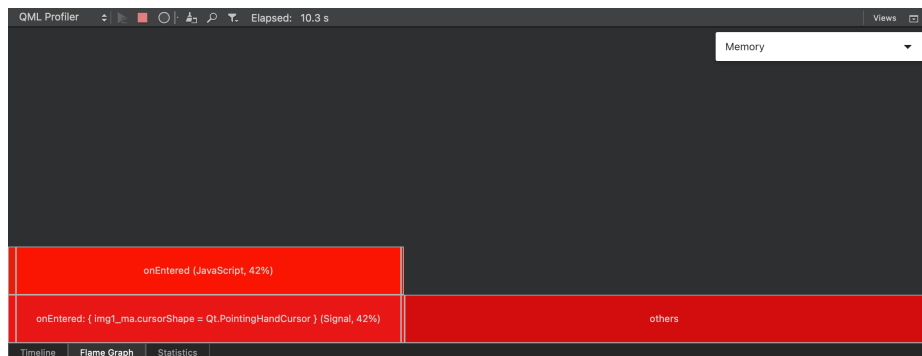


Slika 14: Flame Graph - Total Time

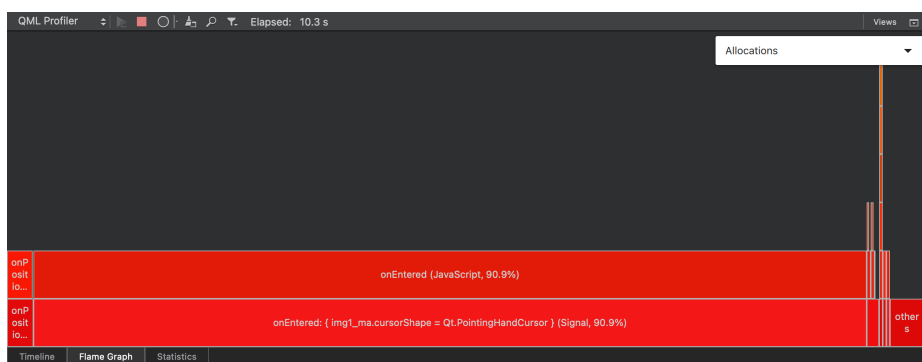
Rezultat analize prikazan kroz Flame Graph, za sekciju memorijskog utroška izgleda ovako [15](#).

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

QML Profiler										Elapsed: 10.3 s										Views	
Location	Type	Time in Percent	Total Time	Self Time in Percent	Self Time	Calls	Mean Time	Details													
<program>		100 %	36.9 ms	0.00 %	0 ns	1	36.9 ms	Main program													
main.qml:...	Creating	58.12 %	21.4 ms	58.00 %	21.4 ms	2	10.7 ms	QtQuick/Image													
main.qml:...	Handling Signal	19.73 %	7.28 ms	1.26 %	464 μs	72	101 μs	onPositionChanged: { if(isResizing){ var delta = Qt...													
main.qml:...	JavaScript	18.47 %	6.81 ms	17.37 %	6.41 ms	72	94.6 μs	onPositionChanged													
main.qml:0	Compiling	13.77 %	5.08 ms	13.77 %	5.08 ms	1	5.08 ms	main.qml													
main.qml:41	Handling Signal	4.07 %	1.5 ms	0.06 %	22.9 μs	3	501 μs	onEntered: { img1_ma.cursorShape = Qt.PointingH...													
main.qml:41	JavaScript	4.01 %	1.48 ms	4.01 %	1.48 ms	3	493 μs	onEntered													
main.qml:...	Handling Signal	1.56 %	574 μs	0.07 %	24.6 μs	1	574 μs	onReleased: { isResizing = false wFile.saveFile(widg...													
main.qml:...	JavaScript	1.49 %	549 μs	1.49 %	549 μs	1	549 μs	onReleased													
main.qml:...	Binding	1.42 %	526 μs	0.50 %	194 μs	52	10.1 μs	xs: base.width - 16													
main.qml:...	Creating	1.10 %	407 μs	1.08 %	400 μs	2	204 μs	QtQuick/Image													
main.qml:...	JavaScript	0.93 %	342 μs	0.31 %	115 μs	52	6.57 μs	expression for x													
main.qml:...	Binding	0.81 %	227 μs	0.00 %	1.32 μs	1	227 μs	width: parseInt(size[0])													
main.qml:13	Creating	0.61 %	225 μs	0.18 %	65.5 μs	2	113 μs	QtQuick/Item													
main.qml:...	JavaScript	0.61 %	225 μs	0.16 %	57.5 μs	1	225 μs	expression for width													
main.qml:...	Handling Signal	0.46 %	168 μs	0.06 %	23.5 μs	1	168 μs	onPressed: { rPoint = Qt.point(mouse.x, mouse.y) is...													
main.qml:...	Binding	0.45 %	168 μs	0.01 %	4.68 μs	1	168 μs	property var size: wFile.readFile(widgetName, "ima...													
Caller	Type	Total Time	Calls	Caller Description	Callee	Type	Total Time	Calls	Callee Description												
											Timeline Flame Graph Statistics										

Slika 17: Flame Graph - Memory