Izveštaj sprovedene analize
Projekat **kOrganizify**

Milena Vidić 1107/2024

Projekat iz predmeta **Verifikacija softvera**, Matematički fakultet, 2025

1 Uvod

Ovaj izveštaj predstavlja analizu i verifikaciju projekta **kOrganizify**, realizovanu u okviru predmeta *Verifikacija softvera* na Matematičkom fakultetu Univerziteta u Beogradu. Cilj projekta je primena različitih alata za proveru ispravnosti i kvaliteta softvera, kako bi se analizirala tačnost, pouzdanost i struktura postojećeg koda.

Tokom analize korišćeni su alati koji pokrivaju statičku i dinamičku analizu, kao i testiranje:

- **Cppcheck** – alat za statičku analizu C/C++ koda.
- **Clang-format** – alat za automatsko formatiranje
- **Valgrind (Memcheck)** – alat za dinamičku analizu i otkrivanje grešaka u radu sa memorijom.
- **Valgrind (Massif)** – alat za profilisanje memorije koji meri zauzeće heap memorije tokom izvršavanja programa i prikazuje trenutke najveće potrošnje.

Analiza je sprovedena na verziji koda označenoj commit-om `006aac3aa0d534318bdc0cee7db898bd864601f0`.

2 Opis projekta

Projekat **kOrganizify** predstavlja aplikaciju za organizaciju zadataka, događaja i obaveza. Implementiran je u programskom jeziku C++, sa modularnom arhitekturom koja omogućava lako proširenje i održavanje. Ključne funkcionalnosti uključuju upravljanje događajima, unos i čuvanje podataka, kao i osnovnu obradu i prikaz informacija korisniku.

Struktura projekta sastoji se od više fajlova koji sadrže logiku, klase i pomoćne funkcije za rad sa događajima i zadacima. Zbog takve arhitekture, projekat je pogodan za demonstraciju različitih metoda verifikacije softvera – od statičke analize i formatiranja, do testiranja i analize ponašanja tokom izvršavanja.

3 Cppcheck

3.1 Opis alata

Cppcheck je alat za statičku analizu koda koji služi za otkrivanje potencijalnih grešaka, nelogičnosti i loših programerskih praksi u C i C++ programima. Za razliku od kompajlera, koji proverava sintaksu i tipove, Cppcheck analizira semantičku ispravnost koda, logičku konzistentnost i stil pisanja, bez njegovog pokretanja. Cilj alata je da ukaže na delove koda koji mogu izazvati greške u izvršavanju, otežati održavanje ili umanjiti performanse. Njegova analiza obuhvata detekciju:

- nedefinisanog ponašanja i potencijalnih bagova,
- curenja memorije i problema sa pokazivačima,
- neiskorišćenih promenljivih i funkcija,
- grešaka u opsegu i inicijalizaciji,
- nedostatka konstruktora kopije ili destruktora kod klasa koje upravljaju resursima.

Cppcheck koristi analizu toka podataka kako bi identifikovao situacije koje kompajler često ne prijavljuje, kao što su logičke greške ili loša praksa u pisanju koda. Pored osnovne analize, moguće je proširiti alat dodatnim konfiguracijama i prilagoditi ga potrebama projekta.

3.2 Rezultati analize

Analiza nije pokazala ozbiljne greške kao što su dereferenciranje `nullptr`-a, curenje memorije, neinicijalizovani pokazivači ili nedefinisano ponašanje. Detektovana su samo manja upozorenja, uglavnom informativnog ili stilskog karaktera, kao i poruke koje potiču iz eksternih biblioteka korišćenih u projektu.

3.2.1 Upozorenja u okviru test koda

Cppcheck je detektovao nekoliko neiskorišćenih promenljivih i stilskih nedoslednosti u okviru test fajlova, poput:

```
../src/kOrganizify/Test/basicEventTest.cpp:9:34: style:
Variable 'actualDuration' is assigned a value that is never used. [unreadVariable]
```

Ove poruke nemaju uticaja na funkcionalnost programa, ali se preporučuje njihovo uklanjanje radi bolje preglednosti koda.

3.2.2 Upozorenja u okviru Catch2 biblioteke (eksterni kod)

Veći deo prijavljenih upozorenja potiče iz datoteke `catch.hpp`, koja pripada eksternom test okviru `Catch2`, korišćenom za jedinične testove. Cppcheck je u ovom fajlu detektovao sledeće tipove poruka:

- **Syntax error** — linije koje sadrže Objective-C sintaksu poput:

  ```
  id obj = [[m_cls alloc] init];
  @try { ... }
  ```

- **Missing return statement** — Cppcheck je uočio funkcije sa neodgovarajućim povratnim vrednostima, npr:

  ```
  Found an exit path from function with non-void return type
  that has missing return statement. [missingReturn]
  ```

- **Unknown macro** — poruka:

  ```
  There is an unknown macro here somewhere. If slots is a macro
  then please configure it. [unknownMacro]
  ```

- **Stilske preporuke** — Cppcheck je sugerisao korišćenje STL algoritama poput `std::transform` ili `std::any_of` umesto ručnih petlji.

Sve navedene poruke mogu se bezbedno zanemariti, jer potiču iz spoljne biblioteke koja nije deo izvornog koda projekta već predstavlja gotov test okvir.

3.2.3 Informativne poruke o hederima

Tokom analize, Cppcheck je generisao veliki broj informativnih poruka o nedostajućim sistemskim hederima standardne biblioteke, na primer:

```
Include file: <string> not found. [missingIncludeSystem]
```

Ove poruke su očekivane i ne predstavljaju problem, jer Cppcheck ne zahteva pristup sistemskim hederima za validne rezultate.

3.3 Zaključak

Na osnovu sprovedene analize može se zaključiti da je projekat `kOrganizify` stabilan, konzistentan i bez ozbiljnih problema u kodu. Sva upozorenja koja je Cppcheck generisao potiču iz test okruženja (`Catch2`) ili su informativne prirode.

Preporučuje se da se:

- zadrži postojeća struktura i stil koda,
- uklone neiskorišćene promenljive radi preglednosti.

Analiza pokazuje da projekat ima dobru osnovu za dalju verifikaciju i testiranje, bez značajnih rizika po stabilnost i ispravnost programa.

4 Clang-Format

4.1 Opis alata

`Clang-Format` je alat namenjen automatskom formatiranju C, C++ i srodnih jezika prema unapred definisanim stilskim pravilima. On ne analizira logiku programa niti detektuje greške, već obezbeđuje konzistentan izgled koda, što doprinosi njegovoj čitljivosti i održavanju. Korišćenjem `Clang-Format`-a, timovi mogu osigurati jedinstven stil bez ručnih intervencija i neslaganja u konvencijama pisanja.

Najčešće korišćeni stilovi uključuju `LLVM`, `Google`, `Mozilla`, `WebKit` i `Chromium`, a korisnik može definisati sopstveni stil putem konfiguracionog fajla `.clang-format`. Alat funkcioniše tako što automatski primenjuje ta pravila na zadate datoteke ili čitave direktorijume, bez promene same funkcionalnosti koda.

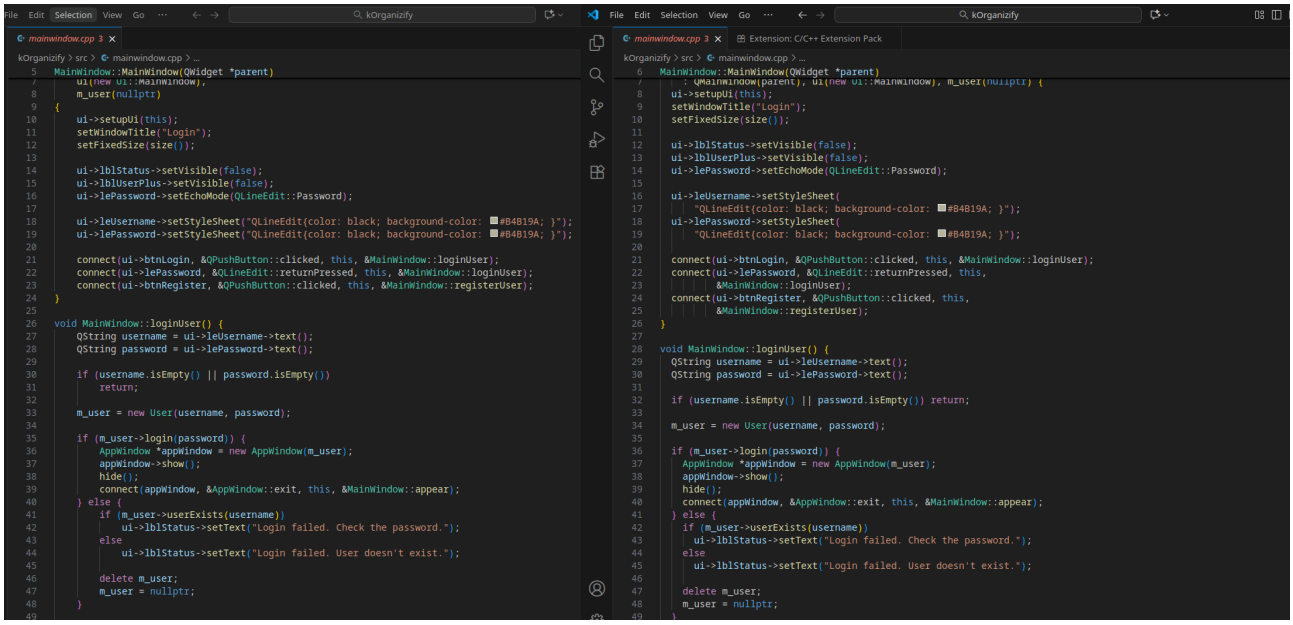
4.2 Rezultati primene

Alat `Clang-Format` je primenjen na sve `.cpp` i `.h` fajlove u okviru direktorijuma `kOrganizify`. Rezultati formatiranja sačuvani su u direktorijumu `formatted_output`, koji zadržava identičnu strukturu kao originalni direktorijum `src`.

Upoređivanjem originalnih i formatiranih fajlova pomoću komande:

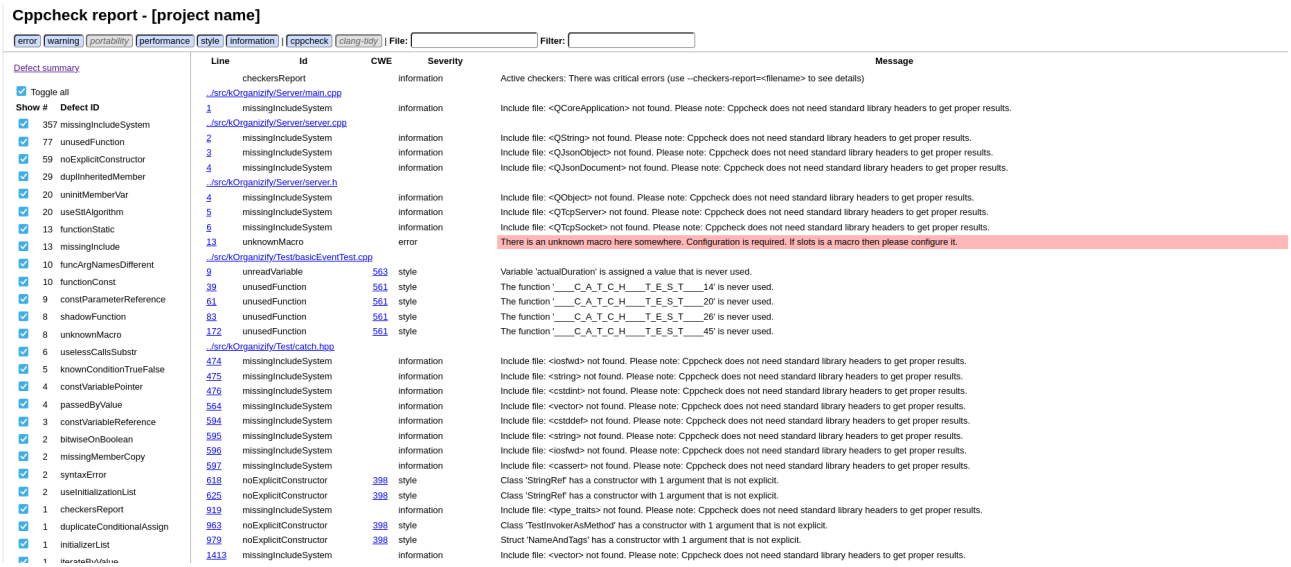
```
diff -rq ../src formatted_output
```

utvrđeno je da su svi fajlovi uspešno formatirani. Promene se odnose isključivo na stil pisanja — uvlačenje, razmake oko operatora, poravnanje zagrada i uniformnost komentara.



Slika 1: Poređenje originalnog i formatiranog fajla `mainwindow.cpp` u VS Code okruženju. Levo je prikazan izvorni kod pre primene `Clang-Format` alata, dok je desno prikazan rezultat nakon formatiranja. Uočava se uniformnije uvlačenje, dosledno razdvajanje operatora razmacima i poboljšana čitljivost koda.

Iako `Clang-Format` poboljšava konzistentnost i strukturu koda, utisak o čitljivosti je donekle subjektivan. Različiti stilovi formatiranja (npr. `Google`, `LLVM`, `Mozilla`) mogu delovati više ili manje pregledno u zavisnosti od ličnih preferencija i navika programera. U konkretnom primeru, formatirani kod ne mora nužno izgledati čitljivije, ali doprinosi ujednačenosti celog projekta.



Slika 2: Prikaz rezultata analize projekta `kOrganizify` pomoću alata `Cppcheck`. Na slici je prikazan HTML izveštaj sa detektovanim upozorenjima različite težine: stilske preporuke, neiskorišćene promenljive, neinicijalizovani članovi klasa i informativne poruke o nedostajućim hederima. Prikazana upozorenja služe kao smernice za poboljšanje kvaliteta i održivosti koda.

5 Valgrind (Memcheck)

5.1 Opis alata

Valgrind je alat za dinamičku analizu C i C++ programa, a njegova najpoznatija komponenta **Memcheck** služi za otkrivanje grešaka u radu sa memorijom. Za razliku od statičkih analizatora, Valgrind izvršava program i prati ponašanje memorije tokom rada.

Memcheck detektuje sledeće vrste problema:

- korišćenje neinicijalizovane memorije,
- curenje memorije (memory leaks),
- pristup oslobođenim ili nevažećim memorijskim adresama,
- nepravilne alokacije i dealokacije,
- čitanje i pisanje izvan granica niza (out-of-bounds access).

Ovi podaci su dragoceni u procesu verifikacije jer omogućavaju rano otkrivanje kritičnih grešaka koje bi u realnom okruženju dovele do prekida programa ili nepredvidivog ponašanja. Memcheck takođe može da se kombinuje sa drugim alatima (kao što su Callgrind ili Cachegrind) za detaljnije profilisanje performansi.

5.2 Rezultati analize

Analiza sprovedena alatom **Valgrind Memcheck** pokazala je da projekat `kOrganizify` ne sadrži kritične greške u radu sa memorijom. Tokom testiranja klijentske aplikacije izvršeno je više korisničkih akcija (prijavljivanje, kreiranje i uređivanje događaja, promene tema i zatvaranje aplikacije), čime je simulirano uobičajeno korišćenje programa.

Memcheck nije prijavio ni jedan slučaj *definitely lost* ili *invalid free*, što znači da ne postoje stvarna curenja memorije. Pronađeni zapisi poput:

```
Syscall param writev(vector[0]) points to uninitialised byte(s)
Invalid read of size 16
```

odnose se na interne pozive unutar Qt biblioteka (`libQt6XcbQpa`, `libQt6Gui`, `libQt6Svg`) i nisu povezani sa izvornim kodom projekta.

U završnom izveštaju stoji da je ukupno zauzeće memorije na izlazu iz programa oko 4MB, bez ikakvih neoslobođenih blokova koji bi ukazivali na curenje:

```
HEAP SUMMARY:
in use at exit: 4,214,260 bytes in 17,335 blocks
total heap usage: 442,626 allocs, 425,291 frees
```

Ove vrednosti potiču iz radnih struktura Qt okvira i prikazuju normalno ponašanje aplikacije. Dakle, može se zaključiti da projekat `kOrganizify` efikasno upravlja memorijom i da tokom tipičnog rada ne dolazi do nevalidnih pristupa, curenja ili inicijalizacionih grešaka.

6 Valgrind (Massif)

6.1 Opis alata

Massif je alat iz paketa **Valgrind** namenjen analizi potrošnje memorije na *heap*-u tokom izvršavanja programa. Za razliku od alata **Memcheck**, koji otkriva greške u radu sa memorijom, Massif meri koliko memorije program dinamički zauzima u svakom trenutku i koje funkcije su odgovorne za najveće alokacije.

Massif omogućava identifikaciju delova koda koji koriste previše memorije, što može biti od ključnog značaja za optimizaciju performansi i smanjenje ukupne potrošnje resursa. Pored ukupne potrošnje, alat prati i veličinu pojedinačnih `malloc`, `new` i sličnih poziva, kao i njihov uticaj na ukupno zauzeće memorije.

Rezultati Massif analize se čuvaju u fajlu `massif.out.<pid>` i mogu se detaljno pregledati pomoću alata `ms_print`, koji generiše čitljiv tekstualni izveštaj o raspodeli memorije po funkcijama i vremenu izvršavanja. Za vizuelni prikaz može se koristiti alat `massif-visualizer`, koji generiše grafikon iskorišćenosti memorije kroz vreme, što olakšava prepoznavanje trenutaka najvećeg zauzeća i eventualnih curenja memorije.

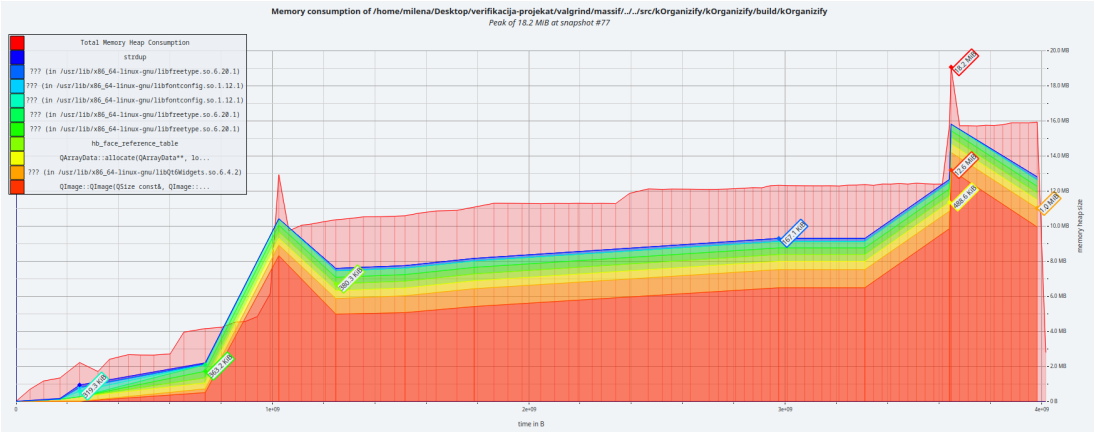
6.2 Rezultati analize

Analiza alatom Massif sprovedena je nad izvršnim fajlom projekta `kOrganizify`, pri čemu je praćeno zauzeće heap memorije tokom pokretanja, korišćenja i zatvaranja aplikacije. Izlazni grafikon (slika 3) prikazuje nagli porast zauzeća memorije u ranoj fazi rada programa, nakon čega sledi postepeno povećanje do maksimalne vrednosti od oko 18 MB, a zatim blagi pad prilikom završetka rada. Ovakav oblik krive ukazuje na tipičan obrazac inicijalizacije Qt GUI biblioteka — inicijalni skok predstavlja učitavanje interfejsa, dok drugi manji porast potiče od dinamičkih elemenata tokom korišćenja.

Iz `ms_print` izveštaja može se uočiti da je vrh zauzeća memorije (snapshot 77) dostignut pri oko 18,2 MB ukupne heap memorije, pri čemu većina alokacija potiče iz Qt modula zaduženih za fontove i grafičke elemente:

```
99.20% (18,200,000B) main
 -> QFontDatabasePrivate::ensureFontDatabase()
 -> QFontconfigDatabase::populateFontDatabase()
 -> QGuiApplicationPrivate::createPlatformIntegration()
 -> Calendar::loadEvents()
```

Ovi podaci pokazuju da najveći deo zauzete memorije pripada inicijalizaciji korisničkog interfejsa i sistemskih resursa. Nakon završetka rada aplikacije dolazi do pravilnog oslobađanja memorije, što potvrđuje da nema trajnih curenja.



Slika 3: Grafikon zauzeća memorije tokom rada aplikacije kOrganizify (Massif Visualizer).

Ukupno je registrovano 89 snapshotova, od čega su 12 detaljno obrađeni. Ovakvi rezultati ukazuju na efikasno upravljanje memorijom i odsustvo ozbiljnih problema u dinamičkoj alokaciji.

7 Zaključak

Korišćenjem različitih alata za statičku i dinamičku analizu, odrađena je provera ispravnosti i kvaliteta koda projekta **kOrganizify**.

Statički alati, poput `Cppcheck`-a i `Clang-Format`-a, doprinose poboljšanju čitljivosti i konzistentnosti koda. `Cppcheck` omogućava rano otkrivanje mogućih logičkih i stilskih nedoslednosti, dok `Clang-Format` obezbeđuje automatsko usklađivanje stilskih pravila, čime se olakšava održavanje projekta. Alati iz paketa `Valgrind` detektuju probleme koji se javljaju tokom izvršavanja programa.

Zajedničkom primenom ovih pristupa postiže se bolja stabilnost, čitljivost i dugoročna održivost projekta, što predstavlja osnovni cilj verifikacije softvera.