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** i **Clang-tidy** – alati za automatsko formatiranje i proveru stilskih i logičkih pravila u kodu.
- **Valgrind (Memcheck)** – alat za dinamičku analizu i otkrivanje grešaka u radu sa memorijom.
- **Jedinični testovi (unit tests)** – testovi koji proveravaju pojedinačne funkcionalnosti sistema.

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

Primena `Clang-Format`-a na projekat `kOrganizify` obuhvatiće automatsko usklađivanje indentacije, razmaka oko operatora, raspored komentara i redosled `#include` direktiva. Rezultati će biti prikazani kroz primere pre i posle primene alata, radi vizuelne ilustracije efekta formatiranja.

5 Valgrind (Memcheck)

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.

6 Jedinični testovi (Unit tests)

Jedinični testovi predstavljaju osnovnu metodu dinamičke verifikacije softvera. Cilj im je da provere ispravnost pojedinačnih funkcija, klasa ili modula – nezavisno od ostatka sistema. Na taj način, greške se mogu detektovati rano, pre nego što utiču na složenije delove koda.

Tipična struktura jediničnog testa obuhvata:

- pripremu ulaznih podataka (Arrange),
- izvršavanje funkcije ili metode koja se testira (Act),
- proveru rezultata u odnosu na očekivane vrednosti (Assert).

U okviru C++ projekata, testovi se najčešće realizuju pomoću okvira kao što su *QtTest*, `Catch2`, `GoogleTest` ili `doctest`, ali se mogu pisati i ručno, bez dodatnih biblioteka. U kontekstu verifikacije softvera, jedinični testovi omogućavaju procenu ispravne funkcionalnosti. Oni se često koriste u kombinaciji sa alatima za merenje pokrivenosti koda, što omogućava uvid u to koji delovi programa su testirani, a koji nisu.

7 Zaključak

Korišćenjem različitih alata za statičku i dinamičku analizu, kao i jediničnih testova, obezbeđena je sveobuhvatna 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, dok jedinični testovi potvrđuju ispravnost funkcionalnosti i daju osnovu za merenje pokrivenosti koda.

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