SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET

CROSS PLATFORM PROGRAMMING,TESTING AND BENCHMARKING

Student: Marko Piskač

Kolovoz, 2016.

Sadržaj

Uvod

U programiranju, cross-platform software je računalni software koji se može implementirati na više platforma. Cross-platform software se može podijeliti u dva tipa: prvi koji zahtjeva izgradnju i kompajliranje za svaku platformu koju podržava, te drugi koji se može direktno izvršavati na bilo kojoj platformi bez potrebe za posebnom pripremom software-a.

Da bi smo za dio koda mogli reći da je cross-platformski, on mora funkcionirati na više od jednoj arhitekturi ili platformi. Razvijanje takvoga programa može biti vremenski vrlo zahtjevno jer različiti operativni sustavi imaju različite API-je npr. Windows koristi drugačiji API u odnosu na Linux.

Postoje dva različita načina razvijanja cross-platformskog programa. Jedan način je da se jednostavno kreiraju više verzija istog programa u drugačijem „source trees“, na način da Windows verzija ima svoj skup izvršnog koda dok bi Linux imao svoju verziju izvornog koda. Taj način će biti skuplji te će zahtijevati više vremena za razvoj a isto tako biti će otežano rješavanje „bugova“ na više strana to jest na više platforma. Ideja je kreirati više od dva programa koji će imati sposobnost ponašati se slično ili isto. Drugi način bi bio razviti program u takvom alatu koji ima sposobnost sakriti razlike između platformi (abstrakcija platforme), na način da program nema informaciju na kojoj se platformi izvršava. Primjer za to možemo naći u Java Virtual Machine, to jest programi koji se izvršavaju u Java Virtual Machine su izvedeni na taj način. Neke aplikaciju su „miks“ različitih metoda razvoja kako bi se dobila krajnja gotova aplikacija.

Izazovi na koje se nalazi tijekom razvijanja cross-platform programa:

- Testiranje cross-platforma aplikacija postaje dosta komplicirano zbog načina na koji se različite platforme ponašaju na greške u izvršnom kodu.

- Različite platforme imaju često različite konvencije za korisničko sučelje. Primjer toga je, aplikacije razvijene za Mac OS i GNOME trebaju imati najvažniji „button“ smješten na desnu stranu prozora, dok Windowsa i KDE imaju obrnutu konvenciju.

- Okruženja u kojim se izvršava cross-platform aplikacija mogu imati nedovoljno dobro razvijene i implementirane zabrane (security flaws)za croos-platform aplikacije, što čini plodno tlo za razvoj virusa.

- Skriptni jezici i virtualne mašine moraju biti prevedene u izvršni kod svaki puta kada je aplikacija pokrenuta, što zahtjeva puno više performansa od računala.

- Programeri su često primorani koristiti najniže značajke koje su dostupne na svim platformama. To može ometati performanse aplikacije i zabraniti programeru da koristiti najnaprednije značajke na platformi.

Pravila kako pisati cross-platform program

Jedan od razloga zašto pišemo cross-platform programe je i taj što nam to pruža kvalitetniji kod. Kompajleri na svakoj platformi imaju različita upozorenja, i mogu nam pružiti dobre savjete pri kodu koji se na jednoj platformi kompajlirao bez greške a na drugoj s greškama i upozorenjima.

Istodobno razvijanje

Kada programer dizajnira ili implementira novi „patch“, on mora razmišljati o svim platformama kojima je namijenjena aplikacija, i tek kada je uspio razviti dio koda koji je isprobao na svim platformama, tek onda može reći da je završio. Po nekim studijama razvijanje koda u programskom jeziku C za više platforma, uzima tek 5% vremena više u odnosu na jednoplatformsku aplikaciju. Ali ako se aplikacija razvija godinu dana za Windows platformu i tek onda se pokuša primijeniti i na Linux platformu doći ćemo do velikih problema. Kada se govori o istodobnom razvijanju, to podrazumijeva da pri dizajniranju aplikacije imamo u vidu da će aplikacija biti višeplatformska te da ćemo biti u mogućnosti nakon razvitka aplikacije, aplikaciju prebaciti na drugu platformu i u nekoliko sati je prilagoditi toj platformi. Postoji više razloga zašto ne bi trebali godinu dana razvijati aplikaciju te je tek onda isprobati na drugoj platformi. Prvi razlog je, dok programer radi istodobno za više platforma na novom feature-u on je svjestan dizajna kakav program mora imati i rubnih slučajeva u programa. Ukoliko se radi samo za jednu platformu i tek nakon godinu dana se ide primijeniti program na drugu platformu, programer će se naći u velikim problemima jer se tada više neće prisjećati dizajna koda i rubnih slučajeva. Osnovni razlog zbog kojega bi programeri trebali istodobno raditi za više platforma je što ponekad programeri dolaze do rješenja prečicom što u neznanju što u nedostatku samokontrole ne brinući o drugoj platformi. Primjer za to možemo naći u korištenju Windows registra. Windows registri su esencijalni u API-ju za spremanje naziv-vrijednost parova na poznatu lokaciju u podatkovnom sistemu, što je slično spremanju naziv-vrijednost parova u XML dokumentu. Ako programer ne uzima u obzir da radi program za više platformi i zapisuje vrijednosti u registar, tada nakon određenog vremena kada želi program prebaciti na drugu platformu naići će na probleme jer sve podatke koje je zapisao u registar na novoj platformi neće postojati. Kako bi se to izbjeglo programer bi trebao podatke zapisivati u XML dokument te naj taj način će program raditi na svim platformama bez promjena. Na kraju, najgora stvar koju programer može napraviti je dati nekoj drugoj firmi ili drugom programeru da nastavi njegov rad. Prvobitni programer najbolje poznaje svoj kod i upoznat je sa mogućim greškama. Prosljeđivanje programa na nekog drugog može dovesti do problema u komunikaciji, do greške pri ponovnom spajanju koda između dvije platforme itd.

Koristiti standardne „C“ tipove podataka

Ovo se čini logično, no najčešća greška koja vodi gomilanju koda kojeg je teže ispraviti kasnije je korištenje platformskih tipova podataka umjesto standardnih „C“ tipova podataka. Primjer za to možemo naći za tip podataka „DWORD“ koji nije specificiran u jeziku C ali je definiran od stane Microsofta kao „typedef unsigned long DWORD“. Očigledno je da pri cross-platform programu treba koristiti originalni „C“ tip „unsigned long“ umjesto „typedef unsigned long DWORD“ ali i dalje je to učestala pogreška kod programera koji su naviknuti na Microsoft okruženje. Najvažnije mjesto gdje treba primjenjivati ovo pravilo je u cross-platform knjižnici. U Macintoshu se na primjer ne može niti pozvati funkciju koja kao argument uzima DWORD, već se treba koristiti unsigned long.

Koristiti već postojeće „ifdef flags“

Ukoliko je potrebno implementirati neki dio koda koja je specifičan za pojedinu platformu, to treba staviti u standard #ifdefs. Dobar primjer za to možemo naći u Visual Studiu koji ima u #ifdefs-u kompajler zastavicu(„flag“) nazvanu „_WIN32“ , koja nam osigurava da će napisani kod se sa sigurnošću izvršavati na Windows platformi. Ne postoji niti jedan razlog zbog čega bi programer mora raditi svoju verziju toga, ukoliko se koristi sintaksa na dolje navedeni način.

```
#ifdef _WIN32
// Microsoft Windows specifični pozivi
#endif
```

Na taj način kompajliranje program će uvijek biti uspješno, bez obzira koje Makefilove ili Visual Studio koristimo i bez obzira ako programer taj dio koda kopira u neki drugu izvršnu datoteku. Mnoge besplatne knjižnice na Internet inzistiraju postavljanje mnogo zastavica ispravno, i ukoliko koristimo kod iz takvih knjižnica prouzročiti će mo mnogo problema prilikom prebacivanja programa na drugu platformu.

Korištenje Unicode-a (UTF-8) za sve API-je

Korištenje Unicode-a je podržano na svim računalima i aplikacijama u svijetu, te bi pri razvitku cross-platform programa trebali korisiti Unicode UTF-8. Svi poznati operativni sustavi kao što su: Windows Vista, Macintosh OS X, Linux , programski jezici: Java, C#, email programi: Microsoft Outlook, Outlook Express, Gmail te svi veći Internet pretraživači koriste UTF-8.

Korištenje dodatnih framwork-ova i knjižnica

Korištenje dodatnih knjižnica ponekad je dobro kako bi svojoj aplikaciji dodali dodatne funkcionalnosti. Primjer možemo dati za OpenSSL knjižnicu koja je besplatna i omogućuje nam brzu enkripciju. Za razliku od knjižnica koje su nam neophodno potrebne, dodavanje knjižnica koje nam ne daju novu funkcionalnost našoj aplikaciji i tu je samo kako bi nam omogućila neke prečice pri programiranju je posve krivo. Cijela pretpostavka u izgradnji cross-platform aplikacije je da se zasniva na čistom C ili C++ kodu, nije potrebno dodavati nikakve dodatne knjižnice kako bi aplikacija bila podržana na svim platformama. Postoji mnogo framework-ova za koje tvrde da će nam skratiti vrijeme u izgradnji naše aplikacije, ali će na kraju samo smanjiti funkcionalnost aplikacije. Kao primjer možemo uzeti Qt, ZooLib, GLUI/GLUT, CPLAT, GTK+, JAPI itd. U najgorem slučaju ti framework-ovi će „napuhati“ našu aplikaciju sa nepotrebnim dodatnim knjižnicama i runtime okruženjima te prouzročiti dodatne probleme sa kompatibilnosti. Ispravan način korištenje dodatnih knjižnica bi bio razvijanje svojih knjižnica a ne korištenje tuđih. Taj način će nam uzeti malo više vremena no to nam pruža potpunu kontrolu nad našom cross-platform aplikacijom.

Korišenje posebnih znakova

Korištenje posebnih znakova treba izbjegavati, jer su ti znakovi u nekim operacijskim sustavima rezervirani za posebne naredbe.

Avoid	Example characters	Reasons
File separators	: (colon) / (forward-slash) \ (backward-slash)	You should avoid using colons and slashes in the names of files and folders because some operating systems and drive formats use these characters as directory separators. Consider substituting an underline (_) or dash (-) where would normally like to use a slash or colon in a filename.
Non-alphabetical and non-numerical symbols	ç™$®	Non alphanumeric characters may not be supported by all file systems or operating systems, or may be difficult to work with when exported to certain file formats such as EDL, OMF, or XML.
Punctuation marks, parentheses, quotation marks, brackets and operators	, [ ] { } ( ) ! ; " ' * ? < > |	These characters are often reserved for special functions in scripting and programming languages.
White space characters such as spaces, tabs, new lines and embedded returns.		Although OS X and Mac OS formatted disks support spaces in filenames, certain processing scripts and applications may not recognize these characters, or may treat your files differently than expected. Consider substituting an underline (_) or dash (-) where you would normally use spaces.

Razvijanje cross-platform programa u timu

Pri razvijanju cross-platform programa u timu najvažnije je postaviti instrukcije u radu te definirati pravila rada na projektu. Svi programeri moraju biti odgovorni i pridržavati se gore navedenih pravila. Što ponajprije znači da moraju istodobno provjeravati funkcionalnost njihovog koda na svim platformama na kojim rade. Za to mogu koristiti kontrolni sistem za izvršni kod (npr. CVS) kao sinkronizaciju između više platforma pri kompajliranju programa. Na taj način kada se doda nova zakrpa (patch) i istestira se npr. Na Windows operacijskom sustavu, kontrolni sistem odmah provjerava ispravnost zakrpe na ostalim platformama. Na taj način se pridržava osnovno pravilo o istodobnom razvijanju programa za više platforma. Ukoliko dođe do neke greške programer može pravovremeno reagirati te ispraviti kod.

Alati

CMake

CMake je cross-platform besplatni open-source software za upravljanje i razvijanje software-a koristeći neovisni kompajler. Dizajniran je na način da podržava hijerarhiju direktorija i aplikacija koje ovise o više knjižnica. Koristi se sa razvojnim okolinama kao što cu Visual Studio (Windows), Xcode (Mac) i Kdevelop (Linux). Ima minimalnu ovisnost, zahtijevajući samo C++ kompajler pri izgradnji programa. CMake može generirati razvojnu okolinu koja će kompajlirati izvorni kod, kreirati knjižnice, izgraditi izvršne datoteke u proizvoljnim kombinacijama. CMake isto tako podržava statične i dinamične knjižnice. Dodatak CMake-u je i to što može generirati cache datoteku koja je dizajnirana kako bi se mogla koristiti s grafičkim editorom. Primjer: kada je CMake pokrenut, on locira datoteke, knjižnice i izvršne datoteke, te informacije se skupljaju u cache koje korisnik može promijeniti prije nego što se generiraju „build files“. Kako sam već rekao CMake je dizajniran kako bi mogao podržati kompleksnu hijerarhiju direktorija i aplikacija ovisno o nekoliko knjižnica. Primjer: CMake podržava projekte koji se zasnivaju na više alata (knjižnica), gdje se svaki alat sastoji od nekoliko direktorija, i tada aplikacija ovisi o alatima i dodatnom kodu. CMake isto tako podržava slučajeve kada izvršne datoteke moraju biti izgrađene u cilju generiranja koda koji je kasnije kompajliran i linkan u krajnju aplikaciju. Korištenje CMake nije komplicirano. Proces izgradnje se temelji na datotekama CMakeLists.txt u svakom direktoriju, uključujući i poddirektorije, koje čine projekt. Svaka datoteka CMakeLists.txt se sastoji od jedne ili više komandi. Svaka komanda ima formu COMMAND (arg) gdje COMMAND predstavlja naziv komande, a arg predstavlja listu argumenata. Cmake nam pruža unaprijed definirane komande, ali isto tako dopušteno je pisanje svojih komandi.

Primjer CmakeLists.txt

```
# stop if cmake version below 2.8
cmake_minimum_required(VERSION 2.8 FATAL_ERROR)
# project name
project(example)
# enable fortran, c, and c++ language
enable_language(Fortran C CXX)
# location of additional cmake modules
set(CMAKE_MODULE_PATH
   ${CMAKE_MODULE_PATH}
   ${CMAKE_CURRENT_SOURCE_DIR}/cmake)
# detect operating system
message(STATUS "We are on a ${CMAKE_SYSTEM_NAME} system")
if(${CMAKE_SYSTEM_NAME} STREQUAL "Linux")
   add_definitions(-DSYSTEM_LINUX)
endif()
if(${CMAKE_SYSTEM_NAME} MATCHES "Windows")
   add_definitions(-DSYSTEM_WINDOWS)
endif()
# example how to set c++ compiler flags for GNU
if(CMAKE_CXX_COMPILER_ID MATCHES GNU)
   set(CMAKE_CXX_FLAGS        "${CMAKE_CXX_FLAGS} -Wall -Wno-unknown-pragmas -Wno-
sign-compare -Woverloaded-virtual -Wwrite-strings -Wno-unused")
   set(CMAKE_CXX_FLAGS_DEBUG   "-O0 -g3")
   set(CMAKE_CXX_FLAGS_RELEASE "-O3")
   set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -fprofile-arcs -ftest-coverage")
endif()
# build executable
add_executable(hello.x src/hello.F90)
# location of header files
include_directories(
   ${PROJECT_BINARY_DIR}
   ${PROJECT_SOURCE_DIR}/src  )
# configure header file
configure_file(
   ${PROJECT_SOURCE_DIR}/cmake/config.h.in
   ${PROJECT_BINARY_DIR}/config.h )
# compile main executable
add_executable(main.x src/main.cpp)
# link library
target_link_libraries(main.x mylib_static)
```

Alati koji koriste CMake:

- Allegro library
- Armadillo – linear algebra library
- Avidemux
- awesome – window manager
- BCI2000
- Blender
- BRL-CAD
- Bullet Physics Engine
- CGAL – Computational Geometry Algorithms Library
- Chipmunk physics engine
- CLion
- Compiz
- Conky
- cURL
- Deal.II
- Doomsday Engine
- Dust Racing 2D
- Drishti
- Ettercap
- Falcon (programming language)
- FlightGear Flight Simulator
- GDCM
- Geant4
- Gmsh
- GNU Radio
- GROMACS
- Hiawatha (web server)
- Hypertable
- Hugin
- iCub robot and YARP
- IGSTK
- Insight Segmentation and Registration Toolkit (ITK)
- KDE SC 4
- KiCad
- libpng
- LAPACK
- LLVM and Clang
- LMMS
- Mir
- MiKTeX
- MLPACK – machine learning library
- MuseScore
- MySQL and MariaDB
- OGRE
- OpenCV
- OpenCog
- OpenCPN
- OpenSceneGraph
- OpenSync
- Orthanc
- Point Cloud Library
- Poppler
- PvPGN
- QGIS
- Qt
- Raw Therapee
- ReactOS
- ROOT
- ROS
- Ryzom
- Scribus
- SDL
- Second Life
- SFML
- Spring RTS
- SuperTux
- Synergy
- Slicer
- Stellarium
- Trilinos
- Vortexje
- The Visualization Toolkit and ParaView
- VXL
- zlib
- PCSX2
- Zdoom
- ZeroMQ

Qt

Qt je cross-platform framework koji ima široku upotrebu u razvijanju softwarea i može se pokretati na različitima software-ima i platformama uz male ili gotovo nikakve promjene u kodu. Qt je razvijen od strane Qt Company te postoje dvije verzije, open source i commercial. Qt se koristi uglavnom za razvoj software-a sa grafičkim sučeljem. Qt koristi standardni C++ jezik sa nekim dodacima koji pojednostavljuju rukovanje s „eventima“ te to pomaže pri razvijanju GUI i server aplikacija koje primaju skup informacija o eventima te ih prema tome trebaju procesuirati. Qt ima podršku za mnoge kompajlere uključujući i GCC C+ kompajler i Visual Studio. Qt predstavlja integriranu razvojnu okolinu koja omogućuje korištenje alata za dizajniranje i razvoj aplikacija posredstvom korisničkog sučelja. Koristeći Qt isto tako mogu se pisati i web skripte.

Značajke:

- Kada jednom napišemo kod on se povezuje na više platformi. Koristeći Qt moguće je napraviti naprednu aplikaciju s korisničkim sučeljem uz ugrađeni operacijski sustav bez ponovnog prepisivanja izvornog koda.

- Bez obzira da li koristimo C++ ili java jezik, Qt nam pruža širok skup „build“ datoteka koje su prilagodljive te nam pružaju grafičko okruženje sve ostalo potrebno za razvoj modernog korisničkog sučelja.

- Brza izgradnja softwarea. Koristeći Qt-Creator moguće je u malo vremena postići velike rezultate. Uz Qt knjžnice te klase više vremena se posvećuje inovacijama nego li brizi oko infrastrukture kodiranja.

- Mogućnost integracija Qt sa Web okruženjem znači da se mogu brzo ubaciti sadržaji i usluge sa Interneta na aplikacijama.

Aplikacije koje koriste Qt

- AMD's Radeon Software
- Autodesk Maya
- CryEngine
- Dragonframe
- FreeMat
- Google Earth
- Krita
- Orange
- Scribus
- Spotify
- Stellarium
- Subsurface
- Telegram
- VirtualBox
- VLC media player
- XnView MP

Qt Creator

Qt Creator je cross-platform C++, JavaScript i QML integrirana razvojna okolina koja je dio SDK za QT GUI. Uključuje grafički debugger i integrirani GUI layout (raspored) s dizajnerom za formu. Koristi C++ kompajler iz GNU kolekcije kompajlera za Lniux te MinGw ili MSVC za Windows operacijski sustav.

Editori

Qt Creator uključuje editor koda i integrira Qt Designer za dizajniranje i građenje korisničkog sučelja iz Qt widgets-a. Editor koda u Qt Creator-u podržava istančavanje sintakse za različite jezike.

Qt Designer je alat za dizajniranje i izgradnju korisničkog sučelja iz Qt widgets-a. Moguće je sastaviti i prilagoditi widgets-e i tada ih testirati koristeći različite stilove u editoru. Widgets-i i forme kreirane sa Qt Designer su integrirane s programskim kodom koristeći Qt „signals and slots“ mehanizme. Korištenje Qt Designer-a uključuje četiri osnovna koraka:

1. izabrati svoju formu i objekte
2. postaviti objekat na obrascu
3. povezati signale i mjesta djelovanja
4. pregled formulara

Qt Quick Designer je alat za razvoj animacija koristeći programski jezik QML.

Qt Widgets

Widgets-i su osnovni blokovi za grafički korisničko sučelje (GUI) koji su ugrađeni u aplikacijama u Qt-u. Postoji više vrsta Widget-a koji su određeni podklasom QWidget-a koja je sama podklasa klase Qobject koja je inače glavna klasa za rukovanje objektima. U Qt-u se može koristi kao skladište za druge dodatke. QWidget se koristi za kreiranje prozora glavnog roditeljskog dodatka u kojemu se nalaze djeca widget-i.

Popis nekih od cross platform alata i okruženja

- **Cairo**: Je besplatni software knjižnica koja se temelji na grafici vektora, neovisno o API-ju uređaja. Dizajnirana da pruža dvodimenzionalno crtanje . Cairo je napisan u C-u i ima podršku za mnoge programske jezike.
- **Eclipse**: Je open source cross-platform razvojna okolina. Implementirana je u Javi s podesivom arhitekturom koja podržava mnoge alate za razvoj software-a. Dodaci su dostupni za nekoliko jezika uključujući Java i C++.

- **FLTK**: Je open source cross platform alat, ali je mnogo jednostavniji jer se ograničava samo na GUI.
- **fpGUI**: Je open source widget alat koji je u cijelosti implementiran u Object Pascal-u. Podržava Linux, Windows i dio Windows CE.
- **GeneXus**: Je Windows software za brzo rješenje kod cross platform aplikacije za stvaranje i implementaciju na temelju prikaza znanja i podrške za C#, COBOL, Java uključujući Android i BlackBery pametne uređaje, Objective C za Apple mobilne uređaje, RPG, Ruby, Visual Basic i Visual FoxPro
- **Haxe**: Je open source cross-platform jezik.
- **Juce**: Je framework napisan u C++, koristi se za pisanje software na velikom broju sustava (Microsoft Windows, POSIX, Mac OS X) bez promjena u kodu.
- **Lazarus**: Je programsko okruženje za FreePascal kompajler, Pruža podršku za stvaranje samostojećih grafičkih i konzola aplikacija te se pokreće na linuxu, MacOSX, Android, WinCE, Windows i Webu.
- **Max/MSP**: Je vizualni programski jezik koji enkapsulira nezavisno platformski kod s platformski specifičnom runtime okolinom u aplikaciji za Mac OS X i Windows.
- **MechDome**: Je cross platform Android runtime. Omogućuje nemodificiranim Android aplikacijama da se pokrenu na IOS i OS X.
- **Mono**: Je open source cross platform verzija Microsoft.NET-A (framework za aplikacije i programske jezike).
- **MoSync**: Je open source SDK za platformu za razvitak mobilnih aplikacija u jeziku C++
- Mozilla application framework: Je open source platforma za razvijanje Mac OS X, Windows i Linux aplikacija.
- **OpenGL:** A cross-platform 3D grafička knjižnica.
- **Simple and Fast Multimedia Library**: Je multimedija C++ API koja pruža niski i visoki level pristupa grafici, audiu, unosu itd.
- **Smartface**: Je cross platform aplikacija ili razvojni alat za stvaranje mobilnih aplikacija za Android i iOS, koristeći WYSIWYG editor za dizajn s JavaScrip editorom koda.
- **Tcl/Tk**
- **WinDev**: *Je integrirana razvojna okolnia za Windows, Linux, .Net i Javu*
- **wxWidgets**: Je open source widget alat koji je i ujedno i framework. Pokreće se na Unix sustavima s X11, Microsoft Windows i Mac OS X. Dozvoljava napisanim aplikacijama na svim navedenim operacijskim sustavima.
- **XPower++**: Je cross-platform IDE za Windows, Linux, Mac OS X te mobilne platforme.
- **Delphi**: A cross platform IDE, koja koristi Pascal jezik za razvoj aplikacija. Podržava Android, iOS, Windows, OS X.
- **Xojo**: Koristi objektno orijentirano programski jezik za kreiranje destkop, web i iOS aplikacija. Xojo kreira kompajlirane aplikacije za Mac OS X, Windows, Linux i Raspberry Pi. Kreira kompajlirane web aplikacije koje mogu biti pokrenute kako samostojeći serveri ili CGI.
- **Ultimate++:** je C++ cross platform framework za razvoj brzih aplikacija koji je orijentiran na produktivnost programera. Uključuje niz knjižnica (GUI, SQL, etc..) i integriranu razvojnu okolinu. Ima podršku za Windows i Unix sustave. U++ se natječe s popularnim skriptnim jezicima dok u isto vrijeme čuva C/C++ runtime karakteristike. Ima svoju integriranu razvojnu okolinu TheIDE, koja ima tehnologiju BLITZ-build za brže C++ rebuild-anje do 4 puta.

Testovi

Mnogi programeri gledaju na testiranje programa kao neki dokaz da su dobro izvršili svoj dio posla i da programi ispravno rade. No to bi trebalo gledati s druge strane i reći da testovi služe tome kako bi se pronašle greške u kodu, jedino kada se pronađe greška možemo reći da je test uspio.

Testiranje je proces koji bi trebao početi mnogo prije samog kodiranja. Početak testiranja proizvoda nakon što je navodno „završen“ je marginalno bolji, ali bi testni proces trebao početi mnogo prije nego što se uopće počne kodirati. Priprema za testiranje počinje u toku faze analize, identifikacije testnih slučajeva uz korištenje robusnih dijagrama. Ranim testiranjem, moguće je eliminirati mnogo više kvarova, prije nego što se uopće pojave. Testiranje slijedi ubrzo nakon preliminarnog dizajna, a onda pisanje unit test kodova tokom implementacije. Moramo biti sigurni da su testovi povezani zajedno za zahtjevima. Ne iz razloga da svaki test mora biti praćen unazad ka zahtjevima, ali bi trebao bar postojati test da dokaže da je svaki zahtjev proveden točno.

DDT (design-driven testing) je dizajn vođen testiranjem. Pruža neprobojnu metodu za proizvodnju test slučajeva, kako bi mogli zaključiti kako su svi mogući scenariji završeni. Možemo također koristiti ovaj proces da napišemo izvršne testove iz testnih slučajeva.

Vrste testova:

Test jedinica (Unit test) - testiranje individualnih software-skih komponenti. Može se koristiti za simulaciju inputa i outputa komponente, tako da može raditi u samostalnom načinu. Testiranje jedinica počinje prije integracije te se izvršava na svakoj izgradnji software-a kroz razvojne faze.

Integrirani test (Integrated test) -testiranje se vrši kako bi se otkrile greške u sučeljima i u interakciji između integriranih komponenti. Vrši se nakon testiranja jedinica.

Test kompatibilnosti - testira se sustava i njegovo surađivanje s ostalim vanjskim sustavima sa kojima zahtjeva komunikaciju.

Test sustava - test funkcionalnosti test sučelja. Funkcionalno testiranje sastoji se od mnogih korištenih slučajeva i procesa ispitivanja te uključuje i testiranje funkcionalnosti zahtjeva. Vrši se nakon integracijskog testa. Test sustava teži k tome da bude fokusiran na razvoj. Njegova namjena je da dokaže da sustav koji je naznačen da je i dostavljen.

Test prihvatljivosti - formalno testiranje provedeno od strane kupca, utvrđuje se da li sustav zadovoljava zahtjeve zapisane u ugovoru. Provodi se u isto vrijeme kako i test sustava.

Beta test - provodi se od strane krajnjeg korisnika koji nije uključen u razvoj. Provodi se prije testa prihvatljivosti.

Test puštanja - Provjerava da li krajnji proizvod precizno odražava dizajn. Osigurava se da je proizvod razvijen u skladu sa specifikacijama i da postiže željene ciljeve

Unit test

Testovi se pišu prije samog izvornog koda koji treba zadovoljiti test. Najprije se testira svaka programska komponenta, nezavisno u odnosu na ostale dijelove koda. Takvo testiranje provjerava da li pojedinačne komponente ispravno funkcioniraju sa svim očekivanim tipovima ulaza, u skladu sa dizajnom komponente. Unit testiranje treba vršiti u kontroliranom okruženju tako da se može komponenti koja se testira predati unaprijed definiran skup podataka kako bi se mogli gledati izlazni podaci. Test treba ponavljati jer može doći do razlika u rezultatima pri jednom izvršavanju komponente i pri više ponavljanja. Dakle, ako očekujemo da će komponenta koja se testira iz stanja A, poslije izvršavanja dati rezultat B, onda to mora biti i pri ponavljanju izvođenja. Može postojati dinamika kretanja stanja komponente sve dok je ona točno unaprijed definirana. Kod testiranjem upravljanog programiranja test se uglavnom sastoji od četiri faze:

- priprema: priprema okruženja za testiranje i definiranje očekivanih rezultata,
- izvršavanje: pokretanje testa,
- validacija: usporedba dobivenih rezultata sa očekivanim rezultatima, i
- čišćenje: postavljanje okruženja onako kako je bilo prije testiranja.

Zašto testirati?

- Provjeriti da manja promjena nije uzrokovala nestabilnost u proizvodu
- Promjena implementacije metode
- Dokumentacija
- 2/3 posla u izgradnji nekog softvera otpada na održavanje (popravljanje bugova)

Prednosti korištenja unit testova:

- Smanjuje se vrijeme debugging-a. Vrijeme izgubljeno na debugging-u je period u kojem se ne kodira.
- Testiramo funkcionalnost koda na niskom nivou jer ako tu postoji greška, ona se "pogoršava" na većim nivoima
- Unit testiranje nam omogućuje da radimo velike promjene u kodu u malo vremenskom roku. Odmah znamo dali će kod funkcionirati ili ne, ukoliko kod nije ispravan, izvršimo promjene u kodu te ponovo testiramo. Na ovaj način možemo uštedjeti mnogo na vremenu.
- TDD nam pomaže shvatiti kada možemo prestati sa pisanjem koda. Test nam je dovoljni dokaz da smo dobro izvršili sa „uštimavanjem koda“, te možemo preći na druge zadatke.

- Unit test nam pomaže da shvatimo dizajn koda na kojemu radimo. Umjesto suhoparnog pisanja koda, programer počinje uviđati sva stanja u kojemu se program može naći te koja stanja očekivati na izlazu.
- Pisanje testa ne oduzima mnogo vremena jer je pisanje samog testa skoro pa trivijalno.
- Unit test nam daje trenutnu povratnu informaciju. To nam uvelike olakšava rad, u svakome trenutku možemo vidjeti ispravnost našeg koda. Ukoliko je došlo do greške u kodu odmah znamo gdje se ta greška nalazi.
- Testovi se mogu koristiti i kao dokumentacija (može se vidjeti kako kod treba koristiti).

Primjer:

U primjeru ću pokazati na koji način radi unit test. Imamo 2 modula. Prvi modul će samo za zadaću imati zbrojiti dva broja.

```
BOOL  dodaj(int a, int b)
{    return (a + b);  }
```

Taj modul ćemo iskoristiti u drugom modulu, to jest u drugoj C funkciji. Ta funkcija provjerava sve moguće slučajeve i vraća TRUE ili FALSE te na govori da li modul prolazi ili ne prolazi test.

```
BOOL dodajTest()
{    if ( dodaj(1, 2) != 3 )
       return (FALSE);
    if ( dodaj(0, 0) != 0 )
       return (FALSE);
    if ( dodaj(10, 0) != 10 )
       return (FALSE);
    if ( dodaj(-8, 0) != -8 )
    return (FALSE);
        ----------→
    if ( dodaj(5, -5) != 0 )
       return (FALSE);
    if ( dodaj(-5, 2) != -3 )
       return (FALSE);
    if ( dodaj(-4, -1) != -5 )
       return (FALSE);
       return (TRUE);
}
```

Kako možemo vidjeti istestirali smo sve moguće slučajeve: pozitivno+pozitivno, nula+nula,pozitivno+nula, negativno+nula, pozitivno+negativno, negativno+pozitivno, negatino+negativno. Svaki test uspoređuje dodatni rezultat s očekivanom vrijednosti i vraća FALSE ako je rezultat vrijednost drugačija od one očekivane. Ako kompajler dođe do zadnje linije, smatramo da su svi testovi uspješno odrađeni te se vraća vrijednost TRUE.

Današnji programi se velike i kompleksne aplikacije zato je važno koristiti alate koji nam omogućuju automatizirano izvršavanje testova. Ti testovi omogućuju puno detaljnije testiranje, a bez koji bi aplikacija bila samo površno testirana. Alati koji nam služe za izvođenje testova mogu se integrirati sa drugim alatima u cilju izgradnje okruženja za sveobuhvatno testiranje. U današnje vrijeme alati se često povezuju s bazama podataka, mjernim alatima, alatima za analizu koda, editorima teksta kako bi se automatiziralo što veći proces testiranja.

CppUnit

CppUnit jer open source framework za testiranje, napisan je za C++ programski jezik. Može se koristi na više platforma. Za druge programske jezike inačice možemo naći u: C# (NUnit), Python (PyUnit), Fortran (fUnit), Java (JUnit).

CppUnit omommgućava kreiranje testova za bilo koju C++ klasu, odnosno za bilo koju njezinu metodu. Svaka C++ klasa može se testirati ali ne moraju sve. Kada se testira klasa, koristi se jedna ili više test procedura. Test procedura je nezavisna od klase, pa se tako ista test procedura može primijeniti na više različitih klasa. Kada se test procedura primjeni na jednu od klasa tada nastaje konkretan test, što znači da je konkretni test u vezi sa samo jednom procedurom i sa samo jednom klasom. CppUnit se može koristiti kroz cijelo vrijeme razvoja software-a, jer se prvo pišu testovi a tek nakon toga metode koje taj test testira. Korištenje CppUnit-a kreira se baza testova koji se izvode detaljno organizirani. Na taj način se stvara sigurnost da sve što je implementirano je i funkcionalno.

Prednosti korištenja CppUnit-a:

- CppUnit je veoma jednostavna pri korištenju i testovi se pišu veoma lako.
- CppUnit čuva bazu testova što je vrlo važno kako bi se kasnije moglo provjeriti da li je program zadržao svoju funkcionalnost kao ranije.
- Pri izvršavanju, testovi napisani u CppUnit-u vrše automatsku provjeru rezultata i prijavljuju greške
- Ukoliko imamo puno testova, oni se mogu organizirati u hijerarhijske cjeline.

Primjeri:

- http://cppunit.sourceforge.net/doc/lastest/cppunit_cookbook.html
- http://www.uow.edu.au/~nabg/222/Lectures/P3UnittestingforC++.pdf

Instalacija:

- Linux: sudo apt-get install libcppunit-dev
- Windows: https://sourceforge.net/projects/cppunit/files/cppunit/1.12.1/

CmakeList.txt i CppUnit

```
set(CMAKE_MODULE_PATH "${CMAKE_SOURCE_DIR}/cmake_modules/")
FIND_PACKAGE(CPPUNIT REQUIRED)
Folder cmake_mod:
SET(CPPUNIT_FOUND "NO")
FIND_PATH(CPPUNIT_INCLUDE_DIR „path“)
# Kod Win32 sustava, debug i release nemaju isti naziv.
IF(WIN32)
  FIND_LIBRARY(CPPUNIT_LIBRARY cppunit
          ${CPPUNIT_INCLUDE_DIR}/../lib
          /usr/local/lib
          /usr/lib)
  FIND_LIBRARY(CPPUNIT_DEBUG_LIBRARY cppunitd
          ${CPPUNIT_INCLUDE_DIR}/../lib
          /usr/local/lib
          /usr/lib)
ELSE(WIN32)
  # Kod Unix sustava, debug i release imaju isti naziv.
  FIND_LIBRARY(CPPUNIT_LIBRARY cppunit
          ${CPPUNIT_INCLUDE_DIR}/../lib
          /usr/local/lib
          /usr/lib)
  FIND_LIBRARY(CPPUNIT_DEBUG_LIBRARY cppunit
          ${CPPUNIT_INCLUDE_DIR}/../lib
          /usr/local/lib
          /usr/lib)
ENDIF(WIN32)

IF(CPPUNIT_INCLUDE_DIR)
  IF(CPPUNIT_LIBRARY)
    SET(CPPUNIT_FOUND "YES")
    SET(CPPUNIT_LIBRARIES ${CPPUNIT_LIBRARY} ${CMAKE_DL_LIBS})
    SET(CPPUNIT_DEBUG_LIBRARIES ${CPPUNIT_DEBUG_LIBRARY} ${CMAKE_DL_LIBS})
  ELSE (CPPUNIT_LIBRARY)
```

```
   IF (CPPUNIT_FIND_REQUIRED)
     MESSAGE(SEND_ERROR "Nije moguće pronaći knjižnicu CppUnit.")
   ENDIF (CPPUNIT_FIND_REQUIRED)
  ENDIF(CPPUNIT_LIBRARY)
ELSE(CPPUNIT_INCLUDE_DIR)
  IF (CPPUNIT_FIND_REQUIRED)
    MESSAGE(SEND_ERROR " Nije moguće pronaći knjižnicu CppUnit“)
  ENDIF(CPPUNIT_FIND_REQUIRED)
   ENDIF(CPPUNIT_INCLUDE_DIR)
```

Benchmarking

U računarstvu, benchmarking predstavlja izvršavanje programa, grupe programa ili drugih operacija u cilju procjene relativne performanse objekta. Termin benchmarking je često povezan sa procjenom performansi i karakteristika hardware-a npr. Performansi procesora pri izvršavanju određenih operacija. Postoje i software-ski benchmarking kojim se testiraju kompajleri ili sustavi baza podataka.

Tipovi benchmarkinga:

1. Pravi programi
   - softver za obradu teksta
   - CAD programi
   - korisnički softver (i.e.: MIS)
2. Kernel
   - sadrži ključne kodove
   - uglavnom izdvojen od glavnog programa
   - popularni kernel: Livermore petlja
   - linepack benchmark (sadrži osnovnu linearnu algebarsku podrutinu pisanu u FORTRAN jeziku)
   - rezultati su prikazani u broju MFLOPS
3. Mikro-benchmark
   - programi dizajnirani da mjere performanse osnovnih računarskih komponenti
   - automatska detekcija parametara računarskog hardware-a kao što su broj registara, veličina cachea, memorijsko kašnjenje
4. Sintetički benchmark
   - Procedura programiranja sintetičkog benchmarka:
     - sakupiti statistiku svih tipova operacija iz puno aplikacijskog software-a
     - uzeti proporciju svake operacije
     - napisati program na osnovu gornje proporcije
   - Tipovi sintetičkih benchmarka su:
     - Whetstone
     - Dhrystone
   - To su bili prvi općenamjenski industrijski benchmarkovi. Oni neće nužno dobiti visok rezultat na modernim računarima sa protočnom obradom.
5. Ulazno/Izlazni benchmark-ovi
6. Benchmark-ovi baza podataka: za mjerenje propusnosti i brzine odaziva sustava za upravljanje bazama (DBMS)
7. Paralelni benchmark-ovi: koriste se na procesorima sa više jezgara

Mjerenje na različitim platformama

Kada govorimo o benchmark-ingu onda nas interesira stvarno vrijeme koje je prošlo u izvršavanju neke operacije. Da bi smo to mogli izmjeriti, moramo mjeriti vrijeme najmanje u mikrosekundama. Svaki operacijski sustav ima jedan ili više načina na koji mogu vratiti vrijednost stvarnog vremena. Neki su brži od ostalih i imaju bolju rezoluciju od ostalih.

OS	clock_gettime	gethrtime	GetSystemTimeAsFileTime and Get SystemTimePreciseAsFileTime	gettimeofday	mach_absolute_time	time
AIX	yes			yes		yes
BSD	yes			yes		yes
HP-UX	yes	yes		yes		yes
Linux	yes			yes		yes
OSX				yes	yes	yes
Solaris	yes	yes		yes		yes
Windows			yes			

GetSystemTimeAsFileTime() i GetsystemTimePreciseAsFileTime()

Na Windows i Cygwin (Linux kompaktibilni alat za Windows) GetSystemTimeAsFileTime() i GetsystemTimePreciseAsFileTime() obje funkcije koriste FILETIME strukturu. Funkcije su slične ali GetSystemTimePreciseAsFileTime() pruža veću preciznost i ima podršku samo za Windows 8 i Windows Server 2012.

Izgled strukture:

```
typedef struct _FILETIME
{  DWORD dwLowDateTime;
  DWORD dwHighDateTime;
} FILETIME, *PFILETIME;
```

Struktura pruža 32-bits i 64-bit vrijeme u jedinicama od 100 nanosekundi.

Dostupnost funkcije GetSystemTimeAsFileTime( ): Cygwin i Windows 2000 + kasnije verzije.

Upotreba u kodu:

```
#include <Windows.h>
        FILETIME tm;
        ULONGLONG t;
#if defined(NTDDI_WIN8) && NTDDI_VERSION >= NTDDI_WIN8
        /* Windows 8, Windows Server 2012 + kasnije verzije ---------------- */
        GetSystemTimePreciseAsFileTime( &tm );

#else
        /* Windows 2000 +kasnije verzije ---------------------------------- */
        GetSystemTimeAsFileTime( &tm );
#endif
        t = ((ULONGLONG)tm.dwHighDateTime << 32) | (ULONGLONG)tm.dwLowDateTime;
        return (double)t / 10000000.0
```

clock_gettime()

clock_gettime() pruža najpreciznije sistemsko vrijeme. Funkcija kao prvi argument stavlja „clock id“ a drugi je timspec struktura koja sadrži vrijeme u sekundama i nanosekundama. Za skoro sve operacijske sustave mora biti povezana s librt(Libary Interfaces and Headers).

Funkcija je opcionalni dio POSIX specifikacija i dostupna je samo ako je _POSIX_TIMERS definiran u <unistd.h> sa vrijednošći većom od 0. Za sada AIX, BSD, HP-UX, Linux i Solaris podržavaju tu funkciju za razliku od OSX koji ne podržava.

Clock_getres() funkcija vraća nam rezoluciju sata, ali ta funkcija nije uvijek implementirana. Za sada radi na AIX, BSD, HP-UX i Linuxu ali ne i u Solaris-u.

POSIX specifikacije definiraju nazive za nekoliko standardnih „clock id“ vrijednosti, uključujući CLOCK_MONOTONIC i CLOCK_REALTIME za dobivanje sistemskog vremena. HP_UX izostavlja CLOCK_MONOTONIC dok ga Solaris definira kao „include“ datoteku. Solaris dodaju svoju nestandardni CLOCK_HIGHRES clock, Linux dodaje CLOCK_MONOTONIC_RAW i BSA dodaje CLOCK_MONOTONIC_PRECISE.

Dostupni „clocks“ na operacijskim sustavima:

OS	Available clocks
AIX	CLOCK_MONOTONIC, CLOCK_REALTIME
BSD	CLOCK_MONOTONIC, CLOCK_MONOTONIC_PRECISE, CLOCK_REALTIME
HP-UX	CLOCK_REALTIME
Linux	CLOCK_MONOTONIC, CLOCK_MONOTONIC_RAW, CLOCK_REALTIME
Solaris	CLOCK_MONOTONIC, CLOCK_REALTIME, CLOCK_HIGHRES

CLOCK_REALTIME (AIX, BSD, GP-UX, Linux i Solaris) je sat sustava koji odbrojava sukladno stvarnom protoku vremena (može se mijenjati posebnim sučeljem)

CLOCK_MONOTONIC (AIX, BSD, Linux i Solaris) je sat sustava koji odbrojava sukladno stvarnom protoku vremena ali se ne može mijenjati

CLOCK_MONOTONIC_PRECISE (BSD) je varijanta CLOCK_MONOTONIC-a koja nam pruža preciznije mjerenje vremena.

CLOCK_MONOTONIC_RAW (Linux2.6.28+ kasnije verzije) j je
varijanta CLOCK_MONOTONIC-a koja nam pruža preciznije mjerenje vremena u suradnji s hardware-om.

CLOCK_HIGHRES (Solaris) je CLOCK_MONOTONIC verzija za Solaris koji pruža vrlo precizno vrijeme.

Upotreba u kodu:

```
#include <time.h>
#if defined(_POSIX_TIMERS) && (_POSIX_TIMERS > 0)
        struct timespec ts;
#if defined(CLOCK_MONOTONIC_PRECISE)
        /* BSD. --------------------------------------------- */
        const clockid_t id = CLOCK_MONOTONIC_PRECISE;
#elif defined(CLOCK_MONOTONIC_RAW)
        /* Linux. ------------------------------------------- */
        const clockid_t id = CLOCK_MONOTONIC_RAW;
#elif defined(CLOCK_HIGHRES)
        /* Solaris. ----------------------------------------- */
        const clockid_t id = CLOCK_HIGHRES;
#elif defined(CLOCK_MONOTONIC)
        /* AIX, BSD, Linux, POSIX, Solaris. ----------------- */
        const clockid_t id = CLOCK_MONOTONIC;
#elif defined(CLOCK_REALTIME)
        /* AIX, BSD, HP-UX, Linux, POSIX. ------------------- */
        const clockid_t id = CLOCK_REALTIME;
#else
        const clockid_t id = (clockid_t)-1;      /* Unknown. */
#endif
        if ( id != (clockid_t)-1 && clock_gettime( id, &ts ) != -1 )
                return (double)ts.tv_sec +
                        (double)ts.tv_nsec / 1000000000.0;
#endif;
```

gettimeofday()

U POSIX operacijskim sustavima, gettimeofday() vraća trenutno vrijeme sustava u timeval strukturi s vremenom u sekundama i mikrosekundama.

```
struct timeval
{       time_t      tv_sec;
        suseconds_t tv_usec;
};
```

Vraća vrijeme polje sa vremenom u sekundama od Unix Epoch i mikrosekundama, minute zapadno od Greenwatcha ili ako stavimo $return_float true onda vrati float. Prvi argument u funkciji je pokazivač na timeval strukturu, drugi argument je opcionalan za timezone strukturu (upišemo li 0 preskočiti će se timezone informacije) koja nam nije potrebna pri benchmark-ingu.

Dostupnost funkcije gettimeofday( )**:** AIX, BSD, Cygwin, HP-UX, Linux, OSX, iSolaris.

Upotreba u kodu:

```
#include <sys/time.h>
        struct timeval tm;
        gettimeofday( &tm, NULL );
        return (double)tm.tv_sec + (double)tm.tv_usec / 1000000.0;
```

mach_absolute_time ()

Na OSX , mach_absolute_time () vraća vrijeme koje je mjereno i spremljeno u vremenskoj bazi mach_timebase_info ( ). Kasnije funkcija ispunjava mach_timebase_info strukturu s brojnikom i nazivnikom.

```
struct mach_timebase_info
{
        uint32_t numer;
        uint32_t denom;
};
```

Kako bi se dobiveno vrijeme moglo pretvoriti u sekunde, dobiveno vrijeme treba pomnožiti s brojnikom, podijelit s nazivnikom tada podijeliti s 1,000,000,000.0 kako bi iz nanosekunda dobili sekunde. Na OSX, match_absolute_time() je preciznije nego li funkcija gettimeofday() te je kao takva i preporučena od strane Apple-a. Vrijeme je monotonično i može se prebaciti u stvarno vrijeme.

Dostupnost funkcije mach_absolute_time( ): OSX

Upotreba u kodu:

```
#include <mach/mach.h>
#include <mach/mach_time.h>

        static double timeConvert = 0.0;
        if ( timeConvert == 0.0 )
        {
                mach_timebase_info_data_t timeBase;
                (void)mach_timebase_info( &timeBase );
                timeConvert = (double)timeBase.numer /
                        (double)timeBase.denom /
                        1000000000.0;
        }
        return (double)mach_absolute_time( ) * timeConvert;
```

time()

U POSIX operacijskom sustavu, time() vraća trenutno sistemsko vrijeme u sekundama. Funkcija je vrlo slična funkcijama gettimeofday() i CLOCK_REALTIME satu za clock_gettime(). Sistemsko vrijeme može skakati naprijed ili nazad ukoliko to administrator želi.

Dostupnost funkcije time(): AIX, BSD, Cygwin, HP-UX, Linux, OSX, and Solaris.

Upotreba u kodu:

```
#include <time.h>

        return (double)time( NULL );
```

Literatura

Linkovi:

- https://en.wikipedia.org/wiki/Cross-platform
- http://www.cs.colostate.edu/saxs/researchexam/CrossPlatformDevelopment.pdf
- https://www.backblaze.com/blog/10-rules-for-how-to-write-cross-platform-code/
- https://msdn.microsoft.com/en-us/library/dn771552.aspx
- https://www.qt.io/
- https://en.wikipedia.org/wiki/Qt_(software)
- https://cmake.org/
- https://en.wikipedia.org/wiki/CMake
- https://www.cs.swarthmore.edu/~adanner/tips/cmake.php
- https://blog.kloud.com.au/2014/05/02/cross-platform-testing-myths-vs-mysteries/
- http://www.drdobbs.com/open-source/testing-linux-code/231903171?pgno=2
- http://www.drdobbs.com/cpp/microbenchmarking-c-c-and-java/184401976?pgno=2
- https://bs.wikipedia.org/wiki/Benchmark
- https://en.wikipedia.org/wiki/Unit_testing
- http://nadeausoftware.com/articles/2012/04/c_c_tip_how_measure_elapsed_real_time_benchmarking
- https://developer.android.com/training/testing/unit-testing/local-unit-tests.html
- http://codeutopia.net/blog/2015/04/11/what-are-unit-testing-integration-testing-and-functional-testing/
- https://www.fer.unizg.hr/_download/repository/ILJ-skripta-testiranje%5B2%5D.pdf
- http://cppunit.sourceforge.net/doc/lastest/cppunit_cookbook.html
- http://www.uow.edu.au/~nabg/222/Lectures/P3UnittestingforC++.pdf

Dodatno

Dodatno sam uz postojeći seminar napravio program za mjerenje vremena (benchmarking). U njemu se mogu mjeriti vremena za zapisivanja na disk i čitanje s diska(sekvencijonalno i random), zatim vremena korištenje cpu-a (iops flops) kroz petlje i na kraju možemo mjeriti vrijeme izvršavanja jedne jednostavne funkcije.
Postoje dvije verzije programa bach_test i bentch_test+getRealTime_function. Prva verzija koristi ugrađenu (C/C++) funkciju clock() dok druga ima funkciju getRealTime() koja na temelju verzije platforme odabire najbolji sistemski sat. Postupak izgradnje getRealTime funkcije opisao sam u seminaru pod naslovom „Mjerenje na različitim platformama“. Kod sam razvijao u Visual studio ali sam ga isto tako testirao u Linuxu (Kdevelop).

Github link:
https://github.com/mpiskac/bench_test_cross_platform/