PT&SCHE

Introduction of part-time and short cycle studies in Serbia

Funded by the Erasmus+ Programme of the European Union

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

UNIVERZITET
METROPOLITAN
BEOGRAD

# KI301 - KONSTRUISANJE SOFTVERA

## Integracija softvera

Lekcija 10

PRIRUČNIK ZA STUDENTE

# KI301 - KONSTRUISANJE SOFTVERA

## Lekcija 10

## *INTEGRACIJA SOFTVERA*

- Integracija softvera
- Poglavlje 1: Integracija softvera-Uvod
- Poglavlje 2: Strategije inkrementalne integracije
- Poglavlje 3: Procedure integracionog testiranja
- Poglavlje 4: Alati za integraciono testiranje
- Poglavlje 5: Integraciono testiranje - Vežbe
- Zaključak

# Uvod

## L10:INTEGRACIJA SOFTVERA

Sadržaj lekcije:

- Integracija softvera-Uvod
- Strategije inkrementalne integracije
- Integraciono testiranje
- Alati za integraciono testiranje
- Poređenje metoda testiranja
- Integraciono testiranje-vežba

# Poglavlje 1

# Integracija softvera-Uvod

## INTEGRACIJA SOFTVERA

*Može doći do kolapsa u toku konstruisanja softvera, naime broj defekata može da izgleda nesavladljiv, napredak u integraciji može da bude nevidljiv.*

Integracija softvera je integracija softverskih jedinica/komponenti u jednu funkcionalnu tj operativnu celinu. Ako se izkonstruiše i integriše softver po pogrešnom redosledu, to ima negativne posledice: otežava kodiranje, otežava testiranje, otežava prepravke ( tj tzv debugging)

Čak može doći do kolapsa u toku konstruisanja softvera, naime broj defekata može da izgleda nesavladljiv, napredak u integraciji može da bude nevidljiv. Pošto se integracija radi posle razvojnog testiranjai zajedno sa testiranjem sistema, integracija se ponekad tretira kao deo testiranja softvera pa se naziva „integracionim testiranjem“. Medjutim, pošto je zadatak integracije kompleksan i obiman, treba je posmatrati kao jednu posebnu i nezavisnu aktivnost.

Sledeće koristi se mogu dobiti od pažljive integacije:

- lakšu dijagnozu defekata, manje defekata , i manje "skela"(less scaffolding),
- manje vremena do prvog operativnog proizvoda
- kraće ukupno vreme razvoja proizvoda
- bolje odnose sa korisnicima
- veće šanse da se projekat uradi uspešno
- pouzdanije procene rokova, bolji kvalitet programa

## FAZNA INTEGRACIJA :

*Problem sa faznom integracijom je da kad se klase integrišu da funkcionišu zajedno po prvi put, novi problemi se pojavljuju, i oni mogu biti bilo gde.*

Do nedavno, fazna integracija je bila obavezna. Sastoji se od sledećih ustanovljenih faza:

- dizajnirati, kodirati, TESTIRATI, dedefektovati (debugging) svaku klasu, što se inače zove razvoj jedinica softvera tj unit development
- kombinovati klase u jedan sistem, što se zove sistemska integracija

- testirati i dedefektovati ceo sistem

Medjutim , problem sa faznom integracijom je da kad se klase integrišu da funkcionišu zajedno po prvi put, novi problemi se pojavljuju, i oni mogu biti bilo gde. Pošto se tipično radi o velikom broju klasa koje nikad nisu radile zajedno pre toga, uzroci mogu biti različiti, npr.

- *Nedovoljno testirana klasa, ili greška u interfejsu izmedju dve klase, ili greška usled interakcije dve klase. Medjutim, sve klase su sumnljive.*

Vrlo je neizvesno gde je lokacija pojedinih problema, i to se kombinuje sa problemom da su svi problemi nastali odjednom kod takve integracije. Takodje , problemi interaktuju medjusobom. Zbog svega ovoga ovakva integracija se zove ‘ eksplozivna integracija’. Problemi mogu biti :Slabo sakrivanje podataka, Globalne varijable, Nedokumentovani interfejsi, Pretpostavke o operisanju greškama tj zaštita od grešaka

# INKREMENTALNA INTEGRACIJA :

*Kod inkrementalne integracije razvije se jedan manji deo sistema koji može da obavlja neku funkciju, npr najmanji funkcionalni deo, i zatim se taj deo testira i dedefektuje.*

Inkrementalna integracija: Kod inkrementalne tj postepene integracije, ne kombinuju se svi delovi programa odjednom, već se ide postepeno, i to na sledeći način:

- Razvije se jedan manji deo sistema koji može da obavlja neku funkciju, npr najmanji funkcionalnideo, najteži deo, ključni deo, ili kombinacija ovih, i zatim se taj deo testira i dedefektuje (tj vrše popravke), i ovo će poslužiti kao jezgro tj kostur na koji se onda kače ostali delovi
- Dizajnirati, kodirati, testirati i debug-ovati novu klasu koja ne pripada prethodnom jezgru
- Integrisati novu klasu sa prethodnim jezgrom, i testirati i debug-ovati kombinaciju jezgra i nove klase, i osigurati da ova kombinacija funkcioniše pre nego se dodaju nove klase

Ponekad, može da se umesto jedne klase doda nekoliko klasa odjednom, ako te klase čine nekakvu celinu, onda se obavlja prvo mini integracija, gde se tih nekoliko klasa integriše, a onda se obavlja integracija sa jezgrom

# PREDNOSTI INKREMENTALNE INTEGRACIJE

*Lakša detekcija greški, jer greška je obično ili u novoj komponenti (interakciji ove komponente sa ostatkom sistema), ili je greška u vezi tj interfejsu izmedju nove komponente i ostatka progra*

Inkrementalni prilaz omogućuje niz prednosti, u poredjenju sa tradicionalnim tj faznim prilazom integraciji softverskih jedinica u jedinstvenu operativnu celinu:

Lakša detekcija greški, jer greška je obično ili

- (1) u novoj komponenti ( tj. u interakciji ove komponente sa ostatkom sistema),
- ili (2) je greška u vezi tj interfejsu izmedju nove komponente i ostatka programa tj sistema,
- pa se zna gde tražiti grešku,
- dok kod fazne integracije je moguća greška u bilo kojoj konekciji ili interakciji dve komponente i to odjednom, kao što je ilustrovano na slici 2.
- Rano se oseti uspeh u toku projekta, i moral programera je bolji, jer ne postoji i ne traje dugo neizvesnost oko integracije

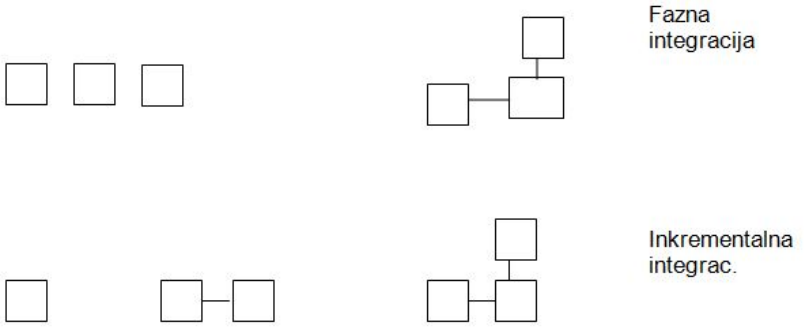
Kada se integrisanje vrši postepeno, tj često, tj frekventno, ima se bolja informacija o stanju završenosti projekta, naime bolje je saznati da 50% sistema funkcioniše nego da je 90% kodiranja gotovo

Odnos sa kupcem je bolji, jer i kupcima se svidja opipljiv progres na projektu, i oni takodje onda imaju bolji moral

Omogućuje brži završetak projekta, jer se ranije kreće sa problemima integracije

# FAZNA I INKREMENTALNA INTEGRACIJA

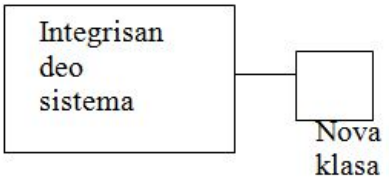
*Slika prikazuje faznu i inkrementalnu integraciju.*



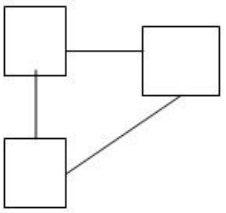

Kod inkrementalne tj postepene integracije, ne kombinuju se svi delovi programa odjednom, već se ide postepeno, i to na sledeći način:
Razvije se jedan manji deo sistema koji može da obavlja neku funkciju, npr najmanji funkcionalnideo, najteži deo, ključni deo, ili kombinacija ovih, i zatim se taj deo testira i dedefektuje (tj vrše popravke), i ovo će poslužiti kao jezgro tj kostur na koji se onda kače ostali delovi
Dizajnirati, kodirati, testirati i debug-ovati novu klasu koja ne pripada prethodnom jezgru
Integrisati novu klasu sa prethodnim jezgrom, i testirati i debug-ovati kombinaciju jezgra i nove klase, i osigurati da ova kombinacija funkcioniše pre nego se dodaju nove klase

# POREDJENJE FAZNE I INKREMENTALNE INTEGRACIJE

*Slika ilustruje poredjenje fazne i inkrementalne integracije.*




Inkrementalna integracija, korak po korak



Fazna integracija, odjednom

Fazna integracija ne može da počne do kasnije faze projekta, pošto su tek sve klase razvojno istestirane. Medjutim kad se klase konačno spoje integracijom i greške se pokažu, to izaziva paniku kod programera, i panično dedefektovanje , umesto metodične detekcije i popravke greške. Za programe sa 2 ili 3 klase, fazna integracija je prihvatljiva. U ostalim slučajevima ne.

# VIDEO VEŽBA

## *Video vežba - Šta je to integraciono testiranje (What is Integration Testing? Software Testing Tutorial)*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

# Poglavlje 2

# Strategije inkrementalne integracije

## STRATEGIJE INKREMENTALNE INTEGRACIJE

*Kod inkrementalne integracije potrebno je odrediti redosled i napraviti vremenski plan integracije, i ovo ima uticaja na konstruisanje celog programa.*

Kod fazne integracije, sve komponente se integrišu u isto vreme, i onda nije potrebno praviti redosled i plan integacije. Već, samo definisati taj kritičan dan, kad se planira ta 'eksplozija' integracije. Medjutim, kod inkrementalne integracije potrebno je odrediti redosled i napraviti vremenski plan integracije, i ovo ima uticaja na konstruisanje celog programa, pa treba izvršiti pažljivo planiranje. Ima nekoliko strategija integracije,

npr. integracija odozgo-nadole,top-down integration,

integracija odozdo-nagore, bottom-up integration,

sendvič-integacija, sandwitch-integation,

'antirizična' integracija, risk-oriented integration,

'funkcionalna' integracija, feature-oriented,

T-integracija, T-shaped integration

Izbor strategije integracije tj kombinovanje strategija integracije u cilju definisanja integracije za odredjeni softver je heuristička**,** i različito za svaki projekt. Koja god da se izabere strategija integracije softvera, preporučuje se svakodnevno konstruisanje, kompilacija (i linkovanje), i kombinovanje u jedan program koji se može egzekutovati, i ovaj se program onda pušta kroz 'dimni test', da se proveri da li softver 'gori' kod izvršavanja.

Kod integracije odozgo-nadole, klasa na vrhu hijerarhije se prvo kodira, i integracija počinje od nje. Na vrhu je klasa, koja obuhvata glavni program, naime main() u Javi ili C++. Zatim se klase integrišu odozgo na dole silazeći sa jednog nivoa na sledeći donji nivo, kako je ilustrovano na donjem dijagramu. Klase na vrhu se dodaju na početku integracije, dok se klase na dnu dodaju na kraju integracije.

# INTEGRACIJA ODOZGO-NADOLE:

*Konačno , ne može se primeniti integracija odozgo-nadole ako ne postoji klasa na vrhu. U mnogim sistemima, ne postoji vrh, tj struktura softvera nije kao na slici.*

Kod integracije odozgo-nadole, interfejsi izmedju klasa moraju biti korektni. Većina greški nisu greške koje su unutar klase već one koje su usled pogrešne interakcije klasa. Ako su interfejsi tačno specificirani, mnogo je manje problema. Specifikacija interfejsa je uradjena ranije, kod dizajna i kodiranja, ali provera interfejsa kod integracije je veoma korisna.

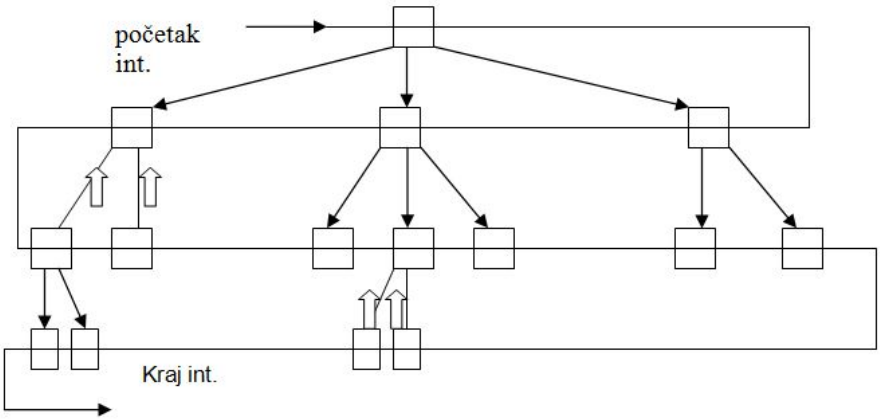
Prednost integracije odozgo-nadole, je u tome što se prvo ispituju najvažnije i najveće klase, a to su klase na vrhu hijerarhije, tako da vrlo rano, na početku projekta, se integrišu oni najvažniji i najkompleksniji delovi, tj kontrolna logika sistema. Druga prednost je da se omogućuje kompletiranje jednog parcijalno funkcionišućeg sistema rano u toku projekta. Npr. ako su korisnički inbterfejsi na vrhu sistema, može se bazni interfejs osposobiti brzo, a detalji ostaviti za kasnije. Takodje , prednost kod integracije odozgo-nadole, je da se kodiranje i integracija može započeti rano u projektu, pre završenog detaljnog dizajna, što ubrzava projekt, i skraćuje rokove. Medjutim , integracija odozgo-nadole ima i nedostataka. Na donjim nivoima softvera ima puno interfejsa. I oni ostaju na kraju integracije odozgo-nadole. Takodje nije neobično da neki problem sa donjeg nivoa utiče na sam vrh softvera, izazivajući promene na vrhu i poništavajući prednosti prethodnog integracionog truda.

Dalje , s obzirom da se ide horizontalno po pojedinim nivoima, ne prate se vertikale, svaka ispravka koja se desi izaziva promene vertikalno, i to se ostavlja za kasnije, pa čitava serija restlova se pjavljuje koja ostaje da se integriše kad se dodje na niži nivo integracije. U stvari , u praksi, rigidnu tj čistu integraciju odozgo-nadole je vrlo teško primeniti, skoro nemoguće. Većina programera koriste hibridne prilaze integraciji, npr odozgo-nadole po sektorima. Čista integracija odozgo-nadole nije preporučljiva.

Konačno , ne može se primeniti integracija odozgo-nadole ako ne postoji klasa na vrhu. U mnogim sistemima, ne postoji vrh, tj struktura softvera nije kao na slici. U mnogim sistemima, korisnički interfejs je na vrhu, ili je program main() na vrhu. Pogodna alternativa integraciji odozgo-nadole je integracija parcijalno odozgo-nadole, gde se primenjuje integracija odozgo-nadole po sektorima, kao dato na slici.

# INTEGRACIJA ODOZGO-NADOLE (PUTANJA INTEGRACIJE)

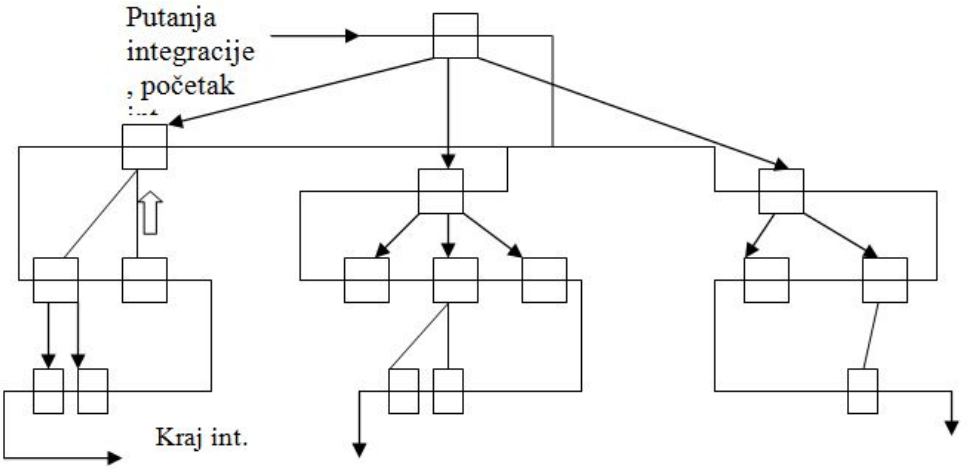
*Slika ilustruje integraciju odozgo-nadole (putanja integracije).*




Kod integracije odozgo-nadole, interfejsi izmedju klasa moraju biti korektni. Većina greški nisu greške koje su unutar klase već one koje su usled pogrešne interakcije klasa. Ako su interfejsi tačno specificirani, mnogo je manje problema. Specifikacija interfejsa je uradjena ranije, kod dizajna i kodiranja, ali provera interfejsa kod integracije je veoma korisna.

# INTEGRACIJA PARCIJALNO ODOZGO-NADOLE

*Slika demonstrira integraciju parcijalno odozgo-nadole.*




Konačno , ne može se primeniti integracija odozgo-nadole ako ne postoji klasa na vrhu. U mnogim sistemima, ne postoji vrh, tj struktura softvera nije kao na slici. U mnogim sistemima, korisnički interfejs je na vrhu, ili je program main() na vrhu. Pogodna alternativa integraciji odozgo-nadole je integracija parcijalno odozgo-nadole, gde se primenjuje integracija odozgo-nadole po sektorima, kao dato na slici.

# INTEGRACIJA ODOZDO-NAGORE

*Glavni problem integracije odozdo-nagore je da se glavne tj najvažnije klase i njihovi interfejsi, tj one klase i interfejsi na vrhu hijerarhije, ostavljaju za kraj integracije.*

:

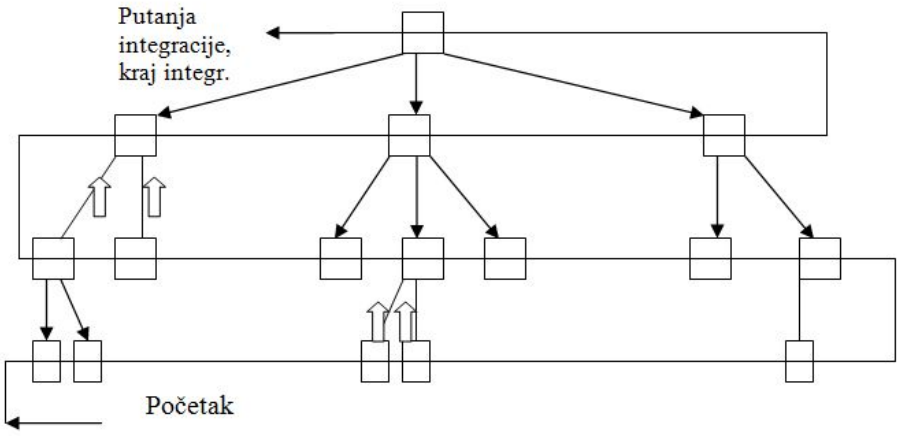
Kod integracije odozdo-nagore, prvo se kodiraju i integrišu klase na dnu, a one klae na vrhu se integrišu poslednje, kao što je prikazano na slici. Dodavanje klasa na niskom nivou, jedna po jedna, idući horizontalno dok se ne završi jedan nivo, čini ovu strategiju inkrementalnom, i pogodnijom u odnosu na faznu integraciju.

Kod testiranja ovih klasa na donjem nivou, potrebno je napisati test-zamene za klase sa gornjeg nivoa. Kako integacija napreduje, ove test-zamene za klase na višim nivoima se zamenljuju pravim klasama. Kod integracije odozdo-nagore, greške se lako otkrivaju, jer su ograničene na jednu klasu koja se dodaje. Zatim , integracija počinje rano u toku projekta. Takodje interfejsi na novou sistema se pojave na početku integracije, što je korisno.

Glavni problem integracije odozdo-nagore je da se glavne tj najvažnije klase i njihovi interfejsi, tj one klase i interfejsi na vrhu hijerarhije, ostavljaju za kraj integracije. Od nekih projekata to može biti veliki problem, ako npr dizajn na visokom nivou nije jasan do kraja. Kod integracije odozdo-nagore neophodno je završiti dizajn softvera celog sistema pre početka integracije. Nije poželjno početi kodiranje na nižim nivoima softvera a da pre toga se ne završi dizajn na visokom nivou softvera. Kao i kod integracije odozgo-nadole, čista tj rigidna integracija odozdo-nagore se retko koristi, već se više koristi hibridni pristup, npr parcijalna integracija odozdo-nagore, kao na slici.

# INTEGRACIJA ODOZDO-NAGORE

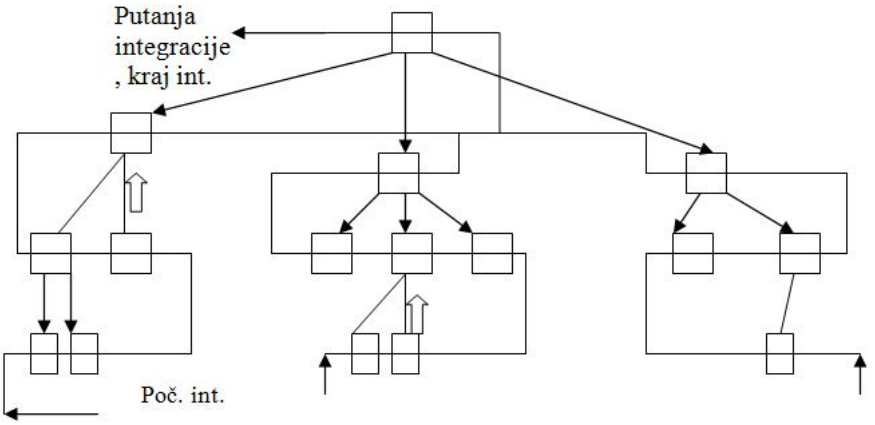
*Na slici je prikazana integracija odozdo-nagore.*



Kod integracije odozdo-nagore, prvo se kodiraju i integrišu klase na dnu, a one klae na vrhu se integrišu poslednje, kao što je prikazano na slici. Dodavanje klasa na niskom nivou, jedna po jedna, idući horizontalno dok se ne završi jedan nivo, čini ovu strategiju inkrementalnom, i pogodnijom u odnosu na faznu integraciju.

# INTEGRACIJA PARCIJALNO ODOZDO-NAGORE

*Slika prikazuje integraciju parcijalno odozdo-nagore.*




Glavni problem integracije odozdo-nagore je da se glavne tj najvažnije klase i njihovi interfejsi, tj one klase i interfejsi na vrhu hijerarhije, ostavljaju za kraj integracije. Od nekih projekata to može biti veliki problem, ako npr dizajn na visokom nivou nije jasan do kraja. Kod integracije odozdo-nagore neophodno je završiti dizajn softvera celog sistema pre početka integracije. Nije poželjno početi kodiranje na nižim nivoima softvera a da pre toga se ne završi dizajn na visokom nivou softvera. Kao i kod integracije odozgo-nadole, čista tj rigidna integracija odozdo-nagore se retko koristi, već se više koristi hibridni pristup, npr parcijalna integracija odozdo-nagore, kao na slici.

# LINK - PRIMER

*Pogledati primer na donjem linku.*

Pogledati linkove:

- http://www.softwaretestinghelp.com/what-is-integration-testing/

# ZADACI ZA VEŽBU

*Zadaci*

- Za izabranu aplikaciju napraviti plan inkrementalne integracije odozgo-nadole?
- Za izabranu aplikaciju napraviti plan inkrementalne integracije odozdo-nagore?

# Poglavlje 3

# Procedure integracionog testiranja

## ŠTA JE TO INTEGRACIONO TESTIRANJE?

*Integraciono testiranje uključuje pisanje "code snippets" za potrebe testiranja integhrisanih modula, pa je to u stvari white box testing. Ali, može se koristiti i black-box testiranje.*

Evo pokaznog primera iz integracionog testiranja koji se nalazi na :

http://www.softwaretestinghelp.com/what-is-integration-testing/

Ako se pogledaju različiti modeli životnog ciklusa, Software developments lifecycle models-SDLC modeli, svi SDLC modeli imaju integraciono testiranje kao jedan sloj testiranja. Integraciono testiranje je može se reći jedan nivo testiranja pre nego tip testiranja, jer se primenjuju iste metode kao kod jediničnog testiranja, pa čak i isti alati.

Integraciono testiranje uključuje pisanje malih parčića izvornog koda (writing code snippets) za potrebe testiranja integhrisanih modula, pa je to u stvari white box testing technique. Ali, koncept integracionog testiranja može da uključi takođe i blackbox technique.

Kod testiranja neke aplikacije koristeći black-box testing technique, imamo kombinovanje niza modula u jednu celinu, gde su ovi moduli u čvrstoj međusobnoj vezi. Tehnika integracionog testiranja, Integration testing technique, može da se primeni na ovakve scenarije kombinovanja modula kao crne kutije.

## KAKO SE OBAVLJA INTEGRACIONO TESTIRANJE?

*The main function or goal of Integration testing is to test the interfaces between the units/modules.*

Uobičajeno je da se integraciono testiranje radi posle jediničnog testiranja ( “Unit testing” ). Pošto su sve individualne jedinice napravljene i testirane, počinjemo sa kombinovanjem ovih jedinično-iztestiranih modula i radimo integraciono testiranje.

Postupak ( ideja) integracionog testiranja je jednostavan:

- dakle, treba integrisati tj. kombinovati jedinično-iztestirane module, jedan po jedan, u složenije celine
- i zatim testirati svaki ovaj složeni modul kao jednu jedinicu.
- pri tome je glavni cilj da se testiraju interfejsi između jedinica/modula.

Dakle, glavni cilj integracionog testiranja ( Integration testing) je testiranje interfejsa između pojedinih jedinica tj. modula (units tj. modules). Pri tome:

- individualni moduli se prvo testiraju u izolaciji od ostalih modula i to je tzv. jedinično testiranje
- posle jedinićnog testiranja pojedinih modula, ovi moduli se integrišu **jedan po jedan**, sve dok se ne integrišu svi moduli, i pri tome se testira kombinovano ponašanje modula
- cilj je da se izvrši validacija da li su zahtevi implementirani korektno
- integraciono testiranje se obavlja u toku razvoja softvera a ne na kraju razvoja softvera

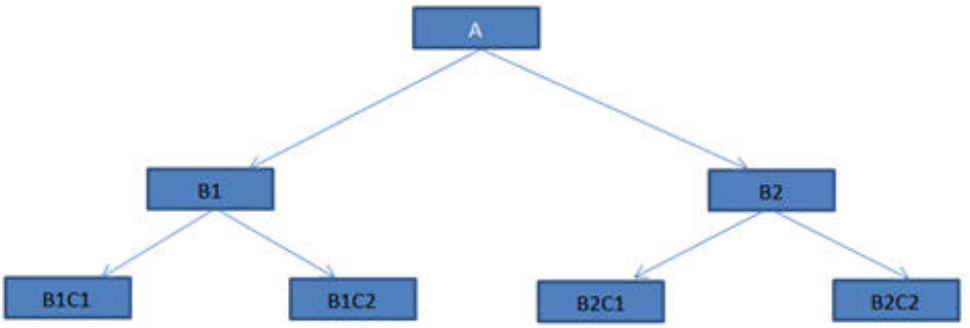
# PRISTUPI INTEGRACIONOM TESTIRANJU

*Postoje dva osnovna prilaza da bi se izvršilo integraciono testiranje: odozgo-nadole i odozdo-nagore.*

Postoje dva osnovna prilaza da bi se izvršilo integraciono testiranje:

- Bottom up approach tj. odozdo-nagore
- Top down approach. tj. odozgo-nadole

Dole je dat primer jednog sistema koji ima 7 jedinica.




Slika 3.1

# PRILAZ ODOZDO-NAGORE

*The Integration testing starts from the lowest module and gradually progresses towards the upper modules of the application.*

Kod prilaza odozdo-nagore, počinje se sa integracijom od najniže jedinice aplikacije, i postepeno se, jedan po jedan modul, idemo na gore. Dakle, integraciono testiranje započinje sa najnižim modulom, i pomeramo se postepeno ka gornjim nivoima tj gornjim modulima aplikacije. Ovaj postupak se nastavlja korak po korak, ssve dok se cela aplikacija ne integriše u jednu celinu. Na kraju se cela aplikacija testira kao jedna jedinica.

Npr. na prethodnoj slici moduli B1C1, B1C2 & B2C1, B2C2 su najniži moduli koji su jedinično iztestirani. Ali, moduli B1 & B2 nisu još razvijeni. Pri tome, funkija modula B1 i B2 je da poziva module B1C1, B1C2 & B2C1, B2C2. Pošto B1 i B2 još nisu razvijeni, potreban nam je neki

program koji će da simulira pozivanje modula B1C1, B1C2 & B2C1, B2C2. Ovi simulacioni programi se zovu "drajveri", DRIVERS.

"Drajveri" su lažni programi ( dummy programs ) koji se koriste da pozivaju niže module, koji zamenjuju gornje module koji ne postoje.

Kod tehnike odozdo-nagore, potrebno je modul-drajver da preda test-slučajeve kao ulazni podatak interfejsu modula koji se testira.

Prednost ovog prilaza odozdo-nagore je da lakše i brže otkriva ako postoje greške u donjim modulima, i lakše ih koriguje. Ali, nedostatak je da glavni program u stvari ne postoji sve dok poslednji moduli na vrhu se ne integrišu i ne iztestiraju, pa greške u dizajnu na višem nivou biće kasnije detektovane.

# TEHNIKA ODOZGO-NADOLE

*This technique starts from the top most module and gradually progress towards the lower modules.*

Kod tehnike odozgo-nadole, počinje se odozgo sa najvišeg modula, pa se postepeno, jedan po jedan modul, spuštamo do najnižeg modula. Prvo se testira modul na vrhu, i onda se niži moduli integrišu i testiraju, jedan po jedan. Proces se završava kad su svi moduli integridsani i iztestirani.

Za naš primer, na slici 1, testiranje počinje od modula A, a niži moduli B1 i B2 se integrišu jedan po jedan. Ali, ovi niži moduli B1 and B2 nisu raspoloživi kada testiramo modul A. Zato, da bi testirali najviši modul A, potrebno je razviti tzv. "skele", “STUBS”.

"Skele", “Stubs”, je komad programa, code snippet, koji prihvata ulaze/zahteve od gornjeg modula i vraća rezultat tj odgovor. Na ovaj način, i ako ne postoje niži moduli, možemo da testiramo viši modul.

Izrada "skela tj. "stubs" može da bude složena i da uzima puno vremena. Pozvani modul može da bude vrlo složen, sa složenom poslovnom logikom, npr. veza sa bazom podataka. Tako da pravljenje "skela" može da oduzima puno vremena isto koliko i pravljenje realnog modula a ne lažnog modula.

"Drajveri" i "skele" su privremeni i veštački delovi programa koji se koriste kod testiranja da zamene nepostojeće module. Oni pozivaju funkcije tj metode i vraćaju kao rezultat odgovor, koji se poredi sa očekivanim ponašanjem tj. očekivanim odgovorom.

# INTEGRACIONI TESTOVI

*Konačno, evo koraci koji su potrebni kod integracionog testiranja, tj. izvršavanje integracionih testova.*

Konačno, evo koraci koji su potrebni kod integracionog testiranja, tj. izvršavanje integracionih testova:

- potrebno je proučiti arhitekturu aplikacije
- identifikovati module
- identifikovati šta svaki modul radi
- identifikovati kako se podaci razmenjuju tj. prenose između pojedinih modula
- identifikovati koji su ulazi i koji izlazi cele aplikacije (koji podaci se unose u sistem i koji podaci se primaju od sistema)
- identifikovati i napraviti test-slučajeve
- integraciono testiranje se može obaviti tehnikom bele kutije i tehnikom crne kutije

# VIDEO VEŽBA

## *Video vežba - Poređenje integracionog i jediničnog testiranja (Integration Vs Unit Testing)*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

# ZADACI ZA VEŽBU

## *Zadaci*

- Za izabranu aplikaciju napraviti plan, korak-po-korak, za integraciono testiranje?

# Poglavlje 4

# Alati za integraciono testiranje

## POPULARNI ALATI ZA INTEGRACIONO TESTIRANJE (ZA PISANJE INTEGRACIONIH TESTOVA)

*Ovde navodimo najbolje tj. najpopularnije alate za integraciono testiranje (Integration Testing Tools and Frameworks).*

Pogledati link: http://www.softwaretestinghelp.com/integration-testing-tools/

**Top 10 Integration Testing Tools to** Write IntegrationTests

Overview of the best Integration Testing Tools and Frameworks.

Postoji više nivoa testiranja, i jedan vrlo važan nivo je integraciono testiranje, “Integration Testing”, gde kombinujemo različite jedinice/module softvera i testiramo ih kao jednu grupu tj. jednu celinu. Pri tome se testiraju interfejsi između modula, i cilj je da se otkriju defekti izazvani integracijom modula.

Cilj integracionog testiranja, integration testing, je da obezbedi da pojedini moduli rade kako se očekuje posle integracije sa drugim modulima.

Mnoge kompanije koriste

- ili kombinovane jedinične testove
- ili koriste funkcionalne testove (end-to-end functional workflow tests).
- na tržištu postoje različiti integracioni test-alati (Integration Testing tools)

Evo liste popularnih integracionih alata i integracionih okvira (integration tools and frameworks): VectorCAST/C++, VectorCAST/Ada,, Citrus Integration Testing, LDRA, SMART INTEGRATION TEST ACCELERATOR (SITA), FitNesse, Rational Integration Tester, Protractor, TESSY, Validate MSG, Steam, Jasmine, eZscript, , Spock for JAVA, Pioneerjs.

## ALAT VECTORCAST/C++

*Npr. VectorCAST/C++ je Vector Software’s VectorCAST alat, popularan za jedinično testiranje i integraciono testiranje*

Npr. VectorCAST/C++ je Vector Software’s VectorCAST alat, popularan za

- jedinično testiranje
- i integraciono testiranje

**Ovaj alat koristi ideju da se jedinični testovi izvršavaju kao individualne komponente, a da integracioni testovi su kombinacija jediničnih testova u okviru nekog modula i da se izvršavaju kao jedna grupa.**
Alat VectorCAST/C++ se koristi za jezik C i C++, i omogućujuautomatizovanje jediničnih i integracionih testova.

# JAVA ALATI ZA PISANJE JEDINIČNIH I INTEGRACIONIH TESTOVA

## *Ovde navodimo popularne alate za pisanje jediničnih i integracionih testova.*

Pogledati link:

https://www.petrikainulainen.net/programming/testing/12-tools-that-i-use-for-writing-unit-and-integration-tests/

**Tools for** Writing Unit and Integration Tests**:**

Automatizovano testiranje je važan deo razvoja softvera, i ovde navodimo alate za to.

Npr. integraciono testiranje, Integration Testing, uz pomoć alata **Maven** , omogućuje da se izgradi Maven-build koji ima različite direktorijume za jedinično i integraciono testiranje.

A amlat JUnit je test-okvir koji omogućuje pisanje i jediničnih i integracionih testova.
Alat JUnit je najpopularniji testing-framework za Javu.

# VIDEO VEŽBA

## *Video vežba - Jedinično integraciono testiranje (Unit Integration Testing)*

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

# ZADACI ZA VEŽBU

## *Zadaci*

- Za izabranu aplikaciju demonstrirati jedinično integraciono testiranje (Unit Integration Testing).

# Poglavlje 5

# Integraciono testiranje - Vežbe

## DNEVNA DOGRADNJA I DNEVNO INTEGRACIONO TESTIRANJE

*Kod integracije softvera se preporučuje "daily build, and daily smoke test".*

Kod integracije softvera, bez obzira na izabranu strategiju integracije, preporuka je da se na dnevnom nivou vrši dogradnja i osnovno testiranje softvera. Ovo se zove "daily build and daily smoke test". Naime, svakog dana :

- svakog dana sve fajlove trenutno izgrađenog sistema treba kompajlirati, linkovati, i kombinovati u jedan funkcionišući program
- zatim izvršiti "smoke test", naime proveriti da li se trenutni proizvod "dimi" kada se izvršava tj. uraditi najosnovnije testiranje

Ovakav proces omogućuje 1) lakšu dijagnozu defekata i 2) povećava kvalitet softvera.

Kada se proizvod svakog dana dograđuje i testira, lakše je otkriti defekt. Jer, u suprotnom, mnogo je teže uraditi dijagnozu defekata i može se desiti da ne otkrijete sve defekte. Ovaj proces dnevnog dograđivanja i testiranja softvera, ima takođe veoma pozitivan efekat na moral programera. Dakle, česta tj. svakodnevna integracija i testiranje omogućuje da se rano otkriju neki problemi, u suprotnom neki problemi mogu da se pojave tek na kraju projekta i da budu vrlo teško otkriveni.

Kod integracije se preporučuje "daily build", mada u nekim organizacijama se vrši izgradnja softvera nedeljno a ne dnevno. Kod nedeljne izgradnje softvera, može da se desi da dođe do problema koji traju po par nedelja.

Pod "smoke test" se podrazumeva da se ceo izgrađeni sistem izloži nekom najosnovnijem testiranju, a ne iscrpnom testiranju, u cilju otkrivanja većih problema kod integracije. Ako dograđujete softver na dnevnom nivou, onda i na dnevnom nivou treba uraditi "dimni test" dakle najosnovnije testiranje. "Dimni test" evoluira tokom integracije, npr. u početku samo testiramo sistem da li daje jednu poruku tipa "hello world", ali postepeno ovaj "dimni teste" postaje sve zahtevniji i sveobuhvatniji.

Pod integracionim testiranjem se podrazumeva kombinovano izvršavanje dve ili više klasa, paketa, komponenti, podsistema,. Ovo testiranje poičinje od samog početka, kad imamo samo par klasa koje možemo da integrišemo i nastavlja se dok se ceo sistem iskompletira. Potsetimo se da razvojno testiranje obuhvata 1)testiranje jedinica, 2)testiranje komponenti, i 3 )integraciono testiranje. Integracija i integraciono testiranje vrše se zajedno.

# IZGRADNJA SOFTVERA TOKOM INTEGRACIJE

## *Evo kako izgleda proces izgradnje softvera tokom integracije*

Proces izgradnje (build process) softvera tokom integracije izgleda ovako:

- 1)kompajlirati sve fajlove, biblioteke, i druge komponente za trenutno izgrađenu verziju softvera (tj. vezije koja se sastoji od određenog broja jedinica/komponenti koje su integrisane u datom trenutku)
- 2)linkovati tj. povezati sve fajlove, biblioteke i druge komponente
- 3)izvršiti osnovno testiranje trenutno izgrađene verzije softvera tj. izvršiti tzv. "smoke test" izgrađene verzije softvera i pri tome eliminisati "bugs" tj. greške u programu ako se tada pojave
- 4)dograditi tj. integrisati sledeću jedinicu/komponentu i vratiti se na početak procesa tj. na korak 1)

Pod drugim komponentama se podrazumevaju npr. "skele" softvera koje se koriste da zamene ostatak sistema koji nije izgrađen.

Termin integracija znači softversku razvojnu aktivnost gde se više odvojenih softverskih komponenti kombinuje tj. integriše u jedinstvenu povezanu celinu. kod manjih projekata, integracija se sastoji od povezivanja jednog relativno malog broja klasa u jednu povezanu celinu,

- što se može uraditi za par sati,
- a kod većih projekata integracija može da traje nedeljama ili čak mesecima.
- za izgradnju softvera tokom integracije mogu se koristiti alati "build tools" koji automatizuju proces izgradnje i osnovnog testiranja softvera
- preporučuje se "daily build and smoke test" tj. da se na dnevnoj bazi vrši dogradnja i osnovno testiranje izgrađene verzije softvera
- na većim projektima se obavezno preporučuje automatizacija dogradnje softvera tj. upotreba "build tools" dok na manjim projektima to nije neophodno

# POKAZNI PRIMER

## *Pokazni primer predstavlja proces integracionog testiranja i aktivnosti u okviru samog procesa.*

Primer integracionog testiranja:

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

Na osnovu programske strukture primera datog u videu, integraciono testiranje polazi od identifikacije hijerarhije unutar programa sa akcentom na glavni deo programa. Moguće je izvršiti testiranje po dubini ili po širini zavisno od potreba razvojnog tima. Ukoliko se testiranje vrši po dubini vrši se identifikovanje glavne kontrolne putanje po definisanom redosledu. Na

taj način, kada je glavna putanja kroz program definisana, prolazi se kroz sve bitne stavke programa i vrši se njihovo ispitivanje.

Ukoliko se razvojni tim odluči za testiranje po širini, vrši se identifikacija neposrednih komponenti programa određenog nivoa kroz program. U tom slučaju testiranje se vrši horizontalnim putem.

Razlike između integracionog i jediničnog testiranja:

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

# PRIMER INTEGRACIONIH TESTOVA

## *Na sledećem linku je dat tutorijal za integraciono testiranje: https://www.guru99.com/integration-testing.html*

Na sledećem linku je dat tutorijal za integraciono testiranje:

https://www.guru99.com/integration-testing.html

Šta je integraciono testiranje?

- Kod integracionog testiranja, individualni softverski moduli se integrišu po nekom logičkom redosledu i testiraju se kao grupa.
- Jedan tipični softverski projekt sastoji se od niza softverskih modula, kodiranih od strane različitih programera.
- Integraciono testiranje se fokusira na proveru komuniciranja podataka između modula.

Integracioni test-slučajevi: Integracioni test-slučajevi se razlikuju od jediničnih test-slučajeva u smislu da se on fokusira pre svega na interfejse između modula tj. tok podataka između pojedinih modula. Dakle, kod integracionih test-slučajeva, prioritet se daje na vezama između modula a ne na funkcionalnost jedinica koje su već iztestirane kod jediničnog testiranja.

**Primer integracionih testova**: Npr. ako aplikacija ima 3 modula (modul 1, modul 2, modul 3), gde je modul 1 povezan sa modulom 2 i modul 2 povezan sa modulom 3. Ovde znači ne interesuje nas testiranje modula 1 jer je on već jedinično iztetsiran, već nas interesuje kako je povezan sa modulom 2, i slično kako je modul 2 povezan sa modulom 3. Dole je data tabela integracionih test slučajeva.

**Test-slučaj Cilj test-slučaja Opis test-slučaja Očekivani rezultat test-slučaja**

**1** Provera veze između 1 i 2 ulaz u modul 1 koji proizvodi ulaz u modul 2 izlaz iz modula 2

**2** Provera veze između 2 i 3 ulaz u modul 2 koji proizvodi ulaz u modul 3 izlaz iz modula 3

# VIDEO TUTORIJAL INTEGRACIONOG TESTIRANJA

## *Dole je dat video tutorijal koji objašnjava integraciono testiranje.*

Dole je dat video tutorijal koji objašnjava integraciono testiranje. Detaljno su objašnjeni različiti prilazi integracionom testiranju, i to kroz praktične primere aplikacija koji se sastoje od više modula. Takođe objašnjena je uloga Stubs i Drivers ("skele" i "drajveri")

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

# POKAZNI PRIMERI - LINKOVI

## *Pokazna vežba*

Proučiti sledeće pokazne primere integracionog testiranja:
http://testingbasicinterviewquestions.blogspot.rs/2012/01/what-is-integration-testing-explain-it.html
https://www.guru99.com/integration-testing.html
http://www.softwaretestinghelp.com/what-is-integration-testing/

# INDIVIDUALNA VEŽBA (ZADACI ZA SAMOSTALAN RAD)

## *Zadaci za individualnu vežbu.*

ZUadaci:

- Za izabranu apliaciju, i za izabrani test-alat, uraditi jedan primer integracionog testiranja koristeći top-down tehniku ?
- Za izabranu aplikaciju, i za izabrani test-alat, uraditi jedan primer intergacionog testiranja koristeći bottom-up tehniku ?
- Za izabranu aplikaciju, uraditi jedan primer integracionog testiranja koristeći alat JUnit i tehniku bele kutije?
- Za izabranu aplikaciju, uraditi jedan primer integracionog testiranja koristeći alat JUnit i tehniku crne kutije?

# DOMAĆI ZADATAK - DZ10

## *10. domaći zadatak je individualan za svakog studenta.*

10. domaći zadatak je individualan za svakog studenta, i dobija se po definisanim instrukcijama.
Domaći zadatak se imenuje:
**KI301-DZ10-ImePrezime-brIndeksa** gde su vrednosti Ime, Prezime i br.Indeksa vaši podaci.

Domaći zadatak je potrebno poslati na adresu asistenta: nebojsa.gavrilovic@metropolitan.ac.rs sa naslovom (subject mail-a) **KI301-DZ10**.

**Posebno je potrebno voditi računa o pravilnom imenovanju mail-a prilikom slanja domaćih zadataka.**
**Napomena:**
**Domaći zadaci treba da budu realizovani u zadatku navedenom razvojnom okruženju i da predstavljaju jedinstveno rešenje svakog studenta.** Prepisivanje i preuzimanje programskog koda sa interneta strogo je zabranjeno.

# Poglavlje 6

# Zaključak

## ZAKLJUČAK

Kod integracije softvera, bez obzira na izabranu strategiju integracije, preporuka je da se na dnevnom nivou vrši dogradnja i osnovno testiranje softvera. Ovo se zove "daily build and daily smoke test". Naime, svakog dana :
sve fajlove trenutno izgrađenog sistema treba kompajlirati, linkovati, i kombinovati u jedan program koji može da se izvrši
zatim izvršiti "smoke test", naime proveriti da li se trenutni proizvod "dimi" kada se izvršava tj. uraditi najosnovnije testiranje
Ovakav proces omogućuje 1) lakšu dijagnozu defekata i 2) povećava kvalitet softvera.
Kada se proizvod svakog dana dograđuje i testira, lakše je otkriti defekt. Jer, u suprotnom, mnogo je teže uraditi dijagnozu defekata i može se desiti da ne otkrijete sve defekte. Ovaj proces dnevnog dograđivanja i testiranja softvera, ima takođe veoma pozitivan efekat na moral programera. Dakle, česta tj. svakodnevna integracija i testiranje omogućuje da se rano otkriju neki problemi, u suprotnom neki problemi mogu da se pojave tek na kraju projekta i da budu vrlo teško otkriveni.
Kod integracije se preporučuje "daily build", mada u nekim organizacijama se vrši izgradnja softvera nedeljno a ne dnevno. Kod nedeljne izgradnje softvera, može da se desi da dođe do problema koji traju po par nedelja.
Pod "smoke test" se podrazumeva da se ceo izgrađeni sistem izloži nekom najosnovnijem testiranju, a ne iscrpnom testiranju, u cilju otkrivanja većih problema kod integracije. Ako dograđujete softver na dnevnom nivou, onda i na dnevnom nivou treba uraditi "dimni test" dakle najosnovnije testiranje. "Dimni test" evoluira tokom integracije, npr. u početku samo testiramo sistem da li daje jednu poruku tipa "hello world", ali postepeno ovaj "dimni teste" postaje sve zahtevniji i sveobuhvatniji.

## REFERENCE

1. Code Complete: A practical handbook of software construction, by S. McConnell, Microsoft Press, ISBN 0-7356-1967-0, https://www.amazon.com/Code-Complete-Practical-Handbook-Construction/dp/0735619670
2. SWEBOK-V3, https://www.computer.org/web/swebok/v3
3. Theory and Problems of Software Engineering - Schaum’s Outline Series, by David Gustafson, ISBN 0-07-137794-8, http://www.mhebooklibrary.com/doi/abs/10.1036/0071377948

Linkovi:
http://testingbasicinterviewquestions.blogspot.rs/2012/01/what-is-integration-testing-explain-it.html

https://www.guru99.com/integration-testing.html
http://www.softwaretestinghelp.com/what-is-integration-testing/