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

Uvod u konstruisanje softvera

Lekcija 01

PRIRUČNIK ZA STUDENTE

KI301 - KONSTRUISANJE SOFTVERA

Lekcija 01

UVOD U KONSTRUISANJE SOFTVERA

- Uvod u konstruisanje softvera
- Poglavlje 1: Aktivnosti kod životnog ciklusa softvera
- Poglavlje 2: Poređenje modela životnog ciklusa
- Poglavlje 3: Konstruisanje softvera-uvod
- Poglavlje 4: Procesni model softvera
- Poglavlje 5: Pripreme za konstruisanje
- Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

L1 - UVOD U KONSTRUISANJE SOFTVERA

L1 - Uvod u konstruisanje softvera - Sadržaj:

Uvod u konstruisanje softvera:

- Aktivnosti kod životnog ciklusa softvera
- Poređenje modela životnog ciklusa
- Konstruisanje softvera-uvod
- Procesni model softvera
- Konstruisanje kod Unified Process-a
- Vežba – UP/RUP kod razvoja softvera

Poglavlje 1

Aktivnosti kod životnog ciklusa softvera

ŽIVOTNI CIKLUS SOFTVERA-AKTIVNOSTI

Detaljni dizajn to je dizajniranje algoritama za pojedine delove (tj. tzv. jedinice (units)) softvera, i dizajniranje pojedinih objekata.

Životni ciklus **softvera** definiše se kao serija različitih aktivnosti koje se dešavaju u toku razvoja softvera. Takodje, pojavljuju se različiti produkti (*deliverables)* proizvedeni tokom životnog ciklusa softvera, npr. softverski izvorni kod (*source code*) i korisničko uputstvo (*user manuel*). Sledeće su aktivnosti koje čine životni ciklus softvera:

1) **Analiza izvodljivosti** (Feasibility study) – Odredjivanje da li je predloženi razvoj svrsishodan(vredan da se izvodi).

2) **Analiza tržišta** (*Market analysis*) – Odredjivanje da li postoji potencijalno tržište za predloženi proizvod.

3) **Odredjivanje zahteva** (Requirements determination) – Odredjivanje (specificiranje) koje funkcije bi softver trebao da sadrži (odredjivanje funkcionalnosti). **Prikupljanje zahteva** (Requirement elicitation): obezbedjivanje zahteva od korisnika.

4) **Analiza domena** (Domain analysis): Odredjivanje koji ciljevi i strukture su zajedničke za postavljeni problem.

5) **Planiranje projekta** (Project planning) – Odredjivanje kako da se razvije softver. **Analiza troškova** (*Cost analysis*)– Procena troškova razvoja softvera. **Vremensko planiranje** (Scheduling) – Izrada vremenskog plana razvoja softvera. **Obezbedjivanje kvaliteta** (*Quality assurance*): definisanje aktivnosti koje će obezbediti kvalitet softvera. **Pregled radova** (*Work-breakdown structure*) – definisanje podciljeva potrebnih da se razvije softver.

6) **Softverski dizajn** (Software design) - Kreirati softver da omogući zeljenu funkcionalnost (da ispuni željene zahteve). **Arhitektonski dizajn**(Architectural design) – Dizajniranje strukture softvera (softverskog sistema). **Detaljni dizajn** (Detailed design) – Dizajniranje algoritama za pojedine delove (tj. tzv. jedinice (*units*)) softvera, i dizajniranje pojedinih objekata. **Dizajniranje interfejsa** (Interface design) – Dizajniranje interfejsa (interakcije) izmedju pojedinih delova softvera.

7) **Implementacija softvera** (Software implementation) – Izgradnja softvera, tj pisanje koda, kao i *debugging*.

ŽIVOTNI CIKLUS SOFTVERA-AKTIVNOSTI-NASTAVAK

Integraciono testiranje to je testiranje tokom integracije pojedinih delova softvera.

8) **Testiranje** (Testing) – Egzekucija (pogon) softvera sa podacima u cilju obezbedjivanja/ provere ispravnog rada softvera. **Pojedinačno testiranje** (Unit testing) - Testiranje pojedinih delova softvera od strane prvobitnog kreatora/proizvodjača softvera. **Integraciono testiranje** (*Integration testing*) – Testiranje tokom integracije pojedinih delova softvera. **Sistemsko tesiranje** (System testing) – Testiranje softvera u uslovima/okruženju koje odgovara operacionim-pogonskim uslovima. **Alfa testiranje** (Alpha testing) – Testiranje od strane korisnika softvera na lokaciji kod proizvodjača softvera. **Beta testiranje** (Beta testing) – Testiranje od strane korisnika na lokaciji kod korisnika softvera. **Testiranje prihvatljivosti** (Acceptane testing) – Testovi u cilju zadovoljenja korisnika/kupca.

9) **Isporuka** (Delivery*)* – Isporuka korisniku efektivnog softverskog rešenja. **Instalacija** (*Installation*) – Instalisanje softvera na loaciji kod korisnika. Treniranje (Training) – Treniranje korisnka da koristi softver. **Tehnička podrška** (Help desk) – Odgovaranje na pitanja korisnika.

10) **Održavanje** (Maintenance) – Ažuriranje i poboljšavanje/modifikovanje softvera u cilju obezbedjenja korisnosti u odredjenom vremenskom periodu.

VIDEO VEŽBA

Video vežba - Softverski razvojni životni ciklus

Pogledati video: Software Development Lifecycle in 9 minutes!

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

VIDEO VEŽBA 2

Video vežba - Životni ciklus softvera detaljno je objašnjen na sledećem video primeru.

Životni ciklus softvera detaljno je objašnjen na sledećem video primeru: Software Development Life Cycle (SDLC) - Detailed Explanation.

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

VEŽBA - PITANJA ZA SAMOPROVERU ZNANJA

Vežba.

Odgovoriti na sledeća pitanja:

- Nabrojati aktivnosti kod životnog ciklusa softvera?
- Zašto softverske ompanije koriste procesne razvojne metodologije?

Poglavlje 2

Poređenje modela životnog ciklusa

POREĐENJE AGILNIH I TRADICIONALNIH METODA RAZVOJA SOFTVERA

Agilne metode razvoja softvera su u stvari procesni okviri.

Agilne metode razvoja softvera (grupa razvojnih metoda) se baziraju na inkrementalnom/ iterativnom razvoju softvera (inkrementalni/evolutivni model), gde zahtevi i rešenja evoluiraju, a ovo se obavlja kroz kolaboraciju izmedju nekoliko timova, uključujući interakciju sa korisnikom. Ove metode promovšu adaptivno planiranje, evolutivni razvoj i isporuku, time-boxed (fiksni rokovi) iterativni pristup, brzinu i fleksibilnost na promene u toku izrade projekta. To je konceptualni okvir (procesni okvir) koji promoviše interakciju izmedju korisnika i softverskih timova. U ove metode spada: AUP (agile unified process)/RUP (rational unified process), Scrum process, Proces extremnog programiranja. Dole je datopoređenje agilnih i tradicionalnih metoda razvoja životnog ciklusa.

Mogu se uporediti agilne metode i tradicionalna metoda:

Agilna metoda	Sekvencijalna metoda
Iterativna	Planirana unapred
Mali koraci	analiza pre dizajna
Adaptivno planiranje	unapred planiranje
Moderno	tradicionalno
Promenljivi zahtevi	fiksni zahtevi
Interakcija ljudi	procesi definisani
Komunikacija	dokumentacija
Mali timovi	veliki timovi
Nepoznati projekti	Poznati projekti

OSOBINE AGILNIH PROCESA

RUP (racionalni ujedinjeni proces) spada u agilne metode razvoja softvera.

U „agilne procese“ spadaju Scrum proces, Ekstremno programiranje, RUP, itd. Agilni procesi su adaptivni, jer se polazi od pretpostavke da softverski projekti nisu lako predvidivi, već da tokom izrade projekta treba se suočavati sa promenama u softverskim zahtevima (zahtevi funkcionalnosti softvera) i u planiranju. Agilni procesi su iterativni, i koriste se kratke i

fiksne iteracije, npr. jednomesečne iteracije ili još kraće iteracije (od 1 do 4 nedelje), i ove iteracije imaju fiksno trajanje. Takodje, koristi se koncizna dokumentacija umesto obimne dokumentacije. Na kraju svake iteracije se vrši prezentacija rezultata tj. softvera i njegove funkcionalnosti korisnicima i vrši se konsultacija sa korisnicima posle svake iteracije. Dakle , agilni pristup je iterativan, inkrementalan i evolutivan.

Takodje, kod agilnog pristupa, imamo organizaciju projekta takvu da se koriste timovi koji sarajuju medju sobom , a svaki tim ima različit zadatak, i to su tzv.cross-functional timovi tj. timovi koji obuhvataju ljude sa različitim iskustvom ali rade na zajedničkom zadatku.

Konačno, kod agilnog pristupa se testiranje vrši paralelno sa programiranje ili ako ne paralelno onda iza svake pojedinačne iteracije a ne samo jednom na kraju projekta posle kodiranja.

Dakle, u agilne metode spadaju, koje se mogu sumirati na sledeći način:

- Scrum metoda, gde se ova metoda fokusira na upravljanje projektom
- Ekstremno programiranje (Extreme programming -XP), koji pokriva neke aspekte razvojnog procesa
- Racionalni unificiran proces (Rational Unified Process-RUP), koji pokriva kompletan razvojni proces

VIDEO VEŽBA

Video vežba - Poređenje agilnih metoda i vodopadnog modela (Agile Vs Waterfall - Four Massive Differences)

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

PITANJA ZA SAMOPROVERU ZNANJA

Odgovoriti na pitanje.

Pitanja za samoproveru znanja:

- Nabrojati agilne procese?
- Nabrojati osobine agilnih procesa?

Poglavlje 3

Konstruisanje softvera-uvod

KONSTRUISANJE SOFTVERA - DEFINICIJA

Ovde definišemo šta je to konstruisanje softvera, značaj konstruisanja softvera, i koje aktivnosti obuhvata konstruisanje softvera.

Konstruisanje (tj. izgradnja tj. neposredna proizvodnja tj. izrada) softvera (software construction) je deo ukupnog procesa razvoja softvera. Razvoj softvera (software development) je vrlo komplikovan proces, i on obuhvata čitav niz aktivnosti:

- početno definisanje problema
- razvoj zahteva
- konstrukciono planiranje
- softverska arhitektura tj. dizajn na visokom nivou abstrakcije
- detaljni dizajn tj. dizajn na niskom nivou abstrakcije
- kodiranje
- eliminacija greški tj. debugging
- testiranje jedinica softvera (unit testing)
- integracija
- integraciono testiranje
- sistemsko testiranje
- korektivno održavanje

Aktivnosti koje pripadaju konstruisanju softvera su:

konstrukciono planiranje, detaljni dizajn, kodiranje, debugging, unit testing, integracija, integraciono testiranje.

Dakle, razvoj softvera se može predstaviti preko sledećih aktivnosti: početno definisanje problema, razvoj zahteva, arhitektonski dizajn, konstruisanje softvera, sistemsko testiranje i održavanje. Medjutim, u toku konstruisanja može doći do modifikacije softverske arhitekture, i modifikacije/dopune specifikacije zahteva, a kod sistemskog testiranja i korektivnog održavanja može doći do modifikacije konstruisanja softvera.

KONSTRUISANJE SOFTVERA - KONKRETNE AKTIVNOSTI

Data je lista konkretnih aktivnosti kod konstruisanja softvera.

Konstruisanje softvera obuhvata sledeće aktivnosti: konstrukciono planiranje, detaljni dizajn, kodiranje, debugging, unit testing, integracija, i integraciono testiranje. Evo liste konkretnih aktivnosti (tj. podaktivnosti) kod konstruisanja softvera:

- verifikacija arhitekture i zahteva
- planiranje testiranja
- dizajn i kodiranje klasa
- dizajn i kodiranje podprograma (routines) tj. klasnih metoda
- kreiranje i imenovanje varijabli i imenovanih konstanti
- izbor kontrolne logike i organizacija blokova klasnih metoda
- testiranje jedinica
- integraciono testiranje
- debugging
- doterivanje programskog koda (formatiranje i komentiranje programskog koda), program formatting i program commenting
- integracija softverskih komponenti
- podešavanje programskog koda u cilju postizanja veće brzine i minimizacije resursa (code tuning)

VAŽNOST KONSTRUISANJA SOFTVERA

Konstruisanje softvera obuhvata veliki deo od totalnog vremena potrebnog za razvoj softvera.

Razvoj softvera se može pretstaviti da se sastoji od:

- početno definisanje problema
- razvoj zahteva
- arhitektonski dizajn
- konstruisanje softvera
- sistemsko testiranje
- korektivno održavanje

Konstruisanje softvera je važno iz sledećih razloga:

- konstruisanje je jedan veliki deo u procesu razvoja softvera, npr. od 30% do 80% u zavisnosti od tipa i veličine projekta
- konstruisanje je centralna aktivnost razvoja softvera, tj. arhitektura softvera i zahtevi softvera se urade pre konstruisanja a sistemsko testiranje posle konstruisanja u cilju verifikacije korektnosti konstruisanja
- produktivnost individualnog programera može da varira veoma puno (faktor 10) pa je potrebno fokusirati se na efikasno konstruisanja softvera
- konstruisanje softvera je obavezni deo razvoja softvera, naime, pre konstruisanja treba pažljivo uraditi razvoj zahteva i arhitektonski dizajn a posle konstruisanja treba uraditi detaljno sistemsko testiranje, ali često u praksi tj. u realnom životu se odmah počne sa konstruisanjem bez pažljivog razvoja zahteva i arhitekture, a sistemsko testiranje se često uradi na brzinu jer je projekat premašio rok završetka

KONSTRUISANJE SOFTVERA-ZNAČAJ

Greške koje nastaju u toku konstruisanja softvera često mogu da se poprave brzo i lako, ako se ova popravka vrši u toku samog konstruisanja softvera.

Konstruisanje tj. izrada tj. izgradnja softvera je u stvari glavni deo pravljenja softvera. Konstruisanje (izrada) softvera, čiji krajnji rezultat je tzv. izvorni kod softvera, tipično čini oko 65% od ukupnog posla na nekom manjem softverskom projektu, a oko 50% na srednjim/ većim projektima. U toku konstruisanja softvera pravi se tipično oko 50% do 75% grešaka. Iz ovoga je već jasno da je konstruisanje softvera oblast gde se može izgubiti puno vremena ako joj se ne pristupi ozbiljno i sistematično.

Greške koje nastaju u toku konstruisanja softvera često mogu da se poprave brzo i lako, ako se ova popravka vrši u toku samog konstruisanja softvera. Medjutim, ako se ove greške ostave neotkrivene i neeliminisane, šteta može biti ogromna. Konstruisanje tj. izgradnja softvera (Software Construction) je glavni deo proizvodnje softvera, i on uključuje pre svega

detaljni dizajn softvera (detailed design),

kodiranje (*coding*),

otklanjanje skrivenih/logičkih tj. izvršnih grešaka (debugging),

integraciju softverskih jedinica (integration),

pojedinačno testiranje i integraciono testiranje (unit testing, integration testing).

Znači, konstruisanje softvera pre svega obuhvata detaljni dizajn, kao i samo kodiranje i otklanjanje grešaka u softveru, ali takodje da bi se ovo pravljenje tj. izrada softvera obavila efikasno i kvalitetno, potrebno je izvršiti planiranje procesa konstruisanja softvera, naročito kod velikih softverskih projekata.

KONSTRUISANJE SOFTVERA-SADRŽAJ

Ovde se daje detaljna lista aktivnosti kod konstruisanja softvera.

Konstruisanje softvera je centralna aktivnost razvoja softvera. Krajnji proizvod konstruisanja softvera je softverski kod tj. tzv. izvorni kod. Konstruisanje tj. izgradnja softvera obuhvata sledeće konkretne aktivnosti:

Dizajniranje klasa i podprograma, tj. detaljni dizajn softvera

Pisanje izvornog koda, naime izv. koda klasa i izv. koda podprograma

Kreiranje i imenovanje varijabli, i imenovanih konstanti

Izbor logičkih upravljačkih struktura, i organizacija blokova instrukcija

Otkrivanje i eliminacija grešaka u sopstvenom kodu, tj. ispravke softvera

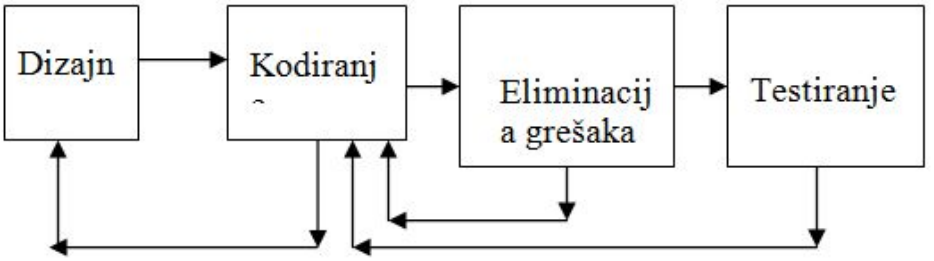
Provera/preispitivanje/profinjavanje detaljnog dizajna

Pojedinačno testiranje pojedinih komponenti softvera, komponenti napravljenih odvojeno od ostatka softvera

Integraciono testiranje interkonektovanih komponenti softvera

ITERATIVNO KONSTRUISANJE SOFTVERA

Ovde je dat grafički prikaz konstruisanja softvera



Slika 3.1 Iterativno konstruisanje softvera

Slika gore ilustruje process konstruisanja softvera, naglašavajući iterativnu prirodu tj. iterativni koncept ovog procesa, tj. potrebu da se ovaj process više puta ponovi u cilju profinjavanja i poboljšanja.
Treba napomenuti, da npr. na slici vidimo da kod testiranja možemo otkriti greške u softveru, i da je onda potrebno vratiti se nazad na kodiranje da bi ispravili grešku. Ali, isto tako moguće je da se kod testiranja otkriju greške koje su u dizajnu, i onda se moramo vratiti na dizajn pa tek onda na kodiranje, da bi ispravili grešku u dizajnu softvera.

PROCES RAZVOJA SOFTVERA

Ovde se proces razvoja softvera pretstavlja kao niz aktivnosti.

Razvoj softvera je komplikovan proces kad se radi o većim projektima tj. većem softveru, i taj proces uključuje čitav niz aktivnosti:

Definisanje problema,

Definisanje zahteva softvera,

Planiranje konstruisanja,

Dizajn softverske arhitekture (architectural design) tj. dizajn strukture softvera (*arhitektonski dizajn*),

Detaljni dizajn tj. dizajn pojedinih komponenti softvera (low-level design),

Kodiranje (Coding),

Otkrivanje/otklanjanje skrivenih/logičkih grešaka, tj. ispravke softvera (Debugging),

Testiranje pojedinih softverskih jedinica (Unit Testing),

Integraciono testiranje (integration testing), Integracija softverskih jedinica(Integration),

Sistemsko testiranje (SystemTtesting),

Korektivno održavanje

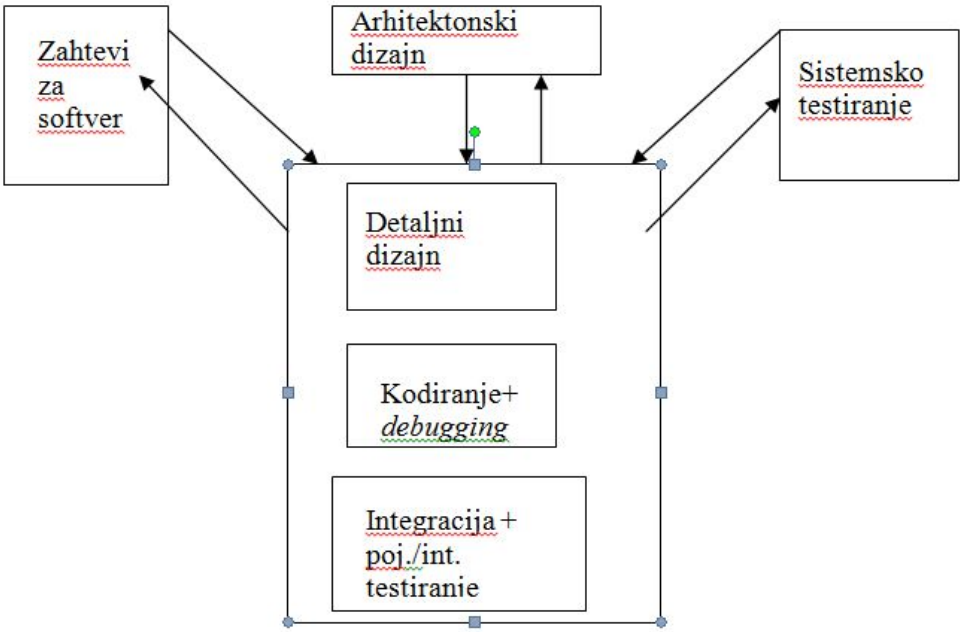
KONTEKST KONSTRUISANJA SOFTVERA

Konstruisanje softvera je deo životnog ciklusa softvera i treba znati kontekst konstruisanja softvera u okviru životnog ciklusa softvera

Bitno je staviti proces konstruisanja softvera u kontekst ostalih aktivnosti koje su obuhvaćene kod razvoja softvera. Na slici dole ilustrovana je interakcija izmedju procesa konstruisanja softvera i ostalih aktivnosti razvoja softvera. I ako se proces konstruisanja obavlja posle arhitektonskog dizajna i definisanja zahteva, ipak u postupku konstruisanja softvera obično se dolazi do ideja o modifikaciji arhitektonskog dizajna i redefinisanja zahteva za softver, kao i modifikaciji plana sistemskog testiranja softvera, što je prikazano na slici. Takodje, vidimo na slici koje su glavne aktivnosti uključene u konstruisanje softvera. U stvari, tek kad je proces razvoja softvera gotov, tek onda može da se sagleda do kraja problem razvoja softvera, i ima slučajeva kada se tek na kraju projekta dodje do ideja o radikalnim izmenama u arhitekturi softvera, detaljnom dizajnu, ili pak definisanju zahteva za softver i sistemskom testiranju softvera.

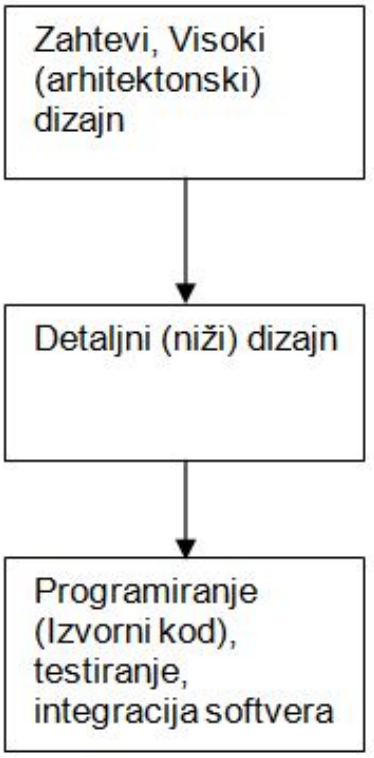
KONTEKST KONSTRUISANJA SOFTVERA-SLIKA

Ovde vidimo grafički prikaz konteksta konstruisanjasoftvera.



NIVOI RAZVOJA SOFTVERA

Na dijagramu vidimo „Visoki“ i „niži“ dizajn.



Slika 3.2 Nivoi razvoja softvera

VERIFIKACIJA SOFTVERA?

Šta je to konstruisanje softvera? Konstruisanje softvera je deo ukupnog procesa razvoja softvera i obuhvata detaljni razvoj softvera.

Konstruisanje softvera (software construction) je jedna disciplina tj. oblast softverskog inženjerstva. To je detaljno kreiranje softvera koje obuhvata:

detaljni dizajn softvera, kodiranje, testiranje jedinica softvera (unit testing), integraciono testiranje, eliminaciju defekata tj. debugging, i verifikacija softvera. Dakle, konstruisanje softvera je jako povezano i značajno se preklapa se sa dizajnom softvera i testiranjem softvera (koje takodje predstavjaju discipline softverskog inženjerstva) i verifikacijom softvera.

Verifikacija softvera obuhvata testiranje softvera (unit testing tj. testiranje funkcija i klasa, zatim testiranje modula tj. grupe klasa, zatim integraciono testiranje gde se testira nekoliko modula zajedno I testiranje sistema gde se testira ceo system, kao i funkcionalno testiranje, i nefunkcionalno testiranje tj. *performance test*). Verifikacija obuhvata i inspekciju programa pre njegovog izvršavanja tj. *code reviews* (gde ima nekoliko tehnika).Konstruisanje softvera podrazumeva upotrebu standarda, npr. standardi za format dokumenata I sadržaj dokumenata. Takodje, kod konstruisanja softvera se koristi UML (Unified Modelling Language) kao standardni model (notaciju) softvera.

Konstruisanje softvera je deo ukupnog procesa razvoja softvera i obuhvata detaljni razvoj softvera. Kod razvoja softvera koriste se mnogi modeli, a jedan broj ovih modela se koristi kod konstruisanja softvera. Npr. UML modeli softvera. Ali takodje i procesni modeli koji opisuju process razvoja softvera: “vodopadni model” (linearni model), ili iterativni modeli kao što su “evolutivni prototajping” (*evolutionary prototyping*), ekstremno programiranje (extreme programming), Scrum model.

KONSTRUISANJE SOFTVERA - METODE I ALATI

Ovaj slajd daje kratak opis aktivnosti kod konstruisanja softvera.

Konstruisanje softvera obuhvata prvo izbor pa onda planiranje primene i najzad primenu:

Modela za konstruisanje softvera, npr. UML-modeli

Metoda za konstruisanje softvera, npr. objektno-orijentisane metode razvoja softvera, ili formalne metode specifikacije softvera, ili metode prototipovanja (prototyping methods, npr. evolutivni metod ili “*throwaway”* metod), ili *data-oriented* metode, itd.

Alata za konstruisanje softvera, npr. Visual Studio, Power Designer, itd.

Izbor metode razvoja/konstruisanja softvera je od velike važnosti za proces konstruisanja softvera: npr. agilni metod razvoja/konstruisanja softvera, ili tradicionalni tj. “vodopadni” metod. S obzirom da je konstruisanje glavni deo procesa razvoja softvera, onda izabrani metod razvoja softvera se odnosi i na proces konstruisanja softvera.

Pomenimo sledeće alate:

Software design tools,alati za kreiranje i proveru dizajna softvera

Software construction tools, alati za proizvodnju softvera, npr. program editors, compilers, code generators, debuggers.

Software testing tools,

Itd.

KORISNI LINKOVI

Evo liste linkova korisnih za bolje razumevanje uvoda u konstruisanje softvera.

Pogledati linkove:

https://en.wikipedia.org/wiki/Software_construction

https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-005-software-construction-spring-2016/

http://swebokwiki.org/Chapter_3:_Software_Construction

https://sites.google.com/site/toolsgsd/tools-1/software-construction-tools

VIDEO VEŽBA

Video vežba - Software Construction in Java

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

PITANJA ZA SAMOPROVERU ZNANJA

Evo liste pitanja, za individualnu vežbu.

Pitanja:

- Navesti konkretne aktivnosti kod konstruisanja softvera?
- Koje aktivnosti prethode konstruisanju softvera, a koje aktivnosti se obavljaju posle?
- Šta je to arhitektonski dizajn?
- Šta je to detaljni dizajn?
- Objasniti uzajamnu vezu između konstruisanja softvera i arhitektonskog dizajna?

Poglavlje 4

Procesni model softvera

PROCESNI MODEL SOFTVERA-DEFINICIJA

Procesni model softvera opisuje procese koji su izvršeni u cilju razvoja softvera.

Procesni model softvera (The Software Process Model) opisuje procese koji su izvršeni u cilju razvoja softvera. Ovaj model obično sadrži sledeće:

Ciljevi

Rezultati

Učesnici

Odluke (opciono)

Notacija koja se koristi može da varira. Standardni procesni model koristi krugove/elipse za ciljeve i procese. Rezultati su predstavljeni kao pravougaonici, i učesnici kao štapaste figure. Često, odluke nisu uključene u model. Odluke se mogu predstaviti pomoću rombova. Protok se prikazuje pomoću strelica, obično s leva na desno i odozgo na dole. Strelice obično nisu označene tekstom. Sledeća su pravila i interpretacije za formiranje korektnih procesnih modela:

Dva cilja ne mogu biti povezana strelicom; ciljevi moraju biti razdvojeni rezultatima;

Cilj nije izvršan ukoliko njegov ulazni rezultat ne postoji;

Postoji jedan ili više početnih ciljeva i jedan ili više završnih ciljeva;

Svi ciljevi moraju biti dostižni sa početnog cilja;

Postoji putanja od svakog cilja do završnog cilja.

PROCESNI MODEL SOFTVERA-PRIMERI

Softverski procesni model može biti deskriptivan a ne samo grafički.

Softverski procesni model može biti deskriptivan, tj. on može opisati šta se desilo u toku jednog razvojnog softverskog projekta. Ovaj opisni model se često koristi kao deo završne analize projekta. Ovo je korisno u cilju identifikacije problema u toku procesa razvoja softvera. Ili, procesni model može biti predvidjajući, tj. on može da se koristi da opiše šta se predvidja

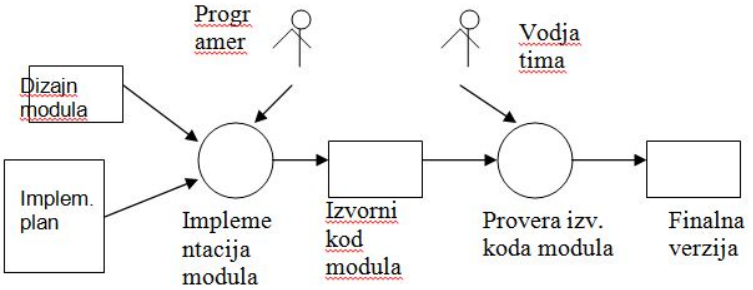
da će se desiti tokom projekta. Takodje može se koristiti za treniranje početnika. Ili za opis netipičnih dogadjaja.

Primer 1:

Na donjoj slici je prikazan procesni model za implementaciju jedne jedinice softvera. Dva su aktera tj. učesnika, vodja tima i programer. Programer je odgovoran za implementaciju te jedinice. programer koristi dizajn jedinice i plan implementacije da zaokruži zadatak implementacije jedinice softvera. Rezultat ove aktivnosti je izvorni kod jedinice. Vodja tima pregleda rezultate implementacije, i ova aktivnost treba da rezultira u prihvatanje implementacije. Ovaj model ne prikazuje eksplicitno šta se dešava ako implementacija nije uspešna. Takodje, ako vodja tima ne želi da prihvati rezultate testiranja, onda se proces obnavlja. To je ilustrovano u sledećem primeru (primer 2).

PROCESNI MODEL SOFTVERA-PRIMER 1

Slika prikazuje procesni model za implementaciju jednog modula softvera.



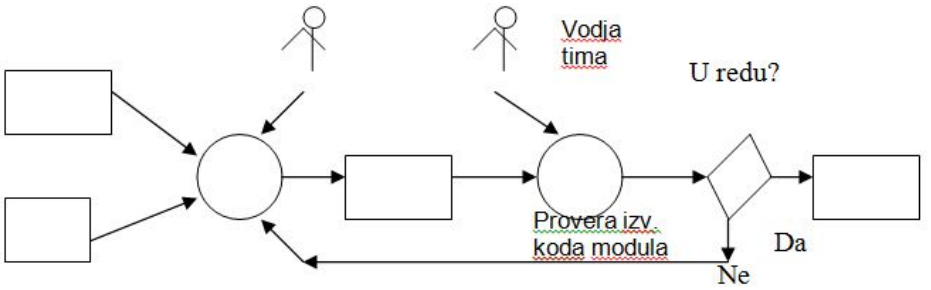
Slika 4.1 Procesni model-primer 1

PROCESNI MODEL SOFTVERA-PRIMER 2

Slika prikazuje procesni model za implementaciju modula softvera, sa uključenim odlukama.

Primer 2:

Dodavanje odluka u model omogućuje da procesni model bude eksplicitan, prikazujući šta se dešava u svim okolnostima.



ZADACI ZA SAMOSTALAN RAD - PRIMENA POWER DESIGNER ALATA

Koristeći Power designer kreirati procesni model dijagram za odabranu aplikaciju.

Zadaci:

- Pogledati priloženi video tutorijal.
- Za potrebe izabrane aplikacije napraviti proces model diagram koji će obuhvatati korisnike koju su dali zahteve za sistem, procese koji se obavljaju unutar aplikacije i odluke koje donose korisnici u različitim koracima aplikacije.
- Takođe, potrebno je prikazati ostvarene rezultate korisnika prilikom toka procesa u aplikaciji.
- Dijagrame raditi u Visual studio ili Power Designer razvojnom okruženju.

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

ZADACI ZA SAMOSTALAN RAD

Evo lista Procesni model – Zadaci.

1. Nacrtaj procesni model za zadatak farbanja zidova sobe. Uključiti sledeće podzadatke: Izbor boje, Kupovina farbe, Pranje zidova, Mešanje farbe, farbanje
2. Nacrtati procesni model softvera za testiranje jednog modula softvera.
3. Ilustrovati primenu procesnog modela kod planiranja procesa konstruisanja softvera.

Odgovoriti na sledeća pitanja:

1. Koja je razlika izmedju modela životnog ciklusa i procesnog modela?
2. Zašto su odluke prisutnije u perspektivnom procesnom modelu nego u deskriptivnom procesnom modelu?
3. Koji je značaj procesnog modela za konstruisanje softvera?

DOMAĆI ZADATAK 1

Na osnovu urađenih primera u vežbama potrebno je uraditi prvi domaći zadatak koji je univerzalan za sve studente.

Domaći zadatak se imenuje:

SE211-DZ01-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) SE211-DZ01.

Posebno je potrebno voditi računa o pravilnom imenovanju mail-a prilikom slanja domaćih zadataka.

Napomena: Prvi domaći zadatak je univerzalan i važi za sve studente, dok ostale domaće zadatke dobijate od asistenta nakon realizacije prethodnog.

Domaći zadaci treba da budu realizovani u razvojnom okruženju koje je definisano domaćim zadatkom i da predstavljaju jedinstveno rešenje svakog studenta. Prepisivanje i preuzimanje programskog koda sa interneta strogo je zabranjeno.

Poglavlje 5

Pripreme za konstruisanje

VAŽNOST PRIPREMA ZA KONSTRUISANJE

Greške u zahtevima i arhitekturi mogu se vrlo loše odraziti na konstruisanje, a greške u konstruisanju su vrlo skupe (njihovo otklanjanje je skupo) u fazi sistemskog testiranja.

Pre nego što počnete sa konstruisanjem softvera, potrebno je izvršiti određene pripreme, i ako pripreme nisu urađene ne treba početi sa konstruisanjem. Npr. arhitektonski dizajn, razvoj zahteva, i planiranje projekta. Naime, veliki rizik za neuspeh projekta je ako se izvrši loš razvoj zahteva (nepotpuni ili pogrešni zahtevi) i li loše planiranje projekta ili nepotpuno ili pogrešno definisanje arhitekture. dakle, cilj priprema za konstruisanje je upravo smanjenje rizika od neuspeha projekta.

Pre nego se počne sa konstruisanjem treba proveriti da li su pripreme kvalitetno urađene. To su:

- planiranje projekta
- razvoj kompletnih i tačnih zahteva
- kvalitetna arhitektura softvera

Ako se ne izvrše kvalitetne pripreme za konstruisanje, to se može veoma negativno odraziti na

- vreme izrade projekta
- i troškove projekta.

Npr. greške u zahtevima (ako se zahtevi ne urade kvalitetno) ako se ne otkriju u fazi razvoja zahteva mogu se otkloniti kasnije u fazi konstruisanja, ali to onda može da košta 5-10 puta više vremena nego da se ta greška otkloni u fazi razvoja zahteva. isto tako, greška u arhitektonskom dizajnu ako se ne otkrije u fazi arhitektonskog dizajna, može se otkloniti u fazi konstruisanja, ali može da košta 10 puta više vremena da bi se otklonila u fazi konstruisanja.

Princip pripreme za konstruisanje je da je potrebno naći grešku u razvoju softvera što pre, jer ako se otkrije kasnije posledice su mnogo veće od onog vremena koje je potrebno da bi se greška otklonila onda kada je uvedena. Npr. greške u konstruisanju, takođe ako se ne otkriju odmah mogu biti 10 puta skuplje da se otkriju kasnije u fazi sistemskog testiranja.

IZBOR LIFE-CYCLE MODELA

U zavisnosti od tipa projekta, može se izabrati i vrsta modela životnog ciklusa.

pre nego počnete sa konstruisanjem treba ustanoviti kojoj vrsti softvera pripada. Npr. imamo sledeće vrste softvera tj. softverskog projekta:

- razvoj internet sajta
- razvoj igara
- razvoj softverskih alata
- razvoj web-servisa
- razvoj medicinskog softvera
- itd

S druge strane, imamo niz life-cycle modela koje možemo koristiti da razvijemo softver:

- agilni razvoj (Extreme Programming, Scrum,)
- evolutivni prototajping
- vodopadni model
- spiralni razvoj
- itd

U zavisnosti od tipa projekta, može se izabrati i vrsta modela životnog ciklusa. Npr. iz iskustva se zna da razvoj intenet sajta se obično radi primenom agilne metode, ili razvoj neke igre npr. se obično radi pomoću evoltivnog prototajpinga, itd. Dakle, treba videti iz prošlosti, koji model životnog cilusa se obično primenjuje za tip projekta koji ćete raditi. Prema tome, pre početka konkretnog rada na projektu, potrebno je odrediti

- tip projekta
- i koji model životnog ciklusa se obično koristi (na osnovu iskustva prethodnih projekata) za taj tip projekta.

PROVERA POČETNE DEFINICIJE PROBLEMA

Pre početka konstruisanja treba proveriti da li je urađena kvalitetna početna definicija problema.

Prvi zadatak na projektu je početna definicija problema (problem definition), i ovo se radi pre razvoja zahteva. Pre početka konstruisanja treba proveriti da li je urađena kvalitetna početna definicija problema. Evo kako treba da izgleda početna definicija problema, a koju treba proveriti pre početka konstruisanja:

- to jednostavan tekstualni iskaz, od jedne ili dve stranice teksta
- to je iskaz koji definiše šta je problem a ne iskaz koji govori o rešenju problema
- ovaj iskaz postavlja temelj za sledeću fazu a to je razvoj zahteva
- definicija problema treba da koristi jezik korisnika softvera a ne tehnički jezik softverista

- problem treba biti opisan sa korisničke tačke gledišta

Ako nemamo dobru početnu definiciju problema, može se desiti da rešavate pogrešan problem tj. da rešavate problem koji nije potreban korisniku.

PROVERA ZAHTEVA

Zahtevi se menjaju tokom izrade projekta, i zato treba koristiti requirement checklist tokom konstruisanja.

Razvoj zahteva tj. "zahtevi" tj. "analiza zahteva" tj. "analiza" tj. "definisanje zahteva" tj. "specifikacija" tj. "softverski zahtevi" opisuju detaljno šta softverski sistem treba da radi tj. to je "funkcionalna specifikacija". Zahtevi se menjaju tokom izrade projekta, i treba ih povremeno proveravati. Eksplicitan i detaljan i kompletan skup zahteva je vrlo važan iz više razloga:

- eksplicitni i detaljni i kompletni zahtevi omogućuju da korisnik softvera može da ih pregleda i složi se sa njima i da spreči softveriste od pogađanja zahteva
- minimiziraju se greška u zahtevima
- greška u zahtevima koje se otkriju kasnije npr. tokom arhitektonskog dizajna može da na velikim projektima izazove velike štete
- specifikacija zahteva je ključna za uspeh projekta, i isto toliko koliko efikasne tehnike konstruisanja jer greške u zahtevima se provlače kroz ceo projekat i izazivaju ogromnu štetu
- važno je napomenuti da u praksi je vrlo čest slučaj da korisnik softvera ne može pouzdano da opiše zahteve na početku projekta, jer tokom projekta korisnik sve bolje razume projekat i onda dolazi do promene ili dopune zahteva
- tokom konstruisanja softvera treba redovno proveravati zahteve a ne samo na početku konstruisanja, npr. posle svakog važnijeg koraka u konstruisanju
- ako su zahtevi veoma loše definisani bolje je odustati od projekta
- proveriti zahteve sa aspekta poslovnog razloga za izradu projekta (bussiness reason for doing the project)
- treba koristiti requirement checklist tokom konstruisanja softvera, gde se proveravaju funkcionalni zahtevi, nefunkcionalni zahtevi , i kvalitet zahteva i kompletnost zahteva

PROVERA ARHITEKTONSKOG DIZAJNA

Bez dobre specifikacije arhitekture u startu, možete da imate pogrešno rešenje problema.

Arhiektura softvera tj. arhitektonski dizajn tj. high-level design tj. top-level design je pisana u dokumentu koji se zove "specifikacija arhitekture". Dobra specifikacija arhitekture puno olakšava konstruisanje softvera, dok loša specifikacija arhitekture puno otežava konstruisanje ili je čak čini nemogućom. Bez dobre specifikacije arhitekture u startu, možete da imate pogrešno rešenje problema. Greške u arhitekturi softvera koje se ispravljaju kasnije su vrlo skupe. Evo od čega se sastoji specifikacija arhitekture:

- organizacija programa tj. podela na glavne blokove programa, npr. za veće sisteme podela sistema na podsisteme, a za male sisteme podela na klase, gde svaki blok je ili klasa ili skup klasa ili skup podprograma koji rade zajedno u cilju ostvarivanja high-level funkcije
- specifikacija arhitekture treba da specificira glavne klase u programu, naime da specificira funkcionalnost glavnih klase i interakcijuglavnih klasa
- specifikacija arhitekture treba da opiše upotrebu baze podataka i fajlova
- specifikacija arhitekture treba da opiše dizajn UI tj. dizajn korisničkog interfejsa, jer arhitektura UI ima veliki uticaj na upotrebu softvera, naime, ako je loš UI onda je softver neupotrebljiv
- opis obezbeđenja bezbednosti softvera
- opis ulaza i izlaza sistema,
- opis procesiranja greški
- itd.

CENA POPRAVKE DEFEKTA U ZAVISNOSTI KADA SE DESIO DEFEKAT I KADA JE OTKRIVEN

Sumiramo šta je i zašto važno proveriti pre početka konstruisanja.

Pre početka konstruisanja treba proveriti:

- koja vrsta projekta i koji model životnog ciklusa izabrati?
- **da** li su zahtevi dovoljno dobro definisani da bi se počelo konstruisanje softvera?
- da li je arhitektura dovoljno dobro definisana da bi se počelo konstruisanje?

Dole je ilustrovana cena greške ako se ne postupi po gornjoj preporuci.

Tabela: Cena popravke defekta u zavisnosti kada se desio defekat i kada je otkriven

Momenat otkrivanja defekta

Momenat uvođenja defekta Zahtevi Arhitektura Konstruisanje Testiranje sistema

Zahtevi 1 3 5 10

Arhitektura 1 10 15

Konstruisanje 1 10

KAKO BIRAMO MODEL ŽIVOTNOG CIKLUSA

Ovde objašnjavamo kako biramo model životnog ciklusa u zavisnosti od tipa projekta.

Sekvencijalni tj. vodopadni model ili slični modeli se biraju ako je projekat okarakterisan na sledeći način:

- zahtevi se praktično ne menjaju tj. menjaju vrlo malo
- dizajn je dobro poznat i zna se skoro potpuno unapred
- tim softverista je familijaran sa takvim tipom projekta
- projekat sadrži mali rizik

Iterativni modeli se preporučuju ako je projekat sledećeg tipa:

- zahtevi nisu precizni i verovatno će se puno menjati
- dizajn je složen i neizvestan
- tim softverista nije familijaran sa takvim tipom projekta
- projekat sadrži puno rizika

VIDEO VEŽBA

Video vežba -Software Development Life Cycle Models.

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

VIDEO VEŽBA 2

Video vežba - Softverska arhitektura -What is Software Architecture?

Pogledati video vežbu: What is Software Architecture?

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

VIDEO VEŽBA 3

Video vežba 3 - Selekcija modela životnog ciklusa (Selecting a Software Life Cycle Model)

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

ZADACI ZA VEŽBU/ PITANJA ZA SAMOPROVERU ZNANJA

Dole je data lista pitanja za individualnu vežbu.

Zadaci za vežbu:

- Dati primere nekoliko različitih projekata, najmanje 3

- Za projekte iz prethodnog primera, izabrati modele životnog ciklusa.

Pitanja:

- Koliko košta kada otkrijete defekt kasno u projektu?
- Kako biste proverili funkcionalne zahteve projekta?
- Kako biste proverili nefunkcionalne zahteve?
- Kako biste proverili arhitektonski dizajn?

Poglavlje 6

Zaključak

ZAKLJUČAK

Konstruisanje softvera obuhvata sledeće aktivnosti: konstrukciono planiranje, detaljni dizajn, kodiranje, debugging, unit testing, integracija, i integraciono testiranje. Evo liste konkretnih aktivnosti (tj. podaktivnosti) kod konstruisanja softvera:

- verifikacija arhitekture i zahteva
- planiranje testiranja
- dizajn i kodiranje klasa
- dizajn i kodiranje podprograma (routines) tj. klasnih metoda
- kreiranje i imenovanje varijabli i imenovanih konstanti
- izbor kontrolne logike i organizacija blokova klasnih metoda
- testiranje jedinica
- integraciono testiranje
- debugging
- doterivanje programskog koda (formatiranje i komentiranje programskog koda), program formatting i program commenting
- integracija softverskih komponenti
- podešavanje programskog koda u cilju postizanja veće brzine i minimizacije resursa (code tuning)

LITERATURA

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

Internet linkovi:
https://en.wikipedia.org/wiki/Software_construction
https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-005-software-construction-spring-2016/
http://swebokwiki.org/Chapter_3:_Software_Construction
https://sites.google.com/site/toolsgsd/tools-1/software-construction-tools