



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.



KI301 - KONSTRUISANJE SOFTVERA

Upravljanje konstruisanjem

Lekcija 02

PRIRUČNIK ZA STUDENTE

KI301 - KONSTRUISANJE SOFTVERA

Lekcija 02

UPRAVLJANJE KONSTRUISANJEM

- ✓ Upravljanje konstruisanjem
- ✓ Poglavlje 1: Ključne odluke kod konstruisanja
- ✓ Poglavlje 2: Konstruisanje kod velikih i malih projekata
- ✓ Poglavlje 3: Upravljanje konstruisanjem
- ✓ Poglavlje 4: Upravljanje konfiguracijom
- ✓ Poglavlje 5: Vežba - Upravljanje konfiguracijom
- ✓ 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.

▼ Uvod

L2 - UPRAVLJANJE KONSTRUISANJEM

L2 - Upravljanje konstruisanjem softvera - Sadržaj:

L2 - Upravljanje konstruisanjem softvera:

- Pripreme za konstruisanje
- Ključne odluke kod konstruisanja
- Konstruisanje kod velikih i malih projekata
- Upravljanje konstruisanjem
- Upravljanje konfiguracijom
- Vežba - Upravljanje konfiguracijom

▼ Poglavlje 1

Ključne odluke kod konstruisanja

KLJUČNE ODLUKE NA POČETKU KONSTRUISANJA

Pre početka konstruisanja treba doneti neke ključne odluke koje se tiču konstruisanja, npr. izbor programskog jezika i praktičnog načina programiranja.

Posle provere zahteva i arhitekture softvera, može se pristupiti donošenju nekih važnih odluka za konstruisanje. Jedna važna odluka je izbor programskog jezika. Ima čitav niz odluka koje treba doneti pre početka konstruisanja:

- izbor jezika, npr. C C++, C-sharp, Java, JavaScript, itd.
- takođe verzija jezika i verzija kompajlera
- da li ćete koristiti standardnu ili nestandardnu verziju jezika
- izabrati programski okvir , npr. J2EE ili Microsoft.NET
- definisati da li će programeri raditi u parovima ili individualno
- izabrati pisanje test-slučajeva pre ili posle kodiranja
- izbor inspekcija programa, npr. formalne ili neformalne inspekcije
- izabrati alate za konstruisanje, npr. editor, alat za refaktorisanje, debugger, test-okvir, itd.
- izabrati integracionu proceduru
- izabrati tehniku za debugging
- itd.

OPISI JEZIKA

Evo liste jezika koji se koriste najviše u današnjoj praksi.

Evo liste jezika, kao i kratak opis programskih jezika:

- Assembly language-tj "assembler", je jezik niskog nivoa apstrakcije gde jedna instrukcija u programu odgovara jednoj instrukciji mašinskog jezika
- C--jezik za opštu namenu, jezik srednjeg nivoa jer ima neke osobine jezika visoko nivoa i neke osobine jezika niskog nivoa, razvijen 1970. ih g.
- C++ je o.o. jezik, razvijen 1980.ih godina, ima vrlo moćnu standardnu biblioteku
- C# je o.o. jezik opšte namene, sa o.o. okruženjem i moćnim alatima za razvoj na Microsoft platformi
- Cobol je razvijen 1960.g., liči puno na engleski jezik, i koristi se pre svega za poslovne aplikacije

- Fortran je prvi jezik visokog nivoa, razvijen 1950.ih godina, a u FORTRAN 90 su dodate klase, pointers, i ovaj jezik se koristi uglavnom za inženjerske aplikacije
- Java je o.o. jezik, koji može da se koristi na svakoj platformi koristeći okruženje koje se zove Java Virtual Machine, i Java se puno koristi za Web aplikacije
- JavaScript je skriptni jezik koji se koristi za programiranje klijenske strane za Web-strane, i ovaj jezik ima vrlo malo veze sa Javom
- Perl je skriptni **string-handlin** jezik, koji se često koristi za administriranje kompjuterskih sistema, za izradu "skripti"
- PHP je **open-source** skriptni jezik
- Python je skriptni o.o. jezik, koristi se za pisanje "skripti" i za malih Web-aplikacija
- SQL je standardni jezik za upotrebu baza podataka (pretraga, ažuriranje, upravljanje)
- VisualBasic je **high-level** i o.o. jezik koja koristi vizuelno programiranje
- Ada-jezik visokog nivoa, opšte namene, baziran na Pascal-jeziku

BITNI FAKTI ZA DONOŠENJE KLJUČNIH ODLUKA

Evo nekih bitnih fakata koji se tiču ključnih odluka koje treba doneti pre konstruisanja.

Bitni fakti:

- Svaki programski jezik ima jake i slabe strane, i kod izbora jezika voditi o ovome računa
- Izabrati programerske konvencije, npr. imenovanje varijabli, konstanti, za komentarisanje i za formatiranje, itd, jer je teško posle menjati program da bi se primenile jezičke konvencije
- Za svaki projekat treba izabrati odgovarajući skup ključnih odluka
- Ključne odluke obuhvataju praksu kod kodiranja, praksu kod timskog rada, praksu kod obezbeđenja kvaliteta i praksu kod izbora alata
- Praksa kod kodiranja obuhvata: kodirajuće konvencije, tretiranje greški, bezbednost softvera, softvera, performanse softvera, itd.
- Praksa timskog rada obuhvata: integracionu proceduru, parno ili individualno programiranje ili kombinacija,
- Pitanja prakse kvaliteta softvera: pisanje test-slučajeva pre ili posle kodiranja, upotreba debugger-a, inspekcija programa od strane drugih programera, itd
- Pitanja alata: izbor **version-control** alata, izbor jezika i verzije jezika i vezije kompajlera, izbor okvira (npr. J2EE ili Microsoft .NET), da li će se koristiti nestandardna verzija jezika, pitanje sledećih alata: editor, alat za refaktorisanje, **debugger**, okvir za testiranje, itd.

VIDEO VEŽBA

Video vežba - Kako Java program funkcioniše (How Java Program Works, Compiler, Interpreter)

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

VIDEO VEŽBA 2

Video vežba - Lista programskih jezika i njihovih upotreba (List of widely used programming languages and their uses)

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

VIDEO VEŽBA 3

Video vežba 3 - Tipovi Java aplikacija (Where It Is Used And Types of Java Application)

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

ZADACI ZA VEŽBU / PITANJA ZA SAMOPROVERU ZNANJA

Ovde je lista pitanja za individualnu vežbu.

Zadaci:

- Dati primere 2 aplikacije
- Za aplikacije iz prethodnog zadatka izvršiti donošenje ključnih odluka za konstruisanje.

Pitanja:

- Detaljno opisati odluke koje se tiču kodiranja a koje treba doneti pre konstruisanja?
- Opisati odluke koje treba doneti pre konstruisanja a tiču se timskog rada programera?
- Objasniti koje odluke o obezbeđenju kvaliteta softvera treba doneti pre konstruisanja?
- Objasniti koje alate treba izabrati pre početka konstruisanja?

▼ Poglavlje 2

Konstruisanje kod velikih i malih projekata

VELIČINA PROJEKTA

Ovde razmatramo kako veličina projekta utiče na konstruisanje softvera?

Može se postaviti pitanje ,kako veličina projekta utiče na konstruisanje softvera (How program size affects construction?)?

Ako ste navikli da radite na malim projektima, vaš prvi srednje-veliki ili veoma veliki projekat može lako da izmakne kontroli, i da postane noćna mora umesto zadovoljstva u napredovanju. Ovde se diskutuje šta možete očekivati da se desi na većim projektima. S druge strane, ako ste navikli na velike projekte, vaš prilaz izradi softvera treba prilagoditi malim projektima, naime vaš prilaz je možda suviše formalan i obiman za male projekte, gde se neke stvari mogu pojednostaviti ili preskočiti kao nepotrebne.

Povećavanje ili proširenje nekog softvera nije jednostavno povećavanje svakog njegovog dela. Npr ako ste napisali neki program tj programski paket od 25.000 linija, upotrebivši 20 inž.-meseci, uz 500 pronadjenih greški kod *field-testing*, i onda treba da razvijete puno poboljšanu verziju tog programa, za koji se očekuje da će imati 250.000 linija programskog koda. I ako je ovaj prošireni program 10 puta veći od originalne verzije, on će zahtevati 30 puta više inž.-meseci, što se tiče greški, on će možda proizvesti 15 puta više greški.

KOMUNIKACIONE VEZE NA PROJEKTU

Ovde analiziramo komunikacione veze na projektu.

Ako ste vi jedina osoba na projektu, jedina komunikacija putanja je izmedju vas i korisnika. Kako se broj ljudi na projektu uvećava, broj komunikacionih putanja se povećava. Medjutim, broj komunikacionih putanja se ne povećava linearno, već skoro kvadratno, sa brojem ljudi što je ilustrovano dole. Za, dvoje ljudi 1 komunikaciona putanja, za 3 ljudi 3 putanje, za 4 ljudi 6 putanja, za 5 ljudi 10 putanja, a za 10 programera 45 putanja. Pod pretpostavkom da svaki programer komunicira sa svakim. Što je više komunikacionih putanja, to je više vremena potrebno i više mogućnosti za komunikacione greške. U cilju smanjenja greški, ove komunikacije se formalizuju, i umesto razgovora zahteva se da se pišu i čitaju dokumenti.



Slika 2.1 Komunikacione veze na projektu

TIMOVI PROGRAMERA

Jedan način merenja veličine projekta je veličina tima programera

Postavlja se pitanje šta je tipična veličina projekta. Jedan način merenja veličine projekta je veličina tima programera. Dole je data gruba procena procenata različitih veličina projekta urađenih od strane različitih timova programera.

Tabela

Veličina tima ----- Procentualni broj projekata

1-3 programera ----- 25%

4-10 ----- 30%

11-25 ----- 20%

26-50 ----- 15%

50+ ----- 10%

VELIČINA PROJEKTA-PRIMERI

Često se postavlja pitanje, šta je tipična veličina projekta i kako se meri veličina projekta?

Pošto veliki projekti angažuju veliki broj programera, oni zapošljavaju veliki broj programera. Dole je to ilustrovano tabelom, gde je dat broj programera u procentima u zavisnosti od veličine projekta.

Tabela

Veličina tima ----- Procentualni broj programera

1-10 programera ----- 15%

11-25 ----- 15%

26-50 ----- 20%

50+ ----- 50%

DEFEKTI-PROCENTI

Procenat greški u zahtevima, dizajnu, i konstruisanju se menja u zavisnosti od veličine projekta.

Procenat greški u zahtevima, dizajnu, i konstruisanju se menja u zavisnosti od veličine projekta. Kod malih projekata, greške u konstruisanju čine 75% od svih greški. Medjutim, na velikim projektima, greške u konstruisanju čine 50% od svih greški, a ostatak čine greške u zahtevima i arhitektonskom dizajnu. Opet, na nekim veoma velikim projektima, npr 500.00 linija programskog koda, greške u konstruisanju opet mogu da čine do 75% od svih greški. Broj greški tj. defekata takodje zavisi od veličine projekta. Broj greški se ne povećava linearno sa veličinom projekta, tj. gustina greški- broj greški po 1000 linija programskog koda- se povećava sa veličinom projekta. To je ilustrovano u donjoj tabeli. Vidi se da se broj greški dramatično povećava sa veličinom projekta, i na vrlo velikim projektima 4 puta veći broj u odnosu na male projekte.

Tabela

Veličina projekta -----Tipična gustina greški

(u linijama koda) ----- (greški po 1000 linija prog.koda)

<2k ----- <25 (greški/KLOC)

2k-16k ----- <40

16k-64k ----- <50

64k-512k ----- <70

>512k ----- <100

PRODUKTIVNOST PROJEKTA

Produktivnost na malim projektima može biti 2-3 puta veća od one na velikim projektima.

Ako se radi o projektu sa samo jednim programerom, onda uticaj na uspeh ili neuspeh je od samo jedne osobe. Medjutim, ako ima 25 ljudi na projektu, onda organizacija projekta ima veliki uticaj na uspeh ili neuspeh projekta. Kod velikih projekata, postoji potreba za formalnom komunikacijom medju članovima tima, i vrsta aktivnosti koje su potrebne se dramatično menja. Naime, kod malih projekata, konstruisanje uzima 65% vremena, na srednjim 50% pripada konstruisanju, medjutim, kod velikih projekata, arhitektura, integracija, sistemsko testiranje su obimniji, pa konstruisanje postaje manje od 50%.

Produktivnost na malim projektima može biti 2-3 puta veća od one na velikim projektima, i može se reći da produktivnost može biti 5-10 puta veća ako se porede najmanji najveći projekti. Npr.

- za projekat od 1k (1000 linija programskog koda), produktivnost može da bude od 2.500 do 25.000 linij programskog koda po programer-godini,
- a za projekat od 1.000k, ovo može da bude od 700 do 10.000, u zavisnosti od vrste softvera,
- a za projekat od 10.000k, ovo može da bude od 300 do 5.000, u zavisnosti od vrste softvera,
- ali za istu vrstu softvera, sa povećanjem veličine projekta se smanjuje produktivnost.

Produktivnost dramatično može da se menja u zavisnosti od:

- vrsta softvera
- ličnih kvaliteta programera
- programskog jezika
- primenjene metodologije
- programskog okruženja
- upotrebe programerskih alata i drugih alata

PROPORCIJA RAZVOJNIH AKTIVNOSTI KOD VELIKH I MALIH PROJEKATA

Na malim projektima, konstruisanje je dominantna aktivnost, dok na većim projektima proporcije za ostale aktivnosti (arhitektonski dizajn, integracija, sistemsko testiranje) se povećavaju.

Kod velikih projekata, potreba za formalnom komunikacijom umesto neformalne komunikacije je neophodna. Npr. ako radite na projektu gde radi samo jedan čovek ili na projektu gde radi 25 ljudi, menjaju se proporcije pojedinih aktivnosti, npr. konstruisanje na malim projektima može biti npr. 70% a na velikim 30%. Pogledajmo sledeću tabelu:

Mali projekat Veliki projekat

Sistemsko testiranje 10% 20%

Integracija 10% 20%

Konstruisanje 70% 40%

Arhitektura 10% 20%

gde konstruisanje obuhvata:

- detaljni dizajn
- kodiranje
- debugging
- razvojno testiranje

Dakle, na malim projektima, konstruisanje je dominantna aktivnost, dok na većim projektima proporcije za ostale aktivnosti (arhitektonski dizajn, integracija, sistemsko testiranje) se povećavaju. U gornjoj tabeli, razvoj zahteva nije prikazan, jer razvoj zahteva se ne menja proporcionalno puno sa veličinom projekta.

LISTA AKTIVNOSTI ČIJE PROPORCIJE RASTU

Čitav niz aktivnosti na velikim projektima zahtevaju značajno vreme, pa se udeo konstruisanja smanjuje.

Evo liste aktivnosti na projektima čija proporcija rastu više nego linearno sa veličinom projekta:

- komunikacije
- planiranje
- upravljanje
- razvoj zahteva
- dizajn interfejsa
- arhitektonski dizajn
- integracija
- otklanjanje defekata
- sistemsko testiranje
- proizvodnja dokumentacije

VRSTE PROJEKATA

Evo liste razloga za izradu softverskih projekata.

Mogu se navesti sledeći razlozi za izradu nekog softverskog razvojnog projekta:

Da se reši neki novi problem (**problem-driven**) ,Da se neki postojeći softver unapredi (**change-driven**)

Da se dovrši neki prethodni plan i projekat (**legacy-driven**) ,

Projekat može biti uradjen: Samo za jednog korisnika , Za tržište za niz korisnika , Za društvenu upotrebu (npr. **open-source softver**)

Kod razvoja novog softvera važno je proučiti postojeći softver, jer po pravilu uvek postoji u ovoj ili onoj formi neko postojeće rešenje. Ako se prouči dobro postojeće rešenje, izbeći će se greške koje već postoje u postojećem softveru. Takodje, treba iskoristiti postojeće komponente, jer ovo može znatno uštedeti troškove.

Projekti mogu biti vrlo raznovrsni:

Kratki projekti (par nedelja ili mesec ili dva meseca) ,Dugi projekti (godinu ili dve)

Potpuno nepoznati tip projekta (nema postojećih tj. urađenih sličnih projekata), Poznat tip projekta (postoje urađeni slični projekti)

REZIME O KONSTRUISANJU KOD VELIKIH I MALIH PROJEKATA

Evo nekih važnih zapažanja.

Bitna zapažanja:

- Kako veličina projekta raste, komunikacija na projektu treba da se podrži na neki način i organizuje, u cilju smanjenja komunikacionih problema
- Produktivnost na velikom projektu je manja nego na malom projektu
- Na velikim projektima se povećava broj grešaka po hiljadu linija programa
- Kod velikih projekata potrebno je izvršiti pažljivo planiranje projektnih aktivnosti
- Nezavisno od veličine projekta treba primeniti sledeće tehnike: disciplinovana praksa kodiranja (komentarisanje, formatiranje, itd), inspekcija dizajna i koda od strane drugih softverista, upotreba dobrih alata, upotreba jezika visokog nivoa , naime ove tehnike su korisne na malim projektima, a na velikim su od izuzetne važnosti
- Kako veličina projekta raste vreme potrošeno na konstruisanje raste približno linearno, ali druge aktivnosti rastu brže nego linearno, pa udeo konstruisanja opada na većim projektima u odnosu na manje

VIDEO VEŽBA

Video vežba -Tips To Managing Huge Projects

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

VIDEO VEŽBA 2

Video vežba 2- Introduction to Scrum - 7 Minutes

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

PITANJA ZA SAMOPROVERU ZNANJA

Lista pitanja o konstruisanju velikih i malih projekata.

Pitanja:

- Koji su komunikacioni problemi na velikim projektima?
- Uporediti produktivnost na velikim i malim projektima?
- Uporediti defekte na malim i na velikim projektima?
- Nabrojati aktivnosti na projektu koje rastu brže nego linearno?
- Šta je to disciplinovana praksa kodiranja?

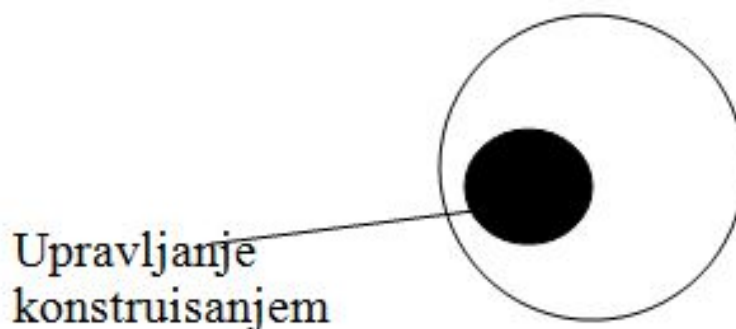
▼ Poglavlje 3

Upravljanje konstruisanjem

UPRAVLJANJE KONSTRUISANJEM-SLIKA

Upravljanje konstruisanjem je deo upravljanja razvojem softvera, a koje je deo opšteg upravljanja softverskom kompanijom

U toku zadnjih nekoliko decenija, upravljanje softverskim razvojem, **software development**, je bilo puno izazova i teškoća. Na donjoj slici prikazano je upravljanje razvojem softvera kao deo opšteg upravljanja a opet gde je upravljanje konstruisanjem deo upravljanja razvoja softvera. Upravljanje razvoja softvera je interakcija izmedju menadžera i softverista.



Slika 3.1 Upravljanje konstruisanjem

TEHNIKE MOTIVACIJE

U praksi je vrlo važno pitanje: kako motivisati kvalitetno kodiranje?

Programski kod je glavni rezultat kod konstruisanja softvera, pa se postavlja pitanje kako motivisati kvalitetno kodiranje, tj. kvalitetno pisanje programa? Postoje sledeće tehnike motivacije za to

Na svakom delu projekta angažovati 2 programera

Pregledati svaku liniju programskog koda, gde dva 'recenzenta' pregledaju programski kod

Overiti programski kod od strane vodećeg inženjera

Nagrade za dobar programski kod

Itd

FAKTORI SLOŽENOSTI PROJEKTA

Ovde ćemo kratko pomenuti probleme kod procene veličine i planiranja softverskih projekata.

Evo liste faktora koji utiču na vreme izrade projekta:

- Veličina baze podataka
- složenost projekta
- sposobnost programera
- sposobnost analitičara zahteva
- kohezija tima
- iskustvo tima u određenoj oblasti
- upotreba softverskih alata
- ponovna upotreba postojećih programa
- količina dokumentacije
- itd

PROCENE VELIČINE PROJEKTA

Veličina softverskog projekta i zahtevani rad tse može proceniti na više načina?

Upravljanje softverskim projektima, software project management, je jedan od značajnih izazova 21-og veka, i u tom kontekstu procena veličine projekta, tj napora potrebnog da se uradi, je najteži aspekt upravljanja softverskim projektom. Npr. tipični softverski projekt je sa jednogodišnjim zakašnjenjem. I 100% preko budžeta. Ovde ćemo kratko pomenuti probleme kod procene veličine i planiranja softverskih projekata.

Veličina softverskog projekta i zahtevani rad tj se može proceniti na sledeće načine:

- Pomoću softvera za procene ,Upotrebe algoritamskog prilaza kao npr COCOMO II
- Angažovanje spoljnjih eksperata za procenu, Pomoću detaljnog sastanka za procenu
- Procena pojedinih delova, i zatim sabiranjem ovih delova
- Procenom od strane programera njihovih sopstvenih zadataka, i potom njihovimsabiranjem
- Oslanjanjem na iskustvo na prethodnim projektima
- Praćenjem procena u toku izrade projekta, i njihovim adaptiranjem da bi se napravile aktualizovane procene

Što se tiče procene samog konstruisanja, construction schedule, i njegovog uticaja na plan izrade, tj na project's schedule, to zavisi koliki deo projekta čini konstruisanje, tj. detaljni dizajn,.kodiranje, debugging, i testiranje jedinica, *unit testing*. Naime, ako je projekat veći proporcija konstruisanja se spušta. Medjutim procenat konstruisanja varira od projekta do projekta i od organizacije do organizacije, zato treba koristiti podatke sopstvene organizacije.

LISTA SOFTVERSKIH MERENJA

Softverski projekti se mogu meriti na brojne načine.

Softverski projekti se mogu meriti na brojne načine. To je potrebno jer, bolje je meriti projekte uprkos nepreciznosti softverskih merenja nego ih ne meriti uopšte, zatim, softverska merenja imaju motivacioni efekt na programere, i ima se informacija o tome šta se ešava sa izradom projekta. Dole je data lista softverskih merenja koja su se pokazala kao korisna za proces razvoj softvera:

- size
- defect tracking
- productivity
- overall quality
- maintainability

MERENJE VELIČINE PROJEKTA

Postoji više načina da se izmeri veličina projekta.

Mere veličine projekta, project size

Napisan broj linija programskog koda

Broj linija komentara

Broj klasa i podprograma

Broj deklaracija

Broj praznih linija

itd

MERENJE PRODUKTIVNOSTI, KVALITETA

Ovde razmatramo merenja produktivnosti, opšteg kvaliteta, i održavanja softvera.

Produktivnost, productivity

Broj radnih sati provedenih na projektu

Broj radnih sati provedenih na svakoj klasi ili podprogramu

Ukupni troškovi projekta

Troškovi po liniji koda

Troškovi po defektu

itd

Opšti kvalitet, overall quality

Totalni broj defekata

Broj defekata u svakoj klasi i podprogramu

Prosečan broj defekata po hiljadi linija koda

itd

MERENJE DEFEKATA I MOGUĆNOSTI ODRŽAVANJA

Merenje defekata je vrlo važan deo softverskih merenja.

Praćenje defekata, defect tracking

Ozbiljnost svakog defekta

Lokacija svakog defekta

Ličnost odgovorna za defekt

Prosečno vreme potrebno da se popravi defet

itd

Sposobnost održavanja, maintainability

Broj javnih podprograma za svaku klasu

Broj privatnih podprograma za svaku klasu

Broj linija koda za svaku klasi i podprogram

Broj linija komentara za svaku klasu i podprogram

Itd

RELEVANTNI STANDARDI

Evo liste relevantnih standarda za upravljanje konstruisanjem (managing construction).

Evo liste relevantnih standarda:

- IEEE Std 1058-1998, Standard for Software project management plans
- IEEE Std 12207-1997, Information technology-Software life cycle processes
- IEEE Std 828-1998, Standard for software configuration management plans
- IEEE Std 1045-1992, Standard for software productivity metrics

POKAZNA VEŽBA-LINKOVI

Pogledajmo korisne linkove koji se tiču upravljanja i planiranja projekta.

Pogledati linkove:

- <https://www.capterra.com/project-management-software/#infographic>
- https://en.wikipedia.org/wiki/Software_project_management
- https://www.tutorialspoint.com/software_engineering/software_project_management.htm
- https://en.wikiversity.org/wiki/Software_metrics_and_measurement#GQM_Steps

REZIME O UPRAVLJANJU KONSTRUISANJEM.

Lista glavnih zaključaka o upravljanju konstruisanjem.

Rezime:

Dobra procena vremena/napora za konstruisanje softvera je veliki izazov ponekad.

Procena vremena/napora za konstruisanje softvera je deo planiranja potreban da bi se projekat završio na vreme.

Ako ustanovite datum isporuke projekta, onda glavni problem ostaje potrošnja ljudskih i tehničkih resursa u cilju isporuke na vreme.

Dobra procena vremena i napora potrebnog za izradu projekta se obezbeđuje upotrebom odgovarajućih metodologija, ažuriranja ovih procena u toku projekta, i upotrebom relevantnih podataka.

Merenja su ključni faktor za upravljanje konstruisanjem?

Postoje načini da se mere razni aspekti projekta, koji se često približni ali ipak vrlo korisni.

Tačno merenje je ključni faktor za tačno planiranje projekta, kontrolu kvaliteta i poboljšanje razvojnog procesa.

VIDEO VEŽBA

Video vežba - Project Management Tutorial: Introduction to Project Management.

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

PITANJA ZA SAMOPROVERU ZNANJA

Pitanja o upravljanju projektom.

Pitanja:

- Opisati i uporediti tehnike motivacije programera?
- Koji faktori utiču na vreme izrade projekta?
- Nabrojati softverska merenja bitna za upravljanje konstruisanjem?
- Kako se meri produktivnost?
- Kako se mere defekti?

▼ Poglavlje 4

Upravljanje konfiguracijom

UPRAVLJANJE KONFIGURACIJOM (CONFIGURATION MANAGEMENT)

Za potrebe upravljanja promenama u programskom kodu (software code changes) se koristi tzv. "version-control software".

Pod upravljanjem konfiguracijom (Configuration management) se podrazumeva :

- identifikacija artifakta (dokumentacije) projekta
- upravljanje promenama u programu, u dizajnu, u zahtevima

Dakle, softverski projekat je izložen promenama/modifikacijama dokumenata tokom svog životnog ciklusa, jer se programski kod menja, dizajn se menja, i zahtevi se menjaju (requirements and design changes) tokom izrade projekta i tokom celog životnog ciklusa. Takođe menjaju se verzije alata (tool versions) koji se koriste, i verzije mašina koje se koriste (machine configurations). Za potrebe upravljanja promenama u programskom kodu (software code changes) se koristi tzv. "version-control software".

TEHNIKA UPRAVLJANJA KONFIGURACIJOM

Ovde dajemo definiciju šta je to upravljanje konfiguracijom.

"Upravljanje konfiguracijom" tj. configuration management je praksa identifikacije artifakata projekta i sistematsko praćenje promena tako da sistem očuva verodostojnost. Drugi naziv je : "upravljanje promenama", i obuhvata analizu promena i praćenje promena i memorisanje kopija sistema u različitim fazama projekta.

Potrebno je pratiti tj. upravljati

- promenama zahteva
- promenama programskog koda
- promenama dizajna

Što se tiče promene programskog koda:

- kod promene programa npr. usled eliminacije grešaka, mogu se pojaviti nove greške, i može se pojaviti potreba za upoređenjem stare i nove verzije programa
- ako imate alat za version-control, koji prati i memoriše razne verzije izvornog koda programa

Version-control softver je vrlo koristan alat na timskim tj. većim projektima. Ovaj alat beleži sve promene fajlova u programu, datum i ime osobe koja je menjala program. Ovaj softver omogućuje puno koristi:

- na timskim projektima to je alat bez koga se ne može
- omogućuje praćenje defekata tj. **defect tracking**

VIDEO VEŽBA

Video vežba - What is Configuration Management?

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

PITANJA ZA SAMOPROVERU ZNANJA

Pitanja iz upravljanja konfiguracijom.

Pitanja:

- Nabrojati alate za upravljanje konfiguracijom softvera?
- Opisati promene zahteva u toku procesa razvoja softvera?
- Opisati promene dizajna u toku razvoja softvera?
- Opisati promene programskog koda u toku razvoja softvera?
- Nabrojati koja je korist od version-control softvera?

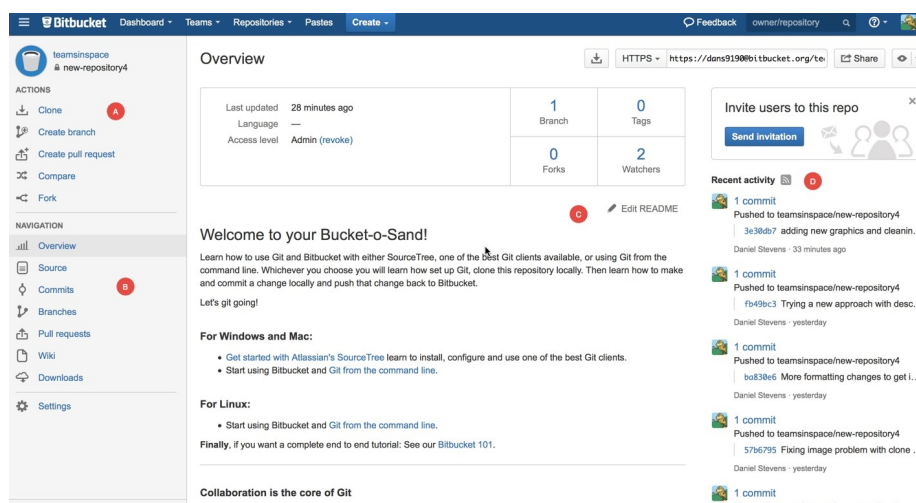
▼ Poglavlje 5

Vežba - Upravljanje konfiguracijom

POKAZNA VEŽBA - PRIMER VERSION-CONTROL ALATA BITBUCKET


Bitbucket je konstruisan za veb hosting usluge (tj. za skladištenje koda samog projekta na kome radi razvojni tim).

Korišćenje Bitbucketa se naplaćuje ukoliko na projektu radi više od pet članova, dok je za manje timove on besplatan. Postoji mogućnost pravljenja velikog broja privatnih repozitorijuma na koje je moguće postaviti izvorni kod aplikacija, kome će moći da pristupi svaki član tima u bilo kom trenutku. Nakon rada na samom kodu i prilikom izmene koda, korisnik pomoću određenih komandi šalje (izvršava commit na repozitorijum) i upisuje izmenu na Bitbucket.



Slika 5.1 Korisnički interfejs Bitbucket alata

Na početku u Bitbucket-u će biti samo izvorni kod (početni kod) dok će se nakon određenog vremena taj kod menjati i biće izvršen veliki broj ubacivanja izmena koda (commit-ova). Nakon svake izmene koda, potrebno je da svi članovi tima provere verziju svog koda na lokalnoj mašini i preuzmu najnoviju verziju koda tako da bi na njoj nastavili rad. Preporuka je da se često commit-uje jer je jednostavnije pronaći grešku u malim izmenama (izmenjeno ispod 100 linija koda).



Author	Commit	Message
ide user ide_ger...	d95b757	add the update for real now
ide user ide_ger...	4392578	making the nodejs build system work with the new system
ide user ide_ger...	61d25e7	Merge branch 'dev' into nodejsbuildsystem
ide user ide_ger...	6894b05	remove _FILES postfixes
ide user ide_ger...	71e9c82	updated the build system to make use of a json file
mrdoob	64b08f4	Merge branch 'master' into dev
phenomnomno...	8090766	Incorrectly named properties in docs.
mrdoob	be752db	GUI: Prettier geometry exporter output.
mrdoob	053e6be	GUI: Some more geometry exporter code.
mrdoob	cd08935	GUI: Geometry exporter starting to work :)
mrdoob	42b7b31	GUI: Exporting vertices by now. Fun hack for downloading "geometry.js" file ^{AA}
mrdoob	f9a8726	GUI: Starting geometry exporter.
mrdoob	60ee5d2	GUI: Modularised properties panel.
mrdoob	8e4c1a9	Merge branch 'master' into dev
mrdoob	29eb9bd	Docs: Fixed color parameter desc in LineBasicMaterial. Fixes #1990.

Slika 5.2 Primer izmena na kodu (Autor, id commita i komentar u kome se nalazi opisana izmena)

TUTORIJAL ZA BITBUCKET

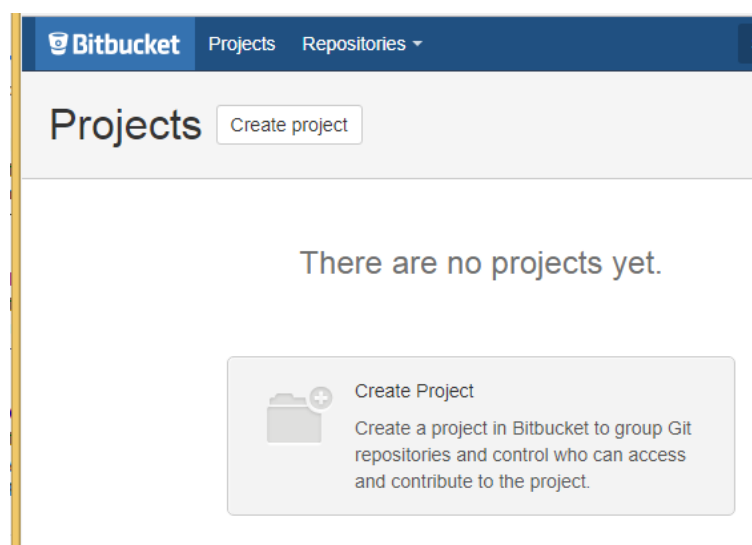
Na linku <http://www.smlcodes.com/devops/bitbucket/bitbucket-tutorial/> može se naći detaljan opis opcija koje pruža Bitbucket.

Na linku

<http://www.smlcodes.com/devops/bitbucket/bitbucket-tutorial/>

može se naći detaljan opis opcija koje pruža Bitbucket, kao i tutorial za Bitbucket gde se opisuje:

- kreiranje projekta pomoću BitBucket alata
- kreiranja novog repozitorijuma pomoću BitBucket alata
- kloniranja repozitorijuma



Slika 5.3 - Kreiranje projekta

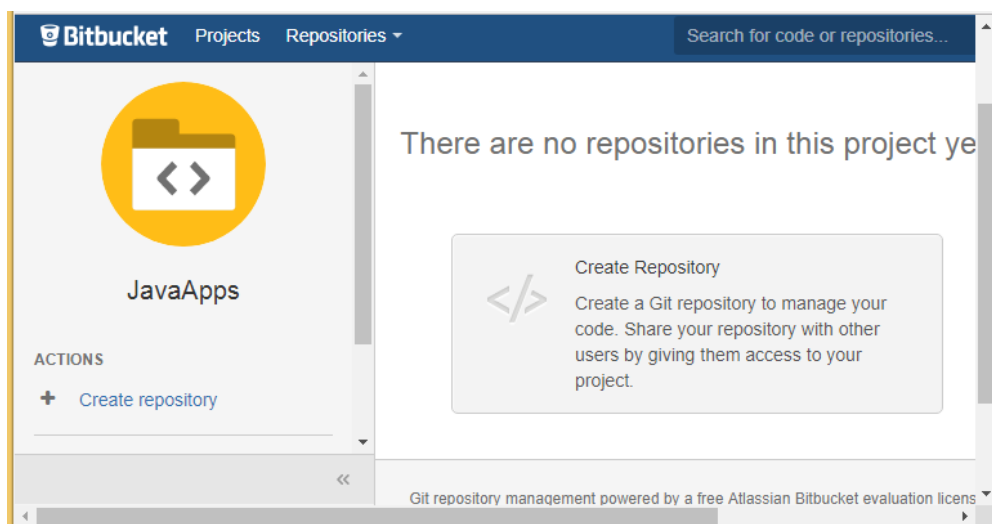
RAD SA BITBUCKET

Ovde ilustrujemo rad sa BitBucket alatom.

Rad sa BitBucket:

Za rad sa bilo kojim Source-control alatom, treba da znamo kako da obavimo sledeće operacije:

- kreiranje repozitorijuma ([Creating Repository](#)),
- dodavanje fajlova ([adding files into repo](#))
- kako da vršimo izmene ([How to commit the changes](#)).



Slika 5.4 - Kreiranje repozitorijuma

VIDEO TUTORIJAL ZA BITBUCKET

Tutorijal za one koji nisu do sada koristili Bitbucket.

Pogledati donji video. To je video tutorijal koji za početnike objašnjava upotrebu alata Bitbucket.

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

INTEGRACIJA BITBUCKET I ECLIPSE

Familijarizacija sa alatom Bitbucket.

Pogledati donji video, gde se objašnjava korak po korak kako se integriše Bitbucket alat i repozitorijum sa Eclipse projektom. U cilju familijarizacije sa alatom Bitbucket.

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

VERSION-CONTROL SOFTVER ZA JAVU: KORISNI LINKOVI

Evo linkvi za upoznavanje version-control softvera za Javu (Source control software).

Evo linkovi za upoznavanje version-control softvera za Javu:

- <http://www.javapractices.com/topic/TopicAction.do?Id=241>
- https://en.wikipedia.org/wiki/Comparison_of_version_control_software
- <https://coderanch.com/t/677176/Deciding-Java-source-control-software>
- https://en.wikipedia.org/wiki/Software_configuration_management
- <https://www.upguard.com/articles/the-7-configuration-management-tools-you-need-to-know>
- <https://energy.gov/cio/downloads/software-configuration-management-scm-practical-guide>

ZADACI ZA SAMOSTALAN RAD

Evo zadataka za individualnu vežbu.

Zadaci:

- Izabrati jedan **version-control** alat za Javu, i upoznati se detaljno sa njim?
- Opisati prednosti i mane alata iz prethodnog zadatka, i uporediti sa alatom Bitbucket.
- Demonstrirati korišćenje **version-control** alata?

DOMAĆI ZADATAK 2

Drugi domaći zadatak se dobija direktno od asistenta ili na poseban zahtev studenta.

Drugi domaći zadatak je individualan za svakog studenta i dobija se po definisanim instrukcijama.

Domaći zadatak se imenuje:

KI301-DZ02-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-DZ02**.

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

Programski kod je glavni rezultat kod konstruisanja softvera, pa se postavlja pitanje kako motivisati kvalitetno kodiranje, tj. kvalitetno pisanje programa? Postoje sledeće tehnike motivacije za to :

- Na svakom delu projekta angažovati 2 programera
- Pregledati svaku liniju programskog koda, gde dva 'recenzenta' pregledaju programski kod
- Overiti programski kod od strane vodećeg inženjera
- Nagrade za dobar programski kod
- Itd

Softverski projekti se mogu meriti na brojne načine. To je potrebno jer, bolje je meriti projekte uprkos nepreciznosti softverskih merenja nego ih ne meriti uopšte, zatim, softverska merenja imaju motivacioni efekt na programere, i ima se informacija o tome šta se ešava sa izradom projekta. Dole je data lista softverskih merenja koja su se pokazala kao korisna za proces razvoj softvera:

- veličina, praćenje defekata , produktivnost, kvalitet, mogućnost održavanja, itd.

LITERATURA

<http://www.javapractices.com/topic/TopicAction.do?Id=241>

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

<https://coderanch.com/t/677176/Deciding-Java-source-control-software>

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

<https://www.upguard.com/articles/the-7-configuration-management-tools-you-need-to-know>

<https://energy.gov/cio/downloads/software-configuration-management-scm-practical-guide>

<https://www.capterra.com/project-management-software/#infographic>

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

https://www.tutorialspoint.com/software_engineering/software_project_management.htm

https://en.wikiversity.org/wiki/Software_metrics_and_measurement#GQM_Steps