SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

SEMINAR

Kontinuirana isporuka aplikacija za mobilne operacijske sustave

Ivan Rep
Voditelj: *izv. prof. dr. sc. Borisu Vrdoljaku*

Zagreb, travanj 2016.

SADRŽAJ

1. Uvod TODO drugi dio

Kontinuirana integracija (engl. *Continuous Integration (CI)*) je, u sklopu softverskog inženjerstva, pristup razvoju programske potpore u kome se radne kopije integriraju s glavnom kopijom nekoliko puta dnevno. Rijetko integriranje promjena dovodi do spajanja velike količine različitog koda i pojave integracijskih pogrešaka koje je prije spajanja potrebno otkloniti. Broj integracijskih pogrešaka može postati toliko velik da je vrijeme potrebno za njihovo ispravljanje duže od vremena utrošenog na razvoj. Kontinuirana integracija otklanja ovaj problem i značajno olakšava proces razvoja programske podrške[5].

Iako je praksa kontinuirane integracije široko prihvaćena na tržištu, ne postoji standard njezine implementacije. Tako se implementacije kontinuirane integracije oslanjaju na listu najboljih praksi koje su se istakle kroz godine korištenja. Cilj ovog rada je, istraživanjem stvarnog korištenja kontinuirane integracije, prepoznati navedene najbolje prakse, kategorizirati ih i prema njima pokušati pronaći najbolju implementaciju kontinuirane integracije. Sustav će biti optimiziran za razvoja programske potpore za mobilne operacijske sustave. Konačan će sustav uz minimalan trud omogućiti korištenje modernog i potpunog sustava kontinuirane integracije.

Uz kontinuiranu integraciju se često spominju i dvije novije prakse. Kontinuirana dostava i kontinuirana isporuka.

Kontinuirana dostava (engl. *continuous delivery*) je, u sklopu programskog inženjerstva, pristup razvoju programske potpore koji nastoji smanjiti vrijeme koje je potrebno za prebacivanje promjena u produkciju. Timovi proizvode programsku potporu u kratkim ciklusima, istovremeno osiguravajući ispravnost glavne kopije u bilo kom trenutku. Programska potpora se gradi i unaprjeđuje inkrementalno čime se nastoji smanjiti trošak, trajanje i rizik razvoja istovremeno povećavajući kvalitetu istog[4].

Kontinuirana isporuka (engl. *continuous deployment*) nadalje svaku ispravnu izmjenu glavne kopije automatski isporučuje korisnicima. Korisnici sustava su podijeljeni u nekoliko razina s obzirom na njihovu funkciju. Promjene se označavaju oznakama koje određuju koja će skupina korisnika vidjeti promjenu[1].

Kontinuirana dostava i isporuka nisu toliko široko prihvaćene prakse kao kontinuirana integracija. Međutim, brojni značajni aktori u svijetu razvoja programske potpore zagovaraju i ističu brojne pozitivne strane navedenih praksi. Zbog toga će drugi dio rada proučiti navedene prakse te ih implementirati u prije ostvarenom sustavu.

2. Kontinuirana integracija

Kontinuirana integracija je praksa u programskom inženjerstvu spajanja svih radnih kopija koda s glavnom kopijom nekoliko puta dnevno. Termin je prvi put predložio i iskoristio Grady Booch 1991. godine tijekom opisa metode danas poznate kao Boochova metoda (engl. *Booch method*)[3]. Kasnije praksu nasljeđuje ekstremno programiranje (engl. *eXtreme Programming (XP)*) koje dodatno potiče praksu integracije nekoliko puta dnevno.

Cilj kontinuirane integracije je izbjegavanje problema poznatog pod popularnim nazivom "pakao integracije" (engl. *integration hell*). Programer preuzima zajedničku (engl. *master*) kopiju trenutačnog koda (engl. *code base*), nad kojom zatim obavlja promjene. Kako vrijeme prolazi, ne samo da se njegov kod mijenja, već se mijenja i zajednička kopija. Čim je duže programerova kopija (engl. *branch*) izdvojena, to je veća vjerojatnost pojave konflikata pri spajanju kopija. Programer u takvom slučaju prije spajanja kopija mora prvo preuzeti glavnu kopiju, otkloniti konflikte koje prouzrokuje njegov kod, te kod dodati u glavnu kopiju.

Nakon nekog vremena kopije mogu postati toliko različite da vrijeme potrebno za spajanje kopija premašuje vrijeme koje je bilo potrebno za obavljanje promjena u izdvojenoj kopiji. Ovaj se problem tada naziva "pakao integracije".

Kontinuirana integracija učestalim spajanjem radne i glavne kopije nastoji izbjeći navedeni problem.

2.1. Provjera ispravnosti

Uz integraciju više puta dnevno, jedini zahtjev kontinuirane integracije je da kod koji se integrira ne izaziva konflikte, odnosno da se oni u slučaju pojavljivanja odmah razriješe. Međutim, u praksi se uz pojam kontinuirane integracije uvijek veže i neka razina provjere ispravnosti obavljene integracije.

Osnovni oblik provjere ispravnosti integracije je provjera ispravne izgradnje koda (engl. *build*). Ova provjera zahtjeva da se kod, nakon obavljene integracije, može

ispravno izgraditi. Zahtjevi provjere, pa tako i njeni rezultati, su vrlo ograničeni.

Zbog toga se od prve pojave kontinuirane integracije u okviru ekstremnog programiranje u proces uključuje i automatizirano testiranje koda. Nad svakom se integracijom pokreću svi testovi te se bilježi njihova prolaznost. Integracija se smatra ispravnom samo ako su svi testovi ispravno izvršeni. Ova praksa dodatno osigurava ispravnost radne kopije, te, ako je izrada testova ispravno obavljena, sprječava narušavanje ispravnosti postojećeg koda.

Pokazalo korisnim u ovoj fazi provoditi i općenitu kontrolu kakvoću (engl. *quality control*), u što ubrajamo mjerenje pokrivenosti koda i sukladnosti sa standardom, mjerenje performansi, pa čak i automatsko kreiranje dokumentacije iz koda. Umjesto provođenja kontrole kakvoće nakon završetka implementacije, provjera se provodi kontinuirano. Ovaj način implementacije kontrole kakvoće poboljšava kvalitetu programske potpore i smanjuje vrijeme potrebno za isporuku osiguravajući maksimalnu ispravnost glavne kopije.

2.2. Kada i gdje provoditi integraciju

U praksi su prisutna dva različita načina implementacije kontinuirane integracije s obzirom na trenutak obavljanja provjera ispravnosti. Implementacije kontinuirane integracije tako dijelimo na trenutnu i odgođenu integraciju.

Trenutna integracija sve provjere ispravnosti provodi nad svakim pokušajem integracije radne kopije s glavnom kopijom. Integracija se smatra ispravnom tek nakon prolaska svih provjera te tada promjene postaju glavnom kopijom. Cijeli proces integracije, odnosno spajanje i provođenje svih provjera ispravnosti, sustav gleda kao jednu atomarnu operaciju.

S druge strane, odgođena integracija provjeru ispravnosti glavne kopije provodi u nekom proizvoljnom trenutku, najćešće nakon određenog broja spajanja ili u nekom unaprijed određenom vremenu. Svako uspješno spajanje, odnosno spajanje s razrješenim konfliktima, postaje glavna kopija bez provođenja dodatnih provjera. Provjere se provode nad više spajanja istovremeno te o svakom pojedinačno prikazuje rezultate.

Vrlo je važno da su rezultati integracije vidljivi što prije. Ne samo da su oni potrebni programeru kako bi mogao nastaviti svoj rad, već oni diktiraju i proces ispravka pogreška. Što je rezultat udaljeniji od trenutka kad je spajanje započelo, to je veća šansa da će eventualni problemi biti zanemareni do nekog drugog trenutna, eliminirajući osnovnu korist kontinuirane integracije.

Zbog navedenog se danas teži trenutnoj integraciji. Ipak, kako bi se cijeli proces

ubrzao, provjera se najčešće odgađa neko kratko vrijeme nakon početka spajanja te se spajanja pokušavaju testirati u manjim skupinama.

S obzirom na mjesto provođenja integracije, implementaciju kontinuirane integracije djelimo na lokalnu i poslužiteljsku. Lokalnu integraciju provodi osoba koja želi obaviti integraciju, najčešće sam programer. Prednost ovog tipa integracije je jednostavnost implementacije te prisutnost programera koji može ručno provesti teže automatizirane dijelove. Poslužiteljska integracija je u automatizirana integracija koja se automatski obavlja na poslužitelju. Njezina implementacija je nešto složenija ali dugoročno daje bolje rezultate.

Lokalna integracije se pokazala vremenski zahtjevnom te vrlo podložnom ljudskoj pogrešci. Zbog toga je ona danas vrlo rijetak te se prvenstveno koristi u slučajevima kada programska podrška ne omogućava željenu funkcionalnost. S vremenom su poboljšanja počela uključivati koncept integracijskih poslužitelja (engl. *build server*) koja samostalno obavljaju proces integracije te po završetku dostavljaju i omogućuju pregled rezultata procesa. Danas je ovaj pristup, uz visoku razinu automatizacije, puno popularniji.

2.3. Praksa

Ne provođenje kontinuirane integracije je skupo, kontinuirana integracija je jeftina.Čestom integracijom pogreške se odmah otkrivaju te se smanjuje vrijeme uloženo u otkrivanje izvora pogreške.

Kontinuirana integracija je široko prihvaćena praksa uspješno implementirana u mnogobrojnim projektima. Ovo se poglavlje bavi najboljim praksama usvojenim kroz godine razvoja te nedoumicama koje još uvijek postoje.

2.3.1. Jedan repozitorij koda

Održavanje jednog repozitorija koda je ne samo jednostavnije, nego daje i bolje rezultate. Važno je repozitorij uvijek održati ispravnim i izgradivim. Drugim riječima, potrebno je omogućiti da se preuzimanjem svježe kopije repozitorija sustav može jednostavno izgraditi. Dodatno, stavlja naglasak na minimalno korištenje račvanja (engl. *branching*) repozitorija. Promjene je bolje integrirati nego stvoriti više verzija repozitorija koje je zatim potrebno odvojeno održavati[2]. Pojava potrebe za račvanjem najčešće upućuje na spori proces distribucije novih funkcionalnosti koji se nadalje pokušava riješiti kontinuiranom dostavom.

2.3.2. Automatiziranje izgradnje i provjera ispravnosti

Automatiziranje procesa kontinuirane integracije smanjuje ukupnu količinu potrebnog posla te poboljšava kvalitetu integracije. Automatizacija uključuje automatizaciju procesa spajanja, izgradnje, provjere ispravnosti i izvještavanja. Cilj je cijeli proces integracije moći pokrenuti korištenjem jedne naredbe. Automatizacija iziskuje ulaganje određenog truda. Zbog toga je korisno na manjim projektima odrediti i utvrditi proces kontinuirane integracije, te ga zatim automatizirati. Drugim riječima, iz lokalne i odgođene integracije, s vremenom prijeći na poslužiteljsku i trenutnu integraciju.

Vrlo je važno proces provjere ispravnosti održati brzim. Problem koji se često javlja u praktičnoj primjeni kontinuirane integracije je odgođeno ispravljanje pogrešaka otkrivenih u procesu kontinuirane integracije. Razlog tome je spor proces provjere zbog kog programer koji je započe integraciju često ne čeka završetak procesa. Zbog toga se eventualne pogreške otklanjaju kasno te se degradira cijeli proces.

2.3.3. Integracijski poslužitelj

Korištenjem integracijskog poslužitelja, provođenje integracije se s lokalnog okruženja prenosi na poslužitelj. Sve se izgradnje i testiranja provode u istoj, lako prilagodljivoj okolini. Integracijski poslužitelji, slično kao i automatiziranje, olakšavaju proces integracije. Jednako tako, i pokretanje integracijskih poslužitelja zahtjeva ulaganje određenog truda. Ovdje proces olakšavaju brojni alati na koje ćemo se osvrnuti u sljedećem poglavlju.

2.3.4. Testiranje u kopiji produkcijske okoline

Testna okolina (engl. *test environment*) i produkcijska okolina se u praksi često značajno razlikuju. Čak i ako one počnu vrlo slične, s vremenom bez potrebnog nadzora divergiraju. Zbog navedenog, izgradnja i testiranje u testnoj okolini može dati krive ili nepotpune rezultate te prelazak u produkciju zahtjeva značajan napor. Zbog navedenog testna okolina treba biti skalirana replika produkcijske okoline.

2.3.5. Jednostavan pregled rezultata

Rezultati integracija trebaju biti javni i lako dohvatljivi. Održavanje najvišeg stupnja kvalitete treba ući u kulturu kompanije te treba postati nešto čime su zaposlenici ponosni. Ovdje je dostupnost rezultata integracije vrlo važna.

2.3.6. Odbijanje neispravnih integracija

Trenutno se u praksi spajanje radne kopije s glavnom obavlja čak i ako neki od testova nisi ispravno izvršeni. U takvom slučaju programer je zadužen za što brže popravljane nastalih grešaka. Drugim riječima, programer je zadužen za održavanje ispravnosti radne kopije. Timovi se obično u početku razvoja dobro nose s ovom činjenicom te pogreške ažurno ispravljaju. Međutim, opterećeni drugim zadacima višeg prioriteta, tim postupno zanemarije neispravne testove te se fokusira na ispunjavanje rokova[7]. Navedeno značajno degradira kvalitetu produkta te u konačnici rezultira većom količinom potrebnog posla. Rješenje ovog problema je odbijanje integracija koje uzrokuju neispravan prilazak čak i jednog testa, bez iznimaka[8].

2.4. Nedostaci

Razvoj okruženja koje podržava kontinuiranu integraciju može zahtjevati izvjesnu količinu posla. Posebno su osjetljivi sustavi i timovi već definirane strukture i načina rada. Promjena ustaljenog načina rada zahtjeva određeni napor. Ovdje iskisniji inežnjeri predlažu postepenu implementaciju, počevši s manjim, fleksibilnijim timom i postupnim proširenjem na ostale djelove. Problemi se rješavaju kako se na njih nalilazi. Čest je i otpor prema promjeni očekivanog ponašanja zaposlenika, ali naglašavanjem pozitivnih strana promjene moguće je doći do željenih razultata.

Implementaciju kontinuirane integracije značajno olakšavaju brojni komercijalni "off-the-shelf" alati. Osvrnimo se na neke TODOOOO

3. Implementacije

Na tržištu je dostupan broj više ili manje prilagodljivih sustava koji implementiraju ili omogućuju jednostavnu implementaciju kontinuirane integracije. Ovo će poglavlje dati pregled te usporediti mogućnosti i karakteristike danih alata.

3.1. Xcode Server

Xcode Server omogućava vrlo jednostavnu kontinuiranu integraciju za korisnike programskog alata Xcode, odnosno za razvoj aplikacija za iOS i OS X operacijske sustave. Xcode Server je kombinacija dva alata, Xcodea, alata za razvoj iOS i OS X aplikacija, te OS X Servera. Xcode Server omogućava automatiziranu integraciju, izgradnju, testiranje, analizu i arhiviranje aplikacija. Također omogućava automatsku dojavu rezultata i generiranje statistika za velik broj parametara[**?**].

Xcode Server, Appleov alat za kontinuiranu intregacijuje, je po mnogima najbolji sistem za kontinuiranu integraciju iOS i OS X aplikacija. Xcode Server je razvijan uz Xcode. Za razliku od ostalih alata, sve su nove funkcionalnosti potrebne za integraciju Xcode projekata dostupne na dan njihovog izdavanja.

Dobre strane Xcode Servera:

- jednostavan za poretanje i korištenje
- uvjek *up to date*
- besplatan za korisnike s Apple Developer akauntom
- samostalno hostan
- testiranje na povezanim uređajima
- vrlo dobro verzioniranje

Loše strane Xcode Servera:

- isključivo Xcode projekti
- ograničena personalizacija

4. Zaključak

Zaključak.

5. Literatura

[1] Agile Alliance. Continuous deployment, 2016. URL http://guide.agilealliance.org/guide/cd.html. [Online; accessed 2-April-2016].

[2] ToughtWorks. Continuous integration, 2016. URL https://www.thoughtworks.com/continuous-integration. [Online; accessed 8-March-2016].

[3] Wikipedia. Booch method — Wikipedia, the free encyclopedia, 2015. URL https://en.wikipedia.org/wiki/Booch_method. [Online; accessed 10-March-2016].

[4] Wikipedia. Continuous delivery — Wikipedia, the free encyclopedia, 2016. URL https://en.wikipedia.org/wiki/Continuous_delivery. [Online; accessed 2-April-2016].

[5] Wikipedia. Continuous integration — Wikipedia, the free encyclopedia, 2016. URL https://en.wikipedia.org/wiki/Continuous_integration#cite_note-1. [Online; accessed 9-March-2016].

[6] Wikipedia. Version control — Wikipedia, the free encyclopedia, 2016. URL https://en.wikipedia.org/wiki/Version_control. [Online; accessed 10-March-2016].

[7] DevOps.com Yegor Bugayenko. Why continuous integration doesn't work, 2014. URL http://devops.com/2014/09/26/continuous-integration-doesnt-work/. [Online; accessed 11-March-2016].

[8] Yegor256. Continuous integration is dead, 2014. URL http://www.yegor256.com/2014/10/08/continuous-integration-is-dead.html. [Online; accessed 11-March-2016].

6. Sažetak

Sažetak.