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, ožujak 2016.

SADRŽAJ

1. Uvod TODO

Kontinuirana dostava (engl. *continuous delivery*) je, u sklopu programskog inženjerstva, pristup razvoju programske potpore koji nastoji smanjiti vrijeme od razvoja do produkcije programske potpore. Timovi proizvode programsku potporu u kratkim ciklusima, istovremeno osiguravajući ispravnost u bilo kom trenutku. Programska potpora se gradi i unaprjeđuje inkrementalno čime se nastoji smanjiti trošak, trajanje i rizik razvoja istovremeno povećavajući kvalitetu istog.

Kontinuirana isporuka (engl. *continuous deployment*) nadalje svaku ispravnu promjenu automatski isporučuje u produkciju. Promjene se često označavaju dodatnim tagovima koje određuju koja će skupina korisnika vidjeti promjenu. Ovaj pristup dodatno poboljšava i ubrzava razvoj.

Kontinuirana integracija (engl. *Continuous Integration (CI)*) je pristup razvoju programske potpore u kom se različite radne kopije učestalo spajaju s glavnom kopijom. Ovaj pristup značajno olakšava razvoj izbjegavajući spajanje velike količine koda. Ovaj pristup nije nužan za kontinuiranu dostavu i isporuku, ali je gotovo uvijek prisutan.

Kontinuirana integracija, dostava i isporuka su široko prihvaćeni pristupi u razvoju programske potpore standardnih sustava. Ovaj će se rad osvrnuti na navedene pristupe i dodatno promotriti njihovu implikaciju na mobilne operacijske sustave.

Ovaj je rad podijeljen u 4(TODO) dijela. Prvi se dio bavi kontinuiranom integracijom i njezinim trenutnim implementacijama.

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*)[1]. Kasnije praksu nasljeđuje ekstremno programiranje (engl. *eXtreme Programming (XP)*) koje dodatno potiče praksu integracije nekoliko puta dnevno[2].

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.

Ubrzo se 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.

TODO: zaokružiti, istražiti što još kompanije ubrajaju pod CI i ostvariti poveznicu na CD

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

Godine prakse su proizvele najbolju praksu koju je korisno sljediti.

Kontinuirana integracija, odnosno čin integrcije radne kopije s glavnom kopijom, se treba odvijati dovoljno učestalo da se greške ne pojavljuju bez programerovog znanja i da ih on može odmah ispraviti. Normalna praksa je pokrenuti proces izgradnje nakon svakog spajanja, a ne nakon određenog intervala. Većina alata pruža ovu opciju.

Sljedeći važan faktor je alat za verzioniranje koji podržava atomična spajanja (engl. *atomic commits*). Niz promjena u skopu radne kopije se smatraju jednom operacijom spajanja.

Kako bi postigli ova svojsta većina se implementacija kontinuirane integracije oslanja na sljedeća svojstva.

2.3.1. Repozitori koda

Sistem za konrolu verzija (engl. *version control system*) značajno olakšava razvoj programske potpore. Grupirajući sve artefakte koji su dio programske potpore, održavanje, djeljenje i izmjena koda postaju puno jednostavniji[3]. CI dodatno zahtjeva da su u repozitorij uključeni svi artefakti potrebni za izradnju, te da je sustav izgradiv iz svježe glavne kopije. Dodatno, stavlja naglasak na minimalno korištenje račvanja (engl. *branching*) kad je ono dostupno. Smatra se boljim promjene itegrirati, nego održavati više verzija programske potpore istodobno.

2.3.2. Automatiziranje izgradnje

Važno je automatizirati izgradnju. Sustav treba moći izgraditi korištenjem jedne komande. Danas na tržištu postoje brojni alati koji omogućuju automatizaciju izgradnje te su oni neizostavni u sklopu CI-a.

Nakon izgradnje potrebno je provesti sve testove kako bi se osiguralo da novi kod nije ugrozio ispravnost postojećeg. Zbog toga se uz automatizaciju izgradnje automatski provodi i testiranje.

Trajanje ovog procesa je potrebno što više ubrzati kako bi programer odmah dobio potrebne podatke. Što duže programer ćeka rezultate izgradnje i testiranja, to postoji veća šansa da na probleme neće odmah reagirati.

2.3.3. Dnevno spajanje s glavnom kopijom

Od svih uključenih u proces izgradnje se očekuje najmanje dnevno spajanje radne verzije s glavnom kopijom. Ovaj zahtjev se nalazi u suštini CI-a, ali ga je vrlo važno konstanto održavati.

Svako od spajanje je zatim potrebno izgraditi kako bi se osiguro integritet kopije.

2.3.4. Testiranje u kopiji produkcijske okoline

Testna okolina (engl. *test environment*) je okolina ostvarena prvenstvno za testiranje, za razliku od produkcijske okoline koja je namjenjena za produkciju aplikacije. U praksi se ovo okoline često razlikuju. Čak i ako počnu vrlo slične, s vremenom bez potrebnog nadrzora divergiraju. Zbog navedenog, izgradnja i testiranje u testnoj okolini može dati krive ili nepotpune rezultate. Rezultat je nepotpuna evalucaija dobivenog rješenja te komplikacije prilikom prelaska u produkciju. Zbog toga testna okolina

treba biti skalirana replika produkcijske okoline. Ovdje se posebno korisnim pokazuju servisi za virtualizaciju koji ostvaruju ujednačenost okolina.

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. Zaključak

Zaključak.

4. Literatura

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

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

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

5. Sažetak

Sažetak.