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

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*) 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.

2. Kontinuirana integracija

Počnimo s najstarijim konceptom, kontinuiranom integracijom, nadalje CI prema (engl. *continuous integration*).

CI je praksa u programskom inženjerstvu spajanja svih radnih kopija 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)*) te dodatno potiče integraciju nekoliko puta dnevno[2].

Glavni 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 kojim zatim obavlja promjene i dodaje nove funkcionalnosti. 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 tada ponovno preuzima glavnu kopiju, otklanja konflikte koje prouzrokuje njegov kod, te konačno spaja kopije.

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 ovaj problem.

2.1. Rani razvoj

Uz provjeru prolaska procesa izgradnje koda (engl. *build*), kontinuirana integracija od svoje prve pojave u sklopu ekstremnog programiranja uključuje i automatizirano testiranje prije procesa spajanja koda. Ova praksa dodatno osigurava ispravnost radne kopije, te, ako je proces testiranja korektno obavljen, sprječava narušavanje ispravnosti postojećeg koda.

Inicijalno je testiranje provodio sam programer u svom lokalnom okruženju, najčešće korištenjem unit testova. Kasnija poboljšanja uključuju koncept integracijskih poslužitelja (engl. *build server*) koja samostalno obavljaju testiranje periodički ili čak nakon svakog spajanja, te najvažnije dostavljala izvješća o testiranju programerima.

Uz automatizirano testiranje, integracijski su poslužitelji ubrzo korišteni i za provođenje općenite kontrole kakvoće (engl. *quality control*). Dodano je statičko i dinamičko testiranje, mjerenje pokrivenosti koda i performansi, pa čak i automatsko kreiranje dokumentacije iz koda. Ovaj način implementacije kontrole kakvoće pokušava poboljšati kvalitetu programske potpore i smanjiti vrijeme potrebno za isporuku. Umjesto provođenja kontrole kakvoće nakon završetka implementacije, provjera se provodi kontinuirano.

2.2. Praksa

Kontinuirana integracija je od svoje prve pojave pa do danas implementirana na mnogobrojne različite načine. Važno je zato izdvojiti najbolje prakse korištene pri implementaciji CI-a i dijelove koji se smatraju standardnim.

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.

2.2.1. 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.

2.2.2. 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.

2.2.3. 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.

2.2.4. 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.

2.3. 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.