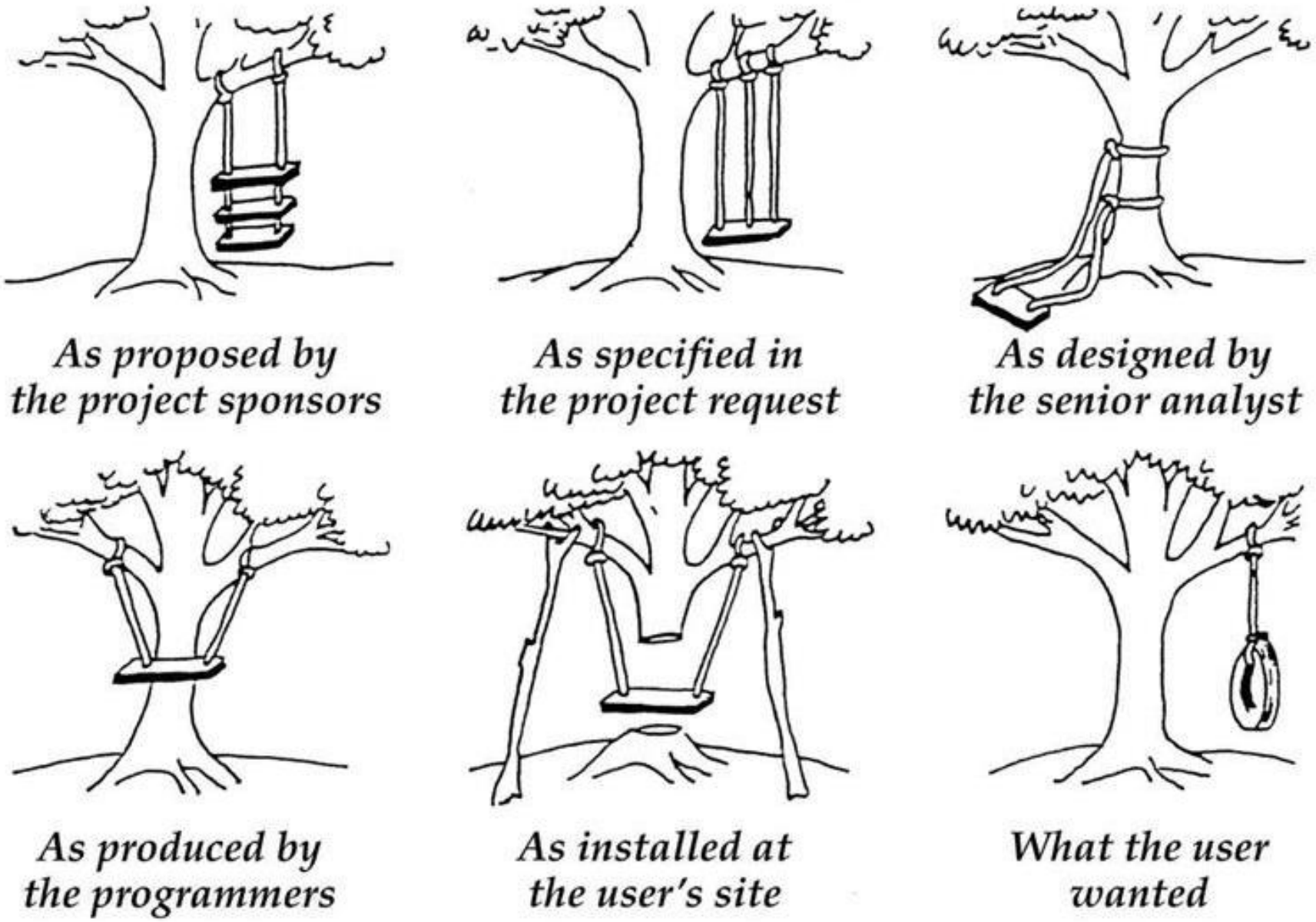
Razvoj primijenjene programske potpore

2. Zahtjevi. Upravljanje konfiguracijom

Pogrešno postavljeni zahtjevi za posljedicu imaju neispunjena očekivanja, a naknadne prepravke su „skupe”



Zahtjevi

- ISO/IEC/IEEE 24765:2010 Systems and software engineering—Vocabulary:
  - uvjet ili sposobnost koje korisnik treba da bi riješio problem ili ostvario cilj.
  - uvjet ili sposobnost koji mora posjedovati ili zadovoljiti sustav, komponenta sustava, proizvod ili usluga da bi zadovoljila ugovor, standard, specifikacije ili neki drugi ugovoreni dokument.
    - Dodatno, prema PMBOK zahtjevi uključuju nabrojane i dokumentirane potrebe, želje i očekivanja sponzora, korisnika i ostalih dionika u projektu
- ISO/IEC/IEEE 29148:2011 Systems and software engineering--Life cycle processes--Requirements engineering
  - izjava kojom se prevodi ili izražava potreba i potrebi pridružena ograničenja i uvjeti

Generalna podjela zahtjeva

- Generalna podjela na funkcionalne i nefunkcionalne

- Funkcionalni opisuju funkcionalnost (što softver mora raditi) te daju opise interakcije s korisnicima, drugim softverom ili hardverom (tko, što, koji podaci)

- Nefunkcionalni su vezani za svojstva ponašanja
  - performance, sigurnost, atribute kvalitete, …
  - razna ograničenja (standardi, OS, …)
  - najčešće se odnose na cijeli sustav, a ne na pojedinačno svojstvo
  - ponekad se nefunkcionalni zahtjevi nazivaju i pseudo-zahtjevi

Analiza (a ne dizajn)

- Zahtjevi odgovaraju na pitanje ŠTO sustav mora raditi, bez ulaženje u implementacijske detalje
  - odnos ulaza i izlaza, sadržaj formi, ali ne izgled formi i razmještaj kontrola
  - slijed operacija, ali ne i detalji implementacije
  - popis provjera ulaznih podataka, ali ne i način prikaza poruka za neispravne podatke
  - što treba poslati vanjskom sučelju, ali ne i kako se šalje
- Rezultat analize služi kao podloga za dizajn
  - Dio odluka prilikom dizajna može biti posljedica rezultata analize, odnosno nefunkcionalnih zahtjeva
    - npr. podržani uređaji, format izvještaja i zapisa, licence, pravila, certifikati …

Vrste zahtjeva

- Osim generalne podjele na funkcionalne i nefunkcionalne neki autori razlikuju
  - [Sommerville] korisničke i sistemske zahtjeve ovisno o detaljnosti zahtjeva i ciljanoj publici
    - Tko je korisnik? Krajnji korisnik ili vlasnik?
  - [Dennis, Wixom, Tegarden] poslovne i sistemske ovisno o količini tehničkih detalja
  - …

- Dobro oblikovani zahtjevi bi trebali imati sljedeću podjelu
  - **poslovni**
  - **korisnički**
  - **funkcionalni**
  - **nefunkcionalni**

Poslovni zahtjevi

- Odgovaraju na pitanje zašto (se radi neki sustav)
  - predstavljaju ciljeve organizacije ili korisničke zahtjeve na višoj razini i ukratko opisuju problem koji treba riješiti
  - U idealnom slučaju zahtjevi vlasnika podudaraju se s poslovnim ciljevima!

- Sadržani u dokumentima u kojima se opisuje vizija i opseg projekta

Primjeri poslovnih zahtjeva (1)

- Sustav za potporu rada udruga
  - evidencija članstva i automatizacija postupka primanja novih članova neke udruge
  - praćenje financijskih podataka udruge i njenih članova
  - poboljšanje procesa prodaje
  - omogućavanje internetske prodaje
  - podrška organiziranju natjecanja i okupljanja

Primjeri poslovnih zahtjeva (2)

- Sustav subvencionirane prehrane
  - Očekivana novčana ušteda
    - Sustav mora biti tako koncipiran da prava na subvencioniranu prehranu može koristiti samo student koji ih je stekao i da ih može koristiti samo u svrhu prehrane.
  - Sustav mora onemogućiti:
    - korištenje subvencije od strane osoba koje nemaju na to pravo
    - zaradu ilegalnih posrednika
    - korištenje subvencije za druge svrhe osim prehrane
    - naplatu usluga koje nisu pružene

Korisnički zahtjevi

- Zahtjevi krajnjih korisnika
  - opisuju zadatke koje korisnik mora moći obaviti služeći se aplikacijama
  - sadržani u opisima slučajeva korištenja, tj. opisima scenarija rada
  - obično se izražavaju u obliku „Korisnik želi/treba/mora moći obaviti…“
    - Može (treba) se navesti i cilj, odnosno koristi koje korisnik ima od te akcije - opravdanje zahtjeva

Primjeri korisničkih zahtjeva

- Korisnik mora moći ostvariti pravo na prehranu kod bilo kojeg pružatelja usluge
  - Novi sustav mora omogućiti da student ostvaruje svoje pravo kod bilo kojeg pružatelja usluge subvencionirane prehrane. Dosadašnja praksa je bila da svaki pružatelj usluga izdaje svoje bonove koji se mogu koristiti samo u određenim restoranima
- Korisnik treba plaćati obroke nakon korištenja pojedinog obroka.
  - Treba izbjeći bilo kakvo plaćanje od strane studenata za potrebe ostvarivanja prava, a posebice unaprijed.
- Korisnik mora moći prijaviti gubitak kartice
  - Potrebno je smanjiti rizik gubitka ostvarenih prava te sustav mora onemogućiti zloporabu stečenih prava.

Funkcionalni zahtjevi

- Odgovaraju na pitanje što (se može/mora napraviti koristeći sustav)
  - definiraju softversku funkcionalnost (očekivano ponašanje i operacije koje sustav može izvoditi) koju treba ugraditi u proizvod da bi omogućio korisnicima obavljanje njihovih zadataka
  - posebno zanimljiva mogućnost programa (*feature*) – skup logički povezanih funkcionalnih zahtjeva koje korisniku omogućuju ispunjavanje poslovnih zahtjeva
- Primjeri:
  - *Nakon što se studentu jednom zavedu prava na matičnoj ustanovi, sustav mora proslijediti informaciju svim pružateljima usluga, odnosno omogućiti distribuirane upite*
  - *Sustav na dnevnoj bazi mora kreirati izvještaje sa statistikom prehrane po pružateljima usluge i vrsti obroka*

Nefunkcionalni zahtjevi

- Odgovaraju na pitanje kako (ili kako dobro sustav mora raditi)
  - posljedica standarda, pravila i ugovora kojih se proizvod mora pridržavati
  - opisi vanjskih sučelja
  - zahtjevi na performance
  - ograničenja na dizajn i implementaciju te svojstva kvalitete
    - preciziraju opis proizvoda navodeći karakteristike proizvoda u različitim dimenzija koja su važne ili korisniku, ili razvojniku.

- Primjer
  - U sustavu prehrane nefunkcionalni zahtjevi mogu biti vezani za oblik korisničke kartice, protokol povezivanja, obvezu fiskalizacije itd…

Kriteriji kvalitete zahtjeva

Zahtjevi moraju odražavati korisnikove želje (**ispravnost**) i sljedeće kriterije

- **Singularnost**
  - Zahtjev ne smije biti kombinacija više zahtjeva
- **Potpunost**
  - opisati sve moguće scenarije, uključujući i one u slučaju pogreške
- **Konzistentnost**
  - zahtjevi ne smiju međusobno biti kontradiktorni
- **Nedvosmislenost**
  - svaki zahtjev se mora moći interpretirati na točno jedan način
- **Izvedivost** (ostvarivost)
  - zahtjevi se moraju moći implementirati
- **Provjerljivost**
  - za zahtjeve se mogu napisati (ponavljajući) testovi za provjeru ispravnosti
- **Sljedivost**
  - Zahtjev mora se moći upariti s izvorom, povezanim zahtjevima ali i budućom implementacijom

Izbjegavati nejasne i općenite izraze

- Primjeri riječi koje treba izbjegavati:
  - najviše, najbolje, lagano za korištenje, efikasno, *user friendly,* visoke kvalitete, on, taj, to, ako je moguće, primjereno, brzo, malo, …
- Ovakvi izrazi se mogu interpretirati na različite načine i/ili nije moguće provjeriti je li zahtjev ispunjen

Primjeri metrika za nefunkcionalne zahtjeve

- Brzina
  - broj transakcija u jedinici vremena, vrijeme odziva
- Veličina
  - broj ekrana, komponenti, vrijednost u (kilo/mega/…) bajtovima, …
- Jednostavnost upotrebe
  - vrijeme treninga, broj pogrešaka prilikom upotrebe
- Pouzdanost
  - prosječno vrijeme prije kvara, postotak dostupnosti
- Robusnost
  - vrijeme oporavka nakon pogreške, postotak događaja koji mogu uzrokovati pogrešku
- Portabilnost
  - broj ili postotak podržanih uređaja (sustava)

Primjer neostvarivog zahtjeva

- *„Proizvod će se trenutno prebaciti između ispisivanja i skrivanja znakova koji se ne mogu tiskati.”*
  - Računala ništa ne mogu napraviti trenutno te je ovaj zahtjev neostvariv.
  - Da li programska podrška sama odlučuje kad će se prebaciti iz jednog stanja u drugo ili je to inicirano akcijom korisnika?
  - Na koji dio teksta će se primijeniti promjena prikaza: da li samo označeni tekst, cijeli dokument ili nešto treće?
  - Jesu li „znakovi koji se ne mogu tiskati” skriveni znakovi, posebne oznake ili kontrolni znakovi? (dvosmisleno, nepotpuno)
- Bolji zahtjev:
  - *„Korisnik će posebno dogovorenom akcijom, odabrati da li će se HTML oznake u trenutno otvorenom dokumentu prikazivati ili ne.”*
  - Sad je jasno da je riječ o HTML oznakama te da korisnik može obaviti određenu akciju, ali nije točno navedeno kakvu (npr. kombinacija tipki, klik miša, potez prsta), što se prepušta dizajnerima.

Primjer nepotpunog zahtjeva

- „Status pozadinskog posla dostavlja se u redovitim intervalima ne kraćim od 60 sekundi.”
  - Što je statusna poruka i pod kojim uvjetima će biti dostavljena? Koliko dugo ostaje vidljiva? Koji dio proizvoda će dostaviti poruku? Koliko dosljedni intervali moraju biti?
- Preciznije i detaljnije bi bilo
  - Modul za nadzor će ispisivati statusnu poruku u za to određeni dio sučelja.
  - Poruka će se ažurirati svakih 60 sekundi (plus minus 10 sekundi) nakon što započne izvođenje pozadinskog zadatka i bit će vidljiva cijelo vrijeme.
  - Ako se pozadinski zadatak izvodi normalno, modul za nadzor će ispisivati postotak obavljenog posla.
  - Modul za nadzor će ispisati „Zadatak obavljen.” nakon što se zadatak obavi.
  - Modul će ispisati poruku o pogrešci ukoliko dođe do zastoja u izvođenju.
- Problem je rastavljen u više zahtjeva jer će svaki zahtijevati posebno testiranje.
  - Ako je više zahtjeva grupirano u jedan lakše je previdjeti neki od njih tijekom izrade ili testiranja.
- Primijetiti da u zahtjevu nije detaljno opisano kako će se poruka i gdje ispisivati. To će biti odlučeno tijekom dizajna !

Primjer nepotpunog i neprovjerljivog zahtjeva

- „Parser će brzo generirati izvješće o pogreškama HTML oznaka, koje omogućava brzi ispravak pogrešaka kada program koriste početnici u HTML-u.”
  - Zahtjev je neprovjerljiv! Kako testirati ovaj zahtjev?
    - Pronaći nekoga tko se smatra početnikom u HTML-u i zatim vidjeti kako brzo će, uz pomoć izvješća, ispraviti pogreške?
    - Što znači „brzo”? 5 sekundi ili 5 minuta?
  - Nije definirano što i kada se tvori izvješće i to čini zahtjev nepotpunim.
- Bolje:
  - *„Na zahtjev korisnika sustav će analizirati HTML datoteku te generirati izvješće koje sadrži broj linije i tekst pronađenih HTML pogrešaka, te opis svake pogreške.”*
    - Ako nema pogrešaka prilikom analize, neće se generirati izvješće.

Nekoliko primjera iz nedavne prakse (1)

- U postojećem sustavu forma je sadržavala polje za unos nalazišta i polje za unos staništa. Korisnik je u pisanom zahtjevu za promjenu naveo sljedeće:

  „preimenovati opis nalazišta u opis nalazišta i staništa”

- Treba li novi sustav imati samo jedno zajedničko polje za unos nalazišta i staništa ili 2 polja s nazivima „opis nalazišta i staništa” te „opis staništa”
- Razjašnjenje zahtjeva:
  - „Ostaju dva polja: 1. opis nalazišta i staništa te 2. Opis staništa, ali tako da na printanoj etiketi ne piše zasebno opis staništa već se pridružuje tekstu pod 1.”
  - Ne samo da prvi zahtjev nije bio jasan i nedvosmislen, nego je u sebi krio jedan inicijalno neotkriveni zahtjev

Nekoliko primjera iz nedavne prakse (2)

- „Dodati opciju za unošenje podzbirki, mora biti moguće dodati nekoliko opcija uz jedan herbarijski primjerak”

  => *„Uvesti herbarske podzbirke i omogućiti povezivanje herbarskog primjerka s više podzbirki”*

- Može li herbarski primjerak istovremeno pripadati i zbirci i podzbirci?
- Može li herbarski primjerak pripadati podzbirkama različitih nadređenih zbirki?

Postavljanje prioriteta

- Nužno svojstvo - Da li korisnik nešto stvarno mora imati?
  - Postoji tendencija da se previše zahtjeva proglasi nužnim!
  - Po definiciji, ako sustav ne uključuje nužne zahtjeve, taj sustav ne može ispuniti svoju svrhu.
  - Treba testirati svaki zahtjev koji se smatra nužnim i probati ga rangirati.
    - Ako se zahtjev može rangirati onda nije obvezan!
    - Nužni zahtjevi se ne mogu rangirati jer su nužni za prvu verziju sustava!
- Poželjno svojstvo - Funkcije koje korisnik želi na kraju imati
  - Ranije verzije sustava mogu biti bez tih zahtjeva.
  - Poželjni zahtjevi mogu i trebaju biti rangirani.
- Neobvezna svojstva - Proizvoljni zahtjevi
  - Svojstva i mogućnosti bez kojih se može (npr. ostvarivanje ostalih prava iz studentskog standarda iz primjera zahtjeva krajnjih korisnika)
  - Iako bi ih lijepo bilo imati, to nisu pravi zahtjevi.
  - Ovi zahtjevi također mogu biti rangirani.
- Alternativno označavanje prioriteta: numerički, po verzijama, …

Dokumentiranje analize (zahtjeva) (1)

- Definicija zahtjeva (*Requirements Definition*)
  - izjava o stanju i ograničenjima sustava te potrebama
  - narativni dokument namijenjen korisniku ili ga piše korisnik
    - poslovni i korisnički zahtjevi te njihovi prioriteti
    - uočeni problemi, ključne pretpostavke i preporuke rješenja

- IEEE standard ISO/IEC/IEEE 29148:2011

  Stakeholder Requirements Specification StRS

1. Introduction
- 1.1 Business purpose
- 1.2 Business scope
- 1.3 Business overview
- 1.4 Definitions
- 1.5 Stakeholders

2. References

3. Business management requirements
- 3.1 Business environment
- 3.2 Goal and objective
- 3.3 Business model
- 3.4 Information environment

4. Business operational requirements
- 4.1 Business processes
- 4.2 Business operational policies and rules
- 4.3 Business operational constraints
- 4.4 Business operational modes
- 4.5 Business operational quality
- 4.6 Business structure

5. User requirements

6. Concept of proposed system
- 6.1 Operational concept
- 6.2 Operational scenario

7 Project Constraints

8. Appendix
- 8.1 Acronyms and abbreviations

Dokumentiranje analize (zahtjeva) (2)

- Specifikacija zahtjeva *Requirements Specification*
  - naziva se i funkcionalnom specifikacijom
  - strukturirani dokument s detaljnim opisom očekivanog ponašanja sustava
  - namijenjen ugovarateljima i izvoditeljima razvoja
  - ugradbeno nezavisan pogled na sustav
    - funkcionalni i nefunkcionalni zahtjevi te njihovi prioriteti
    - model organizacijske strukture (strukturni dijagrami)
    - opis protoka dokumenata (dijagrami toka)
    - model procesa (dijagram toka podataka)
    - konceptualni model podataka (ERD)

- Software Requirements Specification SRS
  - IEEE standard ISO/IEC/IEEE 29148:2011

1. Introduction
- 1.1 Purpose
- 1.2 Scope
- 1.3 Product overview
  - 1.3.1 Product perspective
  - 1.3.2 Product functions
  - 1.3.3 User characteristics
  - 1.3.4 Limitations
- 1.4 Definitions

2. References

3. Specific requirements
- 3.1 External interfaces
- 3.2 Functions
- 3.3 Usability Requirements
- 3.4 Performance requirements
- 3.5 Logical database requirements
- 3.6 Design constraints
- 3.7 Software system attributes
- 3.8 Supporting information

4. **Verification**
(parallel to subsections in Section 3)

5. Appendices
- 5.1 Assumptions and dependencies
- 5.2 Acronyms and abbreviations

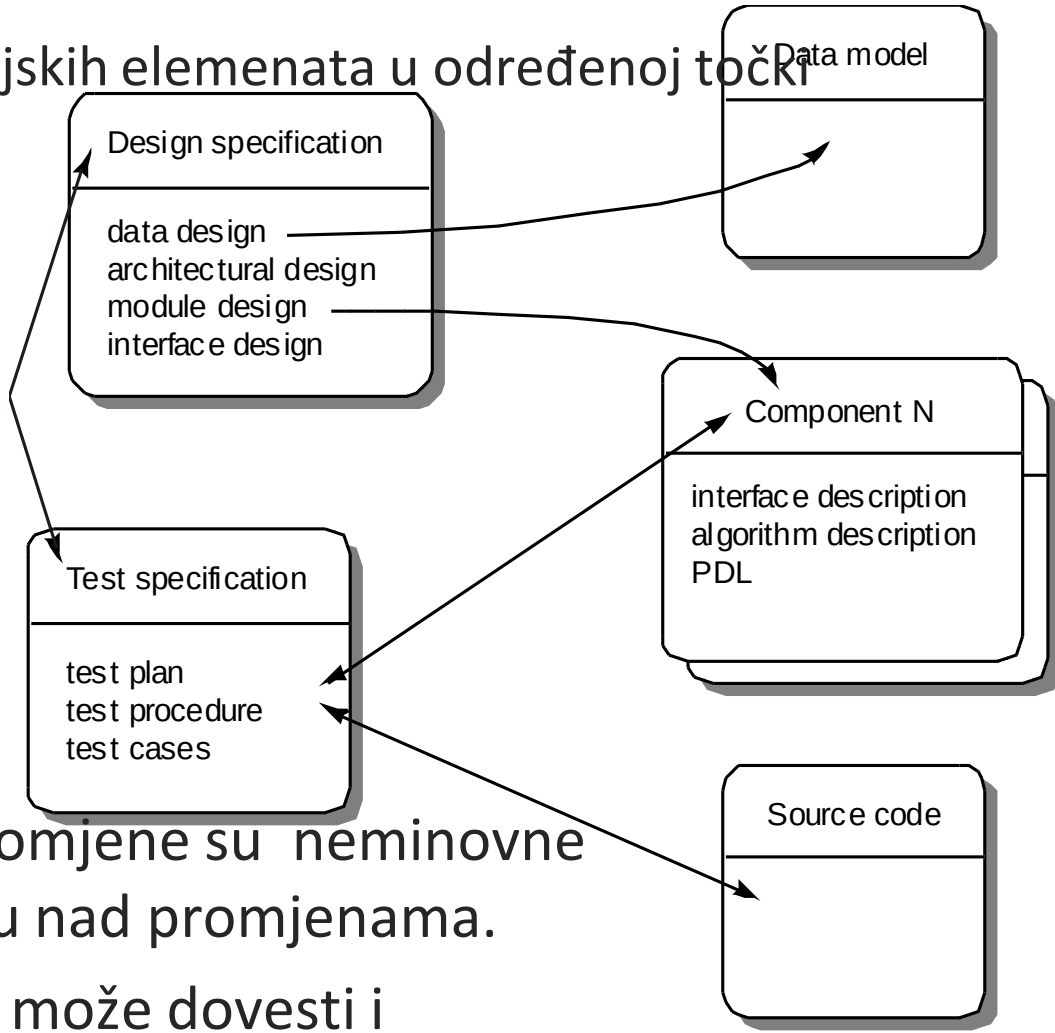
Dokumentiranje analize (zahtjeva) (3)

- Agilne metode često umjesto formalnog dokumenta evidentiraju zahtjeve u obliku korisničkih priča
  - služe kao podloga za procjenu posla
  - razlažu se u manje zadatke
  - odabiru se za izradu po prioritetu
  - pogodno za sustave s nestabilnim ili nejasnim zahtjevima
- Prednosti:
  - jednostavno uređivanje
  - nije potrebno tehničko predznanje za evidentiranje
- Nedostatak:
  - korisnička priča je samo podsjetnik (obećanje) da će se taj zahtjev razraditi detaljnije
  - preporuča se u nekom trenutku imati jasno definiran dokument sa zahtjevima

Upravljanje konfiguracijom

Konfiguracija

- Konfiguracija
  - imenovani skup konfiguracijskih elemenata u određenoj točki životnog ciklusa
- Element konfiguracije
  - agregacija hardvera i/ili softvera koja se tretira kao jedinka u procesu upravljanja konfiguracijom
  - može sadržavati bilo koji artefakt nastao u razvojnom procesu

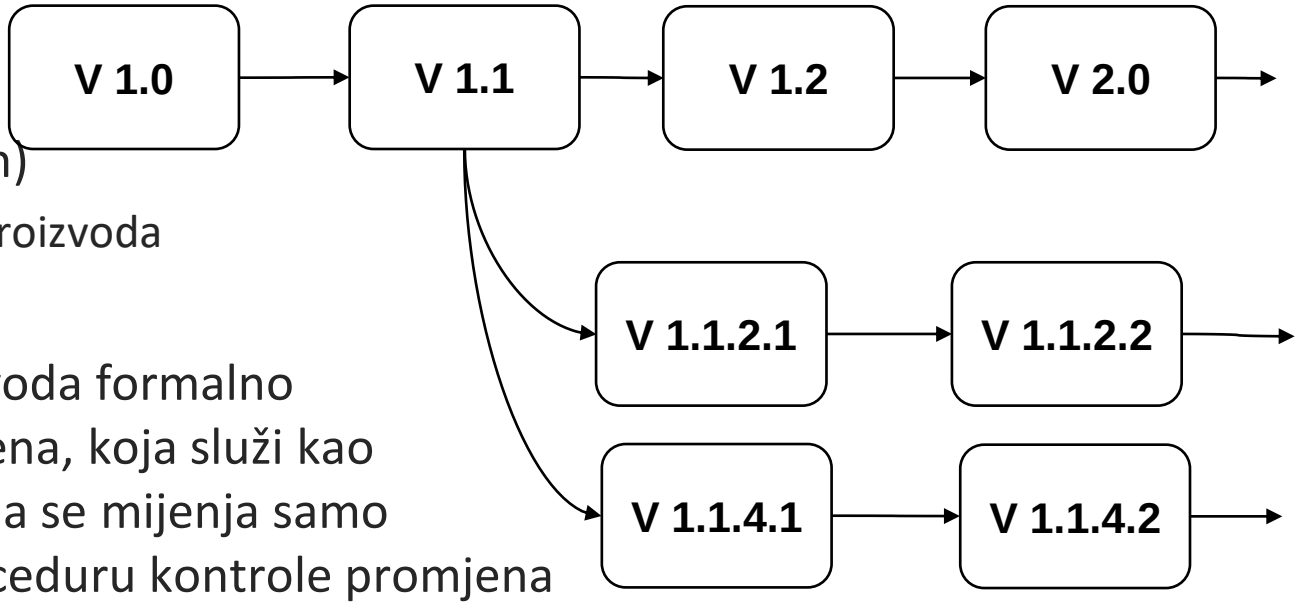


- Prilikom razvoja softvera, promjene su neminovne te je potrebno imati kontrolu nad promjenama.
- Nekontrolirani niz promjena može dovesti i najbolji softver u kaotično stanje

Upravljanje konfiguracijama

- Upravljanje konfiguracijama (engl. *configuration management*) je proces upravljanja promjenama prilikom razvoja softvera
- Sastoji se o 4 vrste aktivnosti
  - upravljanje zahtjevima za promjenu funkcionalnosti
    - proces procjene i odobravanja zahtjeva
  - upravljanje verzijama
    - evidentiranje različitih elemenata konfiguracije
    - upravljanje verzijama izvornog koda
  - integracija i (automatska) izgradnja sustava
  - upravljanje izdanjima
- Sustavi za podršku upravljanju verzijama i izgradnju obično omogućavaju i
  - praćenje problema (engl. *issues tracking*)
  - praćenje pogreški (engl. *bug tracking*)

Verzije konfiguracije



- verzija, inačica (version)
  - određeno izdanje proizvoda
- osnovica (Baseline)
  - specifikacija proizvoda formalno provjerena i usvojena, koja služi kao temelj razvoja i koja se mijenja samo kroz formalnu proceduru kontrole promjena
- objava, isporuka (release)
  - verzija isporučena korisnicima
- revizija (revision)
  - služi umjesto originalne, podrazumijeva izmjene nastale kroz vrijeme (npr. zbog ispravljanja pogrešaka), npr. V1.2
- varijanta (variant)
  - alternativa originalu (hardverska platforma, različiti jezik), živi paralelno s njim, npr. v1.1.2.1

Označavanje verzija (1)

- Postupak pridjeljivanja jedinstvene oznake, imena i broja određenoj verziji
  - ne mora biti ista za internu upotrebu i za isporuku
- Šarolika praksa označavanja, ali većinom u praksi dominira oblik koji počinje s *major.minor*
- Major version/release
  - predstavlja veliko izdanje.
  - vrijednost se povećava prilikom znatne promjene funkcionalnosti
    - Može izazvati prekid kompatibilnosti u odnosu na prethodnu verziju
- Minor version
  - malo izdanje
  - promjene mogu biti značajne, ali dodavanjem novih funkcionalnosti i uz zadržavanje kompatibilnosti s prethodnim verzijama

Označavanje verzija (2)

- Verzija objektne datoteke u .NET Frameworku (assembly) određena je s četiri broja:
  - <major version>.<minor version>.<build number>.<revision>
  - build number - predstavlja ponovno prevođenje istog koda (npr. prilikom promjene platforme, procesora i slično)
  - revision - primjenjuje se npr. prilikom izdavanja sigurnosnih zakrpa i sličnih manjih promjena
- Primjer: Properties \ AssemblyInfo
  - major.minor.* (ili major.minor.build.*) automatski određuje build number i revision
    - build number: broj dana od 1.1.2000.
    - revision: broj sekundi proteklih od ponoći aktualnog dana podijeljen s 2

Označavanje verzija (3)

- .NET Core koristi oblik major.minor.patch-sufix
  - Semantic versioning https://semver.org/
  - Sufiks može biti proizvoljan
    - alfa, beta, rc-2, …
  - Nova verzija obično biva označena jednim brojem više na odgovarajućoj poziciji

- Aplikacije za Google Play Store
  - Korisnik vidi oznaku verzije koju razvojni tim odabere (versionName)
  - Interno se koristi pozitivni cijeli broj (versionCode)
    - raste neovisno o tome povećava li se malo ili veliko izdanje
    - služi za uspostavljanje odnosa među verzijama (prethodna ili nadogradnja)

Označavanje verzija (4)

- Tex – trenutna verzija 3.14159265
  - Svaka nova verzija je nova znamenka broja π
- .NET Core 3.* → .NET 5
  - Preskače se .NET Core 4 da ne izaziva zabunu s .NET Framework 4
  - Mijenja se naziv zbog unificiranja razvoja platformi
- Java 1.0 → Java 1.1 → … → Java 1.4 → Java 5 (umjesto 1.5)
  - Isticanje srednje znamenke verzije
- Eclipse, Windows 10, Ubuntu, … upotrebljavaju godinu i mjesec
  - Ubuntu 20.04, 20.10, …
  - Windows 10 1909, 2004 + build number.update build revision
- Word 2.0 → Word 6.0 → Word 95 → …
- iPhone → iPhone 3G → … → iPhone 8 → iPhone X

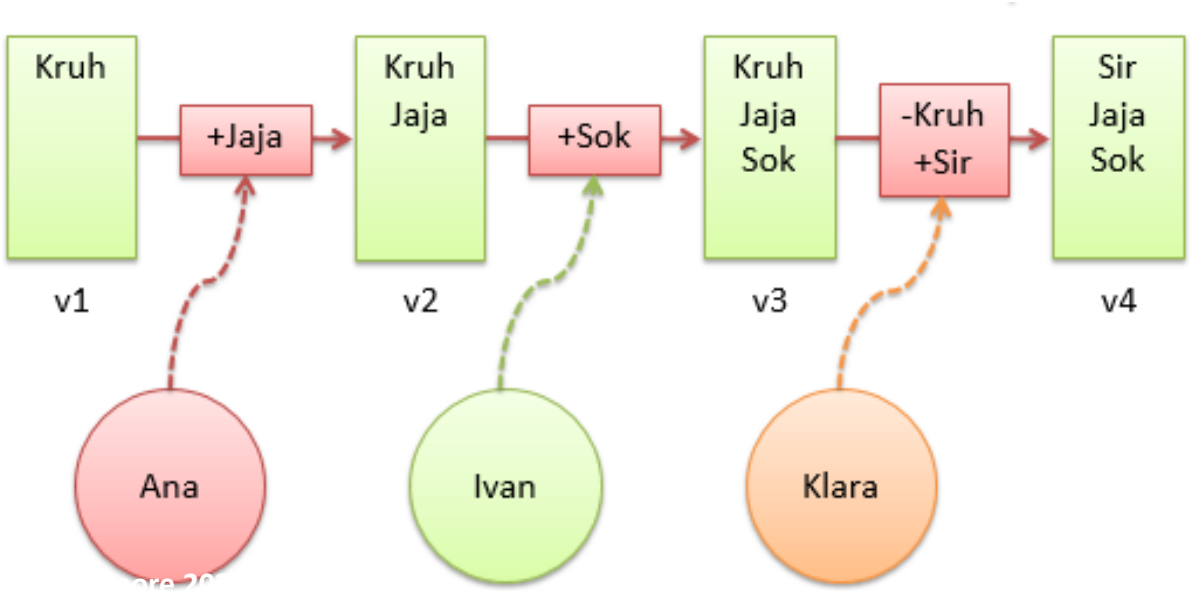
… i puno drugih varijanti kao posljedica usklađivanja varijanti, marketinga, izbjegavanja nesretnih brojeva i slično

Automatsko i ručno verzioniranje

- Automatsko označavanje
  - prednosti:
    - eliminacija ručnog rada (npr. pisanja i izvedbe skripti)
    - ne postoje dvije inačice s istom oznakom
  - nedostaci:
    - oznaka elementa ne podudara se s oznakom cijelog sustava
    - novi brojevi ovise o danu i vremenu prevođenja
    - verzija se mijenja pri svakom prevođenju, neovisno o tome jesu li se dogodile promjene ili ne
- Ručno verzioniranje
  - prednosti:
    - potpuna kontrola nad brojevima verzije
    - moguća je sinkronizacija između verzije pojedinih komponenti i verzije cijelog sustava
  - nedostaci:
    - verzioniranje se mora raditi ručno
    - moguće je napraviti više različitih objektnih datoteka s istim oznakama

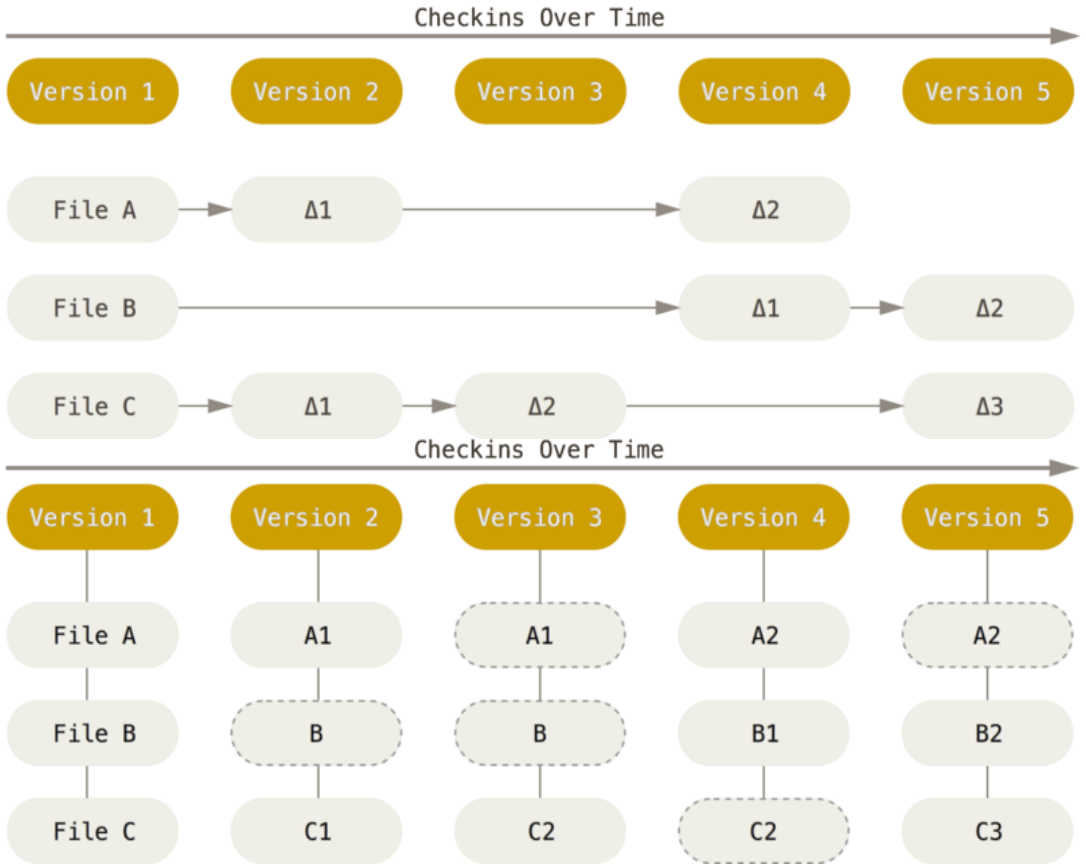
Upravljanje verzijama (izvornog koda)

- Alati za upravljanje verzijama (*Version control systems*) obično omogućavaju
  - identifikaciju verzije i izdanja
  - bilježenje povijesti promjena
  - podršku za neovisni razvoj više programera nad istim softverom
- Najčešće kontrola promjena izvornog koda i konfiguracijskih datoteka
  - ne evidentirati automatski generirane datoteke!
- Repozitorij sadrži zadnju verziju, ali implementira mehanizam rekonstrukcije određene verzije iz povijesti (npr. evidentirajući razlike)
- Niz verzija nastalih izvođenjem iz prethodnih verzija naziva se *codeline*



Karakteristike sustava za upravljanje verzijama

- Identifikacija verzija i izdanja
  - Svaka verzija pohranjena u repozitoriju ima jedinstveni identifikator
- Jedan skup promjena može sadržati promjene na više datoteka
- Kompaktna pohrana
  - Umjesto kopije svake od verzija čuva se zadnja verzija te lista razlika između susjednih verzija



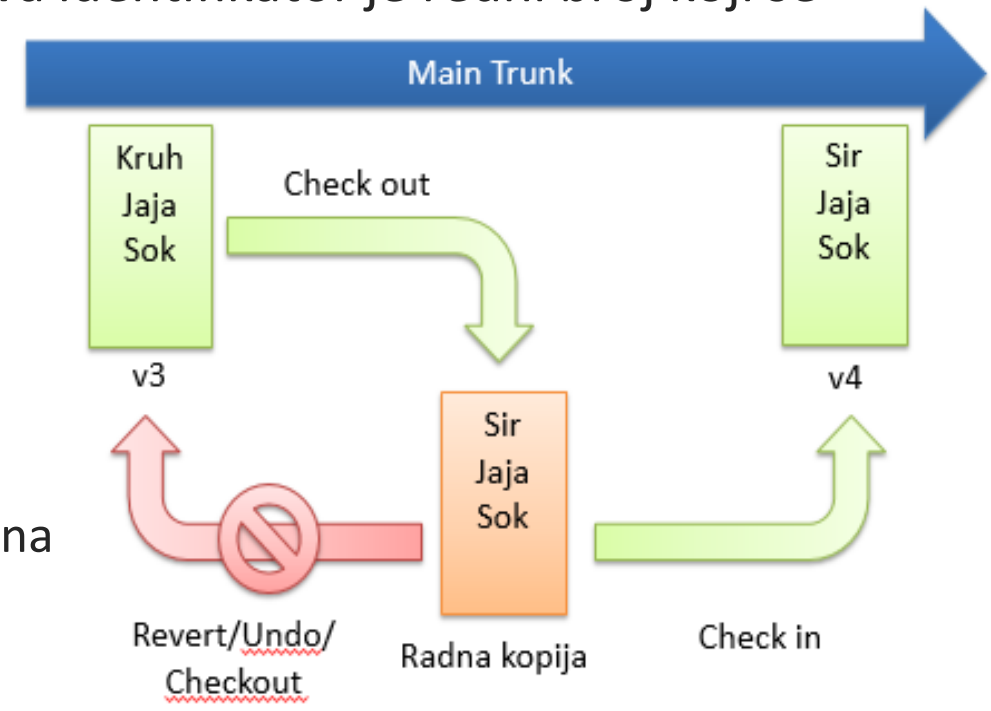
Chacon, Straub, ProGit 2nd Edition, Apress, 2014.

Centralizirani sustavi za upravljanje verzijama

- Subversion, CVS, Team Foundation Server, …
- Samo jedan centralni repozitorij
- Korisnik ima radnu kopiju (working copy, workspace)
  - Za svaku datoteku radna kopija sadrži određenu verziju iz centralnog repozitorija i informaciju o kojoj verziji se radi
  - Neki od alata (npr. TFS) dopuštaju da radna kopija sadrži samo dio centralnog repozitorija
- Promjene na radnoj kopiji započinju *checkoutom*
  - Najava izmjene datoteke
    - Može uključivati prethodni dohvat zadnje verzije datoteke i onemogućavanje ažuriranja te datoteke drugim korisnicima

Promjene na radnoj kopiji

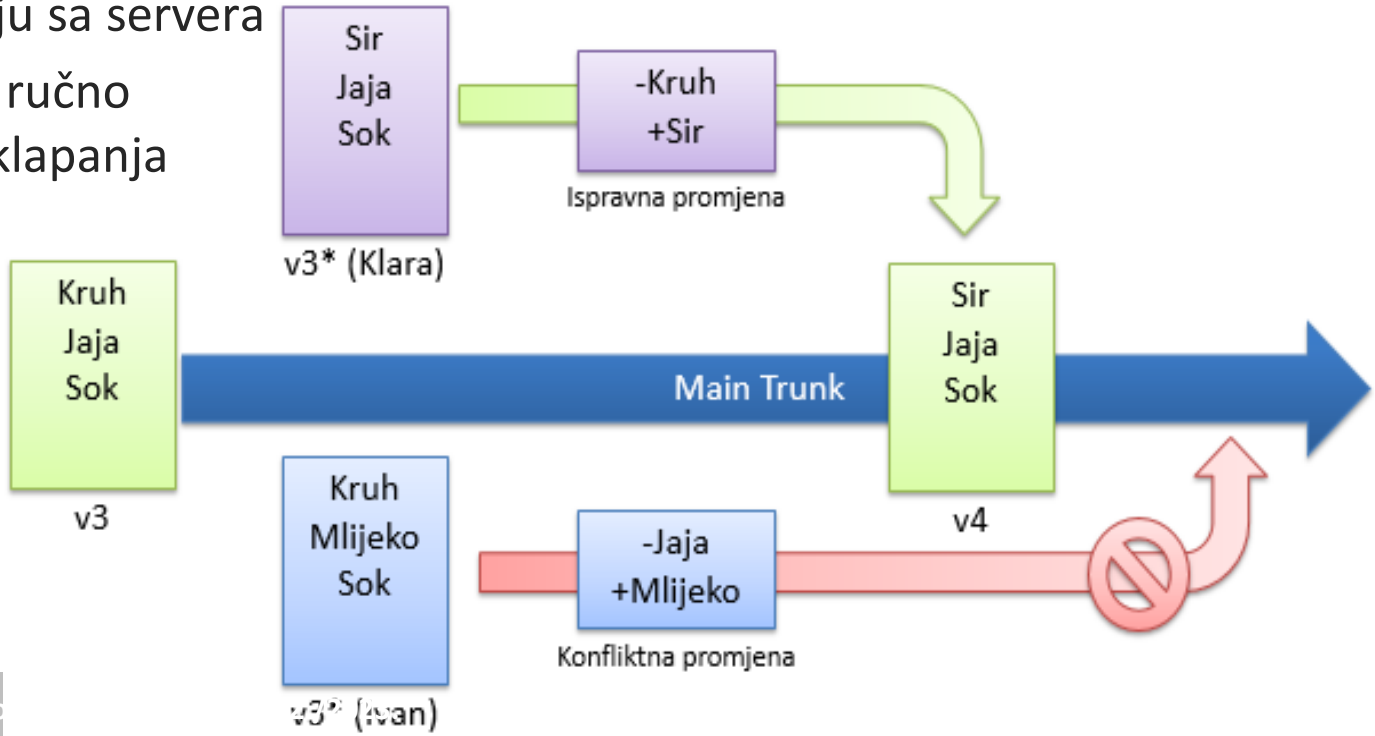
- Korisnik promjene na radnoj kopiji može
  - odbaciti (*undo*, *revert*, *checkout*)
  - potvrditi slanjem na centralni repozitorij (*check-in*, *commit*)
- Skup poslanih promjena naziva se *changeset*
- Svaki skup promjena dobiva jedinstveni identifikator
  - Kod centraliziranih sustava identifikator je redni broj koji se povećava za 1



Trunk – jedinstvena linija razvoja koja nije ničija grana

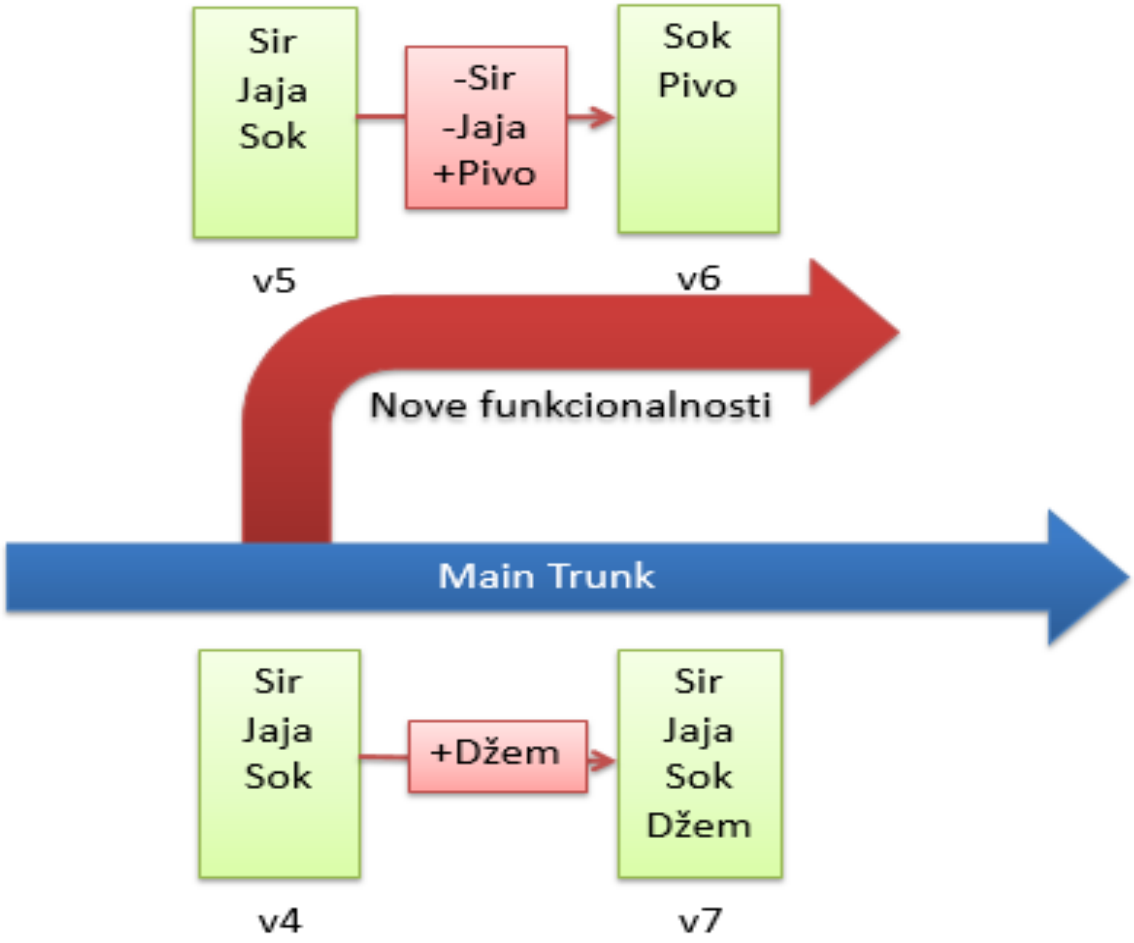
Konfliktne promjene

- Istovremene promjene mogu voditi do konflikta
  - Prvi korisnik će uspješno potvrdi promjene
  - Drugom korisniku će sustav dojaviti da su promjene napravljene u odnosu na verziju koja više nije najsvježija
- Rješenja
  - Odbaciti vlastitu verziju
  - Ignorirati verziju sa servera
  - Automatsko ili ručno rješavanje preklapanja



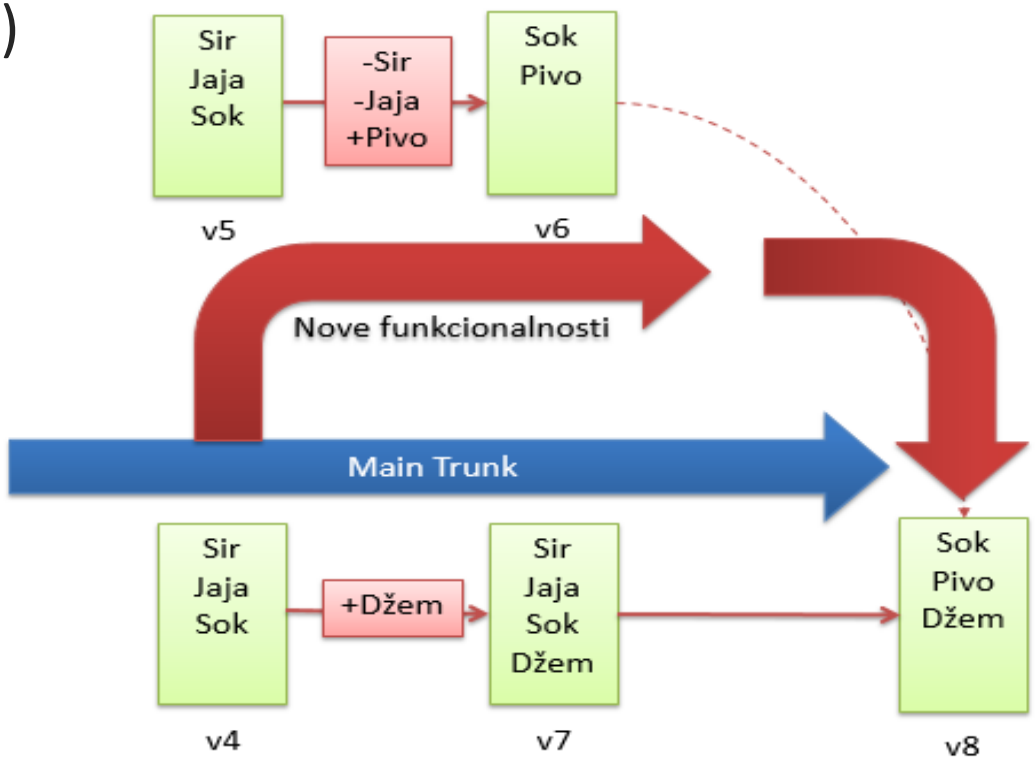
Paralelne grane

- Omogućava istovremeni rad na novim funkcionalnostima uz očuvanje ispravnosti i održavanje glavne (isporučene verzije).
  - Ovisno o sustavu može se raditi o fizički odvojenim kopijama datoteka



Spajanje grana

- Nije nužno – paralelna grana ne mora se nikad spojiti natrag
- Ugradnja promjena iz paralelne grane s promjenama u međuvremenu obavljenim na glavnoj grani
  - paralelna grana se može, ali i ne mora obrisati
- Git dodatno omogućava reproduciranje samo dijela promjena iz paralelne grane (cherry-pick)

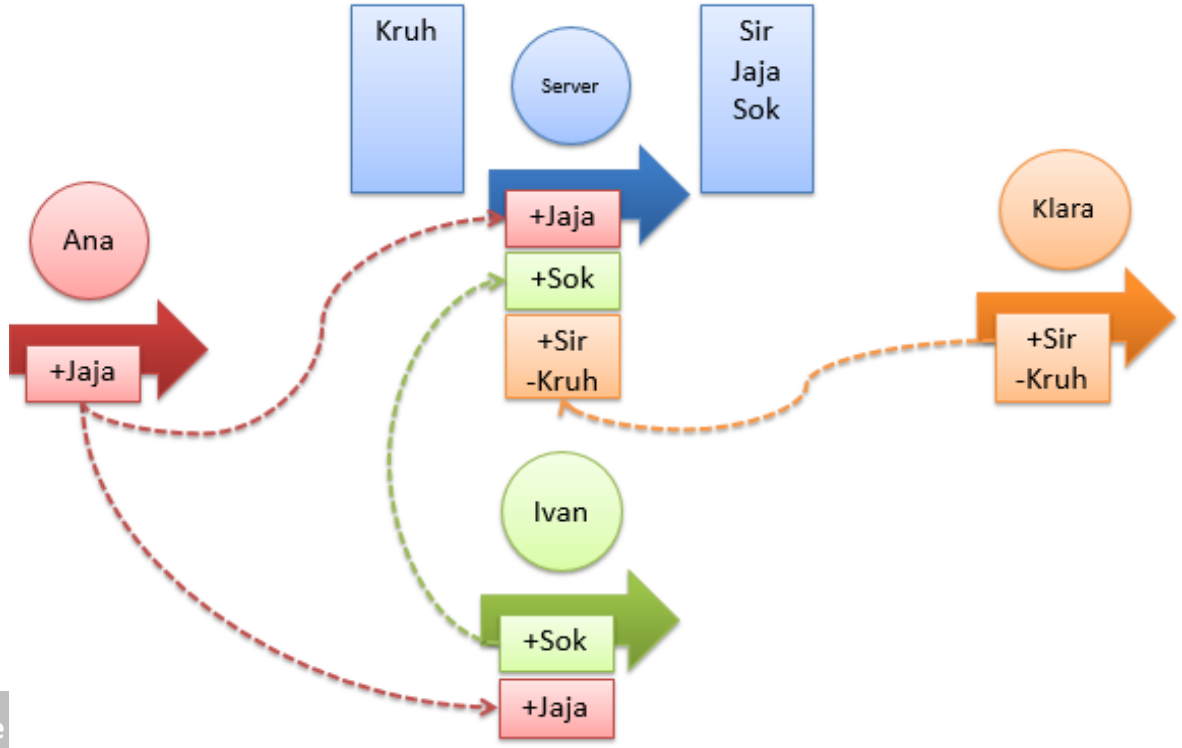


Distriburani sustavi

- Git, Mercurial
- Svaki korisnik ima kompletnu kopiju repozitorija
  - sadržaj verzioniranih datoteka, ali i cijela povijest promjena
  - lakše kopiranje repozitorija na neko drugo mjesto
- Korisnik može evidentirati promjene u izvanmrežnom načinu rada
  - naknadno prenosi svoju povijest na centralni server (*push*)
  - osvježava svoju verziju s verzijom na serveru (*fetch* i/ili *pull*)
    - *pull = fetch + merge*
- Inicijalno stvaranje kopije centralnog repozitorija naziva se kloniranje (engl. *clone*)
- Napomena: centralni repozitorij ne mora postojati!
- Ostali koncepti akcija i problema (konflikata) su isti ili slični kao kod centraliziranih sustava.

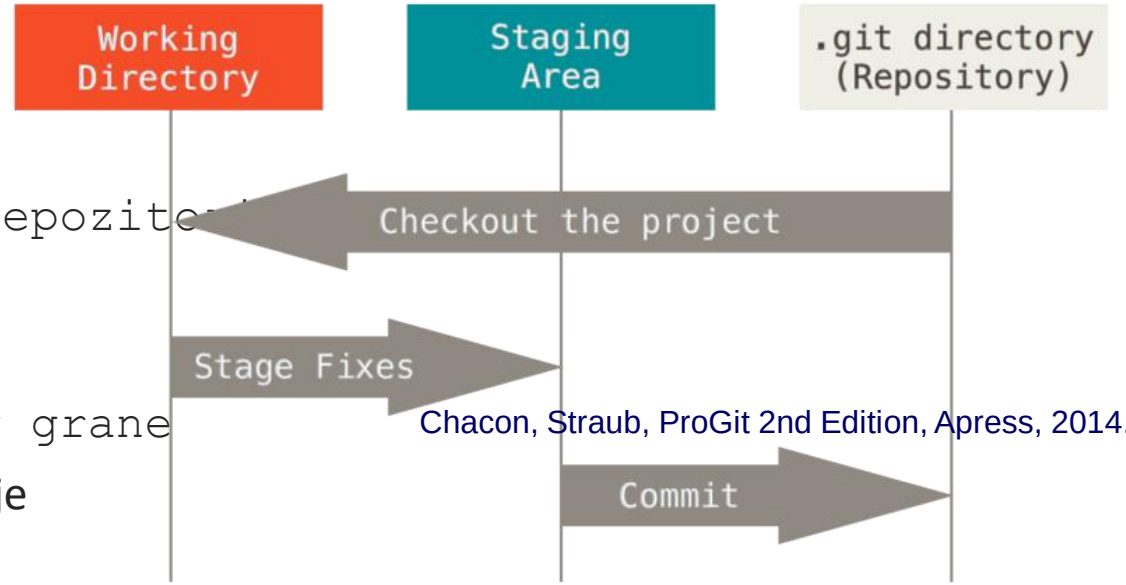
Pojava konflikta u distribuiranim sustavima

- Kod distribuiranih sustava češće dolazi do konflikta
  - Različiti sadržaji repozitorija kod pojedinih korisnika + neovisne promjene
  - Potrebno spajanje promjena
- Svaka promjena mora imati jedinstveni identifikator
- Redni broj nije dovoljan
  - potrebna složenija oznaka (jedinstveni kod, npr. hash)



Tipični scenarij rada s Git-om

- Kloniranje centralnog ili inicijalizacija samostalnog repozitorija
  - `git clone adresa_repozitorija`
  - `git init`
- Promjena grane
  - `Git checkout naziv grane`
- Dohvat i primjena zadnje verzije grane centralnog repozitorija
  - `git pull origin naziv grane`
  - *origin* označava udaljeni repozitorij, naziv glavne grane je master ili main
- Datoteke čije se promjene žele evidentirati dodaju se u međuspremnik (Staging Area)
  - `git add naziv datoteke ili putanja do više datoteka`
- Korisnik
  - potvrđuje `git commit –m "opis promjena"`
  - Ili odbacuje `git checkout naziv datoteke ili putanja`
- Korisnik šalje svoje promjene na server
  - `git push origin naziv grane`



Chacon, Straub, ProGit 2nd Edition, Apress, 2014.

Neki od ostalih pojmova i literatura

- Stash i Unstash
  - Privremeno spremanje izmjena, ali bez potvrđivanja promjena (tj. bez *commita*)
  - Slično kao Shelve/Unshelve kod TFS-a
- Pull request/Merge request
  - Zahtjev nekog korisnika za pregledom njegovog koda i spajanje s nekom drugom (najčešće glavnom) granom

- Literatura za Git
  - http://tkrajina.github.io/uvod-u-git/git.pdf
  - https://git-scm.com/book/en/v2/
  - https://www.atlassian.com/git/tutorials/learn-git-with-bitbucket-cloud

Gradnja sustava

- Prikuplja relevantne objekte i proizvodi određenu verziju softvera
  - Dohvat određene verzije iz repozitorija
  - Kompilacija i povezivanje biblioteka
  - Pokretanje testova (ako postoje)
  - Objava verzije
    - Interno
    - Spremno za instalaciju kod korisnika
    - Automatska isporuka korisnicima
- Pokretanje može biti
  - na zahtjev
  - dnevno (npr. *nightly build*)
  - nakon svake promjene koda u repozitoriju
    - Ili nakon nekoliko akumuliranih promjena
  - …

Okruženja u procesu razvoja i isporuke (1)

- Development
  - razvojno okruženje pojedinog programera, odnosno ono u kojem se odvija razvoj
- Testing
  - Definiran skup testnih podataka i imitacija ovisnosti u cilju osiguranja predvidivosti i ponavljanja testiranja
  - Može se, odnosno poželjno je automatizirati
    - Lokalno testno okruženje i ono s odgovarajućim alatom za kontinuiranu integraciju

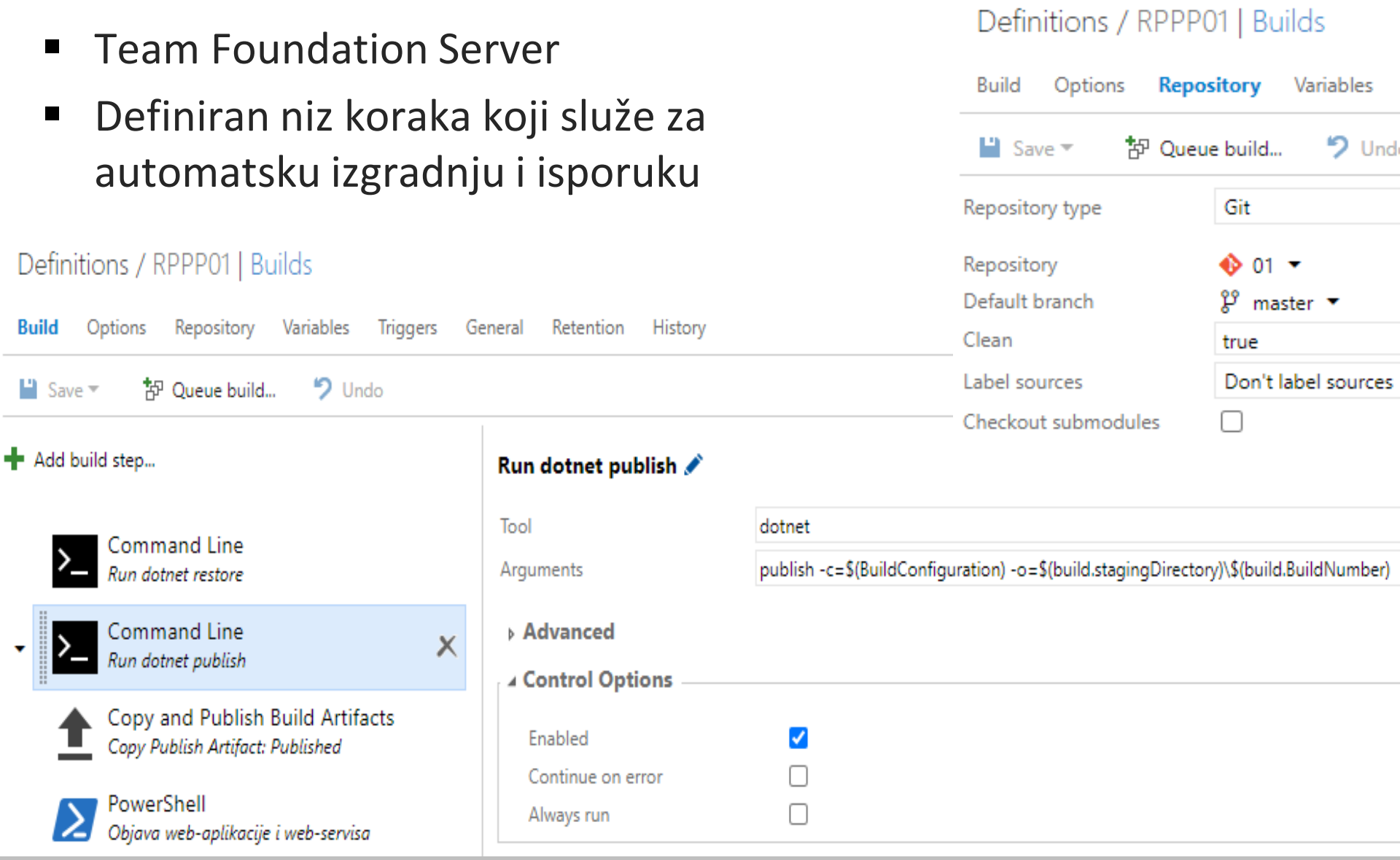
Okruženja u procesu razvoja i isporuke (2)

- Staging
  - okruženje nalik produkcijskom ili kopija produkcijskog okruženja
  - za detektiranje pogreški koje promaknu u lokalnom razvojnom okruženju, vezano za mrežne postavke, ovisnosti o instaliranim bibliotekama i slično
  - pogodno za testiranje performanci i ponašanja pri većem opterećenju
  - obično namijenjeno za internu upotrebu, ali može poslužiti i za pretpregled novih funkcionalnosti
- Production

- U idealnom slučaju ova okruženja su potpuno odvojena, a dijelovi koda ili ponašanje parametrizirano varijablama okruženja

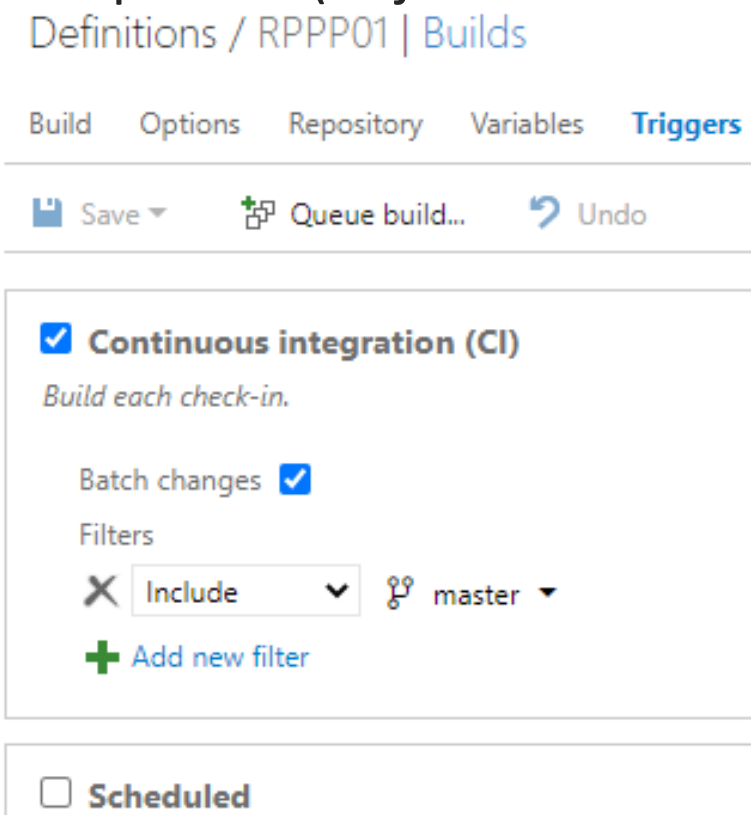
Primjer kontinuirane isporuke na RPPP-u do 2020/21 (1)

- Team Foundation Server
- Definiran niz koraka koji služe za automatsku izgradnju i isporuku



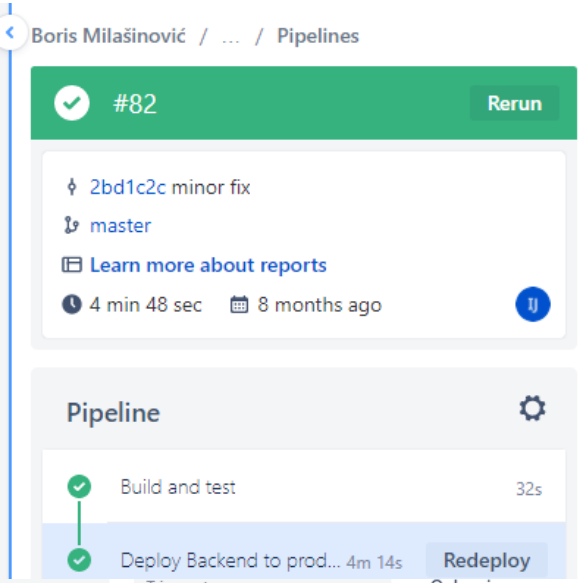
Primjer kontinuirane isporuke na RPPP-u do 2020/21 (2)

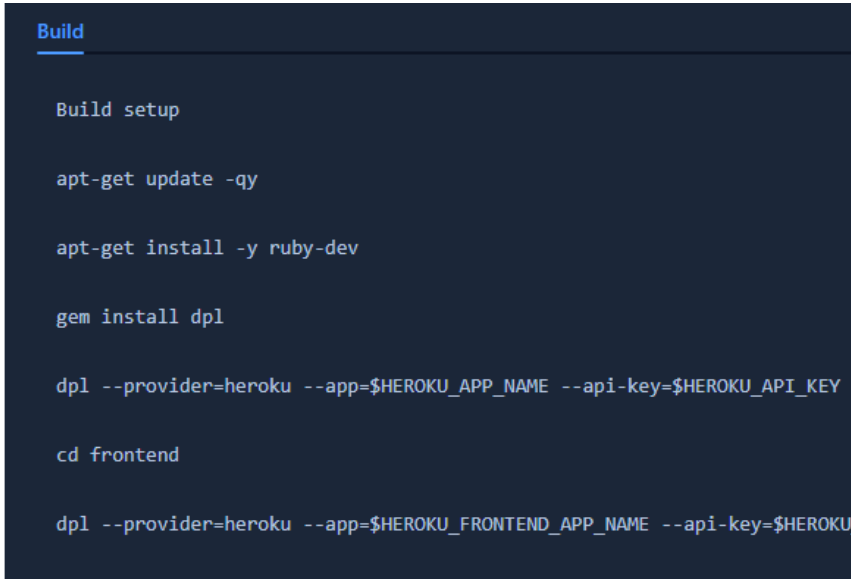
- Izgradnja se pokreće nakon svake promjene u repozitoriju
  - Continuous integration (CI)
    - Iako bez testova to je više Continuous build nego integration
- U definiciji s prethodnog slajda uključena i isporuka (objava web-aplikacije)
  - Continuous delivery (CD)
    - Nova verzija se može odmah dostaviti korisnicima na provjeru u uvjetima nalik produkcijskim
  - Continuous deployment (CD)
    - Nova verzija je odmah objavljena u produkcijskom okruženju
- Prikazani koncept može biti primijenjen i za druga razvojna okruženja

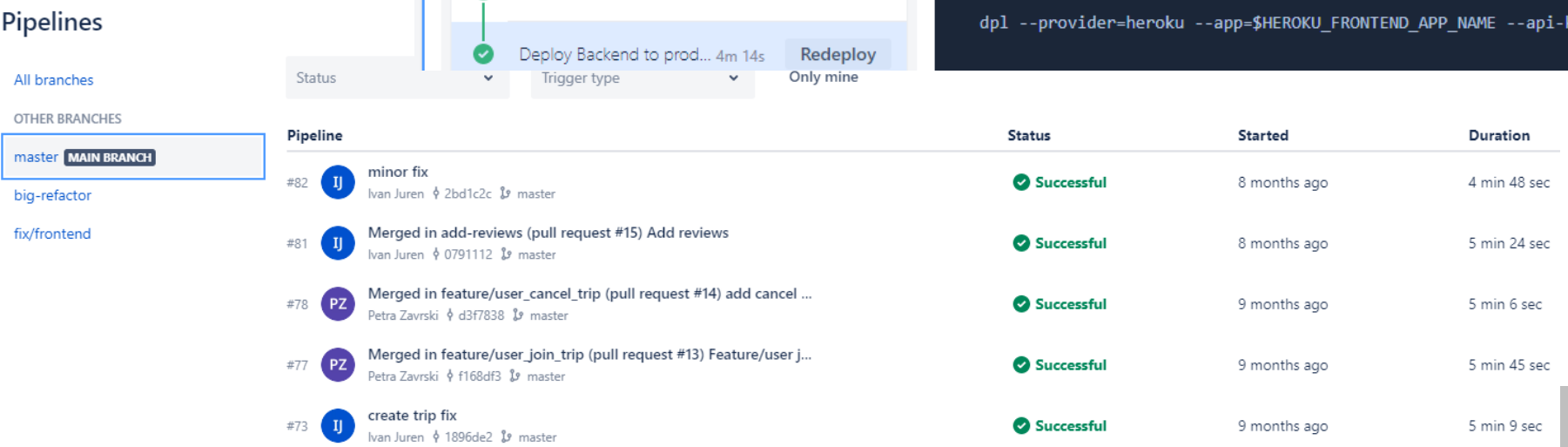


Alati za kontinuiranu integraciju i isporuku

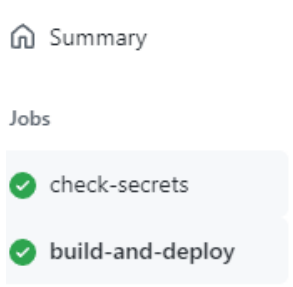
- BitBucket Pipelines, Azure Pipelines, Semaphore CI, GitHub Actions,
- Jenkins, Travis, …

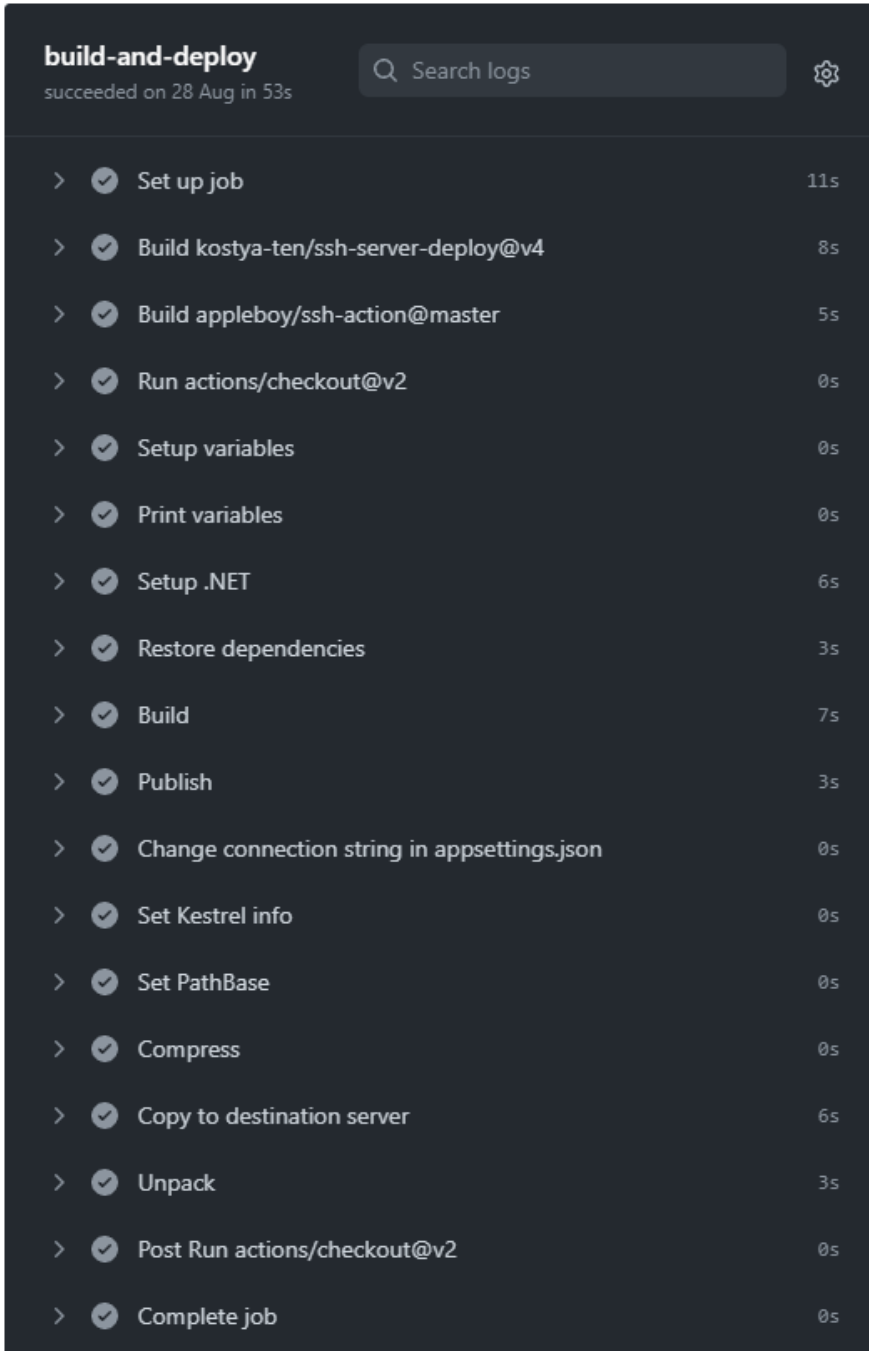






Kontinuirana isporuka na primjeru domaćih zadaća (1)





- Github Actions
  - Niz koraka koji se izvršavaju nakon promjene grane *master*
- Kombinacija Githubove usluge u oblaku
  - *init Docker Container, … restore libraries, build, publish,*

… i „vlastitog hardvera” (fantom.fer.hr VPS, Srce)

Kontinuirana isporuka na primjeru domaćih zadaća (2)

- Primjer: 🗁 https://github.com/FER-RPPP/HomeworkTemplate/blob/master/.github/workflows/dotnet.yml
- Detaljnije prilikom izrade web-aplikacije

```
build-and-deploy:
  needs: check-secrets
  if: needs.check-secrets.outputs.data-available == 'true'
  env:
    # Leave project folder as-is, except if you main project has been renamed (in that ca
    PROJECT-FOLDER : RPPP-WebApp
  runs-on: ubuntu-latest
  steps:
  - uses: actions/checkout@v2
  - name: Setup variables
    id: variables
    run: |
      echo "::set-output name=group::${{ env.GROUP }}"
      echo "::set-output name=port::$((50000 + 100 * ${{ env.GROUP }}))"
      echo "::set-output name=zip-filename::G${{ env.GROUP }}-V$GITHUB_RUN_NUMBER-$(date
      echo "::set-output name=publish-folder::${{ github.workspace }}/published"
  - name: Print variables
    run: |
      echo "Group: ${{ env.GROUP }}"
      echo "Zip filename: ${{ steps.variables.outputs.zip-filename }}"
      echo "Publish folder: ${{ steps.variables.outputs.publish-folder }}"
  - name: Setup .NET
    uses: actions/setup-dotnet@v1
    with:
      dotnet-version: 5.0.x
  - name: Restore dependencies
    run: |
      cd ${{ env.PROJECT-FOLDER }}
      dotnet restore
  - name: Build
    run: |
      cd ${{ env.PROJECT-FOLDER }}
      dotnet build --configuration Release --no-restore
```