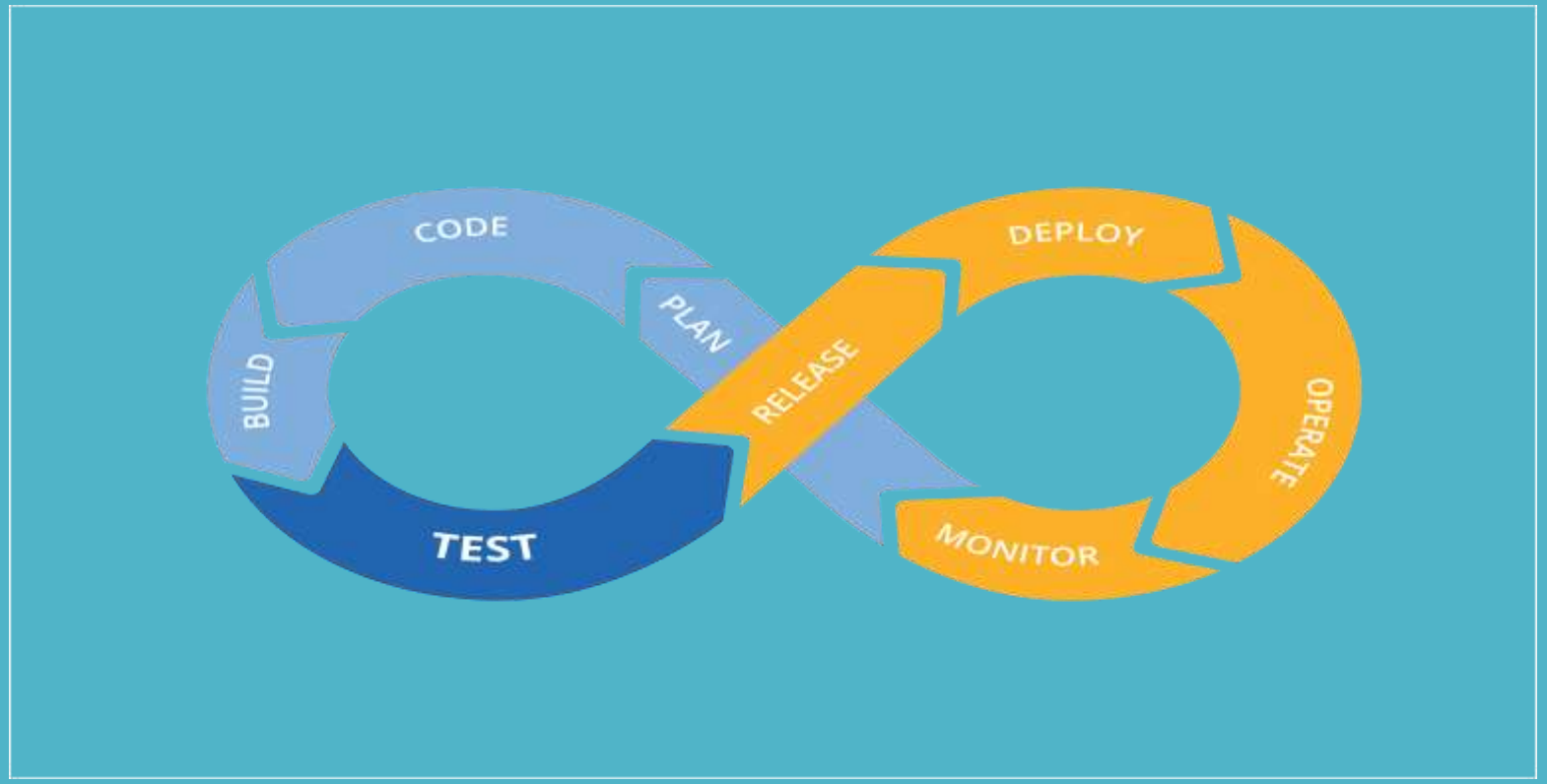
DevOps metodologija



DevOps metodologija je skup praksi koje kombinuju razvoj softverskih rešenja i IT operacija.

To je jedna od najnovijih metodologija koje razvojni timovi koriste prilikom razvoja softverskih rešenja.

Počeci DevOps metodologije bili su 2008. godine, a tek u poslednjih nekoliko godina stekla je veću popularnost unutar IT zajednice.

Ona dobija sve veću popularnost zbog lakoće korišćenja, korišćenja najmonišnijih tehnologija i alata i brzine razvoja softverskih rešenja na ovaj način.

DevOps metodologija je popularna kako među razvojnim timovima, tako i među krajnjim korisnicima softverskih rešenja, pre svega za uštedu vremena ali i neophodne resurse za razvoj i korišćenje softverskih rešenja.

Šta je DevOps?

DevOps je skup praksi koje kombinuju razvoj softverskih rešenja i IT operacija. Ova metodologija, iako je svoje početke doživela 2008. godine, poslednjih godina je samo ostvarila veću popularnost unutar IT zajednice.

Tvorac DevOps-a je Patrik Deboa. On je došao na ideju da DevOps radi kao tester na projektu 2007. Na Agile Infrastrukturnoj konferenciji 2008. upoznaje Endrua Šafera, sa kojim razgovara o svojim idejama. Veće interesovanje za DevOps unutar krugova IT zajednice počelo je nakon predstavljanja "10+ Deploys a Day: Dev and Ops Cooperation at Flickr" 2009. Iste godine, u Belgiji je održana prva DevopsDays konferencija.

Od 2010. godine ova konferencija se održava u mnogim različitim gradovima širom sveta. Alana Braun je 2012. godine objavila prvi Izveštaj o stanju DevOpsa, koji se objavljuje svake godine od 2014. godine. Projekat Feniks, koji su uradili Džin Kim, Kevin Behr i Džordž Spaford, objavljen je 2013. godine i stekao je veliku popularnost i doprineo je upoznavanja šire publike sa konceptom DevOpsa.

Forester Research proglasio je 2017. godinu "Godinom DevOps-a" i saopštio da, prema njihovom istraživanju, skoro 50 odsto kompanija sprovodi DevOps. Od tada sve više kompanija i razvojnih timova primenjuje i koristi DevOps metodologiju u razvoju softverskih rešenja.

U tradicionalnom timu za razvoj softvera najčešća je osoba koja razvija softversko rešenje i osoba zadužena za njegovu isporuku i implementaciju dve različite osobe. Sa pojavom novih načina rada i tehnologija, kao što su metode stalne integracije i isporuke, bilo je moguće da oba ova zadatka obavlja jedna osoba koristeći određene alate i tehnologije. Osoba koja radi na ovoj poziciji često nosi titulu DevOps inženjera.

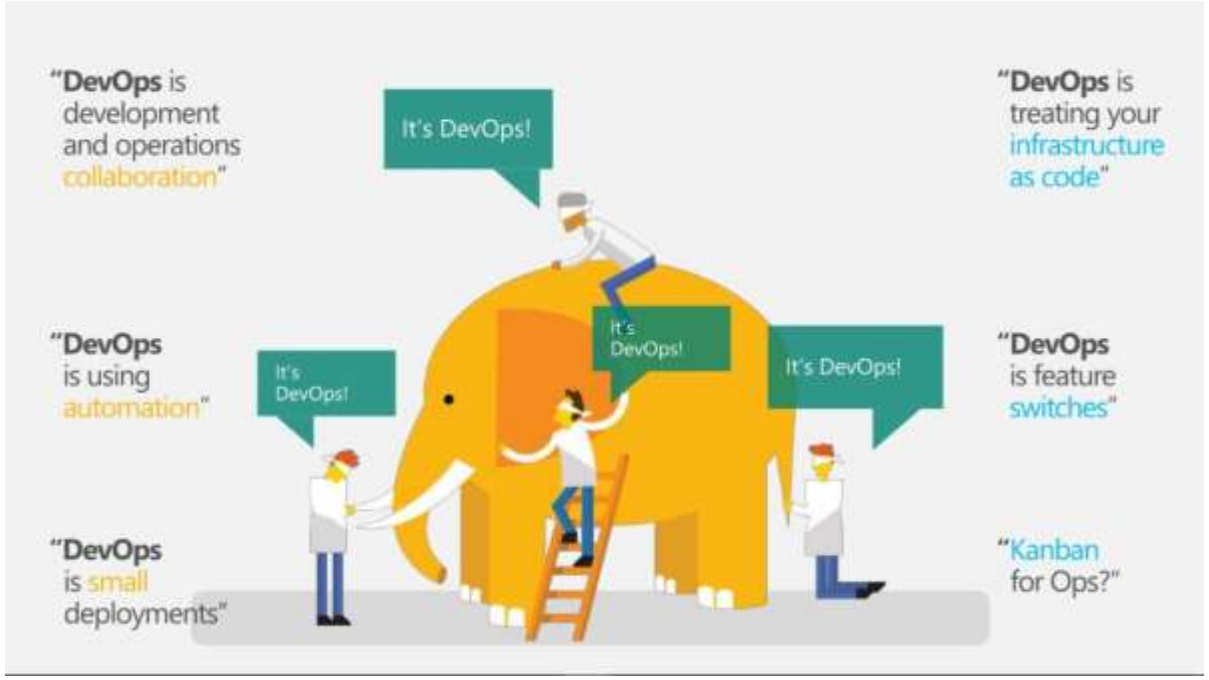
Zadaci DevOps inženjera su, pored razvoja novih funkcionalnosti softverskog rešenja ili ispravljanja propusta u postojećim, onih koje bi inače obavljala osoba zadužena za operativni deo sistema.

Tako su DevOps inženjeri zaduženi za podešavanje i održavanje sistema i alata, kao i sama softverska rešenja.

Pored dobrog poznavanja celokupnog sistema i softverskog rešenja, devOps inženjer mora imati i dobro poznavanje sistema na kojima se radi softversko rešenje, npr. Linux operativni sistem na lokalnom serveru ili u oblaku, baze podataka, alati za upravljanje korisničkim dozvolama i pravima servera, web serveri itd.

DevOps inženjeri, pored dobrog poznavanja alata koje koriste za razvoj softverskih rešenja, moraju imati i dobro poznavanje DevOps alata.

Na primer, potrebno je da budu upoznati sa distribuiranim sistemima za upravljanje izvorima (**GitHub, Mercurial**), cloud computing sistemima (**AWS, Azure, Google Cloud**), sistemima virtuelizacije (**Docker**) i sistemima grupisanja (**Kubernetes**), alatima za automatizaciju infrastrukture (**Chef, Puppet, Ansible**), alatima za korišćenje kontinuirane integracije i kontinuiranim načinom isporuke (**Jenkins, Travis CI**) i alatima za posmatranje.



Slika 1 prikazuje procese koji se odvijaju prilikom razvoja softverskog rešenja u razvojnim timovima koji rade sa DevOps metodologijom razvoja. Procesi prikazani na slici su: **planiranje softverskog rešenja**, **njegova izgradnja**, **kontinuirana integracija, isporuka softverskog rešenja, njegovo korišćenje i kontinuirano primanje povratnih informacija o softverskom rešenju od krajnjeg korisnika**. Ovi procesi se odvijaju ciklično, to jest u beskonačnoj petlji kao što je prikazano na slici. Svi ovi procesi praćeni su kontinuiranom komunikacijom sa krajnjim korisnikom i praćenjem svih aktivnosti tokom razvoja softverskog rešenja.

DevSecOps

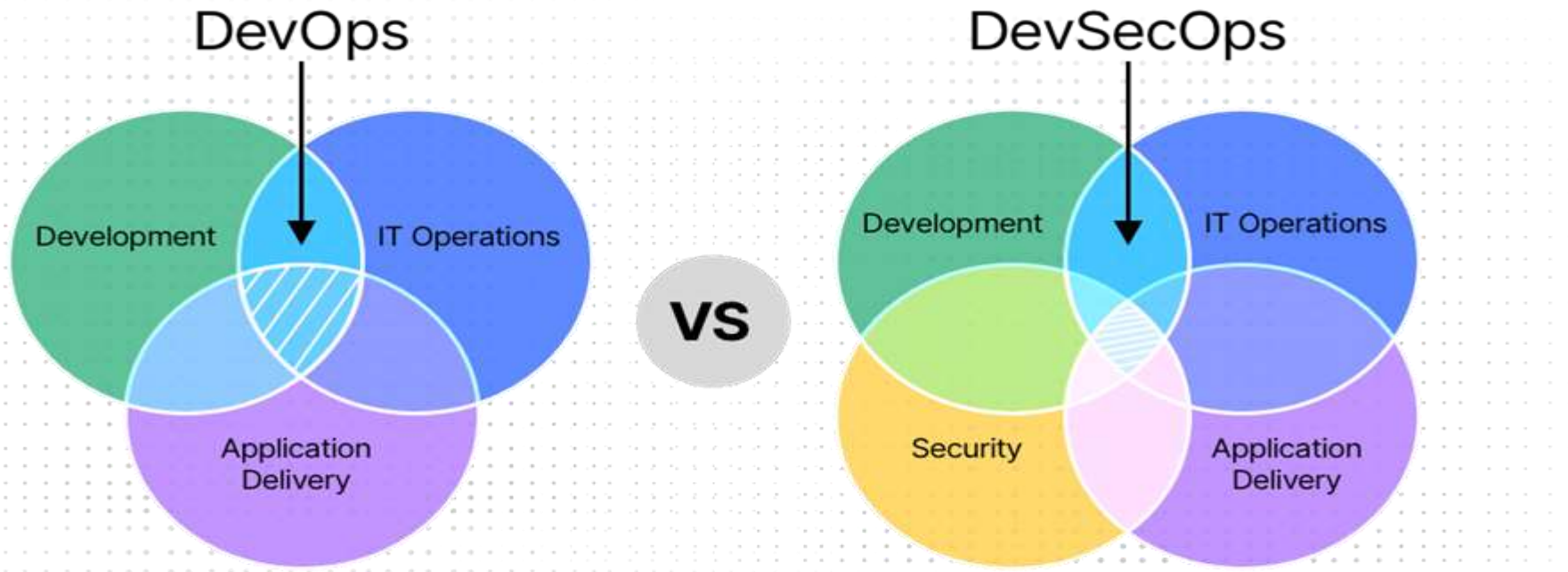
U poslednje vreme se pojavljuje i termin **DevSecOps** unutar IT zajednice. DevSecOps - razvoj, bezbednost, operacije.

Ovaj pojam predstavlja prirodnu evoluciju DevOps-a na način da se kroz tipične procese DevOps automatizacije bezbednosne funkcije integrišu u softversko rešenje.

U prošlosti, bezbednosne funkcije su vrlo često bile samo dodatak softverskom rešenju. U današnjem svetu informacionih tehnologija, kada se sve menja veoma brzo i svakodnevno se pojavljuju novi bezbednosni problemi, postoji potreba da bezbednosne funkcije budu sastavni deo softverskog rešenja.

DevSecOps integriše bezbednosne funkcije u sve Agile i DevOps procese i alatke. Bavi se rešavanjem bezbednosnih problema čim se pojave, kada ih je lakše, brže i jeftinije ispraviti. Infrastrukturna bezbednosna i softverska rešenja u DevSecOps-u su zadužena za sve članove razvojnog tima, a ne pojedince.

Slika pokazuje razlike između tradicionalnih DevOps i DevSecOps. Na slici je moguće videti Venn dijagrame koji pokazuju da je DevOps skup razvoja, operacija i aktivnosti vezanih za isporuku softverskog rešenja krajnjem korisniku (isporuka aplikacija), dakle da je DevSecOps skup svih navedenih aktivnosti sa dodatim aktivnostima vezanim za bezbednost i na taj način objašnjava razliku između ova dva koncepta.



DevOps Always – Git

Primarni alat koji koriste razvojni timovi koji praktikuju DevOps metodologiju razvoja programskih rešenja je distribuirani sistem upravljanja izvorima.

Najpopularniji takav sistem unutar IT zajednice je Git.

Git sistem za upravljanje izvorima nastao je 2005. Nastao je kao odgovor na nedostatke prethodno postojećih sistema.

Git sistem može da radi lokalno, na računaru programera, na lokalnom serveru ili na sistemu u oblaku. Najpopularnija implementacija Git-a u oblaku je GitHub.

Postoje i druge implementacije, kao što su GitLab, Bitbucket, SourceForge i druge

Git - Github

Git kreira udaljeno skladište za softversko rešenje na serveru, u okviru Git sistema. Sav kod i sve promene koje su napravljene na njemu šalju se u udaljeno skladište. Git čuva sve verzije koda koji mu je poslat. Edit je poslat u udaljeno skladište

Git Commit. Svaka izmena poslata Giti mora biti dokumentovana. Dokumentacija o podnetim izmenama je napisana kao komentar u Git commit i zajedno sa izmenom se šalje u udaljeno skladište. Sistem za upravljanje izvornim kodom pamti sve promene u program.

Jedna od karakteristika skoro svih distribuiranih sistema za upravljanje izvorima, uključujući i Git, je grananje (**Branch**).

Grananje vam omogućava da uskladištite nekoliko različitih "kopija" depoa i praćenje svih promena koda poslatih određenoj grani.

Depo se sastoji od glavne grane, u kojoj se obično nalazi proverena, stabilna verzija koda.

Dodatne grane sadrže verzije koda na kojima programeri aktivno rade i menjaju ih. Kada se proveri kôd iz dodatnih grana, promene se objedinje u glavnu granu.

Git - Github

Iz udaljenog skladišta, programeri mogu da preuzmu kôd softverskog rešenja na lokalni računar ako imaju dozvolu za preuzimanje.

Pored programera, kôd iz udaljenog skladišta može da preuzme drugi sistem ili alatka, ako ima dozvolu za preuzimanje.

To omogućava integraciju sa drugim sistemima ili alatkama, kao što su cloud računarski sistemi ili alatke za testiranje softverskog koda, u svrhu kontinuirane integracije / neprekidnih procesa isporuke.

Cloud računarski sistemi

U cilju olakšavanja razvoja softverskog rešenja, ali i izvršenja i korišćenja već završenog softverskog rešenja, koriste se cloud računarski sistemi. Pre pojave cloud računarskih sistema, razvojnih timova i korisnika softverskih rešenja, sve infrastrukturne i sistemske resurse potrebne za razvoj, izvršenje i korišćenje softverskog rešenja morali su sami da obezbede i održe.

Ovakav način rada bio je skup i neefikasan, zahtevao je kupovinu i održavanje infrastrukturnih i sistemskih resursa, kao i timove sistemskih inženjera koji planiraju, grade i održavaju infrastrukturu i resurse.

Korišćenjem cloud računarskih sistema, troškovi izgradnje i održavanja infrastrukture i resursa prenose se na nekog drugog, što omogućava razvojnim timovima i krajnjim korisnicima da uštede vreme i novac i optimizuju troškove razvoja i korišćenja softverskog rešenja.

Glavna **prednost** korišćenja takvih sistema je odsustvo potrebe za kupovinom i održavanjem resursa, kao što su serveri ili sistemi za skladištenje podataka, kao i specijalisti neophodni za planiranje i održavanje ovih resursa.

Pored toga, cloud računarski sistemi imaju veliku dostupnost kao prednost, tako da su resursi dostupni sa bilo kog mesta i u svakom trenutku, bez potrebe za podešavanjem dodatne mrežne infrastrukture.

Prednost korišćenja cloud computing sistema je plaćanje tačno onoliko resursa koliko se koristi, dok infrastruktura koja nije deo cloud computing sistema generiše troškove bez obzira na to da li se koristi ili ne.

Još jedna prednost je jednostavan i brz zakup dodatnih resursa i infrastrukture na sistemu u cloud computing-u, u roku od nekoliko sekundi i uz minimalne dodatne troškove. Cloud računarski sistemi takođe imaju integrisana bezbednosna rešenja, tako da su infrastruktura i resursi na cloud računarskim sistemima automatski osigurani protiv potencijalnih bezbednosnih problema.

Cloud computing sistema takođe ima neke **potencijalne nedostatke**.

Poverljivost podataka je pod znakom pitanja, postoji mogućnost da neovlašćeno lice pristupi podacima kojima ne bi trebalo da pristupi.

Za povezivanje sa Internetom potreban je pristup infrastrukturi i resursima.

Postoji mogućnost nedostupnosti resursa i infrastrukture u slučaju problema sa cloud računarskim sistemom, iako se po pravilu takvi problemi retko i brzo ispravljaju.

Jedan od problema je i "Zaključavanje dobavljača", npr. Iako je cena resursa i infrastrukture na cloud računarskim sistemima generalno niža od korišćenja lokalnih resursa i infrastrukture, zbog greške korisnika ili neovlašćenog pristupa od strane trećeg lica, postoji mogućnost generisanja velikih troškova.

Najpopularniji računarski sistemi u oblaku su AWS (Amazon Web Services), Microsoft Azure i Google Cloud. U prvom kvartalu 2021. godine, AWS je posedovao 32 odsto tržišta cloud computing Sistema.

Alati za korišćenje kontinuirane integracije i kontinuiranog načina isporuke

Dženkins (Jenkins)

Alati za korišćenje metoda kontinuirane integracije i kontinuirane dostavne pomoći programerima, rad na razvoju softverskih rešenja, automatizacija procesa razvoja softverskih rešenja, procesi testiranja i procesi isporuke softverskog rešenja krajnjem korisniku.

Jedan od najpoznatijih i najpopularnijih alata za korišćenje kontinuirane integracije i kontinuiranog načina isporuke je Dženkins sistem automatizacije. Ova alatka je dostupna besplatno i može se izvršiti na svim popularnim operativnim sistemima. Dženkins je softversko rešenje otvorenog koda i zato ima veliku podršku IT zajednice. Jednostavan je za podešavanje i korišćenje, a njegove osnovne mogućnosti se mogu proširiti sa mnogo dostupnih dodataka.

Alati za korišćenje kontinuirane integracije i kontinuiranog načina isporuke

Dženkins podržava integraciju sa većinom drugih distribuiranih sistema upravljanja izvorima, integraciju sa cloud računarskim sistemima kao i drugim DevOps alatima. Pored Dženkinsa, koriste se i drugi alati, od kojih su neki GitHub Actions, GitLab, AWS CodePipeline, Bamboo i drugi.

Ostali DevOps alati

Pored navedenih alata i sistema, DevOps inženjeri često koriste sisteme za **virtuelizaciju** (Docker, Wagrant), **sisteme grupisanja** (Kubernetes, Docker Swarm) i alate za **automatizaciju infrastrukture** (Chef, Puppet, Ansible) i alate za posmatranje performansi softverskih rešenja.

Docker

Docker sistem virtuelizacije je alatka koja olakšava programerima da razviju softversko rešenje, kao i krajnjem korisniku da ga implementiše i izvrši. Doker kontejneri su virtuelne mašine koje se pokreću i izvršavaju na domaćinu. Doker kontejneri sadrže operativni sistem i sve druge uslužne usluge neophodne za izvršavanje softverskog rešenja koje se izvršava u kontejneru Doker.

Prednost korišćenja Dokerovih i Doker kontejnera je mogućnost definisanja predložaka sa uputstvima koja služe za automatizaciju lansiranja Dokerovih kontejnera i softverskih rešenja koja se pokreću unutar njih.

Takođe, prednost Dokera

Kontejneri su takođe njihove male veličine, kao i mala potrošnja sistemskih resursa. Više doker kontejnera može da se pokrene na jednom računaru. Doker kontejneri mogu da se pokreću i koriste lokalno, kao i na udaljenim serverima ili računarskim sistemima u oblaku.

Doker kontejneri se mogu koristiti nezavisno ili u grupama (klasterima). Klasteri su grupe identičnih kontejnera koji se okreću na istom računaru ili serveru. Svrha klastera je da podeli rad između pojedinačnih kontejnera, kako bi se poboljšala dostupnost pojedinačnih kontejnera, kao i brzina izvršavanja poslova.

Svaki kontejner u klasteru radi manji deo ukupnog posla, tako da je potrebno vreme da se ukupan posao manje uradi. Ako jedan kontejner postane nedostupan zbog opterećenja ili zbog greške, poslovi se distribuiraju u druge raspoložive kontejnere. Jedan od najpoznatijih i najpopularnijih alata za izgradnju, održavanje i upravljanje klasterima kontejnera je **Kubernetes.**

Alati za automatizaciju infrastrukture

Alati za automatizaciju infrastrukture, od kojih su neki **Kuvar, Lutkarski i Ansible,** omogućavaju automatizaciju raspoređivanja infrastrukture, upravljanja infrastrukturom i održavanja.

Prednost korišćenja takvih alatki je mogućnost pisanja predložaka u skladu sa kojima se ove alatke posnište, upravljaju i održavaju infrastrukturu. Takve alatke omogućavaju DevOps inženjerima da za kratko vreme automatski podese i pokreću više virtuelnih servera ili Doker kontejnera, bez potrebe za ručnim podešavanjem i pokretanjem.

DevOps inženjeri takođe pronalaze alate za posmatranje performansi softverskih rešenja (eng. primena alatki za praćenje performansi). Ovi alati u pozadini konstantno posmatraju performanse softverskih rešenja i obaveštavaju DevOps inženjere o potencijalnim smanjenim performansama. Takvi problemi se mogu rešiti ručno, iznajmljivanjem dodatne infrastrukture i resursa, a rešavanje takvih problema može biti automatizovano. Neke od ovih alatki su **Datadog, NewRelic, AppDynamics** i druge.

Svi najpoznatiji cloud računarski sistemi podržavaju upotrebu svih navedenih alata.

Životni ciklus razvoja softvera (SDLC)

Životni ciklus razvoja softverskog rešenja je proces koji rezultira softverskim rešenjem maksimalnog kvaliteta, uz minimalne troškove, u najkraćem mogućem roku.

Faze životni ciklus razvoja softvera su najčešće:

- Planiranje i analiza korisničkih zahteva,
- Definisanje zahteva,
- Projektovanje,
- Razvoj rešenja,
- Testiranje rešenja i isporuka
- Implementacija softverskog rešenja.

U prvoj fazi analiziraju se zahtevi korisnika i pravi se plan razvoja softverskih rešenja. Ovo je najvažnija faza životnog ciklusa razvoja softverskog rešenja.

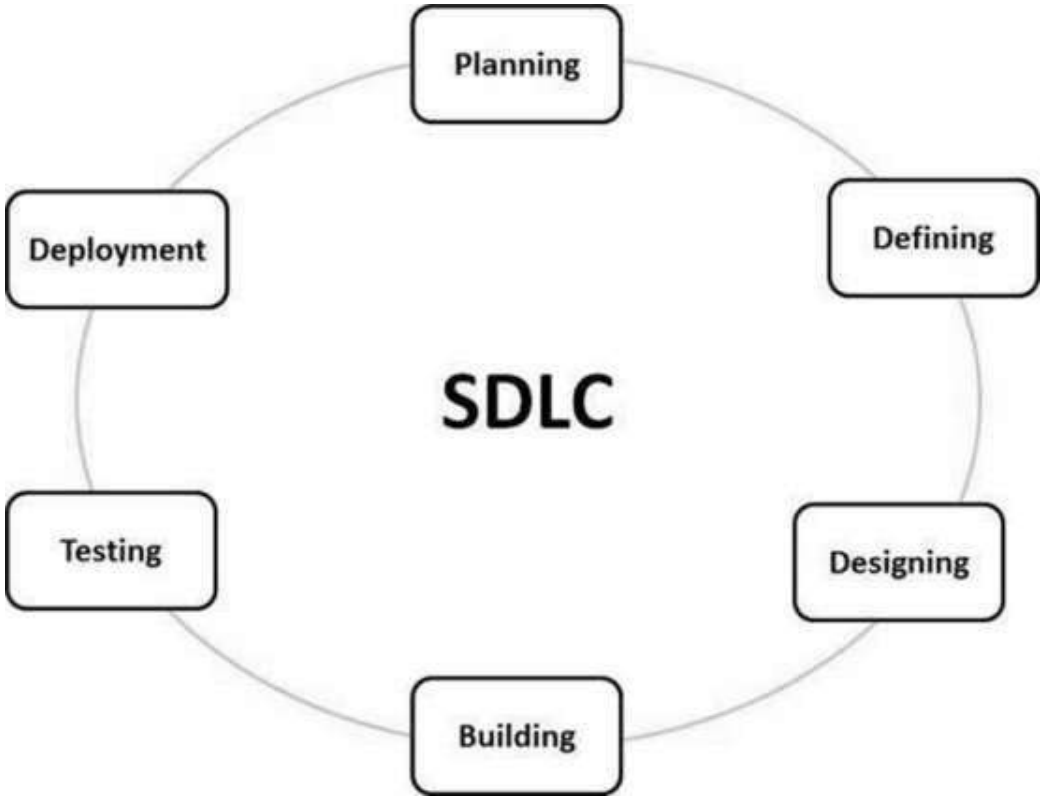
U fazi definisanja korisničkih zahteva, korisnički zahtevi su jasno definisani i dokumentovani, iz kojih je arhitektura softverskog rešenja dizajnirana u sledećoj fazi dizajna.

Faza izgradnje obuhvata sve aktivnosti na izgradnji softverskog rešenja.

Nakon faze izgradnje, u sledećoj fazi, testira se softversko rešenje i uočeni nedostaci se dokumentuju i ispravljaju.

Završna faza je faza isporuke softverskog rešenja krajnjem korisniku i implementacije softverskog rešenja.

Životni ciklus razvoja programskog rješenja



Metodologije razvoja softverskih rešenja

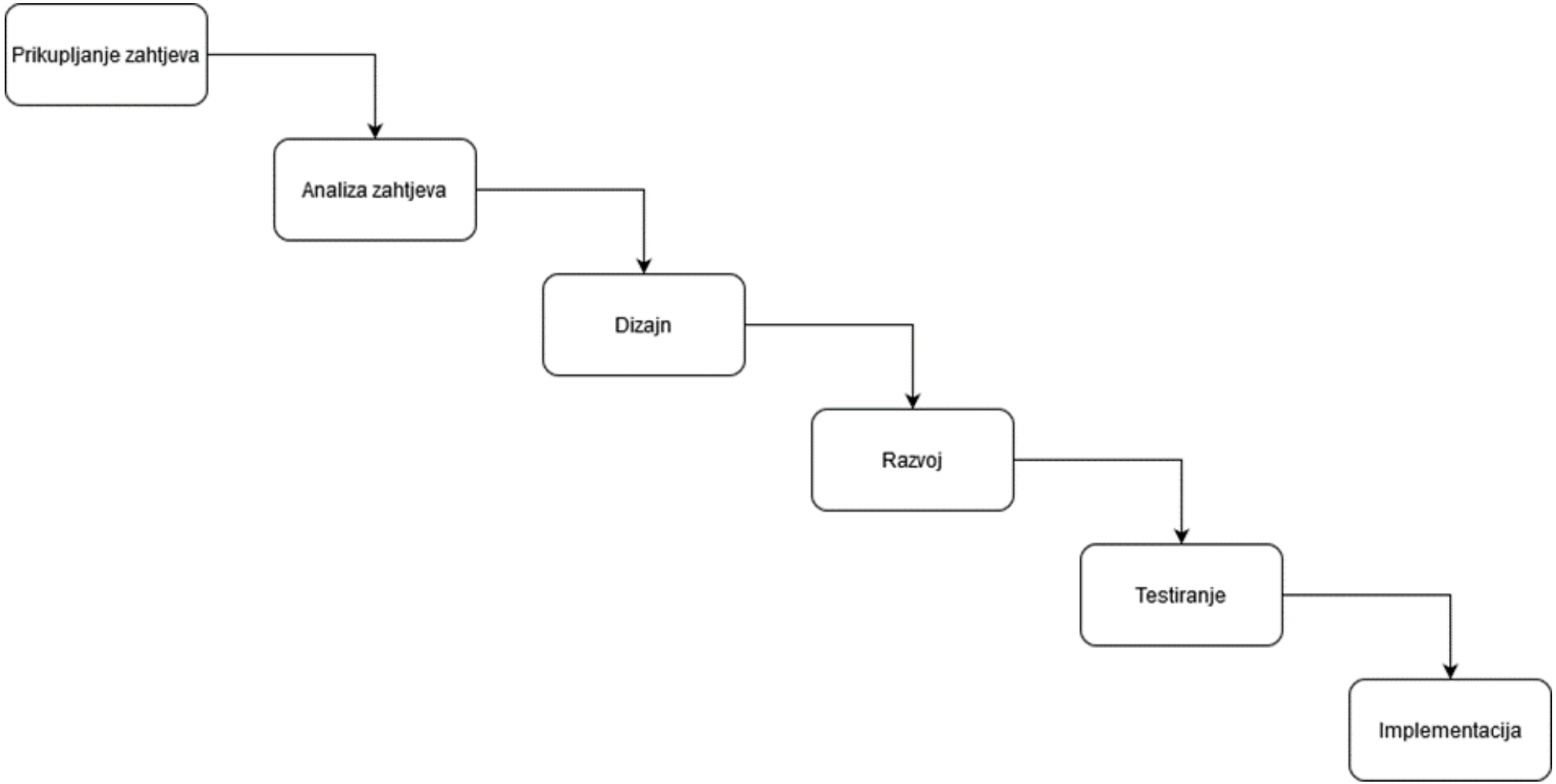
Postoje različiti pristupi razvoju softverskih rešenja koja su opisana metodologijama razvoja softverskih rešenja.

Prva metodologija nastala je **Waterfall**, i možemo reći da pripada prvoj generaciji metodologija za razvoj softverskih rešenja, zajedno sa iterativnom, evolucionom i spiralnom metodologijom.

Druga generacija metodologija, koje se i danas koriste, pripada agilnim metodologijama (**Scrum, Kanban**).

Za DevOps se može kažeti da je prirodno proširenje agilne metodologije i da pripada trećoj generaciji metodologija za razvoj softverskih rešenja.

Metodologija vodopada



U waterfall metodologiji razvoja softverskih rešenja, faze razvoja se dešavaju upravo

određenim redosledom i isključivo jedno za drugim, bez preskakanja faza ili povratka u prethodnu fazu. Metodologija vodopada je procesno orijentisana, što znači da se procesi odvijaju sekvencijalno i da nisu fleksibilni.

Waterfall metodologija razvoja softvera u današnje vreme je prilično zastarela, ali se i dalje može koristiti za kreiranje manjeg i manje komplikovanog softverskog rešenja.

Agilna metodologija

Tradicionalne metode razvoja softverskih rešenja kao što je Vodopad nisu bile prilagođene savremenim korisničkim zahtevima, bile su prespore i ograničene. Vremenom su se pojavile nove metodologije, koje su prilagođene novim uslovima, od kojih su neki RAD (brz razvoj aplikacija) 1991, Scrum metodologija 1995, XP (ekstremno programiranje) 1996.

Godine 2001, 17 programera sastalo se u Juti da razgovara o ovim novim metodologijama. Zaključili su da metode koje koriste imaju neke zajedničke karakteristike. Rezultat njihove diskusije je Manifest za razvoj agilnog softvera, manifest koji objašnjava principe agilne metodologije razvoja softvera:

"Tražimo bolje načine razvoja softvera razvojem softvera i pomažemo drugima da ga razviju. Ovim radom naučili smo da više cenimo:

1. Ljudi i njihovi odnosi jedni sa drugima osim procesa i alata
2. Upotrebljiv softver od iscrpne dokumentacije
3. Saradnja sa klijentom od pregovaranja o ugovoru
4. Reaguje na promene nego da istraje na planu.

Drugim rečima, dok cenimo vrednosti na desnoj strani, više verujemo u one sa leve strane.

Poređenje scrum metodologije i metodologije vodopada

Scrum je najpopularnija agilna metodologija za razvoj softverskih rešenja. Nastao je devedesetih godina prošlog veka, a njegovi autori su Džef Saterlend i Ken Švaber.

Scrum čine Scrum timovi sa svojim ulogama, Scrum događajima, Scrum artefaktima i Scrum pravilima. Scrum čine stručnjaci čiji je zadatak da ostvare ciljeve proizvoda. Svaki Scrum tim sadrži vlasnika proizvoda, razvojni tim. Razvojni tim) i Scrum master.

Vlasnik proizvoda je odgovoran za uvećavanje vrednosti proizvoda koji nastaje kao rezultat rada Scrum tima.

Pored toga, vlasnik proizvoda je takođe odgovoran za upravljanje artiklima u zaostalim stavkama proizvoda. Zaostatak proizvoda je lista stavki koje se neprekidno ažuriraju i ažuriraju. Stavke na listi su sortirane po prioritetu .

Odgovornost Scrum majstora je ispravna implementacija Scrum-a, ne samo unutar Scrum tima već i na nivou cele organizacije. Scrum master pomaže vlasniku proizvoda da upravlja zaostalim proizvodom i uči ih kako da kreiraju jasne i precizno opisane zaostale stavke proizvoda.

Razvojni tim ili članovi razvojnog tima rade na svim aktivnostima vezanim za pravljenje pomaka posle svakog Sprint-a. Na početku svakog sprinta, zajedno sa ostalim članovima Scrum tima, slažu se na kojim stavkama iz zaostalih proizvoda će raditi u tom Sprint-u. Zadatak razvojnog tima na kraju sprinta je da isporuči upotrebljiv i maksimalan kvalitet.

Scrum increment je sprint proizvod koji je spreman za isporuku krajnjem korisniku. Sastoji se od svih artikala iz svih prethodnih sprinta. Krajnji korisnik može da odluči da li će koristiti Scrum inkrement kao softversko rešenje.

Sprint je vremenski period unapred raspoređene fiksne dužine. Tokom Sprint-a, ceo Scrum tim sprovodi aktivnosti sa ciljem da se isporuče koraci. Vlasnik proizvoda prenosi one stavke na kojima će razvojni tim raditi tokom sprinta od zaostalih proizvoda do zaostalih sprinta.

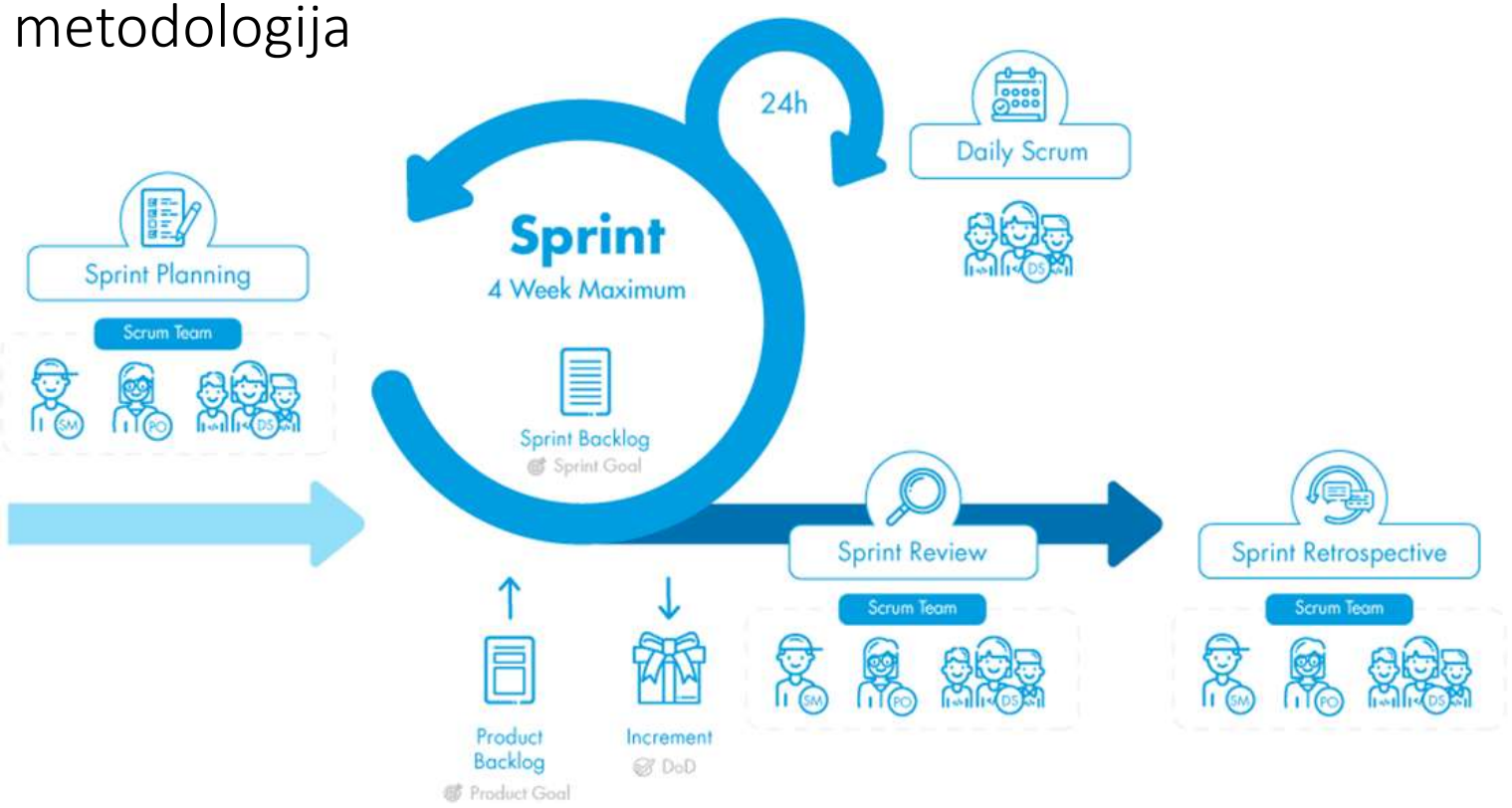
Razvojni tim kreira plan rada za dan na dnevnim Daily Scrum sastancima.

Članovi razvojnog tima informišu jedni druge o zaostalim stavkama sprinta na kojima su radili dan ranije, zaostalim predmetima na kojima planiraju da rade na dan sastanka, kao i o preprekama koje ih sprečavaju da ostvare cilj sprinta.

Nakon završetka sprinta postoji Sprint recenzija gde Scrum tim zajedno sa krajnjim korisnikom govori o aktivnostima obavljenim u prethodnom sprintu.

U pregledu sprinta, Scrum tim analizira aktivnosti obavljene u prošlom sprintu, šta je dobro urađeno u sprintu i šta je loše urađeno, a na osnovu analize planira svoj rad u budućnosti.

Scrum metodologija



Dijagram Scrum metodologije, Scrum timova i njihovih uloga, Scrum događaja i kada se dese i koliko traju, Scrum artefakti i njihova međusobna povezanost.

Svako softversko rešenje razvijeno metodologijom Vodopada prolazi kroz sve faze razvoja, svaka faza razvoja počinje isključivo završetkom prethodne faze, a razvoj se ne može vratiti u prethodnu fazu. To znači da, ako dođe do greške u nekoj od početnih faza, celo rešenje mora biti završeno kroz sve faze i, ako je potrebno, ponovo razvijeno.

Zahtevi korisnika se prikupljaju i dokumentuju na početku razvoja i koriste se za razvoj softverskog rešenja. Nakon završetka početne faze, konačni korisnik više nije uključen u razvoj rešenja. Softversko rešenje je predstavljeno korisniku samo u poslednjoj fazi razvoj, što potencijalno može dovesti do nezadovoljstva korisnika finalnim proizvodom.

Za razliku od metodologije Vodopada, Scrum je veoma fleksibilan.

Razvoj softverskog rešenja ne prolazi kroz unapred definisane faze. Softversko rešenje se obično stalno iteruje, planiraju se nove mogućnosti i dodaju, postojeće se ispravljaju ili dopunjuju.

Konačan korisnik je uključen u razvoj softverskog rešenja sve vreme u vidu vlasnika proizvoda koji, zajedno sa ostalim članovima Scrum tima, učestvuje u razvoju softverskog rešenja. Zahtevi korisnika se stalno prikupljaju i razvoj se može promeniti u skladu sa tim.

Razvoj rešenja počinje odmah, jer nema potrebe za dugoročnim sastancima i planiranjem koji su karakteristični za početne faze razvoja metodologijom Vodopada. Softversko rešenje ili njegovi delovi isporučuju se na kraju svakog sprinta u skladu sa zahtevima korisnika.

U metodologiji vodopada prestaje angažovanje krajnjeg korisnika u razvoju softverskog rešenja nakon početne faze.

Krajnji korisnik nema mogućnost da odluči da se vrati u neku od prethodnih faza životnog ciklusa razvoja softverskog rešenja u cilju ispravljanja nedostataka rešenja ili dodavanja funkcionalnosti koje nedostaju.

Prednost Scrum metodologije je stalna saradnja sa krajnjim korisnikom i konstantno sticanje povratnih informacija od krajnjeg korisnika. Uloge u timu su podeljene što omogućava timu da uradi više posla za manje vremena.

Scrum timovi moraju biti iskusni i motivisani, a ako nisu, to može dovesti do toga da se rokovi prekinu i/ili loše softversko rešenje kao rezultat njihovog rada.

Poređenje DevOps i Scrum metodologije

U tradicionalnom Scrum razvojnom timu, svaki član tima obavlja aktivnosti u svojoj oblasti ekspertize. DevOps inženjeri rade na aktivnostima razvoja softvera, kao i na aktivnostima IT operacija. Razvojni timovi su manji jer poslove više programera obavlja jedna osoba.

Pored alata za razvoj softverskih rešenja i Scrum alata, DevOps timovi koriste i različite DevOps alate i procese koji im pomažu da automatizuju aktivnosti kontinuirane integracije (CI) / Kontinuirana isporuka (CD).

U Scrum-u, pomaci se isporučuju krajnjem korisniku isključivo na kraju sprinta. Krajnji korisnik može da odluči da li će koristiti softversko rešenje. Nasuprot tome, u programu DevOps, softversko rešenje se neprekidno isporučuje korisniku.

Poređenje DevOps i Scrum metodologije

DevOps timovi daju prioritet brzini i učestalosti isporuke. Softversko rešenje se testira pre nego što se sprint završi ili pre isporuke pomaka.

U programu DevOps, softversko rešenje se testira neprekidno, nakon svake izmene kako krajnji korisnik ne bi isporučio neispravno softversko rešenje ili softversko rešenje koje ne ispunjava njegove zahteve.

Testiranje softverskih rešenja i aktivnosti isporuke softvera se obavlja ručno dok DevOps timovi automatizuju ove aktivnosti koristeći različite alate.

Scrum se fokusira isključivo na razvoj softvera dok se DevOps fokusira na razvoj softvera i IT operacije . U Scrum-u se softversko rešenje isporučuje na kraju Sprint-a, dok se u DevOps-u isporučuje neprekidno. Procesi koji se odvijaju prilikom razvoja softverskog rešenja u Scrumu nisu automatizovani, dok su u DevOps-u.

Vizualni prikaz razlika Scruma i DevOpsa

Razlike	Scrum	DevOps
Područje delovanja	Fokus na *software development*	Fokus na *software development* i na *IT Operations*
Isporuka programskog rešenja	Programsko rešenje isporučuje se na kraju *Sprinta*	Softversko rešenje se isporučuje neprekidno
Automatizacija	Procesi se ne automatiziraju	Procesi su automatizirani
Timovi	Veći timovi, svaki projektant se bavi isključivo aktivnostima iz sopstvenih oblasti aktivnosti	Manji timovi, jedan *projektant* specijalizovan za više oblasti aktivnosti