UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

MAGISTRSKO DELO

ANALIZA INFORMACIJSKE REŠITVE ZA PODPORO RAZVOJU INFORMACIJSKIH SISTEMOV V IZBRANEM PODJETJU

Ljubljana, april 2016 ALEŠ VRANEŠIČ

IZJAVA O AVTORSTVU

Spodaj podpisani Aleš Vranešič, študent Ekonomske fakultete Univerze v Ljubljani, izjavljam, da sem avtor magistrskega dela z naslovom Analiza informacijske rešitve za podporo razvoju informacijskih sistemov v izbranem podjetju, pripravljenega v sodelovanju s svetovalcem red. prof. dr. Mirom Gradišarjem.

Izrecno izjavljam, da v skladu z določili Zakona o avtorski in sorodnih pravicah (Ur. l. RS, št. 21/1995 s spremembami) dovolim objavo magistrskega dela na fakultetnih spletnih straneh.

S svojim podpisom zagotavljam, da:

- je predloženo besedilo rezultat izključno mojega lastnega raziskovalnega dela;
- je predloženo besedilo jezikovno korektno in tehnično pripravljeno v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani, kar pomeni, da sem
    - poskrbel, da so dela in mnenja drugih avtorjev oziroma avtoric, ki jih uporabljam v magistrskem delu, citirana oziroma navedena v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani, in
    - pridobil vsa dovoljenja za uporabo avtorskih del, ki so v celoti (v pisni ali grafični obliki) uporabljena v tekstu, in sem to v besedilu tudi jasno zapisal;
- se zavedam, da je plagiatorstvo – predstavljanje tujih del (v pisni ali grafični obliki) kot mojih lastnih – kaznivo po Kazenskem zakoniku (Ur. l. RS, št. 55/2008 s spremembami);
- se zavedam posledic, ki bi jih na osnovi predloženega magistrskega dela dokazano plagiatorstvo lahko predstavljalo za moj status na Ekonomski fakulteti Univerze v Ljubljani v skladu z relevantnim pravilnikom.

V Ljubljani, dne _____ Podpis avtorja:_____

KAZALO

KAZALO SLIK

KAZALO TABEL

UVOD

Informacijski sistem (v nadaljevanju IS) v podjetju naj bi čim bolje podpiral poslovne procese skozi celo svojo življenjsko dobo. To pomeni kontinuirano prilagajanje poslovanja razmeram na trgu, kar zajema spreminjanje poslovnih procesov, produktov ali storitev. Zahtevnost prilagajanja je še toliko težja, ker iz različnih virov (management, stroka, uporabniki, tehnologija, zakonodaja itd.) nenehno prihajajo želje oz. zahteve po dodatnih funkcionalnostih. Te je treba objektivno ovrednotiti ter se glede na različne kriterije smotrno odločiti o nadaljnjih korakih. Ker so si zahteve lahko tudi nasprotujoče, odločanje o vpeljavi sprememb navadno terja kompromise, katerih namen je čim ustreznejša rešitev za podjetje.

Zahtevnost prilagajanja sistema se povečuje tudi zaradi prizadevanj po optimizaciji poslovanja, zmanjševanju stroškov, učinkovitosti, vpeljevanju novih tehnologij in njihovih posledičnih učinkih, s čimer želi podjetje uveljavljati konkurenčno prednost na trgu. Če izvzamemo zakonodajne in ostale regulatorne pogoje, Khan in Zheng (2005, str. 6) menita, da podjetjem ostajata na izbiro le dve strategiji glede nadaljnjega razvoja. Prva je neprestano prilagajanje glede na naravo potreb, druga pa zadovoljitev s čedalje bolj neustreznimi rešitvami, ki jih skozi čas povzroča stagnacija prilagajanja IS. Kot navajata avtorja, so za uspešno evolucijsko prilagajanje IS in tudi za nadzor vseh aktivnostih potrebna tako tehnična kot tudi vodstvena oz. upravljavska znanja. Poudarjata, da je upravljanje evolucije IS navadno kompleksna naloga, ki jo otežuje čedalje večje število povezav znotraj IS ter tudi integracija z ostalimi sistemi oz. aplikacijami.

Cilji raziskave so:

- analizirati dejavnike, ki v obravnavanem podjetju vplivajo na počasne vpeljave sprememb v IS in podati predloge za izboljšave,
- določiti najprimernejše procese za izvajanje avtomatizacije namestitev v obravnavanem podjetju in raziskati ustrezne rešitve,
- analizirati vpliv avtomatizacije namestitev na učinkovitost izvajanja sprememb nekaterih razvojnih aktivnosti IS v obravnavanem podjetju.

Magistrsko delo tako v teoretičnem kot v praktičnem delu vsebuje znanja, pridobljena tekom podiplomskega študija, ki jih dopolnimo z večletnimi izkušnjami, pridobljenimi na delovnem mestu sistemskega inženirja v obravnavanem podjetju.

Magistrsko delo je sestavljeno iz teoretičnega ter praktičnega dela, ki sta razčlenjena na več podpoglavij. V teoretičnem delu je najprej predstavljena avtomatizacija poslovnih procesov znotraj podjetja. Nadaljujemo s podrobnejšim opisom procesa in pripadajočih aktivnostih pri dostavljanju novih verzij aplikacij končnim uporabnikom. Predstavljena je problematika ter nekateri principi razvoja programske opreme, ki se osredotočajo na prej omenjeni proces dostave

ter posledičnega uvajanja sprememb IS. Na koncu teoretičnega poglavja predstavimo določene rešitve avtomatizacije internih procesov v informatiki in posledične učinke za podjetje.

Praktičen del v prvem delu predstavlja opis problematike in potek dostave novih verzij aplikacij v obravnavanem podjetju. Sledi predstavitev orodja IBM UrbanCode Deploy (v nadaljevanju UCD) za avtomatizacijo namestitev, ki je umeščen v proces dostave. Drugi del predstavlja analizo prenovljenih procesov dostave in učinke novega stanja, ter kakšni so prihodnji izzivi. Preko zbranih metričnih podatkov pa je predstavljen tudi finančni vidik upravičenosti vpeljave novega orodja.

Pri izdelavi uporabimo naslednje metode:

- metoda iskanja sekundarnih podatkov po različnih informacijskih virih (Emerald, ProQuest ipd.),
- metoda analize obstoječega in prihodnjega stanja v obravnavanem podjetju,
- analiza stroškov in koristi vpeljave orodja za avtomatizacijo namestitev.

Namen raziskave je v konkretnem podjetju preučiti možnosti vpeljave orodja za avtomatizacijo nekaterih razvojnih aktivnosti, z namenom večje učinkovitosti in boljše kakovosti razvoja IS, ki bi prinašal pozitivne poslovne učinke.

1 AVTOMATIZACIJA PROCESOV V IS

Že koncept t. i. Von Neumannovega modela računalnika, ki danes definira veliko večino računalnikov, temelji na izvajanju avtomatizacije. To pomeni, da večina današnjih računalnikov deluje na vnaprej predvidenem avtomatičnem izvajanju ukazov. V IS je informacijska tehnologija (v nadaljevanju IT) zelo široko uporabljena, zato bi lahko rekli, da avtomatizacija aktivnosti predstavlja eno od prvinskih lastnosti, s katerimi se sooča računalništvo. Zakaj torej IS ni popolnoma avtomatiziran? Dejstvo je, da zaenkrat celotnega delovanja IS v večini primerov ne moremo avtomatizirati, saj bi IS moral biti avtonomen samozadosten sistem, ki bi se odločal, prilagajal in reševal naloge s pomočjo uporabe umetne inteligence. Procesi, ki zadevajo vzdrževanje, upravljanje ali spreminjanje IS, v večini podjetij še niso avtomatizirani. Trenutno tehnologija omogoča reševanje avtomatizacije določenih tradicionalnih procesov, vendar se mora podjetje na prvem mestu vprašati, zakaj bi potrebovali avtomatizacijo določenega procesa in kakšne koristi bi s tem pridobili.

Kot pravi Attaran (2003, str. 441–442), je imela IT v letu 1990 primerljiv vpliv na organizacijske spremembe v podjetjih, ki so bile izvedene v času t. i. taylorizma. IT je v večini primerov zaposlenim olajšala delo in jih opolnomočila, omilila komunikacijske prepreke ter predstavljala tehnološko platformo za prenovo obstoječih poslovnih procesov. Attaran nadaljuje, da je IT ne

toliko preoblikovala, temveč predvsem pospeševala delo. Danes je IT zelo široko uporabljena v podjetjih, saj je postala nepogrešljiva in nam olajšuje življenje tako poslovno kot zasebno.

Prihodnji izzivi v razvijanju IS bodo usmerjeni v povečanje fleksibilnosti IS, ki bo dopuščala enostavnejše prilagajanje podjetja spremembam. V idealnem primeru se tako lahko poslovni procesi poljubno spreminjajo zaradi visoko fleksibilnega IS, kar pomeni, da je celotno poslovanje podjetja v veliki meri odvisno od delovanja IS. Iz podanega primera je razvidno, da postajata IT in prenova poslovnih procesov (angl. *Business Process Reingeniering*, v nadaljevanju BPR) čedalje bolj povezana in soodvisna.

Bill Gates (2001, str. 1–2) razmišlja o realizaciji IS, katerega rešitev naj bi bila pravzaprav nekakšen digitalen nevrološki sistem. Tako bi informacije učinkovito prehajale z namenom maksimalnega in neprekinjenega učenja. Nadaljuje, da zgolj učinkovito narejen IS ne more podjetju zagotoviti uspeha, saj je uspeh podjetja odvisen od številnih dejavnikov: učinkovitih procesov, tržnega deleža, kakovosti storitve, blagovne znamke. Nadalje Gates poudarja, da bo podjetje s slabim poslovnim načrtom neuspešno kljub dobrim informacijam. In obratno, podjetje z dobrim poslovnim načrtom, vendar s slabo izvedenimi ostalimi funkcijami, ki vplivajo na uspešnost podjetja, bo postavljeno v nerentabilen položaj.

Sposobnost prilagajanj ter izkoriščanja novih IT, ki podjetju na kakršenkoli način pripomorejo k ustvarjanju dodane vrednosti, za podjetje pomeni ustvarjanje konkurenčne prednosti. Podjetja so izpostavljena novim izzivom: krajšanje življenjskega cikla izdelka, globalna konkurenca in težnja po zmanjševanju stroškov (Neubauer, 2009, str. 167). Kersten in Verhoef (2003, str. 34–40) omenjata povezovanje poslovanja z ustrezno IT kot enega od dejavnikov uspeha. Uvedba avtomatizacije v katerikoli poslovni proces, kjer je to mogoče brez človeške vpletenosti, pohitri izvajanje, izboljša zanesljivost izvedbe (ni človeških napak) in zmanjša stroške (čas je denar).

Današnja tehnologija omogoča številne možnosti izvajanja avtomatiziranih procesov med različnimi napravami in informacijskimi viri, ki so del IS. Tehnološkim trendom in poslovnim potrebam se prilagajajo tudi podjetja s tradicionalno infrastrukturo IS, ki iščejo nadaljnje možnosti za izboljšave. Pred leti je, npr. virtualizacijska tehnologija omogočila preskok iz fizičnega v virtualno izvajanje praktično česarkoli (npr. virtualizacija operacijskega sistema, podatkovnih naprav, strežnikov, omrežnih naprav itd.). Omogočila je tudi današnji obstoj t. i. računalništva v oblaku. Številni koncepti, rešitve so tako skozi leta prinašali podjetjem dodatne možnosti pri zmanjševanju stroškov ter učinkovitosti IS nasploh. Pogosto pa novi koncepti in tehnologije vplivajo na ustrezen odziv in prilagoditve sprememb tudi v sami organizaciji (način dela, mišljenje, kultura, sprejemanje novosti, kooperativnost, itd.).

Težnje in napori po izboljšanju performanc v podjetjih procesni racionalizaciji in avtomatizaciji zaenkrat še niso doprinesle bistvenih izboljšanj glede potreb podjetij (Hammer, 2000, str. 104–107). Najverjetneje izboljšave niso tako učinkovite, ker se je do sedaj delalo predvsem na avtomatizaciji obstoječih procesov z namenom njihove pohitritve.

Ko se omenja avtomatizacijo in z njo povezane tehnologije, ne moremo mimo poslovnih procesov, saj tehnologija brez pravilne umestitve in načrtovanja ne bo prinesla želenih rezultatov. Namen BPR je, da s holističnim pristopom učinkovito razporedi in poveže ljudi, navade, procese in tehnologijo, kar bi podjetju prineslo čim večjo dodano vrednost (Brown, 2015, str. 1).

V primeru načrtovanja oz. prenavljanja poslovnega procesa je torej pomembno razumevanje njegovega namena ter njegove umestitve v širšo sliko procesov, ki so med seboj velikokrat odvisni. S takšnim pristopom strmimo k čim bolj smiselnemu in ne nazadnje učinkovitejšemu izvajanju procesov, poslovnih funkcij in podjetja nasploh.

Če pogledamo celostno, podjetje z uvajanjem IS pokriva in povezuje čedalje več poslovnih procesov, kar pomeni, da postaja poslovanje podjetja od njega zelo odvisno. To pomeni, da mora podjetje gledati na IS zelo strateško, saj se bodo pravilne in tudi nepravilne odločitve odražale na uspešnosti poslovanja. V primeru visokega deleža podpore poslovnim procesom preko IT je tako hitrost prilagajanja podjetja v veliki meri odvisna od zmožnosti prilagajanja samega IS.

Številni standardi in koncepti (npr. modularnost), ki se danes uporabljajo v IT, omogočajo lažje medsebojno povezovanje in komuniciranje IT virov, kar poenostavlja morebitne avtomatizacijske aktivnosti, vendar pa nivo problematike navadno določa specifika določenega primera. Procesno gledano je vpeljevanje avtomatizacije v veliki meri odvisno od same identifikacije potencialnih procesov in posledične ustrezne celostne rešitve.

Prilagajanje IS v podjetju predstavlja številne povezane in nepovezane procese z njihovimi pripadajočimi aktivnostmi. Ti procesi po navadi predstavljajo infrastrukturo, preko katere se vršijo in implementirajo številne spremembe. Razvoj aplikacij in infrastrukturne rešitve, ki te rešitve razvijejo oz. prilagodijo, so še vedno v domeni ljudi. Učinkovito delo pri spreminjanju IS pa je v veliko primerih pogojeno glede na stanje optimiziranih aktivnosti, načine upravljanja najrazličnejših sistemov in procesov, ki si skupaj vplivajo na zmožnost spreminjanja in prilagajanja IS v podjetju. Optimizirani, ponavljajoči procesi spreminjanja IS lahko s tega vidika koristno pripomorejo k hitrosti in sposobnosti prilagajanja podjetja.

V zadnjem času se v strokovni literaturi pojavlja trend, ki označuje potrebo podjetij po različni hitrosti (angl. *two-speed IT*) razvoja in prilagajanja IS (Bossert, Ip & Laartz, 2014). Na eni strani je tako tradicionalna IT, ki se v vseh poslovnih potrebah s poslovnega vidika ne prilagaja dovolj hitro, na drugi strani pa se pojavljajo številne priložnosti, kot so: oblačne storitve, obdelava velikih količin podatkov (angl. *big data*), različne mobilne aplikacije itd. Togost ter nezaznavanje potreb po hitrih spremembah ima lahko za podjetje negativne posledice. Prilagajanje je nujno na vseh področjih, a vendar je hitrost pogojena tudi od narave storitev. Potreba po veliko hitrejšem ponujanju storitev v IS (npr. mobilne storitve, inovativne rešitve itd.) se navadno pričakuje na področju, kjer stranke zahtevajo predvsem storitve, ki vključujejo tehnološke novosti. Preko posledične hitre dostave inovativnih storitev lahko podjetje uživa prednost prvega na trgu.

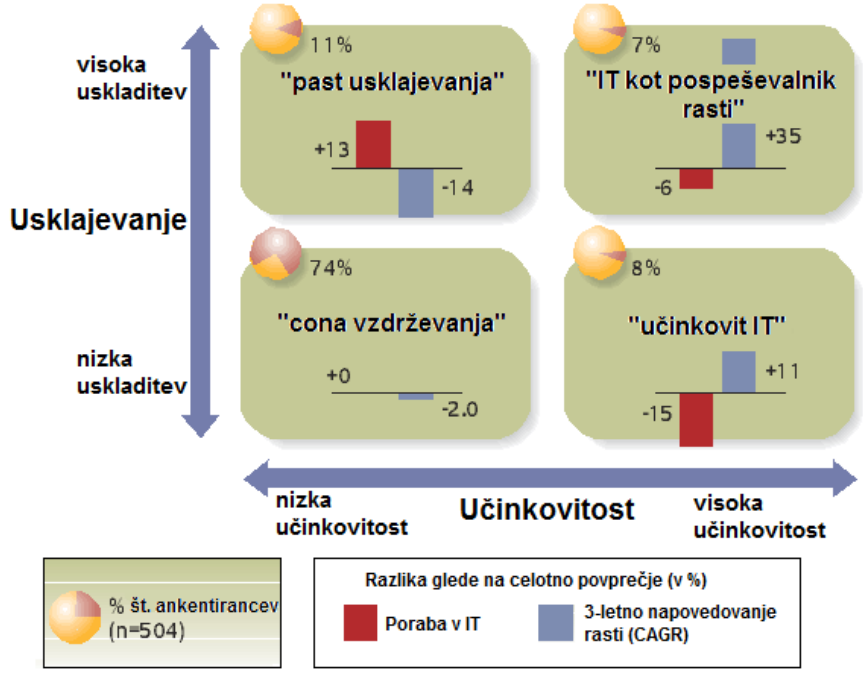
Zanimiva raziskava je bila objavljena v reviji MITSloan Management Review, v kateri avtorji (Shpilberg, Berez, Puryear & Shah, 2007) izpostavljajo, da pri rasti podjetij predstavlja IT zelo pogosto ozko grlo. Sistematična raziskava, v katero je bilo vključenih več kot 500 sodelujočih, ki so zasedali vodilna IT mesta v podjetjih različnih branž, je pokazala zanimiv vzorec. Ta vzorec je prikazan na Sliki 1, in sicer os x prikazuje prizadevanja podjetja po učinkovitosti IS, kjer se od leve proti desni učinkovitost stopnjuje navzgor. Os y ponazarja težnjo podjetja po prilagajanju IS podjetja izključno poslovnim potrebam, kjer predstavlja zgornji rob slike višje vrednosti. Na Sliki 1 se nahajajo tudi štirje kvadranti (Shpilberg, Berez, Puryear & Shah, 2007):

1. IT kot pospeševalnik rasti (angl. *IT-enabled growth*),
2. past usklajevanja (angl. *alignment trap*),
3. cona vzdrževanja (angl. *maintenance zone*),
4. učinkovit IT (angl. *well-oiled IT*).

Kvadranti predstavljajo razmerje med stroški, porabljenimi za IT, ter skupno letno stopnjo rasti prodaje za obdobje 3 let. Kot je razvidno s slike, je najverjetneje vsem podjetjem ciljni kvadrant zgoraj desno (glede na izvedene ankete je bilo teh le 7 %).

Prilagajanje IS je nujno potrebno in zdi se, da maksimiranje naporov informatike v čim boljše prilagajanje IS z vidika poslovnih potreb prinaša neposredne koristi. Raziskava je pokazala zanimive rezultate. Eden izmed teh je tudi, da naravnanost samo k usklajevanju IS poslovnim potrebam povzroča dolgoročno povečevanje kompleksnosti sistemov, infrastrukture, aplikacij, čemur rečemo past usklajevanja. Takšna strategija pripelje podjetje v visoke stroške upravljanja IT in zaradi kompleksnosti v izjemno težko prilagajanje IS, kar vodi podjetje v nekonkurenčnost. V raziskavi je tudi navedeno, da kompleksnost sistema ne more kar izginiti, zato je edini možni način poti do zgornjega desnega kvadranta tako, da začnemo z optimizacijo samega IS in nato s ponovnim prilagajanjem IS poslovnim potrebam.

Slika 1: Kvadranti različnih stanj IS v podjetjih



Vir: D. Shpilberg, S. Berez, R. Puryear & S. Shah, Avoiding the Alignment Trap in IT, 2007.

Raziskava kaže, da če želimo iz kvadranta 3 v kvadrant 1, ne naredimo napake s fokusiranjem samo na usklajevanje podjetja poslovnim potrebam (3. kvadrant). Smer poti je sicer prava, vendar takšna podjetja potujejo po poti, po kateri ne bodo dosegla želenega cilja.

Če želi podjetje torej dolgoročno dosegati konkurenčno prednost, je neprestano usmerjanje resursov IT v učinkovitost IS prav tako pomembno kot samo usklajevanje in prilagajanje IS poslovnim potrebam. Principi, ki bi se jih podjetja morala držati pri doseganju boljše učinkovitosti, so: stremenje k enostavnosti, ustrezno razpolaganje z danimi resursi (fond, zaposleni, maksimizacija koristi in minimizacija stroškov) in merjenje ter nadzorovanje projektov.

Glede na opisano avtomatizacija poslovnih procesov v IS navadno prinaša ravno optimizacijo in zmanjševanje potrebnih resursov. Lahko rečemo, da se avtomatizirane aktivnosti lahko vključuje pri samem dostavljanju prilagojenih aplikacij glede na poslovni nivo (usklajevanje IS) ter tudi pri povečevanju oz. doprinosu k učinkovitosti samega IS. Pri vpeljevanju avtomatizacije poslovnih procesov v podjetje se moramo tako vprašati, na kakšen način bo avtomatizacija prinašala koristi podjetju. Temeljiti pa bi morala na tem, da IS poenostavlja in doprinaša k boljši obvladljivosti in učinkovitosti IS. Najverjetneje je končni cilj avtomatizacije v IS pravzaprav avtonomno prilagodljiv avtomatiziran IS, ki bi bil zmožen ugoditi vsaki želji glede podpiranja in prilagajanja IS. S tem bi bili njegovi odzivi neposredno povezani s strategijo in poslovnimi odločitvami. Do takrat bo najverjetneje preteklo še kar nekaj časa ali pa je morda le utopična želja, a vendar je težnja po avtomatizaciji kakršnihkoli procesov danes močno prisotna pri mnogih podjetjih. Porast uporabe umetne inteligence, avtonomnih sistemov, odkrivanje vzorcev v veliki količini podatkov (angl. *big data*) nakazuje na uporabo sofisticiranih sistemov z višjo dodano vrednostjo.

V zadnjem času se pojavlja trend čim širše uporabe avtomatizacije aktivnosti praktično v vseh panogah industrije, kar podkrepi dejstvo, da smo v 4. industrijski revoluciji (What Is Industrie 4.0 and What Should CIOs Do About It?, 2016).

1.1 Kratka opredelitev IS

Namen IS je učinkovito podpiranje nalog, za katere je bil zgrajen, in s tem prinese olajšanje, pospeševanje ter prinašanje dodane vrednosti uporabnikom. Shim (2000, str. 1) IS enostavno opredeli kot računalniški sistem, ki je namenjen obdelavi podatkov in za rezultat daje informacije. Alter (2002, str. 6) podrobnejše navaja naloge njegovih poslovnih procesov, ki so zajem, prenos, shranjevanje, pridobivanje, manipuliranje in prikazovanje informacij, s čimer podpirajo tudi druge sisteme. Pridobljene informacije so uporabne za notranje in/ali zunanje uporabnike sistema.

IS torej sestavlja tako IT kot tudi ljudje, ki ga upravljajo oz. uporabljajo. Namenjen je čim bolj učinkovitemu podpiranju poslovnih procesov oz. izvajanju aktivnosti v podjetjih. Ustrezno zasnovan IS se hitro prilagaja poslovnim potrebam podjetja in lahko podjetju predstavlja tudi konkurenčno prednost. Znotraj IS obstajajo različni tokovi informacij, ki jih je treba uspešno podpirati, ter tako zagotavljati nemoteno poslovanje podjetja. Obstaja več vrst IS, ki so namenjeni najrazličnejšim poslovnim potrebam, vsem pa je skupna težnja po čim večji učinkovitosti in prilagodljivosti.

Wallace (2015, str. 43) opredeljuje 4 glavne komponente, ki so skupne vsem IS:

- **ljudje** – v interakciji z IS ločimo notranje in zunanje končne uporabnike (Shelly, Cashman & Rosenblatt, 2008, str. 7);
- **tehnologija** – opredelimo jo kot strojno opremo in programsko opremo. Danes je na voljo ogromno kombinacij uporabljenih tehnologij, ki jih lahko podjetje implementira. Izbira mora temeljiti na strateških odločitvah podjetja, kar pomeni, da bo učinkovito podpirala poslovne potrebe in tudi prinašala korist;
- **poslovni procesi** – predstavljajo zaporedje aktivnosti, katerih rezultat je izdelek ali storitev. Procese je treba najprej definirati, določiti nekaj ključnih ter narediti popis ostalih pomembnejših podprocesov. Pri tem je treba določiti lastnike in skrbnike poslovnih procesov, ki so odgovorni za nemoteno izvajanje aktivnosti. Pomembno je nenehno stremenje k optimizaciji, kar posledično pomeni dinamično spreminjanje procesnih povezav;
- **podatki** – v IS prihajajo iz več virov in so zajeti v številnih oblikah. Predstavljajo osnovo za nadaljnjo obdelavo z namenom pridobivanja dodatnih informacij. V IS je integriranih čedalje več naprav, ki na različne načine oddajajo podatke. Prav tako pa dodatno beleženje podatkov zapoveduje zakonodaja (dostopi do občutljivih podatkov itd.), nadzor sistema ipd.

Odločevalci na vodilnih mestih v podjetju, ki so na kakršenkoli način vpleteni v razvoj in nadaljnje odločitve glede razvoja IS, se soočajo s kar nekaj vprašanji oz. izzivi (Godinez et al., 2010, str. 5):

- pridobivanje natančnih in hitrih informacij za nadaljnje odločanje,
- neustreznost oz. nesprejemanje strategije informatike oz. infrastrukture. IS je grajen od spodaj navzgor brez celovitega načrta,
- kompleksnost oz. nepreglednost sistema ter posledično zahtevno obvladovanje sistema,
- slaba kakovost podatkov,
- zahtevna, draga integracija IS (npr. zahtevno ali nemogoče medsebojno celovito povezati transakcijske sisteme, planiranje, analitiko itd.),
- neoptimalno izrabljeni informacijski viri,
- nesoglasja med IT in ostalimi službami (na višjih ravneh),
- previsoki skupni stroški lastništva (angl. *Total Cost of Ownership*).

1.2 Agilen razvoj

Obstajajo različni koncepti razvoja programske opreme, ki predstavljajo načrt realizacije projektov. Razvoj programske opreme se lahko zgleduje po številnih metodologijah, ki jih lahko delimo na tradicionalne in agilne. Tradicionalne metodologije, med katere spada zelo široko sprejet t. i. slapovni model, natančno določa zaporedje, prikazano na Sliki 2. Vsaka faza mora biti popolnoma zaključena, vračanje nazaj ni možno in za zaključek projekta se morajo izvesti vse faze. Slabost te metodologije je nezmožnost prilagajanja projekta trenutnim potrebam oz. spremembam novih želja naročnika. Velikokrat se zgodi, da v prvi fazi, kjer se zbirajo zahteve o končnem produktu, naročnik spregleda oz. še ne ve natančno, kakšno rešitev sploh želi. V nadaljnjih fazah ima lahko naročnik drugačne želje, zahteve glede končnega produkta, za kar je prepozno. Tradicionalne metode so s tega vidika toge, kar za podjetje pomeni počasnejšo prilagodljivost in posledično slabšo konkurenčno prednost.

Slika 2: Faze slapovnega modela

Vir: M. Chang, An Agile approach to library IT innovations, 2010, str. 673.

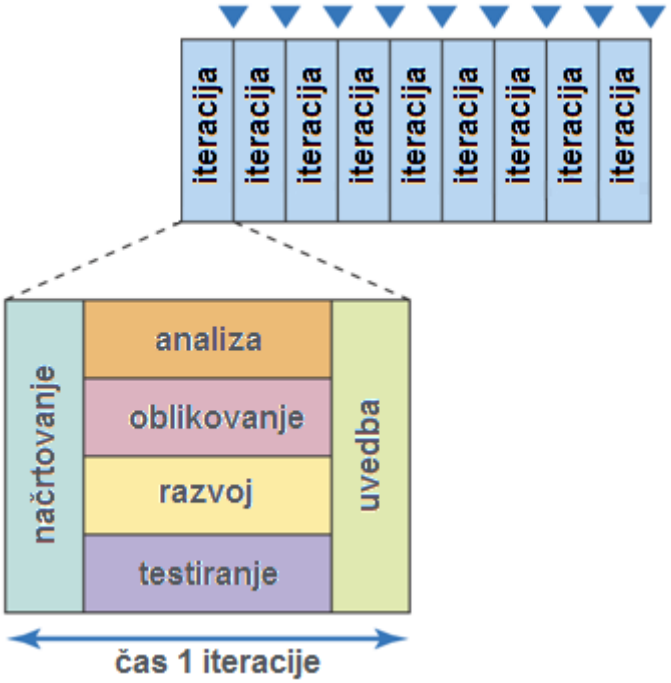
Na drugi strani agilne metodologije predpisujejo razvoj programske opreme po drugačnih principih, saj so nastale ravno zaradi pomanjkljivosti tradicionalnih metodologij: neprilagodljivost, obsežna dokumentacija, detajlno zajemanje zahtev pred začetkom razvoja, težko definiranje vseh zahtev, preden se razvoj sploh začne itd. Vse to je vodilo v iskanje primernejših pristopov za določene projekte.

Agilen razvoj postavlja v ospredje razvijalce in naročnika, kjer je za uspešno izvedbo projekta in kakovosten izdelek nujna učinkovita obojestranska komunikacija. Razvoj poteka ciklično, kjer se ob vsakem intervalu razvija dele celotne rešitve. Glede na situacijo se že razviti moduli lahko dodatno prilagajajo ali optimizirajo, saj je najpomembnejši delujoč izdelek, ki bo naročniku prinašal čim večje koristi. Agilne metode izhajajo iz iterativnega oz. evolucijskega razvoja in

imajo kljub specifikam posameznih metod skupne značilnosti (Larman, 2004, str. 26): preprostost, lahkotnost, timsko delo, pomen komuniciranja, samoorganizirane skupine, prednost razvoja programske kode od samega dokumentiranja.

Kot je razvidno s Slike 3, se v vsaki iteraciji razvoja izvedejo pravzaprav vsi koraki slapovnega modela, s čimer se celoten problem razbije na manjše, bolj obvladljive.

Slika 3: Agilni pristop razvoja programske opreme



Vir: The XP Lifecycle, 2007.

Pravzaprav so pred uporabo izraza agilnih metod nastale številne izvedenke, katerih značilnosti in namen je bil enak. Izraz »agilna metodologija« je bil formaliziran februarja 2001, in sicer ko je 17 strokovnjakov v Manifestu agilnega razvoja programske opreme določilo skupna izhodišča, ki so definirala značilnosti novih, prej imenovanih lahkih metodologij (Beck, et al., 2001).

Agilne metode razvoja poudarjajo (Beck, et al., 2001):

- pomen posameznikov, ki so vključeni v projekt, ter nujnost komunikacije pred procesi in orodji,
- delujoč program pred obsežno dokumentacijo,
- sodelovanje razvijalcev z naročnikom pred pogodbenimi pogajanji,
- odziv na spremembe pred slepim sledenjem planu.

Chang (2010, str. 673-674) navaja, da agilni pristop razvoja programske opreme izhaja iz:

- inkrementalnega modela, ki celoten problem razdeli na obvladljive sklope. Vsak sklop se lahko neodvisno izvaja, kot pri slapovnem modelu. Vsak sklop predstavlja samostojno enoto,

ki je neodvisna od ostalih in se lahko razvija tudi vzporedno z ostalimi (pogojeno od prostih virov). Trajanje izvedbe posameznega sklopa je odvisno od zahtevnosti in širine problematike posameznega inkrementa. Vse razvite sklope na koncu povežemo v celovito delujočo programsko rešitev;
- iterativnega modela, ki se najprej osredotoči na najbolj kritične sklope, od katerih je odvisen uspeh projekta. Končane iteracije še ne pomenijo zaključek sklopa, saj se le delno razvijejo in se kasneje postopoma izboljšujejo ter dopolnjujejo glede na potrebe naročnika.

Izbira katerekoli metodologije je odvisna od specifičnosti projekta, kulture, okolja, pomembnosti oz. kritičnosti, finančnih virov, strokovnih znanj, samoiniciativnosti, motivacije itd. Tako izbira katere od metodologij ne predstavlja edine prave poti, ampak le primernejšo oz. manj primernejšo pot glede na situacijo (npr. sistemi, ki lahko neposredno vplivajo na življenja ljudi, poslovna aplikacija itd.). Ostale bolj znane agilne metodologije so še naslednje: Scrum, XP (Extreme programming), Agile data method, Crystal Clear …

1.3 Dostavljanje sprememb v IS

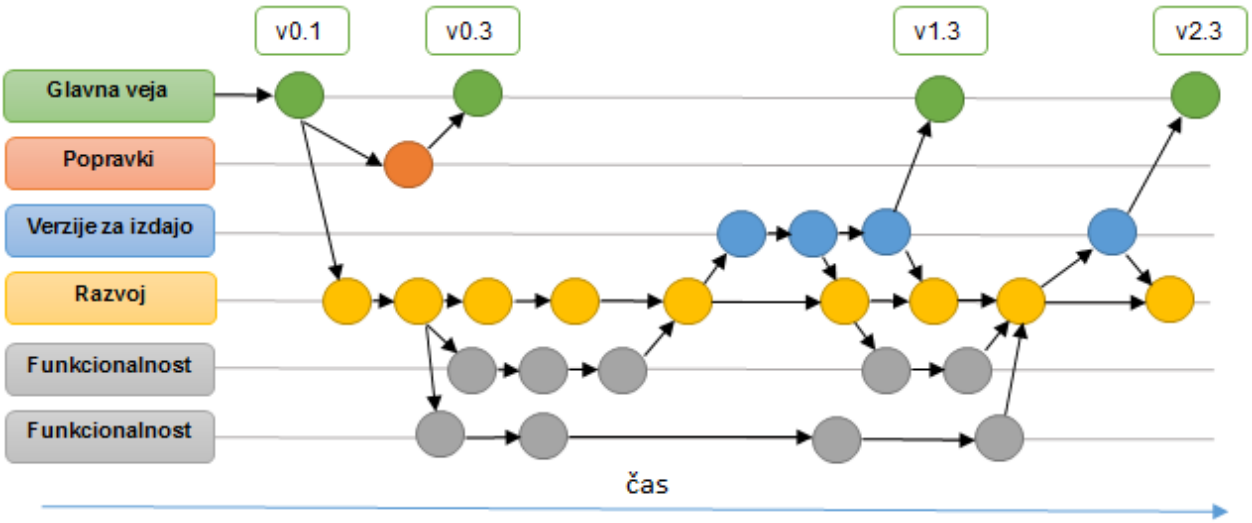
Večina posameznih sprememb v IS je povezana s spreminjanjem funkcionalnosti programskih rešitev in njihovim implementiranjem v produkcijo. Strokovnjaki ugotavljajo, da je čas od same ideje po določeni spremembi in do končne implementacije v produkciji večinoma prepočasen (Humble & Farley, 2011, xxiii). Temu obdobju pravimo čas cikla (angl. *cycle time*), ki je merljiv. Sharma (2012b) poudarja pomen izvajanja čim krajšega cikla, saj ta definira zmožnost podjetja za hitro in pogosto prilagajanje aplikacij v svojem IS.

Celoten namestitveni tok je sestavljen iz več podprocesov, v katere so vključene različne skupine strokovnjakov IS. Vse zaporedne aktivnosti so usmerjene v izvedbo cilja, ki predstavlja delujočo aplikacijo, od katere ima podjetje korist. Da bi bolje razumeli celoten proces nameščanja in povezane podprocese, bosta v nadaljevanju opisana 2 koncepta, ki neposredno vplivata na zmanjšanje časa cikla (Sharma, 2012a). To sta neprekinjena integracija in neprekinjena dostava.

1.3.1 Neprekinjena integracija

Neprekinjena integracija (angl. *Continuous Integration*, v nadaljevanju CI) je avtomatizirana metoda pogostega združevanja in gradnje izvorne kode, ki vključuje izvajanje integracijskih testov. V prvem koraku se izvede proces združevanja programske kode v enoten delujoč programski paket. Spreminjanje oz. razvoj programskih rešitev navadno razvija več razvijalcev hkrati, vsak svoj del. Kot prikazuje Slika 4, razvijalec v skupen repozitorij programske kode odda razvit del programske rešitve, ki mora biti ustrezno verzioniran in pregledan. Tako nastajajo različne veje oz. različice rešitve aplikacije.

Slika 4: Verzioniranje



Vir: V. Driessen, A successful Git branching model, 2010.

Slika 5 prikazuje, da lahko vsako verzijo kode v posamezni veji preko sistema za neprekinjeno integracijo generiramo v enoznačno verzijo aplikacije. Med generiranjem se lahko zgodijo napake oz. se izvedejo testi razvite kode. Če pride do kakršnekoli težave, je treba programsko kodo ustrezno popraviti in ponoviti celoten proces integracije.

Slika 5: Proces CI

Vir: M. Budhabhatti, Test-Driven Development and Continuous Integration for Mobile Applications, 2008.

S CI se tako soočajo razvijalci programske kode, ki morajo svoje rešitve neprekinjeno integrirati z rešitvami ostalih razvijalcev v enotno programsko rešitev. Združevanje spremenjene kode z ostalimi različicami kode navadno vodi do konfliktov, ki so povezani s pravilno integracijo programske kode (soodvisnost narejenih sprememb, nedelovanje združene kode itd.).

Če pride do težave pri gradnji posamezne verzije, se takrat vsi razvijalci osredotočijo na razrešitev problema (Humble & Farley, 2011, str. 105). Bistveno je, da so na voljo vedno delujoče verzije napisane programske kode. Sledenje praksam CI razvojni ekipi omogoča večjo produktivnost, kakovost programske kode, zaupanje in razvoj kompleksnih sistemov.

Za uspešno izvajanje CI programske kode je priporočljivo uvesti (Humble & Farley, 2011, str. 55–57):

- **verzioniranje programske kode:** celoten projekt razvoja in gradnje aplikacije sestavljajo različne verzije. Te vključujejo vse pripadajoče komponente posamezne verzije aplikacije (programska koda, namestitvene skripte, testi, skripte za podatkovno bazo itd.). Za tehnične rešitve problema verzioniranja obstaja veliko sistemov (npr. Git, CrouseControl itd.);
- **avtomatično gradnjo** (angl. *build automation*): realiziran sistem za avtomatično prevajanje integriranih delov programske kode v enoten projekt, ki predstavlja novo verzijo aplikacije. Pri vsaki gradnji nove verzije naj bi bili izvedeni avtomatizirani, vnaprej pripravljeni testi za doseganje konsistentnosti in ustreznosti zgrajene programske kode (Continuous integration, 2015). Sistem za avtomatično gradnjo kode je, npr. Jenkins;
- **medsebojne dogovore**: razvijalci naj nalagajo novo kodo pogosto po manjših sklopih in ne šele takrat, ko je v celoti napisana/popravljena. Tak pristop pripomore k hitrejšemu zaznavanju napak v kodi. Ob morebitnih težavah z integracijo programskih kod nastane bolj omejen obseg sprememb, kjer je lažje najti potencialno težavo.

CI tako zajema aktivnosti, povezane z integracijo izvorne kode, ustrezno verzioniranje vseh artefaktov (sestavni deli aplikacij), gradnjo novih verzij, kakovost programske kode, sintaktično pravilnost, ki kot rezultat izdaja nove verzije aplikacij. Ugotovili smo, da je CI osredotočen predvsem na potek dela, ki zajema ekipo razvijalcev. Izhodi podprocesov, ki pokrivajo razvoj programske opreme, pa predstavljajo pravzaprav vhode ostalim podprocesom, ki jih izvajajo ostale skupine strokovnjakov (npr. testna ekipa, operacije itd.) (Humble & Farley, 2011, str. 105). Ti so vključeni v neprekinjeno dostavo, ki bo opisana v nadaljevanju.

1.3.2 Neprekinjena dostava

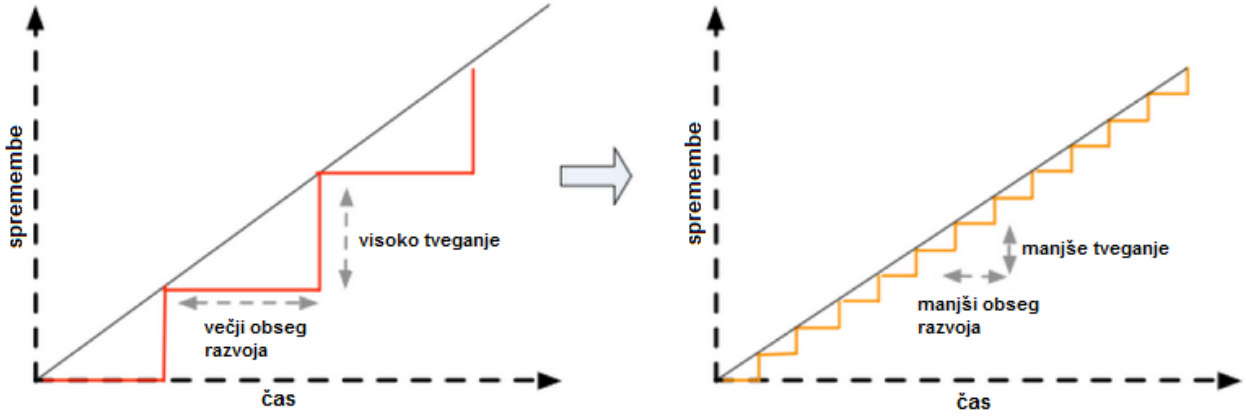
Kot nadgradnja obstoječe CI se je pojavil nov pristop razvoja programske opreme, ki se imenuje neprekinjena dostava (angl. *Continuous delivery*, v nadaljevanju CD). Za lažje razumevanje je treba najprej razložiti naslednja dva izraza:

- **namestitveni tok** predstavlja celoten proces prehodov in testiranja primernosti aplikacij preko številnih okolij, vse do končne izdaje končnim uporabnikom,
- **proces nameščanja ali namestitev** aplikacije pomeni izvedbo aktivnosti pri prilagajanju določenega okolja (testna oz. produkcijska okolja), kjer se prilagodi vse komponente, ki sestavljajo funkcionalnosti posamezne verzije aplikacije (aplikacijski paket, aplikacijski strežnik, konfiguracije, podatkovna baza itd.).

V prvem principu Manifesta agilnega razvoja programske opreme (Beck, et al., 2001) je zapisano: *»Naša najvišja prioriteta je zadovoljiti stranko z zgodnjim in nepretrganim izdajanjem vredne programske opreme.«* CD pomeni zmožnost ponavljanja celotnega procesa od izdane spremembe v izvorni kodi s strani razvoja (CI) ter vse do implementacije aplikacije na produkciji (Duvall, 2011, str. 1). Ragan (2015) meni, da je bistvo tega pristopa v vzpostavitvi ponavljajočih in zanesljivih procesov, ki se inkrementalno izboljšujejo. Da dosežemo kratek čas cikla in tudi visokokakovostne aplikacije po meri podjetja, je treba uvesti naslednje (Humble & Farley, 2011, str. 12):

- **avtomatizacijo** – lahko jo vpeljemo, kjer imamo ponavljajoče aktivnosti, ki jih je treba predhodno ustrezno definirati. Ko se avtomatizacija procesa vpelje, se zmanjša potreba po vpletenosti zaposlenega. S tem se zmanjšajo možnosti za napake in zamude, s čimer hkrati zmanjšamo čas cikla;
- **pogostost namestitev** – omogoča učinkovitejše obvladovanje in progresivno izvajanje sprememb in s tem manjše tveganje pri vpeljevanju novih funkcionalnosti preko številnih manjših namestitev. To nazorno prikazuje tudi Slika 6. Prav tako pa razvojna ekipa prej opazi, ali razvoj poteka v skladu s pričakovanji podjetja ali pa je treba kaj spremeniti. Pogosteje pa se preko različnih nadzornih sistemov lahko spremlja samo odzivnost in delovanje aplikacij. Preko nadzornih sistemov, kjer se meri različne metrične podatke (npr. odzivnost), pa lahko preko pogostih namestitev zaznavamo natančna medsebojna odstopanja. Zgodovina meritev nam lahko pokaže najrazličnejše trende, verzije z majhnimi spremembami pa lahko natančno zaznajo začetek in vzrok neželenih stanj.

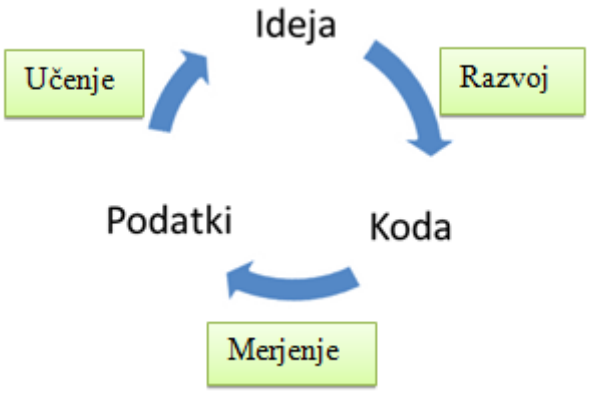
Slika 6: Primerjava različnih pristopov namestitev



Vir: A Robust Delivery Pipeline is the key to your success, 2016.

Reis (2016) pravi, da bi razvoj programske opreme moral temeljiti na uporabi sistematičnega in znanstvenega pristopa/metode. Slika 7 prikazuje konceptualno idejo razvoja posamezne funkcionalnosti. Razvoj končnega izdelka lahko terja veliko časa in finančnih sredstev. Ponavadi se je uspešnost določene ideje preverila šele po realizaciji končnega produkta. Reis predlaga, da začnemo z razvojem čim manjšega sklopa ideje in preverjamo že vmesne rezultate. Po koncu razvoja posameznega sklopa sledi merjenje prej definiranih metrik, ki nam predstavljajo bistvene pokazatelje o uspehu ideje (npr. število obiskov strani, pogostost uporabe določenih funkcionalnosti itd.). Na podlagi zbranih podatkov v sklopu posameznega razvojnega cikla preverimo, ali smo na pravi poti oz. kaj bi še morali spremeniti. Na ta način preko merjenj in učenja v vsakem ciklu ugotovimo, kakšni bodo nadaljnji ukrepi glede na rezultate posameznega cikla.

Slika 7: Znanstveni pristop razvoja



Vir: E. Reis, The Lean Startup Methodology, 2016.

Reisova ideja pravzaprav sovpada s principom CD, ki preko postavljene infrastrukture in procesov omogoča zanesljivo in hitro vpeljavo realizirane spremembe. Preko pogostih implementacij majhnih sprememb na katerakoli okolja prihajamo hitro do novih informacij in izboljšujemo nadaljnje korake razvoja. CD omogoča hitrejši in manj tvegan proces nameščanja novih funkcionalnosti oz. posodabljanj aplikacij vse do končnih uporabnikov. Obstaja veliko orodij, ki omogočajo avtomatizacijo upravljanja sprememb na številnih okoljih v sklopu namestitvenega toka.

V obravnavani literaturi CD poznavalci dojemajo različno. V principu so si pri procesnem vidiku enotni, razlike pa se pojavijo v konceptualnem oz. kulturnem vidiku. Nekateri strokovnjaki CD jemljejo kot povsem procesno tehnično realizacijo namestitvenega problema. Spet drugi pa CD poleg procesnega vidika dajejo organizacijski pomen, pomen sodelovanja, agilnih pristopov ter delovanja vsakega posameznika v duhu doseganja končnega skupnega cilja. Omenjena problematika, ki naj bi jo reševali s sprejetjem kulture DevOps, je podrobneje predstavljena v poglavju 1.3.4.

1.3.3 Namestitveni tok

Osrednji del strategije CD predstavlja namestitveni tok (DeploymentPipeline, 2013), ki je razdeljen na številne aktivnosti. Vhodi v proces predstavljajo razvite dele programske kode posameznega razvijalca, končni izhod pa predstavlja izdajo aplikacije na produkcijsko okolje. Predstavlja celovit proces, ki v ustreznem zaporedju združuje koncepte CI, ter nadaljnje korake pri dostavljanju spremenjene programske opreme do končnih uporabnikov. Če želimo izvajati hitre, zanesljive in ponovljive korake v namestitvenem procesu, je treba proces v čim večji meri avtomatizirati.

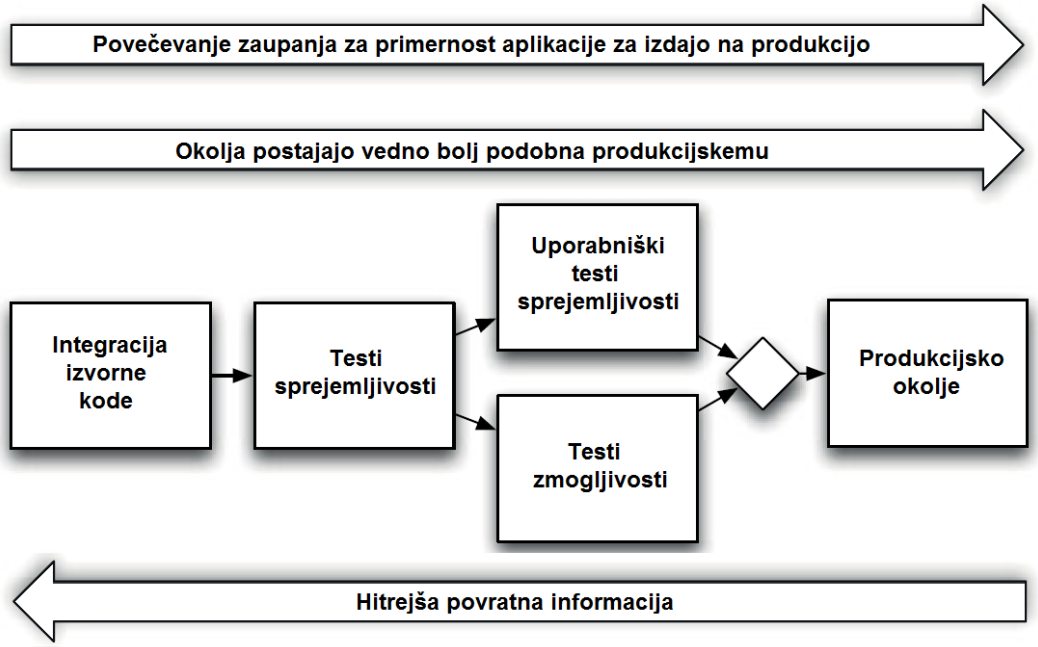
Vsem vpletenim v CD naj bi namestitveni tok na transparenten način omogočal pregled nad spremembami in aktivnostmi nameščanja v procesu (Humble & Farley, 2011, str. 140). Temelji namestitvenega procesa izhajajo pravzaprav iz CI, kjer se preko več korakov izvede vrsta testiranj, s katerimi se potrdi primernost razvitih sprememb (Humble & Farley, 2011, str. 4). V

avtomatiziranem namestitvenem procesu na 4 načine preverjamo produkcijsko primernost kandidata za izdajo (Humble & Farley, 2011, str. 109–110):

- **integracija izvorne kode** (angl. *the commit stage*): preveri se delovanje aplikacije s tehničnega vidika (ustrezno prevedena koda, testi enot (angl. *unit tests*) in analiza kode);
- **avtomatizirani testi sprejemljivosti** (angl. *automated acceptance test stages*): preveri se ustreznost aplikacije tako s funkcionalnega kot nefunkcionalnega vidika (testiranje zmogljivosti, varnosti, skalabilnosti itd.), glede na zahteve naročnika in uporabnikov;
- **ročni testi sprejemljivosti** (angl. *manual test stages*): odkrivanje napak, ročno preverjanje izpolnjevanja zahtev ter uporabnosti aplikacije, ki ga preko avtomatičnega testiranja ne moremo zaznati;
- **faza izdaje** (angl. *release stage*): predstavlja prehod izdaje realiziranih sprememb končnim uporabnikom na produkcijsko okolje.

Na Sliki 8 je prikazano dejansko zaporedje aktivnosti, ki jih mora prestati posamezen kandidat za končno izdajo.

Slika 8: Potek preverjanja primernosti kandidata za izdajo



Vir: J. Humble & D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, 2011, str. 110.

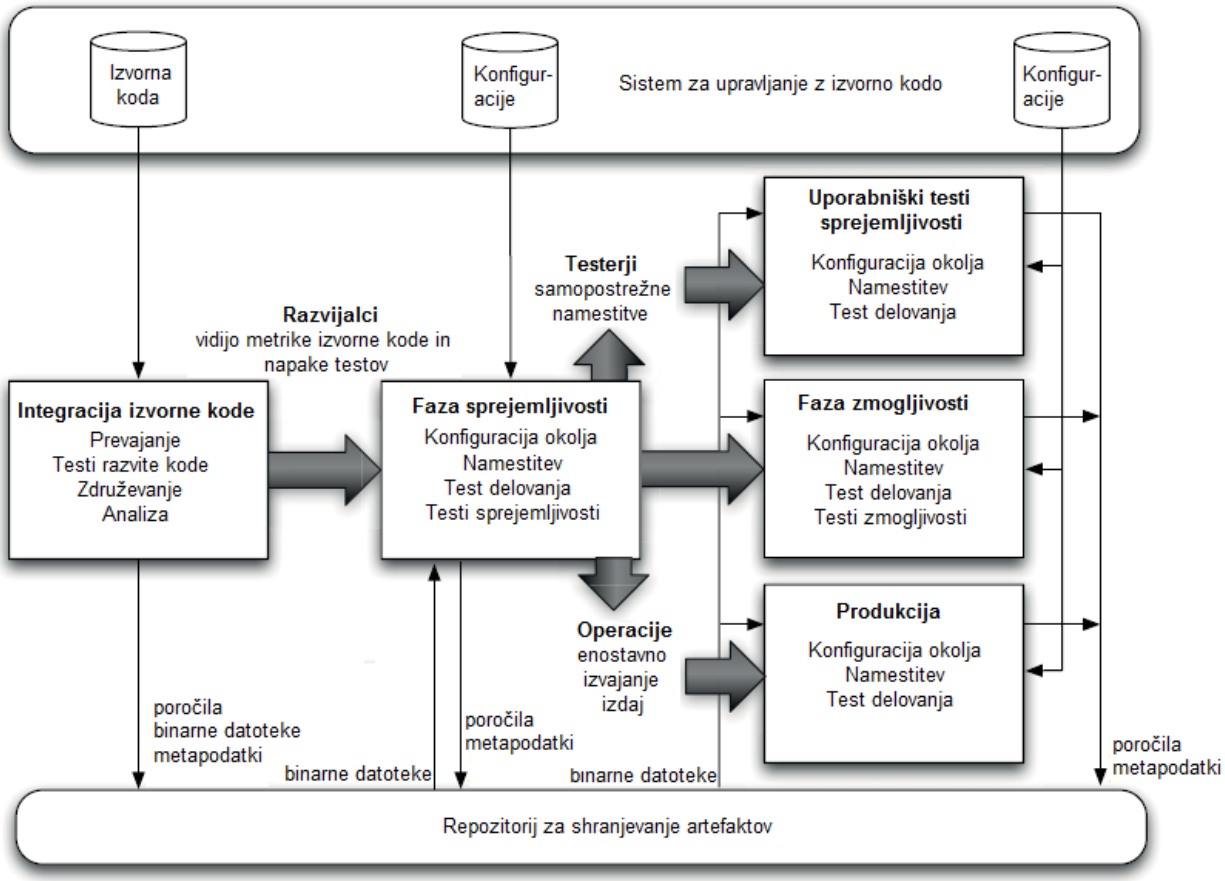
Če se v kateri izmed aktivnosti zazna odstopanje od pričakovanih rezultatov (npr. performančne težave, vsebinska neustreznost, napake itd.), se o tem obvesti razvijalce. Ko ti ugotovljeno neskladje popravijo, se izda nova verzija aplikacije, kjer se morajo ponovno izvesti vse aktivnosti v sklopu namestitvenega toka. Če aplikacija uspešno prestane vse prehode preko testnih okolij, kjer se izvajajo različna testiranja, se takšno verzijo aplikacije lahko izda uporabnikom na produkcijskem okolju. Izvajanje nameščanja na različna okolja ter aktivnosti vseh testov naj bi se izvajale na enak način, saj s tem zagotovimo konsistentnost in primerljivost

s prejšnjimi rezultati nameščanj in testiranj. Namestitveni tok združuje aktivnosti različnih skupin, ki medsebojno sodelujejo (Humble & Farley, 2011, str. 105):

- razvijalci,
- testerji,
- ekipa operacij.

Namestitveni toki se med podjetji razlikujejo, vendar principi in tehnike, ki definirajo obstoj samega procesa, ostajajo pri vseh enaki, tj. eliminiranje slabih kandidatov (Humble & Farley, 2011, str. 3–4). Slika 9 podrobneje prikazuje avtomatiziran namestitveni tok, ki vključuje vpletenost različnih skupin.

Slika 9: Namestitveni tok



Vir: J. Humble & D. Farley, Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation, 2011, str. 111.

Vsaka sprememba (programske kode, konfiguracije itd.) v namestitvenem toku zahteva ponovitev aktivnosti od samega začetka. Praktična uporaba namestitvenega procesa naj bi omogočala čim lažjo uporabo nameščanj (npr. namestitev preko pritiska na gumb) ter testiranj. Testerji morajo imeti popoln pregled tako nad vsemi pripravljenimi verzijami aplikacij kot tudi nad nameščenimi verzijami na vseh okoljih. Priporočljivo je namestitve opravljati na samopostrežen način (angl. *self-service*), kar pomeni, da se ustreznim osebam dodeli pravice za

nameščanje na posameznem okolju. Vsi posegi na okolju so zabeleženi z namenom visoke sledljivosti nad napredkom in procesom izvajanja namestitev in testiranj.

V primeru, da je posamezna verzija aplikacije na kateremkoli koraku procesa neuspešna, se korak prekine. Ker proces omogoča povratno informacijo, se na ta način pripravi nova verzija aplikacije oz. se razišče težavo posamezne verzije. Proces se lahko ponavlja v več iteracijah, dokler ni celotna namestitev uspešna. Sposobnost ponovljivosti namestitvenega procesa omogoča dve stvari:

- izboljšanje ustreznosti oz. kakovosti same aplikacije in
- inkrementalno izboljšanje samega procesa.

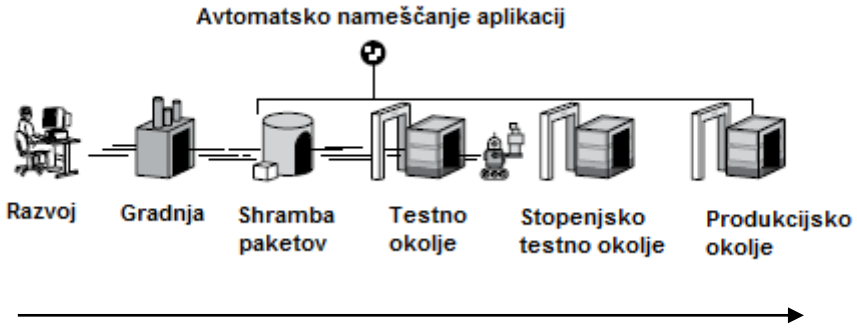
Izboljšava samega procesa naj bi potekala ciklično, kjer nam ciljno stanje predstavlja v čim večji meri avtomatiziran celoten proces, ki poenostavlja celotno namestitev. Pri vpeljanem namestitvenem procesu je pomembno tudi sistematično in disciplinirano merjenje učinkovitosti (Humble & Farley, 2011, str. 138–139). S tem lahko zagotavljamo učinkovite povratne informacije in izvajanje, ki vpliva na bolj kakovostne in hitreje dostavljene spremembe v aplikaciji. Avtorja poudarjata nekaj bistvenih metrik:

- čas namestitvenega cikla (težnja po čim krajšem),
- število vrstic v kodi (razvoj se bo trudil pisati čim krajšo kodo),
- število rešenih napak v kodi, število novih napak v kodi,
- število novih verzij v enem dnevu,
- število napak pri prevodu izvorne kode v enem dnevu itd.

S pomočjo definiranih metrik lahko lažje in celostno proaktivno pristopimo k izboljšanju učinkovitosti, hitrosti, k sodelovanju in zmanjševanju ozkih grl (*angl. bottleneck*), ki nastajajo pri sodelovanju med skupinami ali pa pri posamezni aktivnosti.

Slika 10 predstavlja namestitveni tok s topološkega vidika. Kot začetek procesa prikazuje razvito programsko kodo s strani razvijalca, čemur sledi gradnja ustrezne verzije aplikacije s pripadajočimi testi. Vse pripravljene verzije se odloži v skupen repozitorij, kjer so dosegljive za nadaljnje nameščanje in potrjevanje ustreznosti preko vseh testnih okolij do končnega produkcijskega okolja.

Slika 10: Namestitveni tok – topologija



Vir: Prirejeno po S. Sharma & B. Coyne, DevOps for Dummies, 2015, str. 26.

Nameščanje aplikacij preko različnih testnih okolij vse do produkcijskega predstavlja navadno veliko aktivnosti, ki morajo biti med seboj ustrezno orkestrirane. Sposobnost avtomatičnih namestitev na katerokoli okolje podjetjem omogoča hitrejše ukrepanje v primeru težav, prav tako pa omogoča izdajanje spremenjenih aplikacij glede na poslovne potrebe. Predvsem zadnji korak izdaje sprememb na produkcijo predstavlja izključno poslovno odločitev. Nekatera podjetja se odločijo, da se vsaka najnovejša produkcijsko primerna aplikacija čim prej dostavi tudi uporabnikom. Spet druga podjetja pa zaradi različnih vzrokov dostavijo spremembe z zamikom.

Zakaj potrebujemo toliko namestitev? Tudi če ne želimo vsake najmanjše spremembe na produkciji, je treba v ozadju, torej na testnih okoljih, pretestirati vsako spremembo, ki je bila narejena s strani razvijalcev. V duhu nadgrajevanja aplikacij torej zelo frekventno in z majhnimi spremembami dosežemo večjo varnost in obvladljivost namestitev. Vsako majhno spremembo namestimo na testna okolja in jo v izoliranem okolju podrobno testiramo. Tako lahko vsako manjšo nadgradnjo posebej na varen način dopolnjujemo s predhodnimi verzijami, ki so uspešno prestale faze testiranja. Kdaj se bo določena, uspešno testirana verzija posamezne aplikacije, namestila na produkcijo, predstavlja poslovno odločitev.

Pravilno vpeljan CD omogoča tudi ob morebitnih zaznavah napak na produkcijskem okolju sposobnost hitrega poenotenja stanja, s katerim od testnih okolij. S tem hitro zagotovimo reprezentativno okolje, na katerem lahko ponovimo napako in s tem identificiramo vzrok težav, z namenom hitrejše odprave napak.

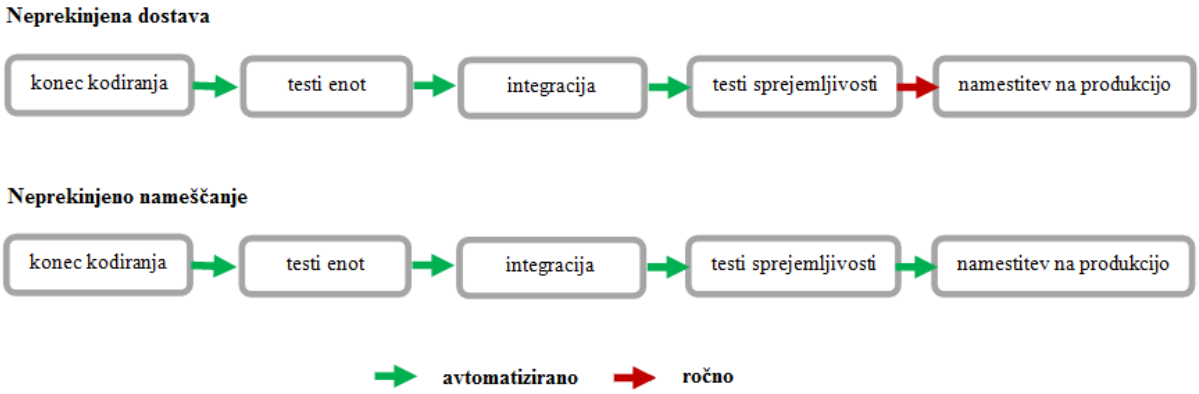
Glede opisanega si je večina avtorjev enotna in tudi glede tega, da je izvedba celotnega namestitvenega procesa tesno povezana z uvajanjem avtomatiziranih aktivnosti. S pomočjo eliminacije počasnega ročnega izvajanja ter optimizacije celotnega procesa namreč želimo doseči čim krajši čas cikla.

Eden izmed pristopov zmanjševanja časa cikla je uporaba **neprekinjenega nameščanja** (angl. *Continuous deployment*, v nadaljevanju CDep). Pogosto prihaja do mešanja dveh izrazov, in sicer CDep in CD. CDep, kot je prikazano na Sliki 11, izvede avtomatično namestitev tudi na produkcijsko okolje, in sicer vedno, ko prestane vse teste (angl. *quality assurance*, v

nadaljevanju QA). CD torej predstavlja zmožnost izvedbe namestitev kadarkoli na katerokoli okolje. CDep vsako uspešno pretestirano spremembo namesti na produkcijo. Pravzaprav gre torej tudi za poslovni vidik izvajanja namestitev, in sicer kdaj podjetje pravzaprav hoče dostavo sprememb. CDep je vsekakor orientiran na skrajševanje časa cikla in hitro uvajanje sprememb. V obeh primerih mora biti infrastruktura nameščanja avtomatizirana in ustrezno zgrajena.

Razlika med odločitvijo izbire CD in CDep ni tehnična, temveč poslovna. Kot razlaga Sharma (2013), je CDep primeren za t. i. *.com* podjetja, ki ponujajo različne storitve ciljnim kupcem na internetu, kjer na produkcijo vpeljejo vsako narejeno spremembo, ki je prestala namestitveni tok vse do konca. Za aplikacije, ki so uporabljene interno, priporoča uporabo CD, saj je izdaja verzij na produkcijo manj frekventna. Tudi če podjetje ne izdaja novih verzij na produkcijo tako pogosto, lahko preko CD hitreje odpravi napake.

Slika 11: Primerjava neprekinjene dostave in namestitve



Vir: Y. Sundman, Continuous Delivery vs Continuous Deployment, 2013.

1.3.4 DevOps

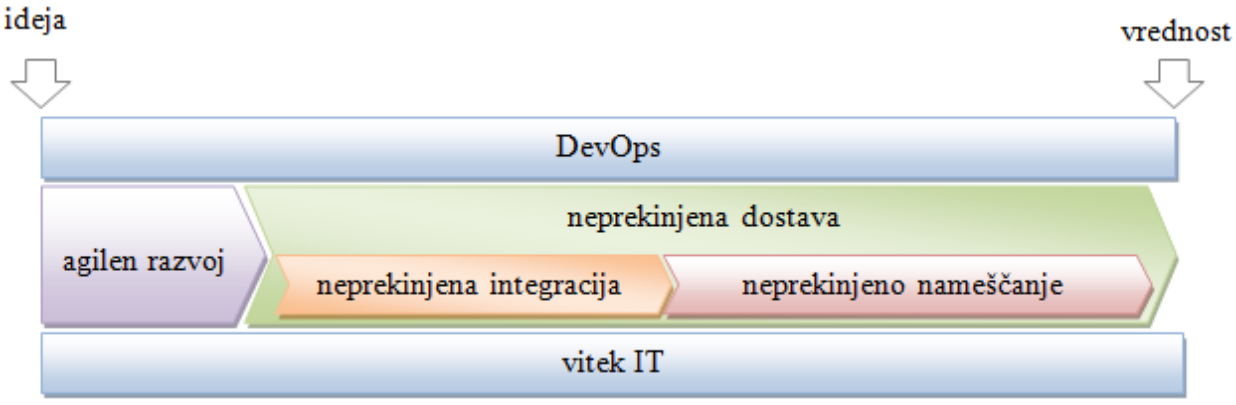
DevOps je relativno nova skovanka dveh izrazov – razvoj (angl. *development*) ter operacija (angl. *operations*), ki predstavlja evolucijo agilnih metodologij (Bradley, 2015). Kot navaja Bradley, samo izrazoslovje sploh ni pomembno. Dejstvo je, da uporaba omenjenih metodologij prinaša kombinacijo naslednjih učinkov: hitrejši čas implementacije sprememb, večjo stabilnost oz. zanesljivost, prav tako pa se sprosti čas za osredotočenje pomembnih nalog, ki z vidika podjetja prinašajo vrednost. Izraz DevOps predstavlja filozofijo, katere namen je ustvariti učinkovito sodelovanje in komunikacijo med razvijalci programske opreme in ostalimi strokovnjaki IT (Ravishankar, 2014). Predstavlja neposredno povezanost oz. komplementarnost z agilnimi metodami razvoja programske opreme.

DevOps omogoča razvijalcem pospešen cikel testiranja sprememb, s čimer prej identificirajo potencialne nepravilnosti (Jackson, 2013). Nova verzija aplikacije je torej v krajšem času implementirana na produkcijsko okolje. Rivera in van der Meulen (2015) menita, da je DevOps filozofija, ki prinaša kulturni premik znotraj organizacije, ki povezuje razvijalce aplikacij in

ostale strokovnjake IT v podjetju. Trenutno DevOps pozicionira kot nišno strategijo, ki bo v letu 2016 prešla v širšo uvedbo v podjetjih.
Večina metodologij razvoja programske opreme ne obravnava detajlnih namestitvenih procesov, ki so na splošno velikokrat prezrti. Namestitveni tok pogosto zahteva sodelovanje različnih strokovnjakov. V tradicionalnih organizacijah imajo ti strogo ločene naloge, hkrati pa potekajo informacije in samo sodelovanje preveč silosno. Zato v praksi obstaja velikokrat prepad med razvijalci in sistemskimi administratorji, a vendar morajo ti pri zagotavljanju čim ustreznejšega IS sodelovati in prispevati k enotnemu cilju. Razvoj programske opreme predvideva tudi testiranje programske kode, kjer se preverja ustreznost rešitve s številnih zornih kotov. Danes imajo podjetja za testiranja namenska okolja, kamor se predhodno namestijo tudi zaključeni programski sklopi, ki so rezultat razvoja. Zaradi problematike prehajanja novih verzij preko različnih okolij in želje po kakovostni in hitri dostavi programske opreme so se začeli pojavljati pristopi, kot je npr. CD, ki ga Bird (2015, str. 22) opisuje kot hrbtenico filozofije DevOps.

Na Sliki 12 je prikazana umestitev DevOps z ostalimi pristopi in metodologijami, katerih namen je uresničevanje zastavljenih ciljev, ki so povezani z razvojem aplikacij v podjetju. Kot vidimo, je za uspešno integracijo te kulture pomembno, da nov način dela posvojijo tisti, ki so vpleteni v proces dostave aplikacij in prilagajanje le-teh. Kot je razvidno s Slike 12, je predpogoj za uspešno izvajanje celotnega procesa razvoja programske opreme vse do končne dodane vrednosti naročniku zasledovanje ideje vitkosti, ki mora biti zastopana na vseh ravneh. Principi izraza vitkega IT izhajajo iz izraza vitke proizvodnje lahkih storitev (Riezebos & Klingenberg, 2009, str. 237). Ta je narekovala eliminacijo vseh nepotrebnih aktivnosti v poslovnih procesih z namenom razbremenitve podjetja balasta. Zavzema se torej za optimizacijo in izboljšave izvajanja poslovnih procesov, ki so med drugim tudi CI in CD. Večina agilnih metod razvoja programske opreme vsebuje koncepte vitkosti. Vse skupine v procesu dostave sprememb končnim uporabnikom naj bi tako upoštevale krovno metodologijo DevOps.

Slika 12: Pristopi v razvoju



Vir: Prirejeno po B. Martin, DevOps, Agile Development, Continuous Delivery und ITIL: Was gibt es für Zusammenhänge und wie man sie nutzen kann, 2014;M. Marschall, How are Lean, Agile, and Devops related to each other?, 2012.

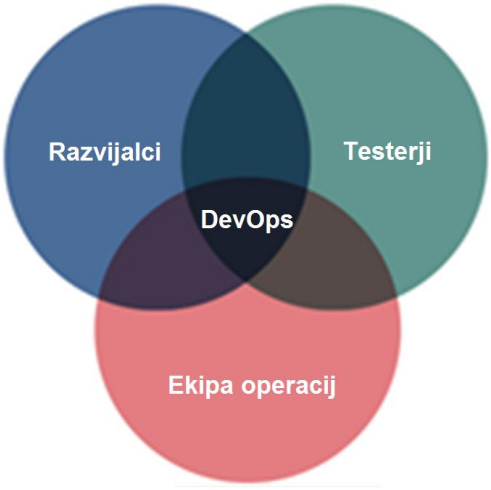
Poudariti je treba, da delovanje razvijalcev in zasledovanje koncepta DevOps ne pomeni nikakršnih sprememb dela, kot ga zapoveduje določena agilna metoda. Krovni koncept DevOps grobo vpliva na delovanje razvijalcev na kulturni in komunikacijski ravni. To pomeni, da se morajo zavedati tako razvijalci kot tudi ostali strokovnjaki IT, da so del procesa in da vsi skupaj zasledujejo višje, skupne cilje, in sicer je to dodajanje vrednosti uporabnikom oz. naročniku. Upoštevanje principov DevOps, agilnih pristopov s strani obeh skupin torej prinaša merljive pozitivne učinke podjetju. Pomembno je, da se vsi trije (vitkost, agilnost, DevOps) vidiki uporabljajo skupaj, saj le tako dosežemo večje učinke na dodajanje vrednosti podjetju (Marschall, 2012).

Učinkovito sodelovanje znotraj kot tudi zunaj različnih skupin je ključnega pomena za uspeh projekta razvoja programske opreme. Ne glede na to, kako uspešni sta skupini, ob neučinkovitem povezovanju, predajanju informacij in sodelovanju lahko prihaja do t. i. ozkega grla (angl. *bottlenecks*) ali pa do neuspešnih namestitev, ki zavirajo razvoj in dostavo programske opreme.

Razvoj programske opreme velikokrat zahteva tudi pridobivanje koristnih informacij s strani uporabnikov. Tudi tukaj sta vključeni obe skupini, ki sta usmerjeni v skupen cilj k prispevanju ustreznih rešitev za podjetje. Gradnja kulture sodelovanja mora biti vzpostavljena na nivoju razvoja in operacij ter z ostalimi vpletenimi, ki imajo ključne informacije v povezavi z reševanjem določenega problema, povezanega z delovanjem programske opreme. Ideja je torej celostno optimizirati izvajanje razvojnih procesov pri dostavi novih funkcionalnosti in z ustrezno kulturo ustvariti predpogoje za k cilju usmerjeno strokovno sodelovanje.

Najlažje lahko pomen komunikacije in gradnje kulture prikažemo s primerom dodelitve naloge operacijam s strani razvijalcev. Dostikrat obstaja mnogo možnih rešitev za določen problem. Od vidikov razvijalcev, njihovih znanj, želja je odvisno, kako si določeno rešitev zamislijo. Naloga posameznika v operacijah (npr. sistemski administrator) je, da ponujeno rešitev, ki neposredno vpliva na njegovo področje, ustrezno preveri. Če ima glede predlagane rešitve kakšne pomisleke, poda tudi konstruktivno mnenje. V tem primeru se je treba strokovno pogovoriti, katera rešitev bi bila najboljša za izvedbo. V nekaterih podjetjih imajo pri dostavljanju razvitih sprememb na produkcijsko okolje namenski oddelek za testiranje. Na Sliki 13 je prikazana sinergija treh skupin, ki skupno najbolje izkoriščajo prednosti, ki jih prinaša DevOps.

Slika 13: Sinergija DevOps

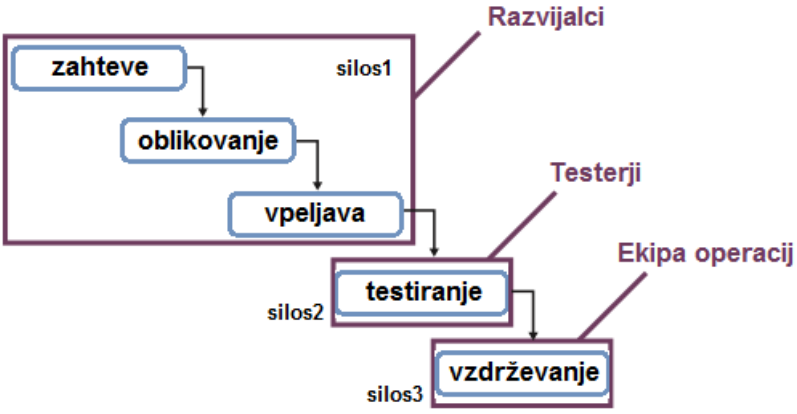


Vir: C. A. Cois, DevOps and Agile, 2014.

Sodelovanje prispeva tudi k učenju in pridobivanju oz. predajanju medsebojnih znanj. Smiselno bi bilo izvesti rotiranje delovnih nalog med razvijalci in operacijami (Ravishankar, 2014). S tem bi imeli vsi vpleteni širše znanje, celovitejše razumevanje in manj informacijskih šumov pri medsebojnem sodelovanju (Hammond, 2011).

Cois (2014) pravi, da je DevOps pravzaprav evolucija agilnega razmišljanja, saj preko komunikacije odstranjuje meje silosov, ki so prisotni pri slapovnem razvoju, kar prikazuje Slika 14.

Slika 14: Organizacijski silosi pri slapovnem modelu



Vir: C. A. Cois, DevOps and Agile, 2014.

Namestitve novih verzij oz. posledično prilagajanje infrastrukture IS zahteva izvedbo nalog sistemskega administratorja. Med samim namestitvenim tokom lahko v fazah nameščanja oz. prilagajanja infrastrukture IS prihaja do najrazličnejših napak (nedelovanje aplikacij …). V tem primeru je proces želje po dostavi novosti uporabnikom ustavljen. Za nadaljevanje razvoja in s tem dostave aplikacij na produkcijsko okolje je torej nujno medsebojno sodelovanje (sistemski administrator, razvijalec). Skupno ugotavljanje vzrokov nedelovanja skrajša čas cikla pri izdaji novih verzij.

Potreba po aktivnem sodelovanju se kaže tudi pri vzdrževanju IS. Spremljanje delovanja aplikacij, na katerem od okolij preko nadzornih sistemov, je ena od takšnih aktivnosti. To zahteva aktivno vpletenost tako razvijalcev kot tudi sistemskih administratorjev. S proaktivnim pristopom se ne izvaja samo prenosa informacij, ampak tudi medsebojno dopolnjevanje različnih strani, ki sta usmerjeni k rešitvi.

Pomembno je tudi, da obe skupini čim bolje poznata medsebojne naloge in specifike, saj se s tem odpravljajo morebitni nesporazumi oz. informacijski šumi. Poleg tega se s sodelovanjem odpravi kulturne omejitve, izboljša se organizacijska klima in poznavanje IS v podjetju. Vse našteto prinaša številne pozitivne učinke, ki ustvarijo spodbudno okolje, kjer obstaja večja možnost za inovacije.

Kot dodaja Allspaw (2010), DevOps temelji na spretnem sodelovanju in medsebojnem komuniciranju v organizaciji. Ravno zaradi prisotnosti mehkih veščin je vpliv DevOps dostikrat težko definirati. Z dobro komunikacijo in sledenjem istim ciljem bodo rezultati na spremembo kulture v podjetju pozitivni, kar bo vplivalo na hitrejše izvajanje procesov z manj napakami.

Potreba po seznanitvi operacij s poslovnimi zahtevami po izdelavi posameznih funkcionalnosti se kaže po skupnem konstruktivnem iskanju najprimernejših celovitih rešitev. Podjetje potrebuje hitre in učinkovite rešitve implementacij novih funkcionalnosti v podjetju, ki bodo prinašale dodano vrednost. Vpetost operacij v razvojne cikle nakazuje potrebo po vpeljavi DevOps, ki zajema spremembe na številnih ravneh (Ravishankar, 2014):

- organizacija – vitkost, optimizacija, zmanjšanje kompleksnosti,
- avtomatizacija procesov,
- sprememba kulture,
- sodelovanje.

1.4 Možnosti in načini avtomatizacij namestitev v IS

Že sam namen informacijskih rešitev in IS je, da avtomatizira določene naloge, ki so se prej reševale ročno. Tako se je podpisovanje papirnih dokumentov, računanje, načrtovanje, statistične analize, raziskovanje itd. izvajalo ročno, kjer so povezane aktivnosti predstavljale časovno potratne in koordinacijsko zahtevne naloge. Računalniška tehnologija in uvajanje informatizacije v podjetja je te aktivnosti skozi čas v čim večji meri avtomatiziralo. Podjetje ima od IS neposredne koristi, zato so vlaganja v informatizacijo in s tem povezano avtomatizacijo aktivnosti postala pravzaprav nujna za konkurenčnost podjetja.

S podobnimi težavami se sooča tudi informatika v podjetjih znotraj lastnih procesov. Uporaba IT je v podjetjih do nedavnega veljala za relativno nov način doseganja večje učinkovitosti v izvajanju ključnih poslovnih procesov. Številne prednosti uporabe IT in tudi nove tehnološke rešitve so podjetja silila v čim obsežnejšo informatizacijo podjetja. Veliko energije in časa se je v

preteklosti tako porabilo za realizacijo želenih nalog. Razvoj in rast IS v podjetjih se kljub zdaj že zreli panogi IT nadaljuje. Novosti v IT podjetjem ponujajo nove možnosti poslovne uporabe in koriščenje prednosti njihove vpeljave. Virtualizacija, računalništvo v oblaku, infrastruktura kot koda (angl. *infrastructure as code*), standardizacija, možnost učinkovite integracije med viri v IS in številne prednosti, ki jih ponuja moderna tehnologija v IS, odpirajo podjetjem nadaljnje priložnosti za izboljšave. Tehnološka zmožnost in tudi uveljavljeni procesi ponujajo možnosti avtomatizacije tudi znotraj lastnih procesov informatike.

Ena od teh možnosti je vsekakor namestitveni tok, ki predstavlja vodilo, po katerem se preko številnih aktivnosti izvaja izdaja novih funkcionalnosti aplikacij do končnih uporabnikov. Namestitveni tok je implementiran v podjetjih, ki se poslužujejo lastnega razvoja aplikacij. Na njegov končen tok aktivnosti in nivo avtomatiziranosti lahko vpliva veliko dejavnikov (velikost razvoja, denarna sredstva, kompleksnost IS, strokovnost, naklonjenost vodstva itd.), njegov obstoj pa je pogojen z izdajo sprememb programskih rešitev končnim uporabnikom. Kako je avtomatiziran, če sploh je, pa je odvisno od podjetja samega. Najverjetneje se v vsakem namestitvenem toku najde prostor za izboljšave, kjer novih izzivov ne manjka že zaradi neprestanega spreminjanja okolja (razvoj, potrebe podjetja, infrastruktura itd.), v katerega je vpet.

Humble in Farley (2011, str. 11–12) se strinjata, da je treba spremembo uvesti čim prej, ko se pojavi upravičena potreba po njej. Čas od nastanka potrebe po spremembi in do njene realizacije mora biti čim krajši, saj tako dosežemo nižje oportunitetne stroške. S hitro vpeljavo lahko podjetje uveljavlja konkurenčno prednost. Vendar avtorja nadaljujeta, da hitrost vpeljave ni dovolj, saj brez ustrezne kakovosti programske rešitve ne dosežemo želenih učinkov. Podjetje si torej mora prizadevati čim hitreje uvajati kakovostne programske rešitve, in sicer na učinkovit in zanesljiv način.

V poglavju 1.3.3 je predstavljen namestitveni tok, ki ga sestavljajo številni koraki in je njegova hitrost dostikrat pogojena s hitrostjo izvajanj ročnih aktivnosti zaposlenih. V idealnem primeru bi hitrost sposobnosti novih izdaj narekoval sam razvoj aplikacij, poslovna odločitev pa bi bila kdaj spremembe do uporabnikov dejansko dostaviti. Avtomatizacija namestitvenega toka ponuja odgovor, kako pospešiti in izboljšati proces prehoda realiziranih sprememb preko vseh korakov testiranja in potrjevanja primernosti sprememb ter pri izdaji realiziranih sprememb na produkciji. Pravzaprav je vpeljava avtomatizacije v namestitveni tok ena od glavnih prioritet razvoja programske opreme v okviru CD.

Avtomatizacijo namestitvenega procesa lahko razdelimo na več povezanih podprocesov, ki jih lahko obravnavamo ločeno. Pomembno je le, da so posamezni vhodi in izhodi med podprocesi med seboj povezani, obvladljivi in transparentni. Procesno gledano je optimalno, da celotni namestitveni tok ne vsebuje t. i. ozkih grl, kar pomeni, da se mora vsak podproces izvajati s podobno propustnostjo. Nerealizacija popolne avtomatizacije procesa nameščanja se navadno odraža v časovnih zastojih ročnega dela. Prvi korak v sklopu avtomatizacije namestitvenega procesa je proces CI, ki je podrobneje opisan v poglavju 1.3.1. Najbolj znana orodja za

neprekinjeno avtomatizacijo so (List of Build Automation Software, 2010): Jenkins, AnthillPro, GitLab CI, Bamboo itd.

Razloge za vpeljavo avtomatizacije v namestitvene procese podkrepijo številne slabosti ročnega nameščanja: veliko koordiniranih aktivnosti, slaba preglednost nad procesom, velika možnost za napake, veliko skritega znanja (čeprav naj bi bilo vse dokumentirano), časovna potratnost, človeški faktor, enolična opravila, ponavljajoče naloge, višja tveganja nedelovanja produkcije, počasne morebitne ročne povrnitve v prejšnje stanje itd. Morebitno nekonsistentno izvajanje ročnih namestitvenih aktivnostih lahko vpliva na težko opazne nepravilne konfiguracijske nastavitve oz. na kakršnakoli odstopanja na podatkovnem nivoju. S takšnimi napakami lahko utrpimo visoko škodo predvsem zaradi zapletene ureditve stanja, npr. transakcij, ki so se izvedle v času prisotnih nepravilnostih. Takšnih napak si podjetja na produkcijskem okolju ne smejo privoščiti. Nepravilnosti različnih nastavitev na testnih okoljih pa lahko vplivajo tudi na nerealne teste aplikacij, s čimer se izniči sam namen teh okolij.

Ker so namestitvene aktivnosti in povezani procesi ponovljivi in tudi vnaprej predvidljivi, se lahko v številnih primerih izvajajo avtomatično, brez ročnega poseganja v sam proces. Avtomatizacije namestitev se lahko lotimo z razvojem lastnih rešitev oz. z nakupom namenskih orodij. Obe rešitvi imata svoje prednosti in slabosti. Če namestitveni tok v okviru CD v podjetju nima posebnih specifik, ki bi pomenile nezmožnost implementacije, s katerim od orodij za avtomatično izdajanje aplikacij (angl. *application release automation,* v nadaljevanju ARA), je verjetno smiselnejši nakup orodja na trgu. Specifika orodij ARA je podrobneje obravnavana v poglavju 1.5.

Betteley (2016) navaja, da avtomatizacija namestitev prinaša prednosti, ki so prikazane na Sliki 15: hitrost, zanesljivost, ponovljivost, preglednost, povrnitev prejšnjega stanja, metrike.

Slika 15: Prednosti avtomatizacije namestitev

Vir: J. Betteley, Deployment Automation Patterns, 2016.

Prilagajanje podjetja, novi produkti, hitro spreminjanje povpraševanja in številni drugi vzroki zahtevajo ustrezne odzive ne samo preko prilagajanja aplikacij, ampak tudi same infrastrukture IS. Skalabilnost, modularnost, elastičnost in standardizacija je samo nekaj lastnosti, ki naj bi jih imela moderna računalniška infrastruktura v IS. Kot je mogoče razbrati iz raziskave »Avoiding the Alignment Trap in IT« (Shpilberg, Berez, Puryear & Shah, 2007), se s tem tudi postavijo ustrezni predpogoji, ki odpravljajo omejevanje oz. ozka grla, ki ga predstavlja IT pri rasti podjetij.

Prilagajanje aplikacij in infrastrukture IS poteka pravzaprav z roko v roki, saj mora biti infrastruktura na eni strani sposobna servirati vse zahteve, na drugi strani pa ponujati podjetju fleksibilno okolje pri podpiranju poslovanja. Pogoji za izvedbo takšne t. i. agilne infrastrukture tičijo predvsem v sami arhitekturni zasnovi in uporabljenih tehnoloških rešitvah. Način, kako takšno infrastrukturo IS čim učinkoviteje upravljati, je ta, da se z le-to manipulira kot s kakršnokoli programsko kodo (angl. *Infrastructure as Code*, v nadaljevanju IAC). IAC predstavlja način upravljanja infrastrukture v IS, kjer se preko opisnega jezika definirajo značilnosti posamezne komponente v IS (strežniki, podatkovna baza itd.) (Rouse,2016).

Danes obstajajo številna takšna orodja za avtomatizirano upravljanje konfiguracij (angl. *configuration management tools*, v nadaljevanju CMT), ki omogočajo uporabo prej opisanih funkcionalnosti v duhu številnih dobrih praks, agilnih pristopov oz. DevOps kulture. Ker opisovane rešitve niso del magistrskega dela, a vendar predstavljajo nadaljevalno stopnjo avtomatizacije procesa CD in infrastrukture v IS, so samo našteta trenutno najbolj znana orodja na trgu: Puppet, Chef, Salt, Ansamble. V primeru, da ima podjetje že avtomatiziran proces CD in prisotno pogosto prilagajanje IS, je o morebitni vpeljavi rešitev CMT vsekakor vredno razmisliti.

Avtomatizacija je sicer zelo generičen in širok pojem, vendar se zaradi namena magistrskega dela osredotočimo predvsem na avtomatiziranje procesa CD. Sicer pa je v informatiki še veliko možnosti in procesov, kjer je mogoče marsikateri proces tudi avtomatizirati. Na splošno je brez uporabe umetne inteligence možno avtomatizirati samo predvidljive procese. Strojno učenje je danes še relativno novo področje, vendar se počasi, a vendarle vključuje tudi v najrazličnejšo avtomatizacijo.

Analitska hiša Gartner avtomatizacijo IT deli na 10 različnih segmentov, ki so prikazani na Sliki 16. Večina avtomatizacijskih rešitev je namenjena konfiguraciji in upravljanju sistemov. Nekatere rešitve omogočajo avtomatizacijski pristop, ki ga definiramo preko najrazličnejših programskih jezikov in s katerim lahko upravljamo številne sisteme (strežnike, usmerjevalnike itd.). Druge rešitve pa omogočajo povezano avtomatizacijo in orkestracijo več procesov skupaj, različnih tehnologij in rešitev upravljanja sistemov. V vseh kategorijah zaenkrat ni niti ene rešitve, ki bi bila zmožna avtomatizirati popolnoma vse scenarije (Naegle, 2015, str. 3).

Slika 16: Različni segmenti avtomatizacije IT

Vir: R. Naegle, IT Market Clock for IT Automation, 2015.

1.5 Orodja za avtomatizacijo namestitev

CD predvideva, da je podjetje zmožno izvesti kakršnekoli spremembe in to kadarkoli na pregleden ter avtomatiziran način. Kot pravita Humble in Farley (2011, str. 253), se mora nameščanje aplikacij izvajati na zanesljiv in dosleden način, kjer se izvede enak proces na vseh okoljih. Vsaka namestitev na posameznem okolju, ki je del namestitvenega procesa, naj bi omogočala vpletenim v namestitveni tok tudi povratne informacije o uspešnosti izvedbe.

Namestitve večnivojskih aplikacij so lahko poljubno kompleksne, hkrati pa je velikokrat pomembna tudi ustrezna orkestracija vseh aktivnosti v sklopu določene namestitve.

Glede na to, da namestitveni tok predstavlja osrednji del magistrskega dela, so v nadaljevanju opisani glavni koncepti orodij ARA, ki modelirajo ročne procese namestitev na standardiziran in avtomatičen način, kar predstavlja eno od ključnih prednosti v sklopu namestitvenega toka. Avtomatizacija omenjenih procesov je ena ključnih nalog strokovnjakov na področju infrastrukture in operacij v IS (angl. *infrastructure and operations (I&O) professionals*) v podjetjih (DeMartine, Bittner, 2015, str. 2). Avtomatizirana namestitev lahko obsega prenašanje najrazličnejših artefaktov, aplikacij, konfiguracij in podatkov na najrazličnejša okolja skozi celoten življenjski cikel aplikacije (Naegle, 2015, str. 25). Avtomatizacija namestitev prinaša tudi konsistenten način sprememb preko testnih okolij vse do izdaje novih funkcionalnosti končnim uporabnikom na produkcijskem okolju. Orodja omogočajo modeliranje, definiranje okolij ter celotne orkestracije namestitev, s čimer zagotavljajo kakovost in hitrost izdajanja aplikacij (Fletcher & Colville, 2015, str. 1). Vpeljava orodja ARA naj bi podjetju prinesla naslednje koristi (Naegle, 2015, str. 26; DeMartine & Bittner, 2015, str. 1):

- agilnost in produktivnost, ki prinašata hitrejšo dostavo aplikacij,
- zmanjšanje stroškov zaradi eliminacije dragih ročnih dostav,
- s pomočjo konsistentnega, standardiziranega in dokumentiranega načina dostavljanja novih verzij aplikacij se zmanjšuje tveganje (npr. nedelovanja produkcije),
- zmanjševanje kompleksnosti sistema,
- enostavnost namestitev,
- nekatera orodja omogočajo modeliranje namestitvenega procesa.

Kot napoveduje analitično podjetje Gartner, bo orodja ARA, kot del iniciative DevOps, do leta 2018 implementiralo kar 50 % podjetij (Fletcher & Colville, 2015, str. 1). Ročno izdajanje novih verzij preko namestitvenega toka terja veliko koordinacije. Orodja ARA podjetjem nudijo stabilnejše, zrelejše in integrirano izvajanje namestitvenega toka, ki nadomešča ročno izvajanje, izvajanje pripravljenih namestitvenih skript ali pa izvajanje namestitvenih aktivnosti, preko katerega od sistemov za CI. Ta orodja omogočajo hitrejše, preglednejše in nadzorovano nameščanje aplikacij preko vseh okolij, vse do implementacije na sami produkciji v sklopu cikličnega spreminjanja IS in prilagajanja spremembam. Spremembe so tako lahko hitreje implementirane na produkcijsko okolje.

Prednost orodij ARA je tudi v tem, da omogočajo najrazličnejšim skupinam znotraj podjetja (skupina operacij, razvijalci, sistemski administratorji, ekipa za testiranje itd.) samopostrežne namestitve (opolnomočenje), kjer je omogočen nadzor in potrjevanje izvedb (npr. produkcijska okolja). Širši skupini izvajalcev je tako preko enotne točke omogočeno enostavno, nadzorovano nameščanje in tudi pregled nad vsem namestitvami. Orodja ARA predstavljajo osrednjo vlogo v sklopu kulture DevOps in so vključena v namestitveni tok pri dostavljanju čim boljših storitev. Z vpeljavo produktov ARA se avtomatizira velik del verige pri dostavljanju novih aplikacij.

Pozitivni učinki se pokažejo pri razvoju programske opreme s frekventnim izdajanjem novih verzij na podlagi CD in DevOps (Fletcher & Colville, 2015, str. 3-4).

Orodja različnih ponudnikov variirajo pri načinu delovanja in modeliranja namestitvenih aktivnosti znotraj namestitvenih procesov, pa tudi v povezovanju z ostalimi rešitvami, ki jih uporabljamo v dostavi programske opreme tekom življenjskega cikla programske opreme (DeMartine, 2015, str. 1). Konceptualno lahko, glede načina uporabe in modeliranja namestitev na nivoju aplikacije, orodja ARA delimo na naslednje sklope (Cherry, 2014).

- **Paketno usmerjen pristop (angl. *package-based approach*)**

Paketno usmerjena orodja ARA realizirajo avtomatizirano nameščanje paketov na različnih strežnikih. Paket predstavlja osrednji del nameščanja, ki vsebuje poljubne spremembe, ki se namestijo na posamezen strežnik. Produkta, ki sta paketno usmerjena, sta BMC Bladelogic (BMC Application Automation) in IBM Tivoli Provisioning Manager (TPM).

- **Deklarativno usmerjen pristop (angl. *declarative-based approach*)**

Podoben pristop kot paketen predstavljajo orodja, ki realizirajo namestitve s pomočjo deklarativnega pristopa. Tudi tukaj je logika podobna, saj se deklarativno navede, kaj vse se bo spremenilo na poljubnem številu strežnikov. Preko deklarativnega jezika se definira spremembo za vsako komponento posebej na posameznem strežniku. Skupek predvidenih sprememb za posamezen strežnik se izvede na različnih strežnikih. Najbolj znan produkt, ki uporablja deklarativni pristop, je Puppet.

- **Ukazni pristop (angl. *imperative-based approach*)**

Zgleduje se po proceduralnih programskih jezikih, kjer se avtomatizirana namestitev definira s pristopom pisanja procedure namestitve, ki se jo realizira z uporabo programskega jezika. Z razvito proceduro, napisano v programskem jeziku, se podrobno definira vse korake, ki se morajo izvesti v sklopu posamezne namestitve. Najbolj znan predstavnik opisanega pristopa je orodje Chef. Tudi tukaj se predvideva ponavljanje enakega postopka na vseh predvidenih strežnikih v sklopu določene namestitve.

- **Procesno usmerjen pristop (angl. *process-based approach*)**

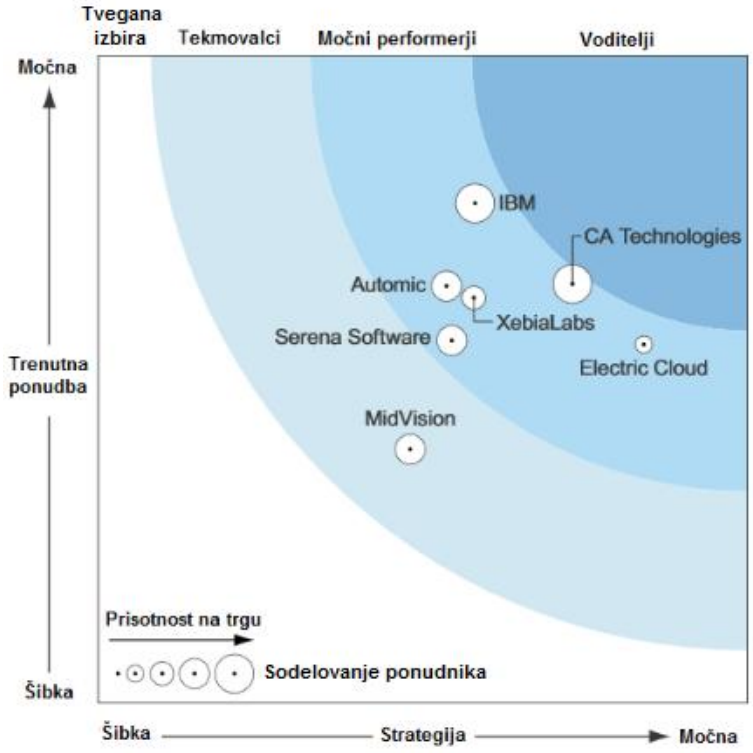
Omogoča avtomatizacijo obstoječih namestitvenih procesov. V nasprotju s prej opisanimi pristopi, ki predvidevajo ponavljanje enakih namestitev na vseh strežnikih, omogočajo procesno usmerjena orodja tudi njihovo orkestracijo. Število sprememb na posameznem strežniku je lahko poljubno veliko in nemalokrat se zgodi, da je vrstni red sprememb na strežnikih zelo pomemben, če želimo uporabljati dosedanje namestitvene postopke. Kar opisni pristop omogoča, je zmožnost orkestracije in pravzaprav uporabe enakih namestitvenih procesov, kjer le-teh ni treba prilagajati zaradi avtomatizacije namestitev, kot pa je to potrebno pri ostalih. Orodja ponujajo vizualno gradnjo procesov, kjer se definira vse komponente, ki bodo vključene v namestitev, vse strežnike, ter definiranje procesno povezane orkestrirane namestitve. Najbolj znana predstavnika procesno usmerjenega pristopa sta IBM UCD in Nolio. Ker je namestitveni tok pravzaprav preslikava ročnega nameščanja, lahko avtomatiziran proces predstavlja dokumentacijo izvajanja namestitev.

- **Lasten razvoj (angl. *generic and custom-built approaches*)**

Podjetje se lahko odloči za lasten razvoj avtomatizacije namestitvenega procesa. Z uporabo skript oz. visokonivojskih programskih jezikov in z vložkom časa ter resursov lahko podjetje razvije avtomatizirane poljubne namestitvene postopke. Ti lahko predstavljajo v celoti avtomatiziran proces nameščanja ali pa poenostavitev aktivnosti pri deloma avtomatičnem nameščanju. Nesmiselno je razvijati rešitve, ki že obstajajo na trgu, saj bi bil razvoj predrag, s čimer rentabilnost izkoriščajo podjetja z ekonomijo obsega. Če le ni namestitveni tok preveč specifičen in nemogoč za implementacijo preko orodij ponudnikov na trgu, je tako najverjetneje bolj smiseln nakup orodij, ki so dostopna na trgu. Izvajanje manjših in enostavnejših namestitev pa je v veliko primerih čisto primerno z realizacijo lastnih skript in postopkov nameščanja.

Forrester je naredil primerjalno analizo sedmih najpomembnejših proizvajalcev ARA na trgu (DeMartine & Bittner, 2015, str. 5): Automic Software, CA Technologies, Electric Cloud, IBM, MidVision, Serena Software, VMware, Clarive, Inedo, HP in XebiaLab. Slika 17 prikazuje rešitve ARA različnih ponudnikov, ki so v graf razvrščeni glede na strategijo in ponudbo ter prisotnost na trgu. Od leve proti desni strani so nanizani ponudniki glede na strateški položaj. Os y prikazuje stopnjo funkcionalnosti, ki jo ponuja posamezen ponudnik, kjer je najvišja vrednost v zgornjem predelu slike. Velikost posamezne točke na sliki predstavlja sodelovanje ponudnikov, kjer pomeni večji obseg kroga boljšo podporo ponudnika. S slike lahko razberemo, da sta najboljša ponudnika IBM in CA Technologies. Stvar prioritete in potreb posameznega podjetja pa je, katera izbira bo najprimernejša.

Slika 17: Zrelost ponudnikov rešitev ARA



Vir: A. DeMartine & K. Bittner, The Forrester Wave™: Application Release Automation, Q2 2015, 2015, str. 5.

2 ANALIZA STANJA PRED UVEDBO NOVEGA ORODJA

V zadnjih letih je bila v podjetju realizirana celostna prenova IS. Prenova je zajemala centralizacijo vseh aplikacij in storitev na enotni lokaciji, vključen pa je bil tudi razvoj ključnih aplikacij, ki jih podjetje potrebuje za poslovanje. Prenova IS in vzpostavljanje lastnega razvoja aplikacij ter njihove dostave je zahtevala tudi vzpostavitev internega namestitvenega toka. Obravnavano podjetje ima svoje temeljne poslovne procese v IS podprte večinoma z razvojem lastnih aplikacij. Skupaj sestavljajo medsebojno odvisno kompleksno mrežo aplikacij, ki kot celota podpirajo poslovanje podjetja. V konkretnem primeru se razvijajo spletne storitve oz. aplikacije, ki so čim bolj šibko sklopljene, s čimer se želi doseči neodvisnost posameznih komponent (Šibka sklopljenost, b. l.), ki se med seboj povezujejo in uporabljajo medsebojne funkcionalnosti.
Spletne poslovne aplikacije se izvajajo na centralni infrastrukturi IT in večinoma temeljijo na platformi Java Enterprise Edition. Z namenom učinkovitega razvoja in uporabe že razvitih funkcionalnosti, enostavne medsebojne integracije in posledičnih nižjih stroškov, je bila zato uporabljena storitveno usmerjena arhitektura (angl. *service-oriented architecture*, v nadaljevanju SOA). Aplikacije tako na enkapsuliran način ponujajo ostalim odjemalcem/aplikacijam uporabo njihovih funkcionalnosti.

V opazovanem primeru gre za organsko rast aplikacij, kjer se nove funkcionalnosti in spremembe vršijo inkrementalno. Z razvojem novih aplikacij, ki uporabnikom in ostalim povezanim aplikacijam ponujajo nove zmožnosti, pa podjetje nadaljuje s prilagajanjem svojega IS (angl. *business alignment*).

V procese nadaljnjega razvoja in vzdrževanja IS so vključeni številni strokovnjaki iz informatike. Razdeljeni so v različne skupine, ki medsebojno sodelujejo ter zagotavljajo uporabnikom čim boljše uporabniške izkušnje in nemoteno poslovanje. Skupine ljudi v informatiki bi lahko razdelili na naslednje:

- razvijalce – razdeljeni so v manjše skupine, vsaka od njih je zadolžena za razvoj določenega sklopa internih aplikacij,
- skupino operacij – skrbi za izvajanje oddaljenih operacij, podporo uporabnikom, klicni center itd.,
- sistemske administratorje – skrbijo za infrastrukturo IS v podjetju,
- ekipo za testiranje.

Kljub sedaj že končanemu projektu prenove IS si podjetje prizadeva za neprestano izboljševanje in prilagajanje celotnega IS. V IS je tako preko 100 ključnih internih aplikacij, ki podpirajo osnovno dejavnost poslovanja (interne aplikacije, internetno poslovanje), ter ostale podporne storitve. Pristop izbire lastnega razvoja omogoča, da se lahko v celoti prilagaja rešitve svojim

poslovnim potrebam. Ob ustreznem nivoju strokovnosti zaposlenih lasten razvoj posameznih funkcionalnosti skrajša čas, prav tako pa zmanjša t. i. transakcijske stroške, ki so višji v primeru odvisnosti razvoja od zunanjih pogodbenih partnerjev. V nadaljnjo obravnavo so vključene aplikacije, ki so za podjetje ključnega pomena ter jih uporablja največji delež ljudi.

V interni razvoj je vključeno večje število internih razvijalcev ter ostalih strokovnjakov, ki skupaj obvladujejo prilagajanje IS v podjetju. Če želi podjetje izkoriščati prednosti internega razvoja, morajo biti vključeni kadri usmerjeni k izboljšavam, delovati timsko in skrbeti za strokovno rast. Skupaj z ustrezno organizacijsko kulturo pa dosežemo predpogoj za učinkovito izvajanje in nadaljnjo rast IS v podjetju.

Aplikacije so grajene po principu večnivojskega aplikacijskega modela. To pomeni, da so različne komponente aplikacije ločene v več nivojih (predstavitveni nivo, logični nivo in podatkovni nivo). Tako vsak nivo opravlja svojo funkcijo, s čimer dosežemo boljšo skalabilnost, preglednost in nadzor nad spremembami (Esposito & Saltarello, 2014, str. 39). Po drugi strani pa avtorja pravita, da ima večnivojska arhitektura tudi slabost, ki vpliva na bolj kompleksen in časovno potratnejši način nameščanja sprememb na vse komponente (Esposito & Saltarello, 2014, str. 129). V večini primerov so različni nivoji aplikacij razporejeni po naslednjih komponentah:

- spletnih strežnikih,
- aplikacijskih strežnikih,
- podatkovnih strežnikih.

2.1 Namestitev

Namestitev aplikacije, ki je razvita po principu večnivojskega aplikacijskega modela, pomeni, da se po potrebi eno ali več pripadajočih komponent ustrezno fizično nadomesti z novejšo ali pa se ji spremeni vsebino. V primeru gručenja strežnikov se mora namestitev izvesti na vseh instancah ter tudi na nivojih, ki sestavljajo aplikacijo. Pri nameščanju moramo upoštevati pravilni vrstni red aktivnosti, ki jih izvedemo (kopiranje, ponovni zagon strežnika, sistemski ukazi, spreminjanje vsebine konfiguracijskih datotek, spreminjanje na podatkovni bazi itd.). V našem primeru namestitev aplikacije največkrat zajema naslednje komponente:

- aplikacijski paket, ki predstavlja zapakirano, prevedeno aplikacijo v datoteki s končnico .ear oz. .war,
- konfiguracijsko datoteko posamezne aplikacije,
- spremembe na podatkovnem nivoju, preko izvajanj skript SQL na podatkovni bazi.

V nadaljevanju je zaradi poenostavitve uporabljen enoten izraz nameščanje ali namestitev aplikacij, ki lahko zajema katerokoli prej navedeno spremembo.

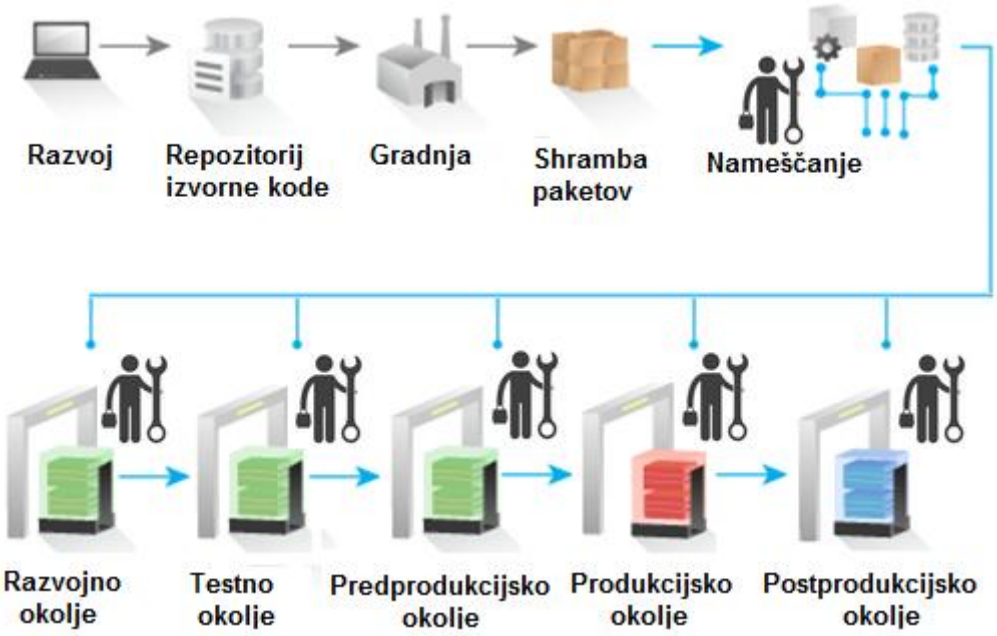
Predvsem pri nameščanju, ki zadeva izvedbo tudi na podatkovni bazi, je pomemben vrstni red nameščanja (orkestracija). Načeloma velja pravilo, da se spremembe izvaja od spodaj navzgor (podatkovna baza, aplikacijski strežnik, spletni strežnik). Kljub lastni usmerjenosti k standardizaciji postavljene infrastrukture in skalabilnem načrtovanju IS pa obstajajo razlike in specifike, ki so vezane na določeno aplikacijo v sistemu. To pomeni, da namestitvene aktivnosti, ki jih mora izvesti sistemski administrator, za vsako aplikacijo niso enake. Prisotne so tudi različne tehnologije (npr. aplikacijski strežniki, podatkovne baze, verzija izvajalnega okolja Jave itd.) in zasnove posameznih podsistemov, na katerih se izvajajo aplikacije. Vse to vpliva na raznolikost nameščanj in nujnost poznavanja vseh internih postopkov, ki niso dokumentirani, prav tako pa kompleksnost in frekventnost namestitev povečuje možnost napak pri samem nameščanju. Vsako namestitev izvede sistemski administrator na podlagi prejetega zahtevka z vsebino namestitvenih navodil. Namestitveno navodilo načeloma sestavlja, kaj se bo nameščalo, na katera okolja se bo namestilo ter tudi seznam potrebnih izvedb ali pa morebitno podrobnejše navodilo.

2.2 Namestitveni tok

Interni razvoj aplikacij poteka v mesečnih ciklih, kjer se izvajajo aktivnosti razvoja, testiranja aplikacij ter njihovo nameščanje preko testnih okolij vse do produkcijskega okolja. Zahteve in prioritete novih funkcionalnosti so dogovorjene na sestankih informatike in produktnih vodij. Viri pobud za nadaljnji razvoj aplikacij so različni. Razvoj aplikacije lahko narekuje zakonodaja, pobude pa največkrat prihajajo s strani uporabnikov oz. produktnih vodij. Omenjene zahteve se definira pred začetkom vsakega cikla nadaljnjega razvoja aplikacij, kjer se tudi uskladi seznam bistvenih funkcionalnih sprememb. Zaradi obvladovanja preko 100 aplikacij, ki skupaj podpirajo delovanje osnovne dejavnosti, so razvijalci razdeljeni v več skupin.

Slika 18 prikazuje pot enomesečnega cikla razvoja aplikacij s postopnim prehajanjem novih aplikacij preko različnih okolij vse do produkcijskega okolja. Glede na rezultate testiranj na posameznih okoljih morajo razvijalci ob kakršnikoli neustreznosti aplikacijo popraviti. Ko je popravek verzije razvit, se izvede ponovna namestitev in testiranje. Ta proces se ponavlja, dokler aplikacija ne prestane načrtovanih testov. Prehajanje aplikacij med okolji predstavlja namestitveni tok (angl. *deployment pipeline*), preko katerega se dostavlja vse spremembe, ki zadevajo prilagajanja aplikacij in posledičnega spreminjanja IS v podjetju.

Slika 18: Enomesečni cikel razvoja aplikacij



Vir: Prirejeno po UrbanCode Deploy, 2016.

Na Sliki 18 so prikazana tudi vsa okolja, preko katerih se izvaja namestitveni tok. Pri namestitvenem toku sodelujejo 3 skupine ljudi, ki so zadolžene za pravočasno dostavo kakovostnih rešitev do končnih uporabnikov:

- razvojna ekipa,
- ekipa za testiranje,
- sistemski administratorji.

Velik delež namestitev predstavljajo tudi najrazličnejši popravki obstoječih aplikacij. V večini primerov gre za verzije, ki odpravljajo performančne neustreznosti oz. zaznane napake s produkcijskega okolja. Popravki načeloma ne potrebujejo dostave na produkcijsko okolje preko celotnega namestitvenega procesa (vseh testnih okolij). Ker je treba vsako narejeno spremembo v aplikacijah predhodno potrditi na testnih okoljih, se neodvisno od namestitvenega cikla vzporedno izvaja tudi proces vzdrževanja aplikacij. Zato se takšna namestitev najprej izvede na predprodukcijskem oz. postprodukcijskem okolju (odvisno od faze 30-dnevnega namestitvenega cikla).

2.3 Okolja

Razlog za obstoj številnih okolij je v faznem testiranju različnih vidikov delovanja aplikacij, končno podobo vseh okolij pa dodaja specifika 30-dnevnega namestitvenega cikla. Vsaka namestitev posamezne aplikacije predstavlja izvedbo različnih aktivnosti, ki skupno predstavljajo uvedbo posamezne spremembe na določeno okolje. Interne spletne aplikacije

temeljijo na Java platformi in večplastnosti (n-nivojska arhitektura). Namestitve v našem primeru največkrat zadevajo poslovni nivo (spremembe na aplikacijskem strežniku) in podatkovni nivo (spremembe na podatkovni bazi). Glede na to, da spremembe funkcionalnosti lahko vplivajo na ustrezne prilagoditve infrastrukturnih komponent v IS (npr. omrežna infrastruktura, interni strežniki, virtualna okolja, širjenje komponent zaradi skalabilnosti itd.), so lahko v posamezno namestitev vključeni tudi infrastrukturni posegi. Takšnih posegov je veliko manj, kot pa običajnih namestitev samo aplikacij, zato se v nadaljevanju posvetimo frekventno intenzivnim in časovno potratnim namestitvenim aktivnostim.

Aplikacije v podjetju podpirajo temeljne poslovne procese, zato je njihovo delovanje izjemno pomembno za zmožnost nemotenega poslovanja podjetja. Obstoj in smiselnost vseh testnih okolij je pravzaprav odvisna od presoje podjetja, kakšno škodo predstavlja posamezen izpad ključnih storitev za poslovanje podjetja. V našem primeru predstavljajo podjetju izpadi visoke stroške, zato so zgrajena številna testna okolja. Ta so namenjena testiranju novih verzij aplikacij ter izvajanju namestitvenih posegov. Kombinacija obojega z ustreznim nadzorom omogoča zanesljivejše povratne informacije o produkcijski ustreznosti sprememb, izvedenih na aplikacijah, še preden te začnejo uporabljati končni uporabniki. Z izvajanjem namestitvenega toka tako zmanjšujemo možnost napak, performančnih težav, nedelovanja in morebitnih neustreznosti spremenjenih funkcionalnostih aplikacij na produkcijskem okolju.

V obdobju 30 dni se realizirajo vse načrtovane spremembe ter se izvedejo vsi testi, s katerimi se potrdi oz. zavrne produkcijsko primernost aplikacij. Vse faze (faza sprejemljivosti, uporabniški testi sprejemljivosti, faza zmogljivosti itd.) testiranja so namenjene odkrivanju in odpravljanju nepravilnosti ter optimizaciji aplikacij z različnih vidikov. V primeru uspešnih testov se cikel zaključi z namestitvijo na produkcijsko okolje. Če se v fazi testiranja ugotovi, da bi bil predviden prehod preveč tvegan, se lahko cikel tudi časovno podaljša. Namestitveni cikel tako zajema postopne prehode preko vseh testnih okolij, vse do končne produkcijske implementacije novih verzij aplikacij. Preko namestitve aplikacij in implementacije sprememb na posamezno okolje se pravzaprav izvede tudi test ustreznosti samega procesa namestitve kot tudi kasneje testi samih aplikacij. Z uspešno izvedbo namestitev na testnih okoljih se zagotovi preverjen proces namestitev, s čimer zmanjšamo verjetnost napak, ko izvedemo enak postopek namestitve tudi na produkcijskem okolju. Izvajanje namestitvenega toka oz. celotnega cikla se predhodno načrtuje in se tudi uskladi časovnico vseh aktivnosti. V nadaljevanju so podrobneje opisana vsa okolja, preko katerih se izvaja namestitveni tok:

- **razvojno okolje** – namenjeno sprotnemu testiranju izvedenih sprememb programske kode s strani razvijalcev. Razvijalci naj bi na razvojnem okolju konstantno preverjali primernost svojih rešitev. Testiranje na razvojnem okolju je bolj primerno kot na lokalnem računalniku razvijalca, saj so testi bolj realni, na tem okolju se namreč izvaja glavne soodvisne aplikacije. Kot priporoča CD, se naj bi nameščanje novih verzij aplikacij izvajalo čim bolj intenzivno. Pomembno je, da razvijalci neprestano testirajo čim manjše zaključene sklope sprememb, saj je tako lažje odkriti potencialne napake ter je mogoče imeti vedno delujoče verzije, ki jih postopoma razvijajo. Po vsaki namestitvi razvijalci preverjajo delovanje posameznih delov

aplikacije (angl. *unit test*), kjer so bile narejene spremembe. Glede na fazo cikla razvoja in odločitve odgovornih se po potrditvi ustreznosti verzije določene aplikacije ta namesti na ostala testna okolja;

- **testno okolje** – tukaj se izvajajo področni (angl. *unit*), funkcionalni in regresijski testi novih verzij aplikacij. Infrastruktura testnega okolja uporablja nekatere rešitve produkcijskega okolja (razporejanje prometa, gručenje itd.), vendar je to okolje veliko bolj poenostavljeno in manjše od produkcijskega;

- **predprodukcijsko okolje** – to okolje je infrastrukturno in kar se da podobno produkcijskemu. Na predprodukcijskem okolju se izvajajo intenzivni performančni in regresijski testi. Ker je zelo podobno produkcijskemu, se uporablja tudi za namene širših uporabniških testiranj in izobraževanj uporabnikov. Tu se namestijo potrjene verzije s testnega okolja, ki so v načrtu za produkcijsko namestitev. Preko performančnih testov ter z metričnim spremljanjem odzivnosti dobi ekipa za testiranje s pomočjo različnih nadzornih sistemov ključne informacije o ustreznosti novih verzij. Ob vsakem novem popravku aplikacij, ki so nameščene za predprodukcijsko okolje, se mora ponovno izvesti celoten namestitveni tok, vključno s testiranji, vse do predprodukcijskega okolja;

- **produkcijsko okolje** – ob ustreznih spremembah aplikacij, ki se potrdijo tudi s številnimi testi, se glede na poslovno odločitev spremembe izvede tudi na produkcijskem okolju. Zaradi zahteve po nemotenem delovanju produkcije se namestitve izvajajo izven delovnega časa, praviloma ponoči. Pri nameščanju na produkcijsko okolje se izvede enak postopek namestitve, kot se je izvedel na predprodukcijskem okolju. Vsaka sprememba na produkciji mora biti potrjena, nadzorovana, po izvedbi pretestirana ter ne nazadnje dokumentirana. Po namestitvi na produkcijsko okolje se s tem zaključi razvojni cikel posameznih funkcionalnosti in sprememb, ki so bile predvidene v času enega meseca. To pa ne izključuje neprestanega izboljševanja performanc oz. popravkov trenutnih funkcionalnosti. Napake in performančne težave se zaznava z najrazličnejšimi nadzornimi sistemi in orodji, kjer se razvijalcem posreduje čim bolj ustrezne informacije za njihovo odpravo. Izvedeni popravki morajo prav tako preiti faze testiranja na testnih okoljih;

- **postprodukcijsko okolje** – specifika tega okolja je, da v obdobju, ko so na predprodukcijskem okolju nameščene aplikacije z novimi funkcionalnostmi, prevzame postprodukcijsko okolje vlogo predprodukcijskega za testiranje popravkov in napak s produkcije. Ker se z novimi funkcionalnostmi aplikacij po navadi spreminja tudi podatkovni model, obstaja verjetnost, da popravki starih verzij, ki so razvite za različni model, sploh ne bi delovali.

Po končanju mesečnega razvojnega cikla z zaključno namestitvijo na produkcijskem okolju se celoten proces z novimi cilji ponovi od samega začetka. Neodvisno od faze mesečnega razvojnega cikla se najrazličnejši popravki aplikacij, predvsem zaradi napak oz. performančnih težav, izvajajo na dnevnem nivoju. Produkcijska primernost teh mora biti prav tako potrjena na predprodukcijskem okolju, preden se popravek namesti na produkcijo.

Opisani mesečni razvojni cikel glavnih aplikacij predstavlja eno pomembnejših in obsežnejših konstantnih izvajanj sprememb IS v podjetju. Pri ostalih aplikacijah lastnega razvoja ter zakupljenih rešitev se uporablja podoben koncept potrjevanja ustreznosti rešitev na ostalih, temu namenjenih okoljih. Testna procedura potrjevanja primernosti teh rešitev pa je odvisna od pomembnosti samih storitev.

Celoten namestitveni tok je vpet v široko okolje različnih izvajalcev in sistemov, preko katerih se le-ta izvaja. Zaradi lažjega razumevanja namena namestitvenega toka ter pomena ostalih komponent so v nadaljevanju s topološkega vidika opisani sistemi in komponente, ki sestavljajo celotno vpeljavo sprememb na produkcijsko okolje:

- **sistem za vodenje projektov Jira:** predstavlja centralni sistem za upravljanje sprememb. Tu se vodi načrtovanje, nadzor in izdajanje zahtev posameznikom za izvajanje aktivnosti, ki so povezane z razvojem, testiranjem in nameščanjem novih verzij aplikacij. Glede na pripravljene plane za nadaljnji razvoj aplikacij v sklopu posameznega mesečnega cikla se nove zahteve preko sistema Jira dodeli razvijalcem. Tu se vodijo vse spremembe (informacije razvijalcev, namestitve itd.), ki se jih naredi na vseh okoljih, vključno z vodenjem namestitvenega toka;

- **sistem za upravljanje z izvorno kodo SVN:** služi za združevanje razvitih programskih podsklopov s programskimi kodami, odloženimi na skupnem repozitoriju. Omogoča razreševanje najrazličnejših konfliktov pri združevanju medsebojne programske kode. Razvijalci tako dostopajo do repozitorija, kjer se nahajajo številne veje (variacije) razvite programske kode za posamezno aplikacijo in kjer lahko integrirajo svoje rešitve z obstoječimi verzijami na repozitoriju;
- **sistem za gradnjo programske kode Jenkins:** naloga sistema Jenkins je gradnja integriranih delov s sistema SVN v prevedeno novo verzijo aplikacije;

- **ftp strežnik:** je namenjen odlaganju vseh komponent (programski paketi, konfiguracije itd.) na enotno mesto, ki se bodo namestile v sklopu namestitvenega procesa;

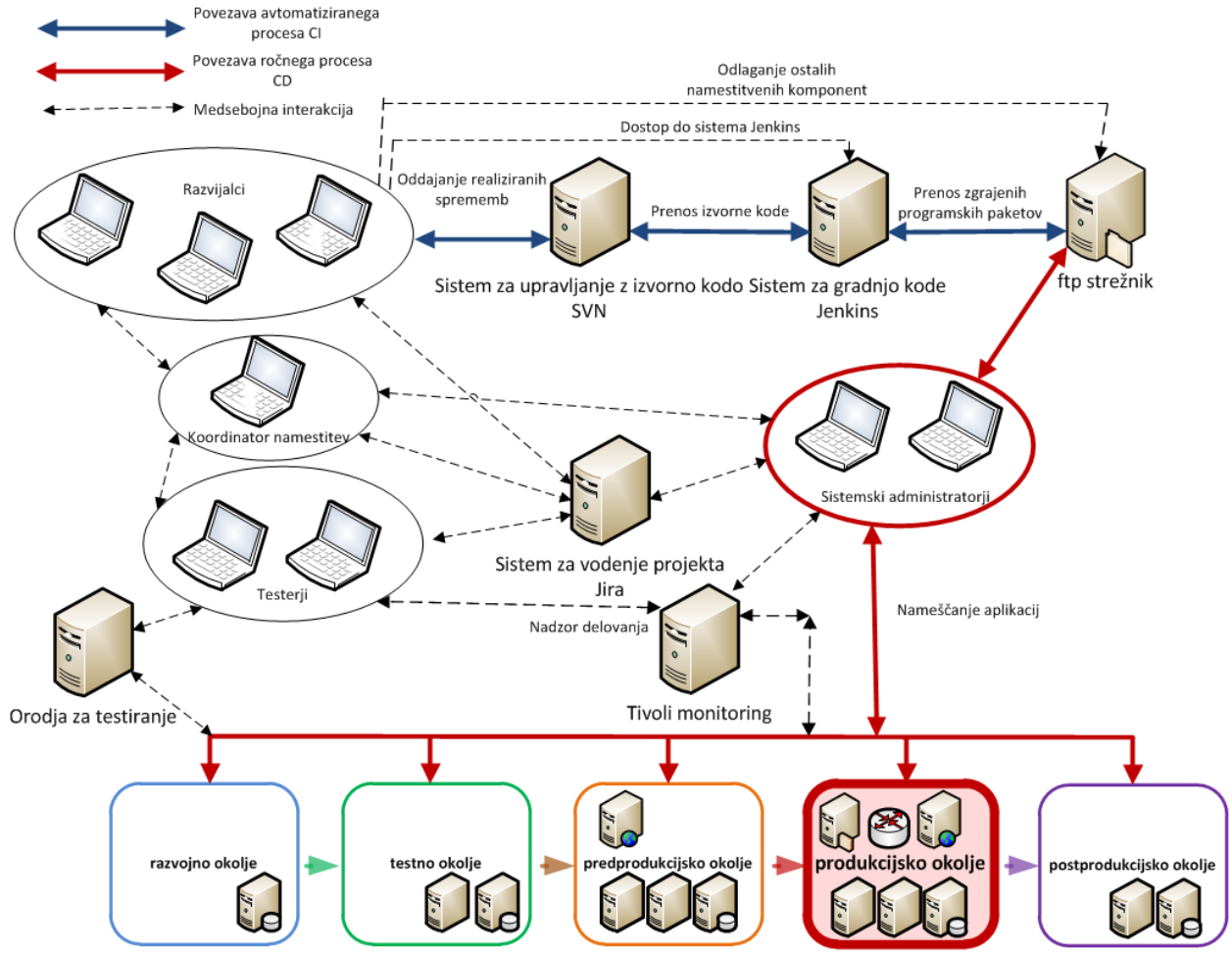
- **IBM Tivoli Monitoring:** je sistem za nadzor delovanja in odzivnosti aplikacij, strežnikov ter ostalih infrastrukturnih delov, ki so ključni za nemoteno delovanje storitev na posameznem okolju. Skupen portal omogoča vizualen pregled performančnega delovanja aplikacij ter obveščanje ustreznih oseb preko elektronske pošte in mobilnega obveščanja ob kritičnih dogodkih. Tako osebe kot primere dogodkov se predhodno definira;

- **orodja za testiranje:** lastne oz. kupljene rešitve (Sahi Pro, AutoHotKey, ProxySniffer) se uporabljajo v sklopu različnih vidikov testiranja primernosti novih verzij aplikacij. Preko različnih rešitev se izvajajo avtomatizirani in ročni testi sprejemljivosti, kot so testi zmogljivosti, funkcionalni testi, regresijski testi, uporabniški testi itd.;

- **okolja v namestitvenem procesu:** vsako okolje posebej predstavlja izolirane zaključene sklope, ki so sestavljeni iz številnih komponent (npr. proxy strežniki, spletni strežniki, razporejevalec bremen, gruča aplikacijski strežnikov, podatkovna baza itd.). Predstavljajo predpogoj za delovanje aplikacij, ki se na teh okoljih izvajajo. Ker so Java aplikacije večnivojske, lahko posamezna sprememba v aplikaciji vpliva na posege na številnih nivojih v sklopu namestitve. Vsako okolje ima svoj namen v sklopu namestitvenega procesa, ki je opisan v nadaljevanju.

Slika 19 s topološkega vidika predstavlja celotno infrastrukturo, kamor je vpet namestitveni tok. V namestitvenem toku sodelujejo tri skupine, in sicer razvojna skupina, skupina za testiranje in sistemski administratorji. Kot povezovalni člen med vsemi skupinami je v proces namestitev vključen tudi koordinator namestitev. Ta skrbi za razdeljevanje namestitvenih nalog in njihovo realizacijo, ki jih prejme s sistema Jira, ter za hitrejši in učinkovitejši pretok informacij med skupinami. Prisotnost koordinatorja pa ne izključuje neposrednega stika s skupinami. Izvajanje namestitvenega procesa poteka preko prikazanih sistemov, kjer je CI del realiziran avtomatično, preostali del CD pa se izvaja ročno. Pri izvajanju namestitvenega procesa omenjene skupine uporabljajo prikazane sisteme.

Slika 19: Infrastruktura v namestitvenem toku



Uporaba novih funkcionalnosti in sprememb na produkcijskem okolju zahteva izvedbo številnih korakov, ki se začnejo s poslovnimi idejami. Pobude in zahteve za nove aplikacijske spremembe prihajajo s strani uporabnikov, vsebinskih skrbnikov, zakonodaje, vodstva. Zaradi potreb po nadzoru in celostnem pregledu nad prilagajanji aplikacij v ta namen enkrat mesečno zaseda odbor za upravljanje sprememb. Odbor sestavljajo izbrani izvršni direktorji različnih služb, namen zasedanja pa je skupno odločanje glede nadaljnjih aktivnostih na področju sprememb funkcionalnosti aplikacij. Skupina potrjuje oz. zavrača predlagane aktivnosti, prav tako pa v seznam prioritet vključuje predloge s strani članov odbora. Nove predloge pripravita tudi vodja razvoja aplikacij in vsebinski skrbnik posamezne aplikacije. Za sprejete predloge je pred izvedbo treba preveriti tehnično zmožnost, nato se jih po potrditvi zapiše v sistem Jira.

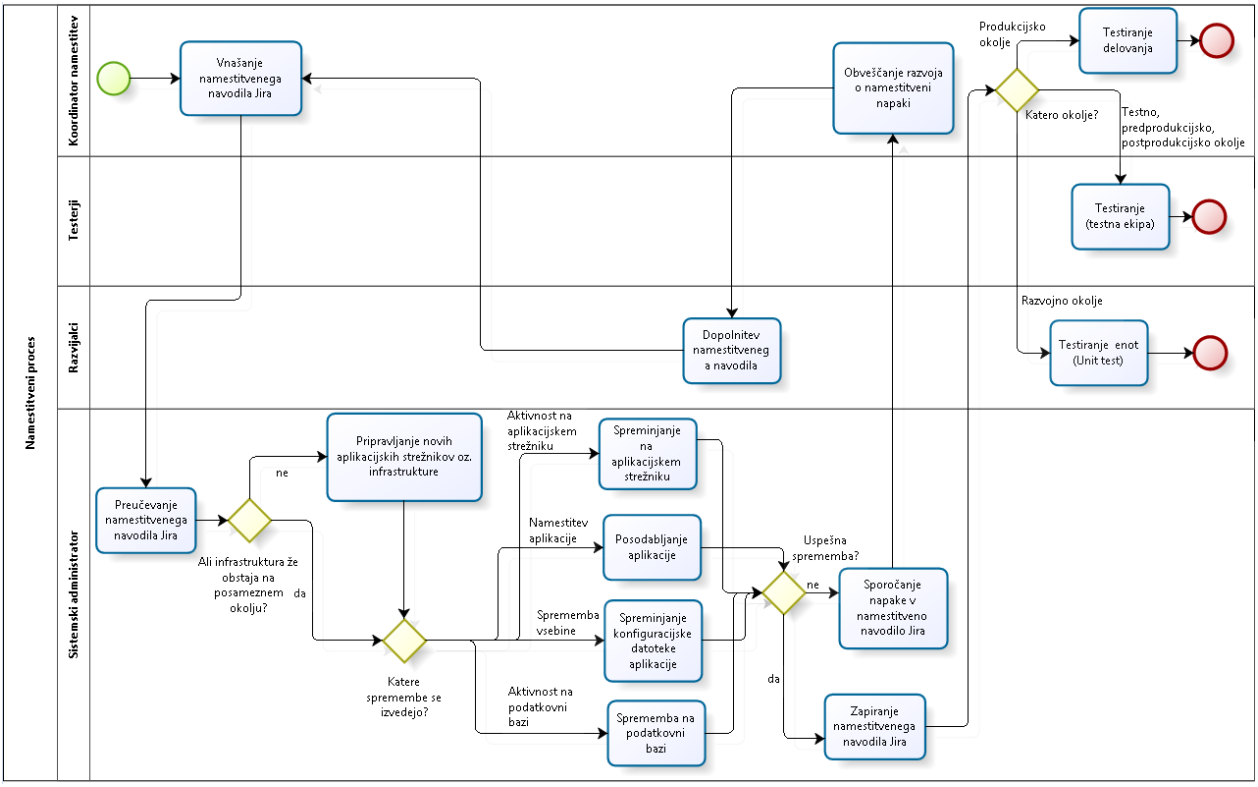
V sistemu Jira, ki je prikazan na Sliki 19, so dosegljive vse sprejete zahteve. Glavna stran v sistemu Jira vsebuje globalne spremembe z vsemi zahtevami, roki, historičnim prikazom glede stanja določene aktivnosti. Iz teh splošnih sprememb se probleme razdeli na podprobleme in se tekom razvojnega cikla ter izvajanja namestitvenega toka odpre številne podzahteve. V podjetju se tako ciklično, kjer vsak cikel traja 1 mesec, ponavlja enomesečni proces zajema zahtev razvoja novih funkcionalnosti, samega razvoja, nameščanja na vsa okolja, izvajanja testiranj ter končne izdaje uporabnikom v produkcijskem okolju.
Proces razvoja novih funkcionalnosti se tako začne z zahtevami v sistemu Jira. Razvijalci na podlagi prejetih nalog razvijajo funkcionalne spremembe, kjer razvite podsklope programske kode z lokalnega računalnika sproti združujejo z ostalimi sodelavci v združene verzije kode, dostopne vsem razvijalcem. Združevanje razvite kode posameznika z ostalo kodo, ki se dopolnjuje in skupaj predstavlja programsko kodo posamezne aplikacije, se izvaja preko sistema SVN (Slika 19). Ko je določen podproblem oz. del kode razvit, se preko sistema Jenkins zgradi nova verzija aplikacije. Vsaka zgrajena verzija predstavlja nov programski paket posamezne aplikacije. Ta se po uspešni izgradnji odloži na posebno mapo na datotečnem sistemu na ftp strežniku. Mapa s programskim paketom ima razumljivo enoznačno ime, programski paket je prav tako ustrezno verzioniran. Razvijalci v mapo z določeno verzijo aplikacije po potrebi prenesejo še morebitne konfiguracijske datoteke oz. ostale komponente aplikacij. Na datotečnem sistemu so tako dosegljive številne verzionirane aplikacije, ki so na voljo sistemskim administratorjem, ki aplikacije ustrezno implementirajo v sistem. Ko je pripravljena nova verzija aplikacije, koordinator namestitve napiše namestitveno navodilo Jira ter ga naslovi skupini sistemskih administratorjev.

Razvoj sledi načrtom namestitvenega toka in diktira hitrost ter frekventnost nameščanja. Celotna vpeljava sprememb, vključno z izvajanjem namestitvenega toka, zahteva neprestano komunikacijo med vsemi ter vodenje projekta. Vsaka namestitev ne glede na okolje ali fazo namestitvenega toka se izvede preko generičnega procesa, prikazanega na Sliki 20. Slika 20 tako prikazuje posamezno namestitev, ki je del namestitvenega toka in ga izvede skupina sistemskih administratorjev ob vsakem prejetem namestitvenem navodilu Jira. Sistemski administratorji preko pripetega navodila ter ustreznih datotek s ftp-ja, ki so definirane v prejetem zahtevku Jira, pričnejo s procesom namestitve. Ko sistemski administratorji prejmejo navodilo, preverijo, ali za izvedbo navodila morebiti še kaj manjka oz. če so prisotna kakršnakoli neskladja. Pri

namestitvah je zelo pomembno, ali gre za aplikacijo, ki se bo nameščala na obstoječo infrastrukturo, ali pa so potrebni večji infrastrukturni posegi.

Aplikacije so večnivojske in so sestavljene iz različnih komponent (programski paket številnih modulov .ear ali .war, aplikacija, konfiguracijske datoteke ali pa podatkovni nivo), ki se jih ustrezno implementira v sistem. Glede na prejeto navodilo s strani razvoja koordinator namestitev glede na vsebino navodila naloge izvedb razpošlje ustreznim osebam, ki so zadolžene za določene izvedbe (npr. skrbnik aplikacijskih strežnikov, skrbnik podatkovne baze itd.). Pri izvajanju namestitvenih aktivnosti na posameznih komponentah je zelo pomemben vrstni red, zato koordinator namestitev skrbno nadzira potek posamezne namestitve. Odvisna od samih sprememb je lahko posamezna namestitev zelo enostavna, lahko pa zajema številne spremembe, še posebej, če gre pri tem tudi za infrastrukturne spremembe. Ob vsaki uspešni namestitvi se zavede izvedbo posamezne namestitve v zahtevek Jira in obvesti koordinatorja namestitev o končani izvedbi. Ta preveri delovanje nameščene aplikacije in s tem se namestitev zaključi.

Slika 20: Proces nameščanja



V obravnavanem podjetju je cikel razvoja posameznih aplikacij prav tako odvisen od njihovega namena. Na grobo bi lahko posodabljanje aplikacij delili na dva dela, in sicer na transakcijske aplikacije, ki podpirajo poslovanje podjetja, ter ostale aplikacije (podporne, integracijske, spletne itd.) in storitve, ki se uporabljajo znotraj in zunaj podjetja. V prvem primeru govorimo o inkrementalnem ponavljanju namestitev 30-dnevnega cikla, v drugem primeru pa se namestitveni tok izvaja pogosteje.

Kompleksnost sistema se neprestano povečuje, število zaposlenih pa ostaja na približno enaki ravni. Zaradi nujnosti po dosegljivosti storitev se večina vzdrževalnih del izvaja izven delovnega časa. Tako se predvsem pri strokovnjakih, ki so že sedaj najbolj delovno obremenjeni, opaža konstantno povečevanje števila nadur. Iz tega lahko sklepamo, da so ključni izvajalci najverjetneje preobremenjeni, s čimer se povečujejo možnosti tudi za človeške napake (visoko tveganje za produkcijsko nedelovanje) in za slabšo kakovost opravljenega dela. Prav tako se zaradi človeškega faktorja prepočasi nameščajo nove verzije aplikacij preko testnih do končnih produkcijskih okolij. Ročno nameščanje je poleg opisanih predstavljalo še ostala tveganja:

- nepreglednost izvedenih aktivnosti za ostale, vpletene v namestitveni tok,
- zahtevno koordiniranje vseh namestitvenih nalog,
- nujnost po čim višji dosegljivosti sistemskih administratorjev,
- pogosto delo izven delovnika,
- preobremenjenost inženirjev,
- težaven pregled zgodovine vseh namestitev (preko mailov določenih oseb),
- plačevanje nadur,
- akumulacija specifičnega znanja, potrebnega za izvajanje namestitev pri posameznikih,
- pomanjkanje sistemskih administratorjev za izvajanje namestitev,
- ozka grla,
- intenzivnost nameščanja,
- slaba dokumentacija,
- izvajanje nalog višjih prioritet s strani sistemskih administratorjev, zato se je z izvedbo namestitev pogostokrat čakalo, s čimer se je podaljševal čas cikla.

Zato se pojavlja potreba po boljši preglednosti procesov, organizaciji in optimizaciji. Ker določene naloge v sklopu posodabljanj predstavljajo že vnaprej znana zaporedja aktivnosti, so se že v preteklosti interno razvijale delno avtomatične procedure za olajšanje dela, ki pa niso celostno reševale omenjenih potreb. Zato je bila v informatiki prisotna želja po vpeljavi dobrih praks ter orodij, ki bi bile zaposlenim v pomoč pri vsakdanjem delu. Na ta način bi se zaposlenega razbremenilo in mu omogočilo več časa za umsko zahtevnejše naloge.

3 ANALIZA STANJA PO UVEDBI NOVEGA ORODJA

3.1 IBM Urban Code Deploy

IBM UrbanCode Deploy je produkt proizvajalca IBM, ki omogoča izvajanje avtomatičnih namestitev kompozitnih aplikacij. Današnje aplikacije večinoma sestavljajo številne komponente, ki se lahko posamezno ali v celoti spreminjajo in vplivajo na delovanje aplikacije kot celote (Continuous Delivery, 2016). Namenska rešitev avtomatizira proces dostave (angl.

process of deliver) aplikacij preko vseh testnih okolij, do končne izdaje novih verzij končnim uporabnikom na produkcijskih okoljih (Jackson, 2013).

UCD vsem vpletenim v proces CD prinaša transparentnost, preglednost, nadzor, verzioniranje, možnosti potrjevanja namestitev, opolnomočenje (Barosa, 2015, str. 3). Preko definiranja komponent vseh nivojev kompozitne aplikacije (aplikacija, konfiguracije, podatkovne spremembe) ter modeliranja orkestriranega procesa namestitev se lahko ta uporabi pri namestitvah na različnih predpripravljenih okoljih. Nameščanje je možno preko enostavnih namestitev na ukaz tudi na daljavo oz. preko nastavitev urnika (angl. *schedule*). S pomočjo uporabe vtičnikov je možna integracija s številnimi sistemi (Jira, Jenkins, podatkovne baze, aplikacijski strežniki, izvajanje skriptnih ukazov itd.).

Kot je navedeno na uradni strani IBM, UCD predstavlja eno izmed vodilnih rešitev ARA na trgu, ki se dopolnjuje s praksami DevOps. IBM s pomočjo DevOps produktov omogoča podjetjem izvajanje stalne dostave programske opreme, kjer orodja predstavljajo infrastrukturo za hitro dostavljanje novih tržnih priložnosti, s čimer lahko podjetja izkoriščajo konkurenčno prednost pred drugimi. Kot zagotavljajo, naj bi uvedba produktov UrbanCode podjetju omogočala:

- nižanje stroškov: manj dela s strani zaposlenih, krajše zastoje v procesu stalne dostave, manj napak,
- hitrejše uvedbe sprememb preko povečanja frekventnosti namestitev,
- zmanjševanje tveganj: predpripravljeni procesi namestitev, avtomatično nameščanje, nadzor in sledljivost nad namestitvami.

IBM UrbanCode portfelj predstavlja naslednje produkte (UrbanCode Deploy, 2016; UrbanCode Deploy with Patterns, 2016; Continuous integration and build management, 2016; UrbanCode Release, 2016):

- **IBM UrbanCode Deploy** – orodje za avtomatizacijo namestitev na vseh okoljih,

- **IBM UrbanCode Deploy with Patterns** – razširitev UCD omogoča izdelavo vzorcev, kjer so poleg aplikacij vključene tudi ostale infrastrukturne komponente,

- **IBM UrbanCode Build** – poenostavlja upravljanje sistema za gradnjo programske kode pri dostavljanju novih verzij programskih paketov,

- **IBM UrbanCode Release** – omogoča boljši pregled in koordinacijo izvajanj kompleksnih namestitev številnih aplikacij.

Načeloma vrstni red gradnje avtomatizacije namestitev v UCD ni pomemben. Pomembno je le to, da ima na koncu celoten proces nameščanja definirane vse potrebne sestavne dele, ki so med seboj ustrezno povezani. V nadaljevanju bodo predstavljeni glavni sestavni deli in koncepti pri modeliranju aplikacij in gradnji procesov nameščanja:

- **agenti** (angl. *Agents*): nahajajo se na ciljnih sistemih, kamor želimo namestiti komponente aplikacij oz. izvajati oddaljene ukaze. UCD je tako preko agentov povezan z oddaljenimi strežniki. Agenti so pravzaprav podaljšana roka sistema UCD, kateri izvajajo naloge, ki se jih upravlja s centralne točke. Agenti morajo biti zato nameščeni na vseh virih, kjer bo UCD opravljal namestitvene naloge;

- **viri** (angl. *Resources*): posamezni viri predstavljajo skupek ciljnih strežnikov, na katerih so nameščenimi agenti, ki skupaj tvorijo pomenljivo skupino (npr. vsi strežniki produkcijskega okolja). Cilj grupiranja je poenostaviti in logično vključiti vse strežnike, na katerih se bodo izvedle namestitvene aktivnosti (namestitev programskih paketov, sprememba konfiguracije, sprememba na podatkovni bazi);

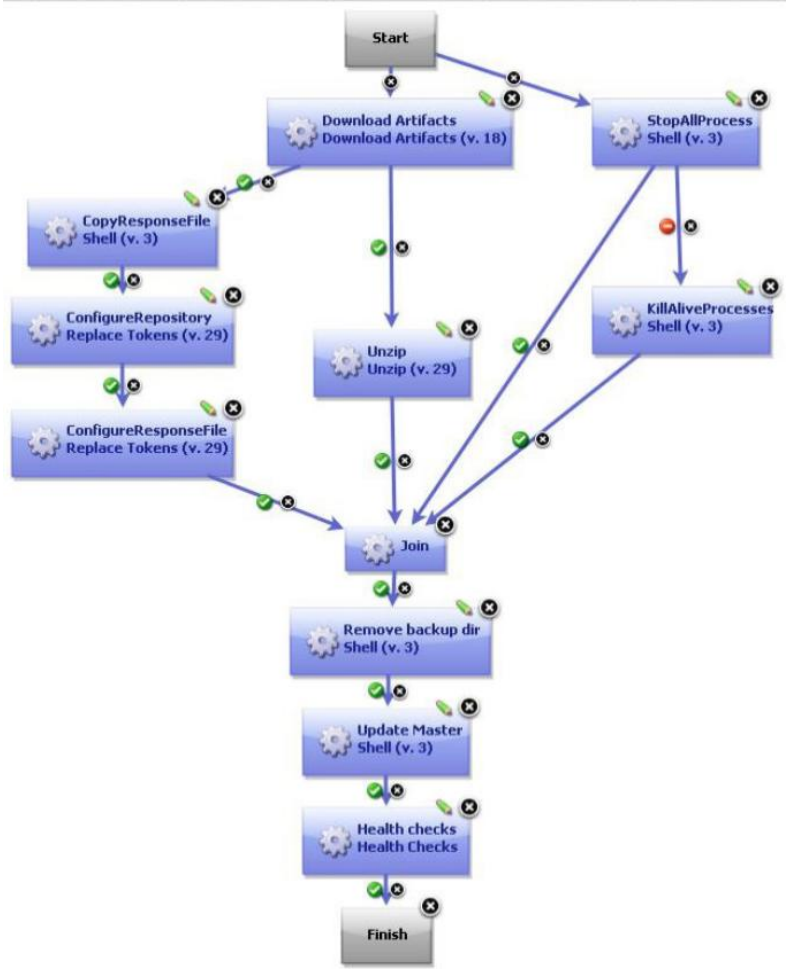
- **komponente** (angl. *Components*): so gradniki večnivojske aplikacije. Na centralnem namestitvenem sistemu se za vsako komponento fizično hranijo pripadajoče verzije oz. artefakti (angl. *Artifact*) (npr. programski paket spletne aplikacije, skripte SQL, fizične datoteke itd.). Artefakte se lahko uvaža iz različnih virov. Največkrat predstavljajo izvore artefaktov najrazličnejši sistemi za hrambo in verzioniranje izvorne kode (angl. *Subversion repository*), npr. SVN, GIT itd., datotečni sistemi in najpogosteje sistemi za gradnjo aplikacij (angl. *Continuous integration server*);

- **posnetki** (angl. *Snapshots*): predstavljajo funkcionalnost sistema UCD, ki omogoča shranjevanje kombinacij verzij artefaktov posameznih komponent, ki se skupaj namestijo v sklopu namestitve. Kombinacije različnih verzij artefaktov ob namestitvi kompozitne aplikacije predstavljajo določeno verzijo posnetka. Prednost posnetkov je v tem, da s tem poenostavimo namestitev, saj ob velikem številu komponent, ki bi se morale namestiti preko vseh okolij, skupaj definiramo kot del krovne verzije posnetka;

- **procesi** (angl. *Processes*): predstavljajo zaporedje predvidenih korakov, ki se izvedejo po vnaprej predvideni proceduri. UCD omogoča več vrst procesov (Processes, 2016):

    - **generični procesi –** so neodvisni od aplikacij in komponent. Aktivnosti procesa se izvedejo preko nameščenih agentov na oddaljenih strežnikih;
    - **procesi komponent** – predstavljajo zaporedje aktivnosti, ki se izvedejo pri nameščanju artefaktov določene komponente. Komponenta ima lahko definiranega enega ali več procesov, ki se jih glede na izbiro lahko uporabi pri nameščanju. Procese se gradi v posebnem urejevalniku, kjer se, kot je razvidno s Slike 21, dodajajo poljubne aktivnosti;

Slika 21: Primer procesa nameščanja komponente v UCD[1]



Vir: I. Gorga, F. Barillari &M. Andreuzzi, Implement continuous delivery using IBM UrbanCode Deploy, 2014.

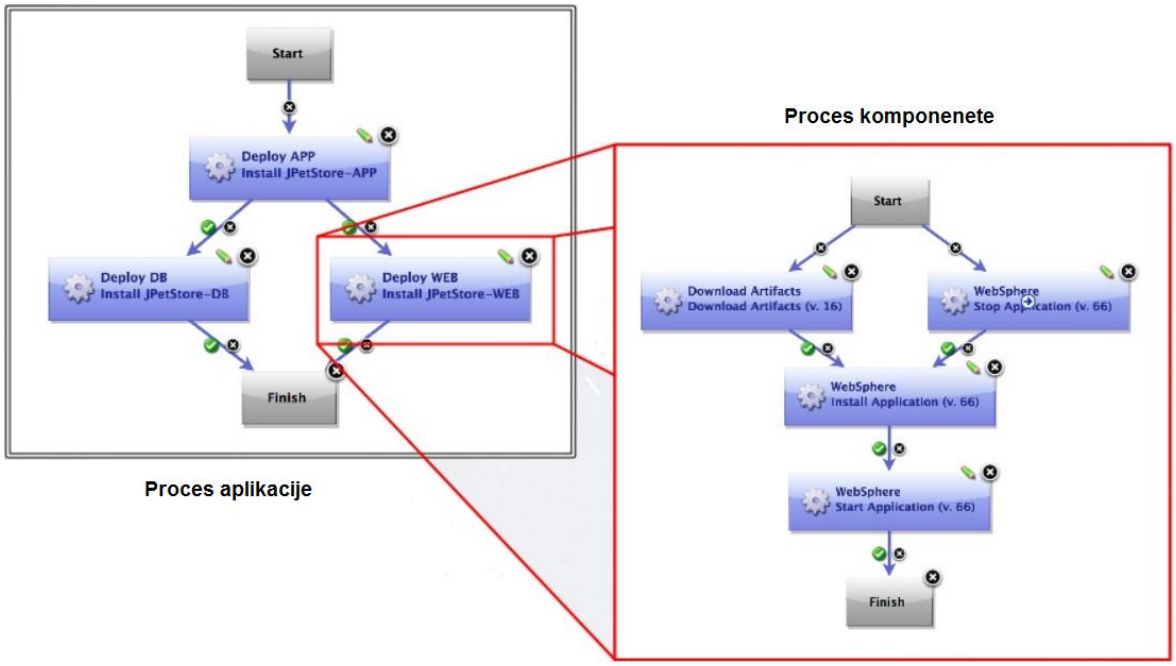
V urejevalniku so upravljalcu procesa na voljo različni moduli, ki omogočajo številne funkcionalnosti (prenos artefaktov, upravljanje aplikacijskih strežnikov, ukazna lupina itd.). Moduli so dosegljivi preko različnih vtičnikov (angl. *plugins*), ki so na voljo na uradni spletni strani produkta in jih je mogoče dodatno uvažati v orodje UCD. Vtičnike lahko razvijamo tudi sami ter jih vključimo v UCD, s čimer so v pomoč pri izvajanju namestitvenih aktivnosti. Aktivnosti znotraj procesov se lahko izvajajo zaporedno oz. z možnostjo vejitve vzporedno, hkrati pa je mogoče celoten potek aktivnosti kontrolirano izvajati preko uporabe najrazličnejših pogojev;

- **procesi aplikacije** – se nahajajo nivo višje od procesov komponent. Na tem nivoju se smiselno združuje in orkestrira izvajanje komponentnih procesov. V vseh namestitvenih aktivnostih, ki so del namestitvenega toka, se za posamezno aplikacijo izvede isti že vnaprej definiran aplikacijski proces v UCD. Na ta način se zagotovi konsistentnost pri prehajanju aplikacij preko številnih okolij v namestitvenem toku. Slika 22 prikazuje povezavo med procesom aplikacije in

[1] Slika 21 je izdelana v UCD in njen namen je prikazati način gradnje posameznega procesa.

procesom komponent, kjer vidimo, da je proces komponent pravzaprav podmnožica aktivnosti, ki se izvedejo v celotnem procesu aplikacije.

Slika 22: Povezava med procesom aplikacije in procesom komponent[2]



Vir: M. Wagner, Product Overview: The New IBM UrbanCode Deploy 6.0., 2013.

- **aplikacije** (angl. *Applications*): definirana aplikacija predstavlja ciljno stanje, ki mora za izvedbo namestitev vsebovati vse ustrezne sestavne dele. Tako mora aplikacija vključevati: komponente, procese, okolja ter druge nastavitve, ki skupaj avtomatizirajo proces nameščanj posameznih aplikacij;

- **okolja** (angl. *Environments*): predstavljajo zaključen sklop fizičnih ciljev (aplikacijski strežniki, podatkovne baze itd.), ki sestavljajo določeno okolje (testno, produkcijsko itd.). Vsaka aplikacija ima definirano eno ali več okolij. S tem jasno določimo medsebojno izolirana okolja, kamor se bodo namestile aplikacije.

3.2 Analiza prenovljenih procesov

Namestitveni tok se je vse od svoje vzpostavitve naprej inkrementalno postopno izboljševal. Avtomatizacija procesa CI je predstavljala prvi korak v smeri učinkovitega in preglednega načina dostavljanja razvitih sprememb internih aplikacij na produkcijsko okolje. Povezano avtomatizirano zaporedje aktivnosti CI združuje naslednja 2 koraka:

- združevanje različic kode (SVN),
- integracija izvorne kode, izvajanje integracijskih testov, gradnja aplikacij z integrirane izvorne kode (Jenkins).

[2] Slika 22 je izdelana v UCD in njen namen je prikazati povezavo med procesom komponent in procesom aplikacije.

Preostali del namestitvenega procesa pa je predstavljal nameščanje in testiranje nameščenih verzij aplikacij preko različnih okolij (razvojno, testno, predprodukcijsko okolje). Nameščanja so se izvajala pretežno ročno, kar je običajno predstavljalo časovno potratno aktivnost. Slabosti ročnega nameščanja so opisana v poglavju 2.1. Kasneje pa se jih je omililo z razvojem namestitvenih skript. Te so v proces prinesle nekoliko večjo transparentnost, saj se je po vsaki namestitvi ustreznim prejemnikom razposlala elektronska pošta z izvajalcem in vsebino namestitve. Do neke mere je bilo možno izvajati namestitve avtomatično preko nastavljenega urnika, a vendar je tudi ta rešitev imela nekatere slabosti. Za vsako nameščanje je bilo treba ročno prilagajati skripte, kar je pomenilo časovno še daljše izvajanje namestitev. Za pripravo celotne namestitvene infrastrukture so bili zadolženi člani sistemskih administratorjev. Poleg tega skripte niso zajemale celotnega procesa vseh komponent, ampak so reševale samo del namestitve, ki je vezan na aplikacijske strežnike in datotečne strežnike. Izvajanja na podatkovni bazi, ki jih je frekventno sicer veliko manj, so bila realizirana s strani skrbnikov podatkovnih baz. Proces je vključeval koordinatorja namestitev, ki je skrbel za usklajeno nameščanje in izvedbo celotnega namestitvenega navodila preko sistema Jira.

V namestitveni tok je bilo vključenih okoli 100 različnih aplikacij, ki so bile nameščene na vseh okoljih. Namestitev je največkrat predstavljala posodobitev programskega paketa oz. spreminjanje konfiguracijskih datotek na aplikacijskih strežnikih. Tako sta bila največkrat obremenjena dva člana sistemskih administratorjev, ki sta skrbela za aplikacijske strežnike. Ob namestitveno frekventnih dnevih sta bila oba sistemska administratorja zelo obremenjena, še posebno ob odsotnosti katerega od njiju. Namestitvene naloge so se pokazale kot zelo izčrpne naloge, saj jih je bilo mogoče velikokrat opravljati skozi celoten dan oz. odvisno od potreb razvoja in izdaj. Preobremenjenost se je izkazovala z zastajanjem dela na ostalih področjih rednega dela in različnih projektih.

Določen delež namestitev je predstavljal tehnično in koordinacijsko zahtevnejše aktivnosti, kjer je moralo biti v namestitveni proces vključenih tudi več različnih strokovnjakov. Ravno zaradi potrebe po poznavanju vrste specifik infrastrukturnega okolja ter številnih pooblastil do ključne infrastrukture (npr. produkcija) za izvedbo namestitvenih nalog teh nalog v tistem trenutku ni bilo možno razporediti na ostale zaposlene (npr. skupina operacij). Kakršnakoli napaka v sklopu namestitvenega procesa je predstavljala nepravilno delovanje oz. nedelovanje aplikacij.

Nameščanja na testna okolja so potekala med delovnim časom, na produkcijska okolja pa v poznih večernih urah, tudi med vikendi. Povprečno število namestitev na mesec v letu 2014 preko ročnih namestitev je bilo 396. Pogosti cikli izdaj novih funkcionalnosti (enomesečni cikel) in intenzivnost dnevnih popravkov aplikacij na najrazličnejša okolja sta pokazala neprimernost obstoječih rešitev. Poleg tega pa sta za izvajalce namestitvenega procesa predstavljala velike obremenitve, s čimer se je posredno ogrožalo tudi delovanje samega produkcijskega okolja. Na tako odgovornem mestu, ki sistemskemu administratorju omogoča izvajanje česarkoli na strežnikih, lahko ob preobremenjenosti prihaja do napak, ki se odražajo v nedelovanju poslovanja, kar ima za podjetje hude finančne posledice. Zaradi opisanih razlogov se je konec

leta 2014 začelo iskati primerne rešitve za optimizacijo namestitvenega toka in zmanjševanje tveganj nasploh (same namestitve, razbremenitev sodelavcev, fokusiranje na ključne naloge itd.).

Z uvedbo nove rešitve smo želeli pravzaprav odpraviti prej opisane težave, hkrati pa omogočiti prepustnost nameščanj, ki jo narekuje razvoj in poslovni del. Vsekakor je bilo vodilo pri iskanju novih rešitev tudi finančna sprejemljivost. Ena od možnosti je bil lasten razvoj orodja, ki bi omogočalo nameščanje aplikacij povsem prilagojeno glede na potrebe podjetja, saj uporabljene skripte to niso omogočale. Ideja pravzaprav ni imela kaj dosti možnosti za uspeh, saj bi bil lasten razvoj finančno nesprejemljiv.

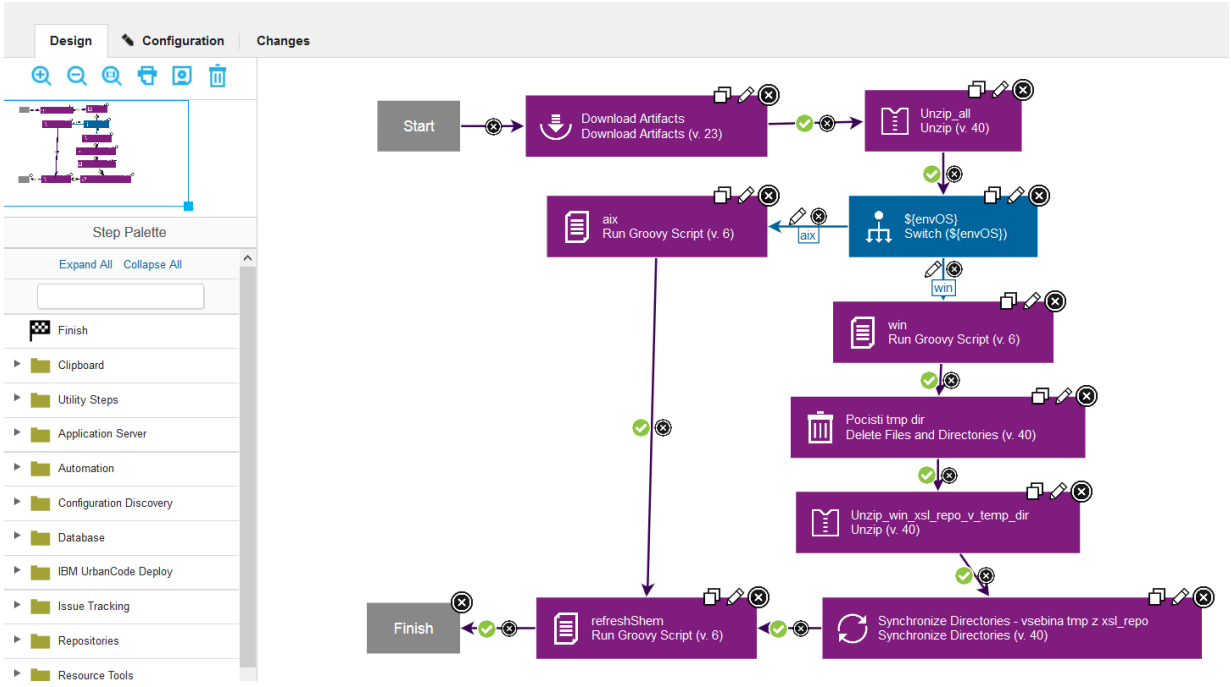
Druga možnost je bila nakup orodja ARA, ki naj bi ustrezal različnim kriterijem. S pregledom trga ponudnikov ARA je bilo rešitev kar nekaj, vendar so le redke ustrezale internim specifikam nameščanja. Želja je bila namestitveni tok čim bolje medsebojno povezati. Naša zahteva je bila tudi procesno avtomatizirati številne aktivnosti, ki se izvajajo v sklopu ene namestitve.

Podjetje se je odločilo za IBM UCD, kar je bil tudi rezultat metode točkovanja (angl. *scoring method*). V poglavju 1.3 so podrobneje predstavljeni kriteriji in rezultati same metode. Ponudnik nam je omogočil demonstracijo produkta UCD in kasneje njegovo testno implementacijo. Testiranje delovanja za potrditev koncepta (angl. *proof of concept*) smo opravili tako, da smo izvedli poskusne namestitve, s tem smo namreč potrdili izvedljivost avtomatiziranih namestitev. Tako smo aprila 2015 začeli postopoma izvajati avtomatične namestitve v omejenem obsegu z novo postavljenim sistemom za avtomatično nameščanje.

Vsaka aplikacija ima lahko glede na svojo specifiko nameščanja definiran svoj proces nameščanja, ki ga lahko uporablja poljubno število aplikacij. Končno število definiranih procesov namestitev tako predstavljajo specifike nameščanj posameznih aplikacij, ki jih je treba pri modeliranju posameznih procesov ustrezno upoštevati. Pri modeliranju procesa nameščanja smo poskušali nove procese čim bolje optimizirati in jih narediti robustne.

Na Sliki 23 je tako prikazan definiran proces za eno od internih aplikacij. Kot se vidi s Slike 23, se lahko proces veji in poljubno usmerja. Poleg tega pa omogoča UCD vključitev številnih funkcionalnosti različnih vtičnikov, ki so dostopni na uradni spletni strani produkta. Funkcionalnosti vtičnikov inženirju poenostavljajo gradnjo procesov, saj ti omogočijo številne možnosti, kot so: izvajanje različnih skriptnih jezikov, najrazličnejši ukazi in upravljanje z aplikacijskimi strežniki, upravljanje podatkovnih baz, povezovanje s sistemom Jira.

Slika 23: Proces aplikacije[3]

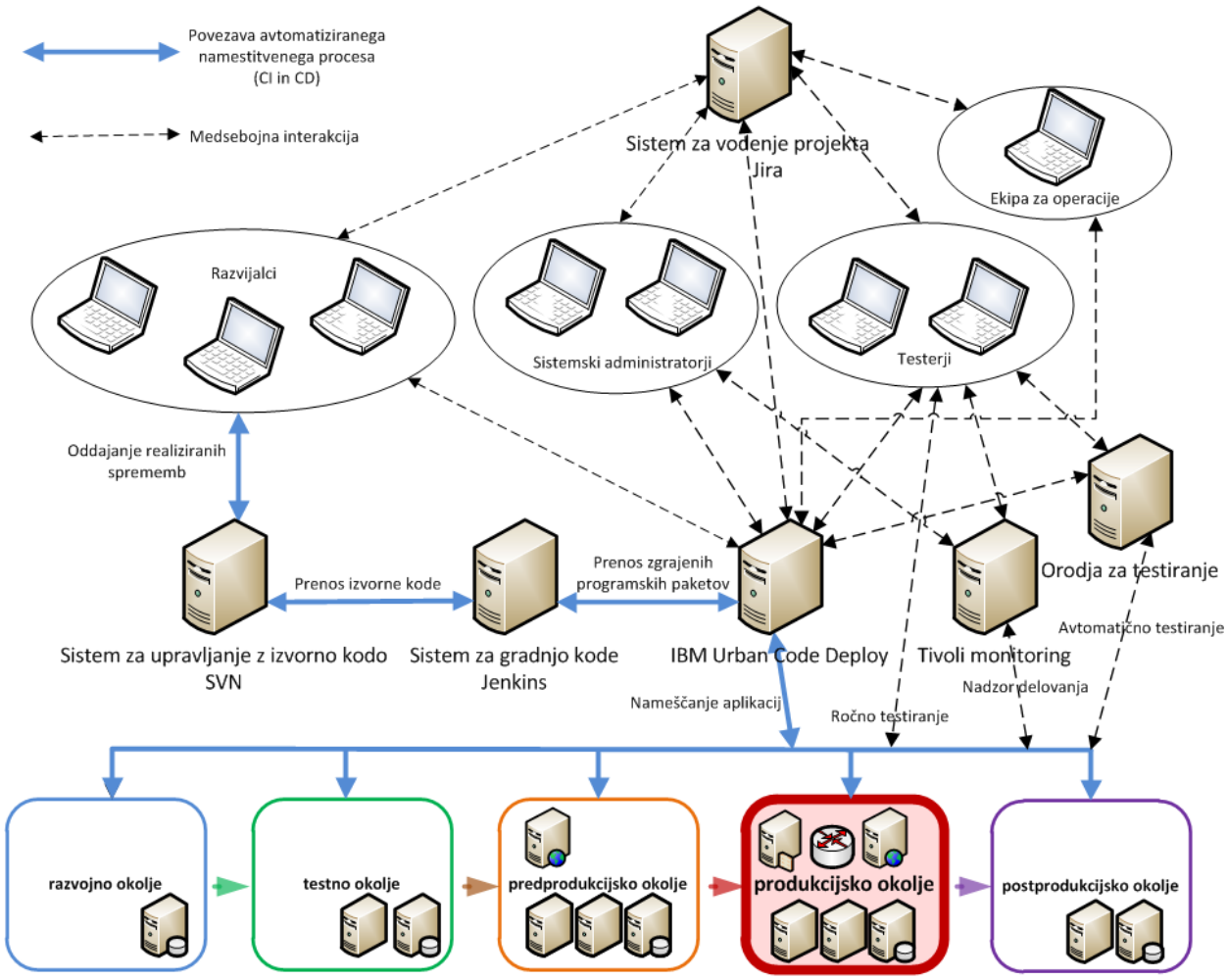


Slika 24 prikazuje končno stanje avtomatiziranega namestitvenega toka. UCD predstavlja osrednjo točko, kamor lahko dostopa in izvaja namestitve praktično vsakdo, ki se ga dodeli v skupino z določenimi pravicami. Zgrajen sistem uporabnikom na samopostrežen način ponuja enostavne namestitve. Ti ne potrebujejo predhodnih znanj, saj so vse procedure in nastavitve sistema uporabnikov zakrite. Administracijo sistema upravljajo sistemski administratorji, ki skrbijo za implementacijo novih aplikacij, okolij in prilagajanje avtomatiziranih namestitev. Da so namestitve na pomembnih okoljih nadzorovane, smo za predprodukcijsko in produkcijsko okolje uvedli potrjevanje namestitev. To pomeni, da lahko uporabniki izberejo namestitev, izvedbo pa potrdijo odgovorne osebe, ki skrbijo za pravilen potek izvajanja namestitvenega toka. Preko samopostrežnega nameščanja in namestitev po urniku vpletenim v namestitveni tok omogočajo neodvisnost in vnaprejšnje definiranje poteka namestitev, brez potrebnih prisotnosti ob samih namestitvah.

Z implementacijo orodja UCD smo dosegli večji obseg avtomatizacije namestitvenega toka CD. Nov proces predvideva vključitev skupine operacij, ki jih bodo izvajali za operativne naloge nameščanja. Dolgoročno se tudi ne predvideva koordinatorja namestitev, saj bodo ustrezno orkestrirane aktivnosti nameščanja definirane znotraj procesov orodja UCD.

[3] Slika 23 prikazuje urejevalnik za gradnjo procesov v UCD.

Slika 24: Infrastruktura za avtomatizacijo namestitvenega toka



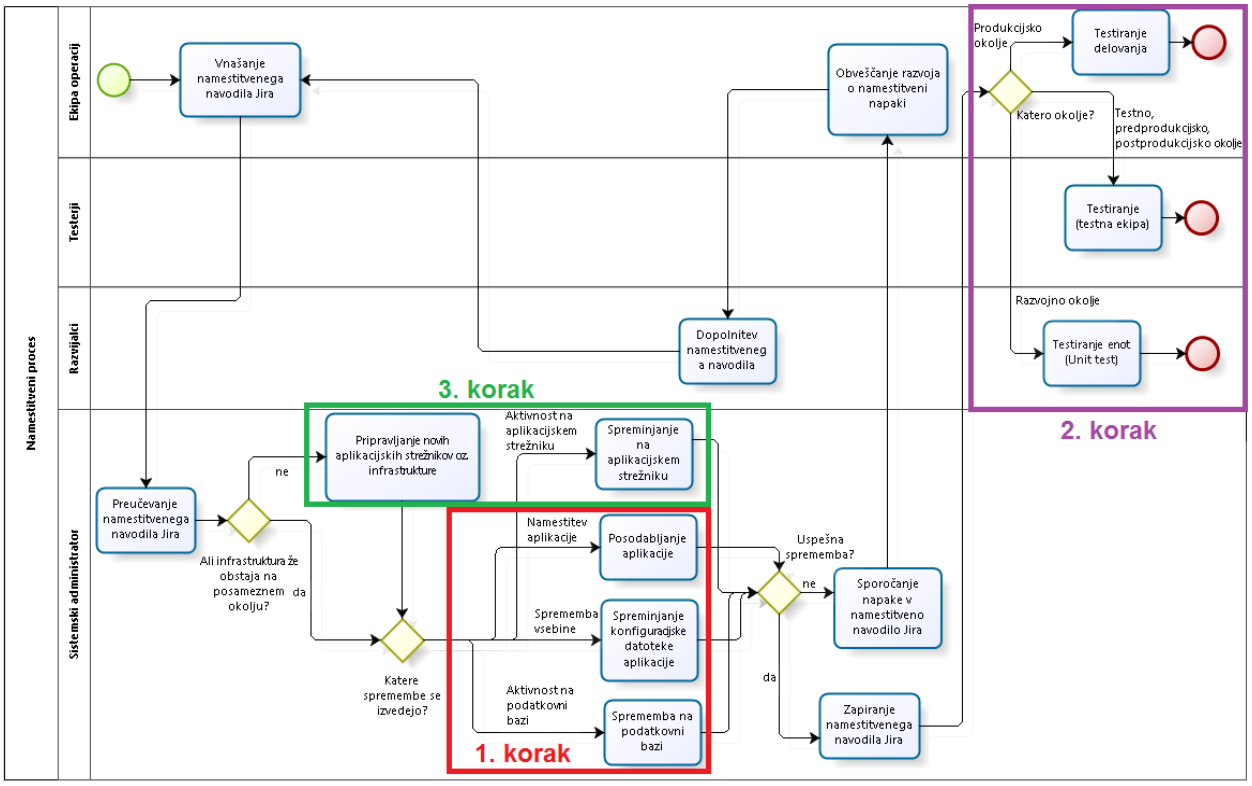
Do sedaj se je realiziralo nameščanje aplikacijskih paketov na aplikacijske strežnike in izvajanje sprememb na datotečnih sistemih, kar pa ni naše ciljno stanje. Uporabna vrednost samega orodja UCD se bo povečala ob vključitvi aktivnosti orkestrirane namestitve na vseh nivojih (večnivojske aplikacije). V našem primeru moramo v že zmodelirane namestitve aplikacij vključiti nameščanje konfiguracijskih datotek ter spreminjanje vsebine podatkovne baze. S tem se bo zaključil prvi korak implementacije, ki je označen z rdečo barvo na Sliki 25. Ker proces nameščanja omogoča samopostrežen pristop, lahko namestitev izvede kdorkoli z ustreznimi pravicami (razvijalci, testerji, sistemski administratorji, ekipa operacij). Slika 25 prikazuje primer procesa nameščanja aplikacije, ki jo izvede sistemski administrator.

Ker se je do sedaj implementacija UCD na splošno izkazala za uspešen projekt, imamo za nadaljevanje avtomatizacije široko podporo vodstvenih kadrov. Nadaljnji korak izvedbe avtomatičnega izvajanja (drugi korak) je vključiti klice testiranja delovanja aplikacij (angl. *smoke test*) po vsaki izvedeni namestitvi. S tem se bomo znebili ročnega preverjanja delovanja po vsaki namestitvi, zaupanje uspešnosti dostave temelji predvsem na produkcijskem okolju. UCD omogoča povezovanje tudi z nekaterimi orodji za testiranje različnih vidikov primernosti

novih verzij aplikacij. Z uspešno integracijo teh orodij v namestitveni tok bi dodatno avtomatizirali celoten proces dostave in s tem izboljšali učinkovitost razvoja.

Dolgoročen korak 3, označen na Sliki 25, pa je uvesti orodja za avtomatizacijo sistemov oz. infrastrukture. Številni vtičniki v UCD ponujajo integracijo z različnimi rešitvami CMT. Celostni pristop orkestriranega procesa razvoja IS, ki bi zajemal tako aplikacijski kot tudi infrastrukturni del, predstavlja želeno stanje avtomatizacije opazovanih procesov, s čimer bi preko podobnih pristopov in že uveljavljenih praks nameščanja aplikacij razširili sposobnost upravljanja tudi infrastrukturnega dela IS. To bo podjetju prinašalo zmožnost hitrega, učinkovitega in transparentnega prilagajanja ne samo aplikacij, ampak tudi zmožnosti uporabe konceptov modularnosti, elastičnosti in sposobnosti učinkovitega prilagajanja infrastrukture v IS.

Slika 25: Cilji izboljšav procesa nameščanja



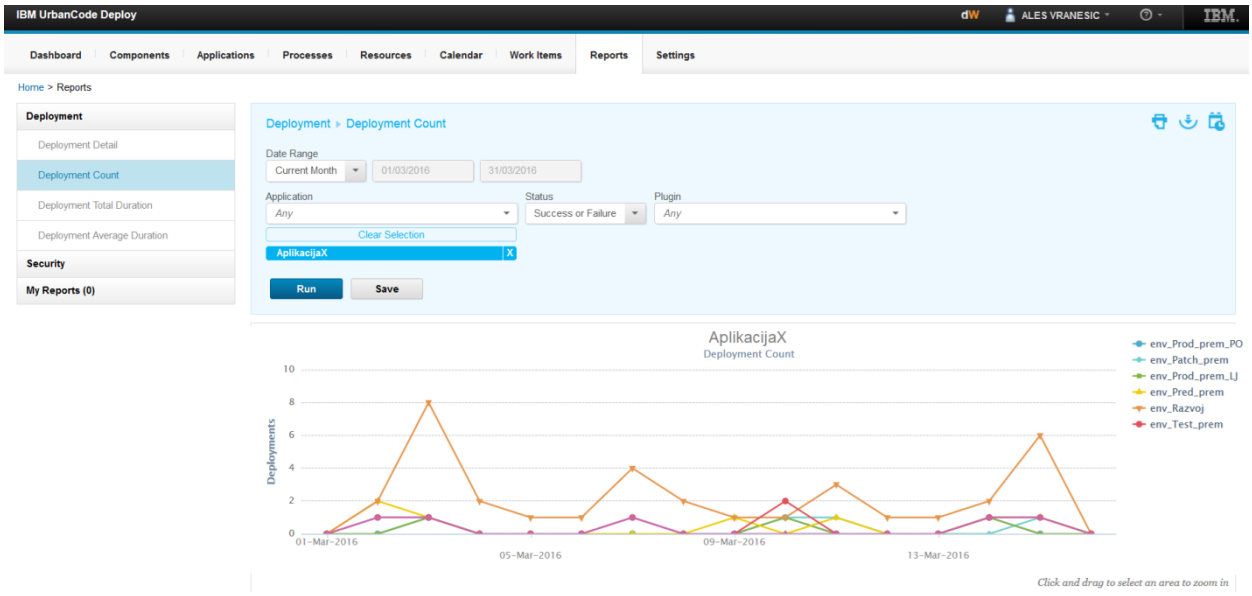
3.3 Analiza zgodovinskih podatkov namestitev

Historični podatki ročnih namestitev so zabeleženi le v prejetih sporočilih v elektronski pošti sistemskih administratorjev. Zato je bilo zbiranje teh podatkov časovno potratno, a vendar nujno, da lahko za obdobje od leta 2014 do februarja 2016 slikovno prikažemo frekventnost namestitev po posameznih okoljih. Zbrani podatki so tudi ključni za nadaljnji izračun upravičenosti vpeljave orodja UCD.

Od aprila 2015 dalje, ko je že bil uveden UCD, smo podatke enostavno pridobili preko uporabniškega vmesnika oz. izvoza v datoteke Excel, kar je tudi ena izmed prednosti UCD. Za

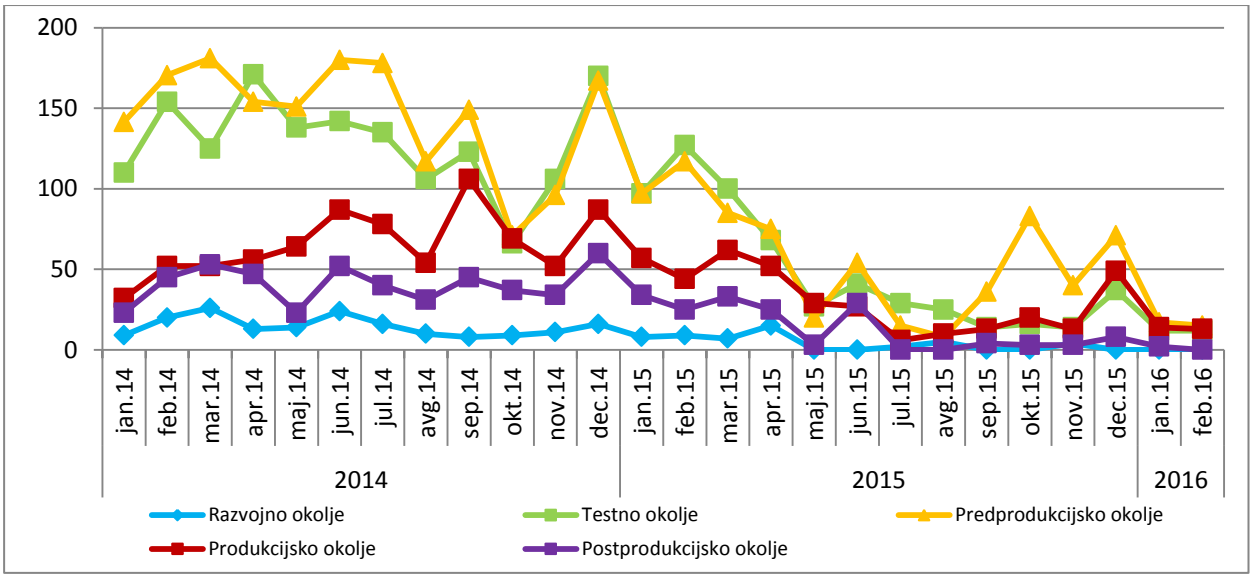
pripravo poročil lahko v vmesniku izberemo poljubno število aplikacij, prav tako pa se vidi revizijska sled vsake namestitve. Na Sliki 26 je prikazan spletni vmesnik, kjer je grafično prikazana pogostost namestitev za posamezno aplikacijo na vseh okoljih, kjer se ta izvaja.

Slika 26: Spletni vmesnik UCD



Slika 27 prikazuje število izvedenih ročnih namestitev po mesecih in okoljih od leta 2014. V povprečju je bilo v letu 2014 izvedenih 396 ročnih namestitev na mesec. Aprila 2015 so se namestitve začele izvajati preko UCD, kjer se na Sliki 27 opazi konkreten upad števila namestitev. Aktivnosti ročnega nameščanja po aprilu 2015 so bile odraz aplikacij, ki zaradi svoje specifike oz. ostalih razlogov še niso bile vključene v orodje UCD.

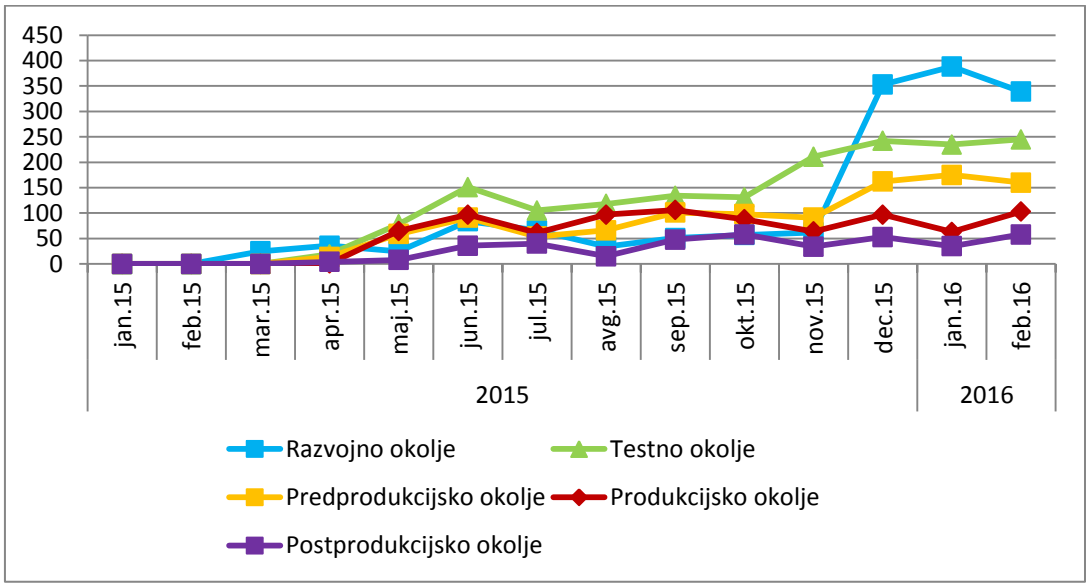
Slika 27: Mesečni prikaz števila izvedenih ročnih namestitev po okoljih



Slika 28 prikazuje pogostost namestitev preko UCD. Opazi se obraten trend naraščanja števila namestitev. Te so se začele iz meseca v mesec povečevati na vseh okoljih. Najbolj izstopata

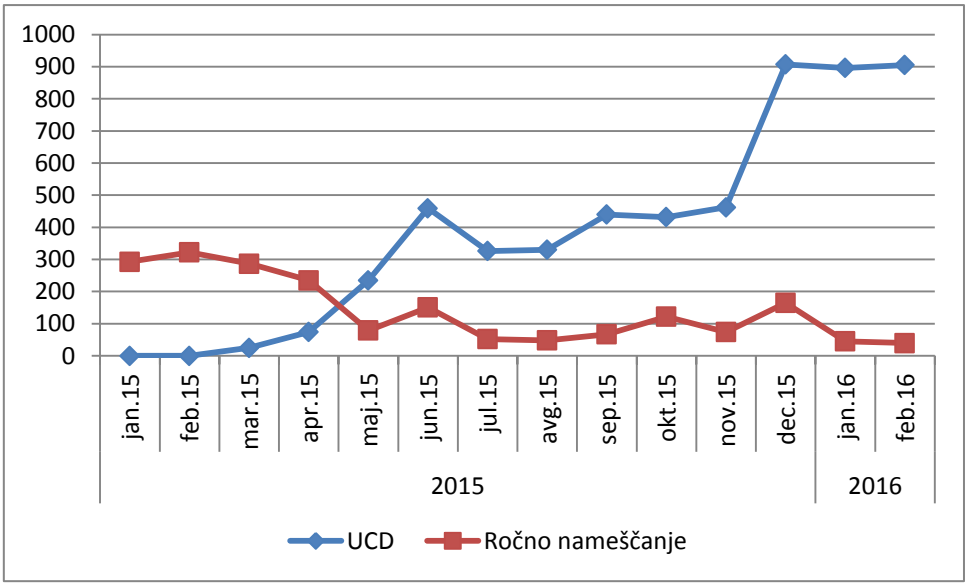
število namestitev na testna in razvojna okolja. V začetku decembra 2015 smo uvedli avtomatično nameščanje vseh najnovejših verzij aplikacij iz repozitorija, ki se pozno popoldne namestijo na razvojno okolje. S tem se je prihrani čas in to pomeni tudi boljšo sliko razvojnega okolja, ker imajo razvijalci vsak dan na voljo operativno in posodobljeno razvojno okolje, pripravljeno za testiranje aplikacij. Število namestitev na testna okolja je pokazalo, da so bile dejanske potrebe nameščanja večje. Zaradi prevelike birokracije, pisanja namestitvenih navodil, čakanja na izvedbo vseh aktivnosti, obremenjevanja ostalih sodelavcev so zahteve za nameščanje prihajale redkeje, kot pa je bila dejanska potreba, ki se je izkazala preko izvajanja v samopostrežnem načinu.

Slika 28: Mesečni prikaz pogostosti namestitev preko UCD



Slika 29 prikazuje časovno primerjavo števila ročnih namestitev in avtomatičnih namestitev. V grafu je jasno prikazana točka preloma med aprilom in majem 2015. Z začetkom leta 2016 se število ročnih namestitev skoraj približa 0. Konec decembra 2015 se je v avtomatične namestitve implementiral dodaten sklop aplikacij, ki so vplivale na povečano število namestitev. S tem smo vključili preko 80 aplikacij, ki so implementirane v sistem UCD z vsemi okolji in pripadajočimi procesi. Pričakujemo lahko, da bodo zaposleni s postopnim dojemanjem uporabnosti frekventnih in enostavnosti hitrih namestitev brez kakršnegakoli spodbujanja samoiniciativno povečevali frekventnost nameščanja. Vse to bo imelo vpliv na povečevanje skupnega mesečnega števila namestitev ter na hitrejšo povrnitev naložbe.

Slika 29: Primerjava ročnih in avtomatiziranih namestitev



Do marca 2016 se je v orodje UCD po postopnem vključevanju vključilo okoli 70 % vseh aplikacij. Pri tem vključene aplikacije v avtomatični proces nameščanja predstavljajo večinski delež.

Vse aplikacije niso bile vključene predvsem zaradi pragmatičnega pristopa vključevanja najbolj frekventnih in pomembnih aplikacij, s čimer smo z vloženim časom za modeliranje dosegli čim večjo pokritost avtomatičnih namestitev. Ostali razlogi se skrivajo tudi v specifiki nameščanj teh aplikacij, kjer ni bilo mogoče uporabiti že modelirane procese v orodju UCD.

Ker namestitve trenutno zajemajo le nameščanje aplikacijskega paketa in ne tudi ostalih komponent (konfiguracijska datoteka in izvajanje skript SQL), bomo v bližnji prihodnosti morali implementirati tudi te sklope v sklopu posamezne namestitve. Kljub temu da namestitve aplikacij v večini primerov zajemajo le namestitev aplikacijskega paketa (.ear oz. .war), pa bomo z vključitvijo ostalih komponent dosegli bistveno manj koordinacije in še boljšo izrabo sistema UCD. S tem se bodo tudi pohitrila kompleksnejša orkestrirana nameščanja, kar bo pomenilo hitrejšo dostavo novih aplikacij končnim uporabnikom in zmanjšanje časa cikla. Poudariti je treba, da bodo morali biti vsi artefakti (aplikacijski paket, konfiguracijska datoteka, skripte SQL itd.), ki se bodo uporabljali v sklopu namestitev zaradi transparentnosti, tudi ustrezno verzionirani.

V sodelovanju z razvojem aplikacij bo UCD omogočal enostavno in hitro manipuliranje s konfiguracijskimi in ostalimi spremembami na različnih nivojih, s čimer se bo omogočilo tudi nove možnosti pri samem dostavljanju novih verzij aplikacij. Preko vključevanja zastavic, ki predstavljajo vrednosti spremenljivke v konfiguracijskih datotekah, bomo lahko učinkovito vklapljali oz. izklapljali določene funkcionalnosti aplikacij. S tem bomo lahko še učinkoviteje in nadzorovano krmilili samo dostavo novih funkcionalnosti končnim uporabnikom. Aplikacije bodo lahko nameščene na produkcijo veliko prej, preden bodo dostavljene določene

funkcionalnosti preko vklopa zastavic. Na ta način se bo lahko reševalo celo tveganje pri dostavljanju sprememb, saj bomo preko učinkovitega upravljanja s konfiguracijskimi datotekami lahko izdajali spremembe postopno, v omejenem obsegu uporabnikov.

Implementacija UCD je že sedaj v podjetje prinesla mnogo izboljšav. Če odvzamemo finančni vidik, se je inženirjem sprostil čas za pomembnejša dela in tako lahko prispevajo višjo dodano vrednost. Poenostavila se je koordinacija in transparentnost namestitev, prav tako pa se je izboljšalo zadovoljstvo vseh vpletenih v namestitveni tok. Zaznali smo, da se zaradi upada ročnih aktivnosti pojavlja manj človeških napak, prav tako pa so povrnitve stanj, v primeru težav z aplikacijo, bistveno hitreje izvedene.

Zanesljivost in ponovljivost namestitev ponuja tako vsem vpletenim v namestitvenem toku kot tudi managerjem večje zaupanje in manj stresne odločitve, o sicer tveganih posegih spreminjanja delovanja aplikacij oz. IS. S stališča teorije bi bilo treba v prihodnosti izvajati čim bolj frekventne namestitve novih verzij aplikacij, ki bi predstavljale minimalne spremembe. Na ta način bi se zmanjšalo tveganje večjih nepravilnosti aplikacij in sprotno preverjanje ustreznega razvoja aplikacij, tako razvoja kot uporabnikov.

3.4 Analiza stroškov in koristi

V tem poglavju so predstavljeni ekonomski vidiki upravičenosti vpeljave orodja IBM UCD v obravnavano podjetje.

Ko smo v podjetju zaznali potrebo po vpeljavi ene od rešitev ARA, smo začeli z iskanjem primernih produktov na trgu. K odločitvi za izbiro orodja IBM UCD so pripomogli rezultati metode točkovanja, ki jih najdemo v tabeli 1. Primerjali smo 7 konkurenčnih produktov ter parametre, ki smo jih razdelili na 5 sklopov in so v tabeli 1 nanizani na levi strani. Sklopi so: interni infrastrukturni pogoji, oblačni infrastrukturni pogoji, aplikacijski pogoji, funkcionalnosti orodja ARA in cena. Glede na potrebe obravnavanega podjetja smo parametrom dodelili uteži. Med vsemi orodji je doseglo največ točk orodje IBM UCD, pred orodjema CA Release Automation in Serena Deployment Automation.

Tabela 1: Primerjava orodij za avtomatično nameščanje

	Utež	Automic One Automation	CA Release Automation	Electric Cloud ElectricFlow Deploy	IBM UCD	MidVision RapidDeploy	Serena Deployment Automation	XL Deploy
Interni infrastrukturni pogoji	0,36							
Možnost integracije z orodjem CI Jenkins	0,12	0,12	0,12	0,12	0,12	0,12	0,12	0,12
Podpora virov OS AIX	0,12	0,12	0,12	0,12	0,12	0,12	0,12	0,12
Podpora virov OS Win	0,12	0,12	0,12	0,12	0,12	0,12	0,12	0,12
Oblačni infrastrukturni pogoji	0,04							
Povezljivost z Microsoft Azure	0,02	0,00	0,02	0,02	0,02	0,02	0,02	0,02
Povezljivost z VMware vSphere	0,01	0,01	0,01	0,01	0,01	0,00	0,01	0,01
Povezljivost z IBM Bluemix	0,01	0,00	0,00	0,00	0,01	0,00	0,00	0,00
Aplikacijski pogoji	0,12							
Izdajanje kompozitne aplikacije	0,12	0,12	0,12	0,12	0,12	0,12	0,12	0,12
Funkcionalnosti orodja ARA	0,38							
Verzioniranje izdaje	0,02	0,02	0,02	0,01	0,02	0,02	0,02	0,02
Verzioniranje konfiguracij	0,02	0,02	0,02	0,01	0,02	0,02	0,02	0,02
Možnost povrnitve v prejšnje stanje pri nameščanju	0,08	0,08	0,08	0,00	0,08	0,08	0,08	0,08
Kombiniranje avtomatičnih in ročnih korakov	0,02	0,02	0,02	0,02	0,02	0,02	0,02	0,02
Elektronsko obveščanje	0,03	0,03	0,03	0,03	0,03	0,03	0,03	0,03
Sporočila v uporabniškem vmesniku	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01
Uporabniški vmesnik za nameščanje	0,01	0,01	0,01	0,01	0,01	0,01	0,01	0,01
Podpora	0,10	0,40	0,50	0,20	0,50	0,10	0,40	0,30
Sledljivost namestitve	0,04	0,04	0,04	0,00	0,04	0,04	0,04	0,04
Razširljivost funkcionalnosti	0,05	0,15	0,20	0,25	0,25	0,20	0,15	0,20
Cena	0,10	0,18	0,30	0,10	0,30	0,50	0,42	0,34
Skupaj	**1,00**	**1,45**	**1,74**	**1,15**	**1,80**	**1,53**	**1,71**	**1,58**

Analitsko podjetje Forrester je naredilo raziskavo na podlagi intervjujev v 4 podjetjih, ki so že vpeljala IBM UCD. Kot pravijo, vpeljava IBM UCD lahko pospeši dostavo aplikacij, razbremeni inženirje, ki so se ukvarjali tudi z nameščanji, in zmanjša tveganja, povezana z namestitvami.

Poudarek so dali na koristi, stroške, prilagodljivost in tveganje. Rezultati raziskave so pokazali naslednje koristi (Forrester Research, Inc, 2015, str. 4):

- povečano produktivnost pri razvoju aplikacij za vsaj 15 %,
- hitrejši čas dostave aplikacij ter večjo produktivnost pri namestitvah,
- zmanjšanje možnosti neuspelih namestitev – vsa podjetja so navedla zmanjšanje neuspelih namestitev, s čimer so se posledično zmanjšali tudi stroški,
- zmanjšanje stroškov za namestitev za 97 %,
- hitrejše namestitve za 75 %, povečanje zanesljivosti namestitev,
- vpeljava standardizacij pri namestitvah,
- izboljšana transparentnost in skalabilnost v namestitvenih procesih,
- izboljšano zadovoljstvo vseh zaposlenih.

IBM (2013, str. 2) je naredil raziskavo na podlagi stališč strank, ki se pri odločanju vpeljave orodja ARA pogostokrat pri tehtanju koristi osredotočajo predvsem na nekatere vidike (hitrost dostave, transparentnost itd.). Pri tem pa velikokrat pozabijo upoštevati vpliv vpeljave na samo organizacijo. IBM nadaljuje, da vpeljava avtomatizacije namestitev prinaša naslednje:

- izboljšano produktivnost, saj se ročno delo nadomesti z avtomatiziranim nameščanjem,
- zmanjšanje napak pri nameščanju, hitrejša izvedba namestitev, zmanjšan čas nedelovanja produkcije zaradi nameščanj,
- skrajšanje časa nedelovanja produkcije v času nameščanja,
- izboljšano transparentnost in nadzor nad namestitvami.

Vse skupaj naj bi podjetju prinašalo številne prednosti, kot so hitrejša dostava aplikacij, izboljšana agilnost poslovanja, eliminacija ozkih grl v namestitvenem procesu, manjši stroški in manjša tveganja, povezana z nedelovanjem produkcijskega okolja (IBM, 2013, str. 2).

Upravičenost vpeljave lahko utemeljimo z izračunom neto sedanje vrednosti (angl. *net present value*, v nadaljevanju NSV) za določeno obdobje. Pozitiven rezultat pomeni zeleno luč za vpeljavo orodja. Enačba 1 prikazuje vse denarne tokove D_k prihodnjih n let z diskontno stopnjo i, ki so zmanjšani za začetno investicijo I_0.

$$NSV = \sum_{k=1}^{n} \frac{D_1}{1+i} + \frac{D_2}{(1+i)^2} + \cdots + \frac{D_n}{(1+i)^n} - I_0 = \left(\sum_{k=1}^{n} \frac{D_k}{(1+i)^k}\right) - I_0 \qquad (1)$$

Tabela 2 prikazuje NSV za IBM UCD v izbranem podjetju. V izračunu je upoštevano naslednje.

- **Začetna investicija:** zajema nakup 8 licenc za sočasno izvajanje 8 agentov (IBM UrbanCode Deploy Session per Simultaneous Session License + SW Subscription & Support 12 Months) in vključuje stroške vzdrževanja in strokovne podpore v prvem letu. Ena licenca stane 26.441 € (View Pricing and Buy, 2016).

- **Stroški vzdrževanja UCD:** prvo leto jih ni, saj so vključeni v začetno investicijo. Naslednja leta predstavljajo 20 % začetne investicije.
- **Stroški virtualnega strežnika:** so sestavljeni iz stroškov strojne opreme, licenc in vzdrževanja, varnostnega kopiranja ter elektrike in klimatizacije.
- **Strošek namestitev UCD:** vključuje skupno število vseh namestitev preko UCD, pomnoženo z ocenjeno lastno ceno ene namestitve preko UCD.
- **Skupaj stroški:** predstavljajo skupen seštevek vseh letnih stroškov, povezanih z UCD.
- **Prihranki namestitev UCD:** število vseh namestitev je pomnoženo z ocenjeno lastno ceno ročnih namestitev.
- **Prihranki odpravljanja napake pri namestitvah:** ocenjen odstotek napak pri ročnem nameščanju je 5 %.
- **Skupaj prihodki:** skupen seštevek vseh letnih prihodkov.

Tabela 2: Podatki za izračun NSV

Leto	**1 (v EUR)**	**2 (v EUR)**	**3 (v EUR)**	**4 (v EUR)**	**5 (v EUR)**
Začetna investicija	–211.528				
Stroški vzdrževanja UCD	0	–42.306	–42.306	–42.306	–42.306
Stroški virtualnega strežnika	–300	–300	–300	–300	–300
strojna oprema	–165	–165	–165	–165	–165
licence in vzdrževanje	–90	–90	–90	–90	–90
varnostno kopiranje	–30	–30	–30	–30	–30
elektrika in klimatizacija	–15	–15	–15	–15	–15
Strošek namestitev UCD	–3.600	–4.914	–5.160	–5.418	–5.689
Skupaj stroški	**–215.428**	**–47.520**	**–47.765**	**–48.023**	**–48.294**
Prihranki namestitev UCD	108.000	147.420	154.791	162.531	170.657
Prihranki odpravljanja napake pri namestitvah	2.700	3.686	3.870	4.063	4.266
Skupaj prihodki	**110.700**	**151.106**	**158.661**	**166.594**	**174.924**
Dobiček/izguba	–104.728	103.586	110.895	118.571	126.629
Kumulativa	–104.728	–1.142	109.753	228.324	354.953

V tabeli 3 je pri izračunih uporabljena diskontna stopnja 9%. Analiza NSV za obdobje 5 let podjetju potrjuje smotrnost investicije, ki se bo glede na izračun povrnila v 2 letih in 4 dneh. To potrjuje tudi ROI z vrednostjo 84%.

Tabela 3: Rezultati NSV

Leto	1 (v EUR)	2 (v EUR)	3 (v EUR)	4 (v EUR)	5 (v EUR)	Skupaj
Diskontirani stroški (i= 9%)	-215.131	-47.114	-47.339	-47.576	-47.824	**-404.984** €
Diskontirani prihodki (i= 9%)	101.560	127.182	122.515	118.019	113.688	**582.965** €
NSV (i= 9%)	-113.571	80.069	75.176	70.443	65.864	**177.980** €
ROI (v %)						**84 %**
Doba povračila investicije (v letih)						**2,01**

Na končne rezultate je imelo največji vpliv zmanjšanje števila delovnih ur zaposlenih v primeru UCD, ki je pri ročnih namestitvah predstavljalo največji strošek. Za nadaljnjo povečanje ROI mora podjetje v avtomatična nameščanja vključevati čim več aplikacij, s čimer bo izkoriščalo ekonomijo obsega. V primeru, da bomo v podjetju izkoriščali prednosti vpeljevanja majhnih sprememb v aplikacije in s tem zelo frekventne namestitve, lahko pričakujemo bistveno povečanje ROI. Vpeljava UCD pa prinaša še dodatne neoprijemljive koristi, kot so: zadovoljstvo uporabnikov, razbremenitev sistemskih administratorjev rutinskih del (znotraj in zunaj delovnega časa), poenostavljeno koordinacijo in pregled nad namestitvami, s samopostrežnimi namestitvami opolnomočenje zaposlenih, enostaven nadzor in pregled nad namestitvami.

SKLEP

Če želijo podjetja na trgu dolgoročno uspešno poslovati, se morajo neprestano prilagajati in boriti za tržni delež. Uporaba IS je v mnogih podjetjih že dolgo obvezna, saj podpira izvajanje temeljnih poslovnih procesov. Prednost, ki jo prinaša IT v podjetja, ni samo zmanjševanje stroškov poslovanja, ampak tudi platforma, ki preko ustrezne in inovativne rabe ponuja trgu nove oz. diferencirane izdelke oz. storitve. IS skozi čas skupaj s podjetjem raste in se spreminja glede na trenutne potrebe, s čimer se povečuje njegova kompleksnost. Hitro se lahko zgodi, da heterogenost sistema, obremenjenost s starimi rešitvami, nenehno dograjevanje in prilagajanje IS pripelje do težko obvladljivega in vsesplošno neoptimiziranega sistema. Vse to se izraža z nezmožnostjo hitrega in uspešnega prilagajanja ter z visokimi stroški upravljanja, kar podjetju ne prinaša takšnih koristi, kot bi si jih želelo.

Mantra neučinkovitega IS je največkrat kompleksnost sistema. Z zasledovanjem uporabe modularnosti, skalabilnosti, standardizacije, avtomatizacije itd. v IS postavljamo predpogoje za poenostavitev samega upravljanja. Avtomatizacija procesov v IS v podjetje navadno prinaša optimizacijo in zmanjševanje potrebnih resursov. Vpeljevanje avtomatizacije zmanjšuje stroške, omogoča hitrejše izvajanje in zmanjšuje prostor za človeške napake. S pomočjo avtomatiziranih procesov lahko prav tako lažje in učinkoviteje obvladujemo kompleksnost. Poleg tega se ljudem odpravlja enolična in dolgočasna dela, s čimer avtomatizacija pozitivno vpliva tako z ekonomskega kot tudi z organizacijskega vidika. Kljub temu je potrebna previdnost pri samem

modeliranju in vpeljevanju avtomatizacij poslovnih procesov. Avtomatizirani procesi morajo biti robustni, s čimer naj bi zajemali čim več robnih primerov. V nasprotnem primeru lahko avtomatizacija prinese več škode kot koristi zaradi posledičnega nepravilnega delovanja in s tem povezanimi stroški. Pri vpeljevanju avtomatizacije procesov in ponovnem modeliranju lahko posredno vplivamo na zmanjševanje kompleksnosti sistema, saj se takrat ukvarjamo s podrobnim pregledom obravnavanega procesa.

V podjetjih, ki imajo realiziran interni razvoj večjega števila aplikacij, se danes informatika sooča s številnimi izzivi. Poslovni procesi, ki omogočajo poslovanje podjetja, so pravzaprav velikokrat abstrahirani/definirani v internih aplikacijah. Nenehno spreminjanje situacije na trgu zahteva učinkovito prilagajanje podjetja novim razmeram. Leon C. Megginson (1963, str. 4) je zapisal naslednjo misel, ki naj bi jo črpal iz knjige O nastanku vrst (Charles Darwin, 1859):

»*It is not the strongest of the species that survives, nor the most intelligent that survives. It is the one that is the most adaptable to change*.[4]«

Navedeni citat velja tudi za podjetja, umeščena v poslovno okolje, kjer med seboj tekmujejo in bijejo boj za obstanek. Tako morajo biti aplikacije ob pojavitvi kakršnihkoli potreb (vsebinske, funkcionalne spremembe) s strani podjetja čim prej prilagojene. Hitrost in zmožnost dostavljanja realiziranih sprememb od same ideje do produkcijske uporabe je sestavljena iz veliko aktivnosti. Celotna veriga dostave mora omogočati hiter, zanesljiv, transparenten in stroškovno učinkovit proces vpeljevanja sprememb v IS.

Z opisano problematiko se ukvarja pristop razvoja programske opreme CD. Ta združuje več medsebojno povezanih podprocesov in skupin zaposlenih, ki so vključeni v proces dostavljanja novih verzij aplikacij končnim uporabnikom. Hrbtenico celotnega procesa predstavlja namestitveni tok, kjer vhodi v proces predstavljajo razvite dele programske kode posameznega razvijalca, končni izhod pa predstavlja izdajo aplikacije na produkcijsko okolje. Preko številnih testov primernosti aplikacij se v procesu namestitvenega toka eliminira neustrezne kandidate verzij aplikacij, z namenom zagotavljanja ustreznih verzij aplikacij pri izdaji do končnih uporabnikov.

DevOps kultura se ukvarja s pomenom sodelovanja in podiranja preprek med skupinami z različnimi znanji in nalogami. Večina metodologij razvoja programske opreme ne obravnava detajlnih namestitvenih procesov, ki so na splošno velikokrat prezrti in dostikrat predstavljajo ozka grla pri izvajanju sprememb v IS oz. dostavljanju novih verzij aplikacij na produkcijo. Ravno te razlike in različni pogledi naj bi botrovali neuglašenosti skupin, naravnanih v skupen cilj. Tako CD kot tudi DevOps izpostavljata avtomatizacijo ponavljajočih procesov, ki je ena od njunih glavnih sporočil. Avtomatiziran proces dostave omogoča hiter, zanesljiv, transparenten in varen (manj človeških napak) cikel vpeljevanja sprememb. Lahko rečemo, da se avtomatizirane

[4] »*Ne preživijo najmočnejše vrste niti ne najinteligentnejše, temveč tiste, ki se najbolje prilagodijo spremembam.*«

aktivnosti lahko vključuje tako pri samem dostavljanju prilagojenih aplikacij glede na poslovni nivo (usklajevanje IS) kot tudi pri povečevanju oz. doprinosu k učinkovitosti samega IS.

Orodja ARA rešujejo problem nameščanj aplikacij preko številnih okolij, na avtomatiziran način z vsemi omenjenimi prednostmi, ki jih prinaša avtomatizacija. Na ta način se ustvari pogoje, da lahko razvoj aplikacij poteka zelo agilno. Z majhnimi in hitrimi namestitvami novo razvitih verzij aplikacij je omogočen progresiven in nadzorovan neprekinjen proces izvajanja sprememb. S tem poslovni del prejema hitrejše informacije o ustreznosti rešitev in na proaktiven način sodeluje pri nadaljnjih spremembah, prilagoditvah in ukrepih, povezanimi z nadaljnjim razvojem, s čimer podjetje bolje izrablja namen informatike in koristi lastnega razvoja.

V opazovanem podjetju smo zaznali prednosti, ki jih prinašajo orodja ARA. Kot sistemski administratorji smo vpeti v namestitveni tok ter pripadajoče namestitvene procese. Ker poznamo posebnosti namestitev in njihovega izvajanja, smo pri vpeljavi orodja UCD sodelovali od začetka. Skrbimo za povezovanje orodja UCD z ostalimi sistemi, modeliranje namestitev aplikacij, vzdrževanje in pravilno delovanje namestitev aplikacij.

Produkt UCD smo vključili v celoten proces dostave novih verzij aplikacij vse do končnih uporabnikov. S tem se je namestitveni tok dodatno avtomatiziral v procesu nameščanja aplikacij preko različnih okolij. Uspešnost izvedb ročnih namestitev je bila odvisna od specifičnega, nedokumentiranega znanja posameznikov. Z vpeljavo UCD so postali namestitveni postopki preprosti, kjer dokumentacijo nameščanj predstavljajo modelirani procesi nameščanja in pripadajoče komponente v samem orodju UCD. Teorija transparentnosti, zanesljivosti, ponovljivosti in hitrosti namestitvenih postopkov se z uporabo orodja UCD pokaže kot uspešna tudi v praksi.

Zaposlene je implementirano orodje razbremenilo ponovljivih in časovno potratnih aktivnosti, s čimer so se lahko osredotočili na pomembnejše delovne naloge. Samopostrežen način nameščanja je v proces nameščanja vključil številne zaposlene iz različnih skupin. Namestitve na zahtevo in trenutno potrebo posameznika so od same vzpostavitve orodja pokazale bistveno povečanje števila namestitev. To nakazuje, da so bili prejšnji ročni procesi in potrebna koordinacija preveč kompleksni in časovno potratni, kar je lahko marsikoga odvrnilo od izdajanja namestitvenega zahtevka, če le to ni bilo nujno.

V podjetju se trudimo v orodje UCD vpeljevati čim več aplikacij in avtomatiziranih procesov spreminjanja infrastrukture IS, s čimer bomo izkoriščali ekonomijo obsega. S tem bomo dosegli boljše rezultate tako pri dostavljanju novih verzij kot tudi pri zadovoljstvu uporabnikov. Povečevanje frekventnosti namestitev lahko pričakujemo z vključevanjem vseh komponent, ki sestavljajo večnivojsko aplikacijo (programski paket, konfiguracijska datoteka, spremembe na podatkovni bazi). Ekonomsko upravičenost vpeljave UCD je podkrepila finančna analiza, ki je pokazala, da se investicija povrne v 2 letih z vrednostjo ROI 84 %.

Kot smiselno nadaljevanje vpeljevanja avtomatizacije, povezanega s prilagajanjem IS, ki je pravzaprav neposredno povezano tudi z razvojem aplikacij, je avtomatizacija infrastrukture v IS (aplikacijski strežniki, podatkovne baze, omrežna infrastruktura in ostale komponente). Zavedati se je treba, da sam nakup orodij za avtomatizacijo procesov ne pomeni brezpogojnih pozitivnih učinkov. Nakup orodij oz. rešitev mora temeljiti na potrebah podjetja in ustrezni uporabi, ki upravičuje potencialno investicijo. Takšna orodja sicer rešujejo izvajanje avtomatizacije procesov, vendar na koncu še vedno predstavljajo le sisteme, ki jih je kot prvo treba pravilno implementirati, kot drugo pa ustrezno uporabljati.

LITERATURA IN VIRI

1. *A Robust Delivery Pipeline is the key to your success.* Najdeno 24. februarja 2016 na spletnem naslovu http://www.panthacorp.com/continuous-delivery-for-business

2. Allspaw, J. (2010, 9. december). DevOps: These Soft Parts. *AgileWebOperations*. Najdeno 23. januarja 2016 na spletnem naslovu http://www.agileweboperations.com/devops-these-soft-parts

3. Alter, S. (2002). *Information Systems: Foundation of E-Business* (4th ed.). New Jersey: Pearson Education, Inc.

4. Application release automation (ARA). (b.l.). V *Gartner IT Glossary*. Najdeno 9. septembra 2015 na spletnem naslovu http://www.gartner.com/it-glossary/application-release-automation-ara/

5. Attaran, M. (2003). Information technology and business-process redesign. *Business Process Management Journal*, *9*(4), 440-458.

6. Avison, D., & Fitzgerald, G. (2003). *Information Systems Development: Methodologies, Technique, and Tools (3rd ed.)*. New York: McGraw Hill Education.

7. Barosa, R. (2015, 15. julij). *Demo: Automatic deployment to z/OS and other platforms using UrbanCode*. Orlando: Share, Inc.

8. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, j., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D. (2001). Manifest agilnega razvoja programske opreme. Najdeno 20. decembra 2015 na spletnem naslovu http://www.agilemanifesto.org/principles.html

9. Betteley, J. (b.l.). Deployment Automation Patterns. *DZone*. Najdeno 28. februarja 2016 na spletnem naslovu https://dzone.com/refcardz/deployment-automation-patterns

10. Bird, J. (2015). *DevOps for Finance*. Sebastopol: O'Reilly Media, Inc.

11. *BlueGreenDeployment*. Najdeno 18. septembra 2015 na spletnem naslovu http://martinfowler.com/bliki/BlueGreenDeployment.html

12. Bossert, O., Ip, C., & Laartz, J. (2014, december). A two-speed IT architecture for the digital enterprise. *McKinsey & Company*. Najdeno 15. decembra 2015 na spletnem naslovu http://www.mckinsey.com/insights/business_technology/a_two_speed_it_architecture_for_the_digital_enterprise

13. Bradley, T. (2015, 29. september). The business benefits of DevOps. *Devops.* Najdeno 3. oktobra 2015 na spletnem naslovu http://devops.com/2015/09/29/business-benefits-devops/

14. Bridgwater, A. (2015, junij). Inside DevOps: How it really works. *ComputerWeekly*. Najdeno 3. oktobra 2015 na spletnem naslovu http://www.computerweekly.com/feature/Inside-DevOps-How-it-really-works

15. Brown, N. (2015). Improving Performance Through Business Process Re-engineering: Aligning People, Process and Technology to Ensure Success. *Datamark*. Najdeno 15. novembra 2015 na spletnem naslovu http://www.datamark.net/uploads/files/improving_performance.pdf

16. Budhabhatti, M. (2008, januar). Test-Driven Development and Continuous Integration for Mobile Applications. *Microsoft*. Najdeno 5. januarja 2016 na spletnem naslovu https://msdn.microsoft.com/en-us/library/bb985498.aspx

17. Chang, M. (2010). An Agile approach to library IT innovations. *Library Hi Tech*, *28*(4), 672-689.

18. Chaffey, D. (2002). *E-Business and E-Commerce Management*. London: Pearson Education Ltd.

19. Cherry, P. (2014, 26. januar). Approaches to Application Release Automation. *DevOps*. Najdeno 15. oktobra 2015 na spletnem naslovu http://devops.com/2014/01/26/approaches-to-application-release-automation/

20. Cois, C. A. (2014, 13. november). DevOps and Agile. *SEI*. Najdeno 10. marca 2016 na spletnem naslovu https://insights.sei.cmu.edu/sei_blog/2014/11/devops-and-agile.html

21. *Continuous Delivery*. Najdeno 20. decembra 2015 na spletnem naslovu http://martinfowler.com/bliki/ContinuousDelivery.html

22. *Continuous Delivery*. Najdeno 20. januarja 2016 na spletnem naslovu https://developer.ibm.com/urbancode/products/solutions-continuous-delivery/

23. *Continuous Integration*. Najdeno 10. decembra 2015 na spletnem naslovu https://en.wikipedia.org/wiki/Continuous_integration#Automate_the_build

24. *Continuous integration and build management.* Najdeno 2. decembra 2015 na spletnem naslovu http://www-03.ibm.com/software/products/en/ucbuild

25. DeArdo, C. (2015, 23. julij). The Calculus of DevOps. *Devops.* Najdeno 17. septembra 2015 na spletnem naslovu http://devops.com/2015/07/23/the-calculus-of-devops/

26. DeMartine, A. (2015, 29. januar). Market Overview: Application Release Automation Tools. *Forrester*. Najdeno 5. februarja 2016 na spletnem naslovu https://www.forrester.com/Market+Overview+Application+Release+Automation+Tools/fulltext/-/E-res119004

27. DeMartine, A., & Bittner, K. (2015, 14. april). The Forrester Wave™: Application Release Automation, Q2 2015. Najdeno 15. februarja 2016 na spletnem naslovu http://electric-cloud.com/wp-content/uploads/2015/04/The_Forrester_Wave__Appli.pdf

28. *DeploymentPipeline.* Najdeno 13. januarja 2016 na spletnem naslovu http://martinfowler.com/bliki/DeploymentPipeline.html

29. Driessen, V. (2015, 5. januar). A successful Git branching model. *Nvie*. Najdeno 10. januarja na spletnem naslovu http://nvie.com/posts/a-successful-git-branching-model/

30. Duvall, P. M. (2011). Continuous Delivery. *Dccia.* Najdeno 20. januarja 2016 na spletnem naslovu http://www.dccia.ua.es/dccia/inf/asignaturas/MADS/lecturas/10_Continuous_Delivery_Dzone_Refcardz.pdf

31. Edwards, D. (2010, 23. februar). What is DevOps? *Dev2ops*. Najdeno 9. septembra 2015 na spletnem naslovu http://dev2ops.org/2010/02/what-is-devops/

32. Erl, T. (2009). *SOA Design Patterns*. Boston: Pearson Education, Inc.

33. Fletcher, C., & Colville, R. J. (2015, 20. julij). Market Guide for Application Release Automation Solutions. *Gartner*. Najdeno 15. oktobra 2015 na spletnem naslovu https://www.gartner.com/doc/3097026/market-guide-application-release-automation

34. Forrester Research, Inc. (2015, avgust). *The Total Economic Impact™ Of IBM UrbanCode: Achieving Velocity In An Enterprise DevOps Environment*. Cambridge: Forrester Research, Inc.

35. *Gartner Says By 2016, DevOps Will Evolve From a Niche to a Mainstream Strategy Employed by 25 Percent of Global 2000 Organizations.* Najdeno 10. oktobra 2015 na spletnem naslovu http://www.gartner.com/newsroom/id/2999017

36. Gates, B. (2001). *Business @ the Speed of Thought*. Harlow: Pearson Education Limited.

37. Godinez, M., Hechler, E., Koenig, K., Lockwood, S., Oberhofer, M., & Schroeck, M. (2010). *The Art of Enterprise Information Architecture: A System-Based Approach for Unlocking Business Insight*. Boston: Pearson Plc.

38. Gohring, N. (2012, 20. marec). 'NoOps' Debate Grows Heated. *PC World.* Najdeno 18. septembra 2015 na spletnem naslovu http://www.pcworld.com/article/252264/noops_debate_grows_heated.html

39. Gorga, I., Barillari, F., & Andreuzzi, M. (2014, 30. julij). Implement continuous delivery using IBM UrbanCode Deploy. *IBM*. Najdeno 1. marca 2016 na spletnem naslovu http://www.ibm.com/developerworks/cloud/library/cl-softlayer-urbancode-trs/

40. Gualtieri, M. (2011, 7. februar). I Don't Want DevOps. I Want NoOps. *Forrester*. Najdeno 9. septembra 2015 na spletnem naslovu http://blogs.forrester.com/mike_gualtieri/11-02-07-i_dont_want_devops_i_want_noops

41. Hammer, M. (2000, julij-avgust). Reengineering Work: Don't Automate, Obliterate. *Harvard Business Review*, str. 104-107.

42. Hammond, J. (2011, 9. september). The Relationship Between Dev-Ops And Continuous Delivery: A Conversation With Jez Humble Of ThoughtWorks. *Forrester*. Najdeno 11. novembra 2015 na spletnem naslovu http://blogs.forrester.com/jeffrey_hammond/11-09-09-the_relationship_between_dev_ops_and_continuous_delivery_a_conversation_with_jez_humble_of_thought

43. Hulme, G. V. (2014, 14. maj). DevOps: Getting Past Audit. *Devops*. Najdeno 21. septembra 2015 na spletnem naslovu http://devops.com/2014/05/14/devops-getting-past-audit/

44. Humble, J., & Farley. D. (2011). *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation.* Boston: Pearson Education, Inc.

45. IBM Corporation. (2013, januar). *ROI: The Value of Deployment Automation.* Somers: IBM Corporation.

46. Jackson, J. (2013, 22. april). IBM snatches UrbanCode for devops prowess. *Infoworld.* Najdeno 17. septembra 2015 na spletnem naslovu http://www.infoworld.com/article/2614257/application-development/application-development-ibm-snatches-urbancode-for-devops-prowess.html

47. Kersten, B., & Verhoef, C. (2003). IT portfolio management: a banker's perspective on IT. *Cutter IT Journal, 16*(4), 34-40.

48. Khan, K. M., & Zheng, Y. (2005). *Managing Corporate Information Systems Evolution and Maintenance*. Hershey: Idea Group Publishing.

49. Kim, G. (2014, 22. maj). Enterprise DevOps Adoption Isn't Mandatory — but Neither Is Survival. *The Wall Street Journal*. Najdeno 9. septembra 2015 na spletnem naslovu

http://blogs.wsj.com/cio/2014/05/22/enterprise-devops-adoption-isnt-mandatory-but-neither-is-survival/

50. Kim, G., & Humble, J. (2014). The Science Behind the 2014 Puppet Labs DevOps Survey Of Practice. *IT Revolution.* Najdeno 21. septembra 2015 na spletnem naslovu http://itrevolution.com/the-science-behind-the-2013-puppet-labs-devops-survey-of-practice/

51. Kotter, J.P. (1995, marec-april). Leading Change: Why Transformation Efforts Fail. *Harvard Business Review*, str. 59-67.

52. Kovačič, A. (2005). Management poslovnih procesov, Prenova in informatizacija poslovanja. Ljubljana: GV založba.

53. Kovačič, A., Jaklič, J., Indihar Štemberger, M., & Groznik, A. (2004). Prenova in informatizacija poslovanja. Ljubljana: Ekonomska fakulteta.

54. Krill, P. (2015, 8. januar). 7 cool tools for doing devops right. *InfoWorld*. Najdeno 15. oktobra na spletnem naslovu http://www.infoworld.com/article/2866574/devops/7-cool-tools-for-doing-devops-right.html#slide1

55. Larman, C. (2004). *Agile and iterative development: a manager's guide.* Boston: Pearson Education, Inc.

56. *List of Build Automation Software*. Najdeno 20. novembra 2015 na spletnem naslovu https://en.wikipedia.org/wiki/List_of_build_automation_software

57. *Manifesto for Agile Software Development.* Najdeno 18. januarja 2016 na spletnem naslovu http://agilemanifesto.org/

58. Marschall, M. (2012, 12. julij). How are Lean, Agile, and Devops related to each other? *AgileWebOperations*. Najdeno 23. januarja 2016 na spletnem naslovu http://www.agileweboperations.com/lean-agile-devops-related

59. Martin, B. (2014, 9. oktober). DevOps, Agile Development, Continuous Delivery und ITIL: Was gibt es für Zusammenhänge und wie man sie nutzen kann. *ITIL*. Najdeno 10. januarja 2016 na spletnem naslovu http://blog.itil.org/2014/10/itil/devops-agile-development-continuous-delivery-und-itil-was-gibt-es-fuer-zusammenhaenge-und-wie-man-sie-nutzen-kann/

60. Megginson, L. C. (1963). Lessons from Europe for American Business. *Southwestern Social Science Quarterly*, *44*(1), 3-13.

61. Naegle, R. (2015, 13. avgust). IT Market Clock for IT Automation, 2015. *Gartner*. Najdeno 15. oktobra 2015 na spletnem naslovu https://www.gartner.com/doc/3111921/it-market-clock-it-automation

62. Neubauer, T. (2009). An empirical study about the status of business process management. *Business Process Management Journal*, *15*(2), 166-183.

63. Post, G., & Anderson, D. (2003). *Management information systems* (3rd ed.). New York: McGraw-Hill Higher Education.

64. *Processes*. Najdeno 10. februarja 2016 na spletnem naslovu https://www.ibm.com/support/knowledgecenter/SS4GSP_6.1.1/com.ibm.udeploy.doc/topics/comp_workflow.html

65. Pullen, W. (2015, 23. februar). Five Key Components of Application Release Automation (ARA). *Devops*. Najdeno 9. septembra 2015 na spletnem naslovu http://devops.com/2015/02/23/five-key-components-of-application-release-automation-ara/

66. Ragan, T. (2015, 30. oktober). Continuous Delivery versus Continuous Deploy. *DevOps*. Najdeno 1. februarja 2016 na spletnem naslovu http://devops.com/2015/10/30/continuous-delivery-versus-continuous-deploy/

67. Ravishankar, N. (2014, 18. april). DevOps and Agile. *ScrumAlliance*. Najdeno 8. januarja 2016 na spletnem naslovu https://www.scrumalliance.org/community/articles/2014/april/devops-and-agile

68. Reid, L. (2015, 22. junij). The Simple Math of DevOps. *Devops*. Najdeno 9. septembra 2015 na spletnem naslovu http://devops.com/2015/06/22/the-simple-math-of-devops/

69. Reis, E. (b.l.). The Lean Startup Methodology. *The Lean Startup*. Najdeno 27. februarja 2016 na spletnem naslovu http://theleanstartup.com/principles

70. Riezebos, J., Klingenberg, W., & Hicks, C. (2009, maj). Lean Production and information technology: Connection or contradiction? *Computers in Industry*, *60*(4), str. 237-247.

71. Rivera, J., & van der Meulen, R. (2015, 5. marec). Gartner Says By 2016, DevOps Will Evolve From a Niche to a Mainstream Strategy Employed by 25 Percent of Global 2000 Organizations. *Gartner*. Najdeno 9. septembra 2015 na spletnem naslovu http://www.gartner.com/newsroom/id/2999017

72. Rouse, M. (b.l.). Infrastructure as Code (IAC). *TechTarget*. Najdeno 16. januarja 2016 na spletnem naslovu http://searchcloudcomputing.techtarget.com/definition/Infrastructure-as-Code-IAC

73. Sharma, S. (2012a, 25. september). Understanding DevOps – Part 2: Continuous Integration and Continuous Delivery. *Wordpress.* Najdeno 15. decembra 2015 na spletnem naslovu https://sdarchitect.wordpress.com/2012/09/25/understanding-devops-part-2-continuous-integration-and-continuous-delivery/

74. Sharma, S. (2012b, 13. december). Understanding DevOps – Part 5: Infrastructure as Code. *Wordpress.* Najdeno 15. decembra 2015 na spletnem naslovu https://sdarchitect.wordpress.com/2012/12/13/infrastructure-as-code/

75. Sharma, S. (2013, 16. oktober). Understanding DevOps – Part 6: Continuous Deployment vs Continuous Delivery. *Wordpress.* Najdeno 15. decembra 2015 na spletnem naslovu https://sdarchitect.wordpress.com/2013/10/16/understanding-devops-part-6-continuous-deployment/

76. Sharma, S., & Coyne, B. (2015). *DevOps for Dummies*. New Jersey: John Wiley & Sons, Inc.

77. Shelly, G. B., Cashman, T. J., & Rosenblatt, H. J. (2008). *System Analysis and Design* (7th ed.). Boston: Thomson Course Technology.

78. Shim, J. K. (2000). *Information systems and technology for the noninformation systems executive: an intergrated resource management guide for the 21st century.* Boca Raton: CRC Press LLC.

79. Shpilbeg, D., Berez, S., Puryear, R., & Shah, S. (2007, 1. oktober). Avoiding the Alignment Trap in IT. *MITSloan Management Review*. Najdeno 5. januarja 2016 na spletnem naslovu http://sloanreview.mit.edu/article/avoiding-the-alignment-trap-in-it/

80. Sundman, Y. (2013, 5. februar). Continuous Delivery vs Continuous Deployment. *Crisp*. Najdeno 10. decembra 2015 na spletnem naslovu http://blog.crisp.se/2013/02/05/yassalsundman/continuous-delivery-vs-continuous-deployment

81. Šibka sklopljenost. (b.l.). V *iSlovarju*. Najdeno 20. decembra 2015 na spletnem naslovu http://www.islovar.org/izpisclanka.asp?back=iskanje_zadnji.asp&id=10281

82. *The Internet of Things Enables Digital Business*. Najdeno 5. junija 2015 na spletnem naslovu http://www.gartner.com/technology/research/internet-of-things/

83. *The New Methodology.* Najdeno 21. septembra 2015 na spletnem naslovu http://www.martinfowler.com/articles/newMethodology.html

84. *The XP Lifecycle*. Najdeno 3. januarja 2016 na spletnem naslovu http://www.jamesshore.com/downloads/art-of-agile/xp-lifecycle.pdf

85. *UrbanCode Deploy*. Najdeno 20. februarja 2016 na spletnem naslovu https://developer.ibm.com/urbancode/products/urbancode-deploy/

86. *UrbanCode Deploy with Patterns*. Najdeno 20. februarja 2016 na spletnem naslovu https://developer.ibm.com/urbancode/products/urbancode-deploy-with-patterns/

87. *UrbanCode Release.* Najdeno 20. februarja 2016 na spletnem naslovu https://developer.ibm.com/urbancode/products/urbancode-release/#media/0/

88. Vaughan-Nichols, S. J. (2015, 3. februar). DevOps theory for beginners. *CSC Blogs.* Najdeno 9. septembra 2015 na spletnem naslovu http://blogs.csc.com/2015/02/03/devops-theory-for-beginners/

89. *View Pricing and Buy*. Najdeno 20. marca 2016 na spletnem naslovu https://www-112.ibm.com/software/howtobuy/buyingtools/paexpress/Express?P0=E1&part_number=D1165LL,D1168LL,D1166LL,D1169LL,D12TILL,D12TKLL,D12TJLL,D12TMLL,D12TPLL,D12TRLL,D12U1LL,D12U3LL,D12U5LL,D12U6LL,D12U7LL,D12U9LL,D15HWLL,D15I1LL,D15I4LL,D15I5LL&catalogLocale=de_AT&Locale=de_AT&country=AUT&PT=jsp&CC=AUT&VP=&TACTICS=&S_TACT=&S_CMP=&brand=SSMGUC

90. Wagner, M. (2013, 1. oktober). Product Overview: The New IBM UrbanCode Deploy 6.0. *SlideShare*. Najdeno 20. februarja 2016 na spletnem naslovu http://www.slideshare.net/Urbancode/product-overview-the-new-ibm-urbancode-deploy-60

91. Wallace, P. (2015). *Introduction to Information Systems* (2nd ed.). London: Pearson Education Ltd.

92. *What is DevOps?* Najdeno 17. septembra 2015 na spletnem naslovu http://theagileadmin.com/what-is-devops/

93. *What is DevOps?* Najdeno 18. septembra 2015 na spletnem naslovu http://newrelic.com/devops/what-is-devops

94. *What Is Industrie 4.0 and What Should CIOs Do About It?* Najdeno 1. marca 2016 na spletnem naslovu http://www.gartner.com/newsroom/id/3054921

PRILOGE

SEZNAM KRATIC

ARA – avtomatično izdajanje aplikacij (angl. *application release automation*)
BPR – prenova poslovnih procesov (angl. *business process reingeniering*)
CD – neprekinjena dostava (angl. *continuous delivery*)
CDep – neprekinjeno nameščanje (angl. *continuous deployment*)
CI – neprekinjena integracija (angl. *continuous integraton*)
CMT – orodja za avtomatizirano upravljanje konfiguracij (angl. *configuration management tools*)
IAC – infrastruktura kot koda (angl. *infrastructure as code*)
IS – informacijski sistem (angl. *information system*)
IT – informacijska tehnologija (angl. *information technology*)
NSV – neto sedanja vrednost (angl. *net present value*)
QA – zagotavljanje kakovosti (angl. *quality assurance*)
ROI – donosnost investicije (angl. *return on investment*)
SOA – storitveno usmerjena arhitektura (angl. *service-oriented architecture*)
UCD – UrbanCode Deploy