UNIVERZA V LJUBLJANI
FAKUKLTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Sašo Knap

Združitev in prilagoditev metodologij Scrum in ekstremno programiranje za organizacijo razvoja programske opreme v podjetju

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: izr. prof. dr. Marko Bajec

Ljubljana, 2009

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01586/2009
Datum: 01.09.2009

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **SAŠO KNAP**

Naslov: **ZDRUŽITEV IN PRILAGODITEV METODOLOGIJ SCRUM IN EKSTREMNO PROGRAMIRANJE ZA ORGANIZACIJO RAZVOJA PROGRAMSKE OPREME V PODJETJU**

MERGING AND ADAPTATION OF SOFTWARE DEVELOPMENT METHODOLOGIES SCRUM AND EXTREME PROGRAMMING

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Za podporo razvoju programske opreme so danes na voljo številne metodologije. Posebej popularne so agilne metodologije, ki so navadno enostavnejše in lažje prilagodljive konkretnim potrebam. Ker je skoraj vsaka agilne metodologija v nečem dobra ali boljša od drugih, se je težko odločiti, kateri bi sledili. V okviru diplomske naloge preučite možnost združevanja agilnih metodologij ter njihovega prilagajanja konkretnemu podjetju. Kot primer vzemite metodologijo SCRUM in Extreme Programming.

Mentor:
prof. dr. Marko Bajec

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Franc Solina

*Namesto te strani **vstavite** original izdane teme diplomskega dela s podpisom mentorja in dekana ter žigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden odda izdelek v vezavo!*

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani **SAŠO KNAP**,

z vpisno številko **63010062**,

sem avtor diplomskega dela z naslovom:

Združitev in prilagoditev metodologij Scrum in ekstremno programiranje za organizacijo razvoja programske opreme v podjetju

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom
  izr. prof. dr. Marka Bajca,

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,

- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne 10.12.2009 Podpis avtorja:

Zahvala

Zahvaljujem se mentorju, izr. prof. dr. Marku Bajcu, za usmeritev pri izbiri teme in za strokovni pregled diplomske naloge.

Izredna zahvala gre mojim najbližjim, staršem in Nini, za vso podporo v času študija in v času pisanja diplomske naloge.

Hvala tudi vsem kolegom, ki so mi kakorkoli pomagali.

Kazalo vsebine

Seznam uporabljenih kratic in simbolov

BDD	–	Vedenjsko voden razvoj (Behavior Driven Development)
FDD	–	Funkcijsko voden razvoj (Feature Driven Development)
RITE	–	Hitro iterativno testiranje in vrednotenje (Rapid Iterative Testing and Evaluation)
RUP	–	Ogrodje objektnega procesnega modela (Rational Unified Process)
TDD	–	Testno voden razvoj (Test Driven Development)
XP	–	Ekstremno programiranje (Extreme Programming)

Povzetek

Pri razvoju programske opreme je bistvenega pomena odzivnost na zahteve naročnika in čim bolj natančna časovna ocena trajanja projekta. Stare metodologije razvoja programske opreme so za hitri agilni razvoj neprimerne, zlasti za manjša podjetja.

Namen diplomskega dela je predstavitev združitve dveh agilnih metodologij za razvoj programske opreme, vse skupaj pa prilagoditi tako, da se najbolje obnese za primer podjetja in novega razvojnega procesa v podjetju ter utemeljiti razloge za prehod na ta način agilnega razvoja.

Diplomsko delo na začetku predstavi zgodovino in pregled klasičnih in novejših agilnih metodologij za razvoj. Podrobneje sta opisani agilni metodologiji ekstremno programiranje in Scrum, ki ju avtor v zadnjem poglavju združi in prilagodi za potrebe razvoja v malem podjetju.

Ključne besede

Agilne metodologije, Scrum, ekstremno programiranje, združitev in prilagoditev agilnih metodologij za razvoj programske opreme

Abstract

Responsiveness to the requirements of the client and as accurate an estimate of duration of the project as possible are essential components of software development. Traditional software development methodologies are inappropriate for fast agile development, especially in small companies.

The purpose of this thesis is to present the mergence of two agile software development methodologies applied to the case of a company to correspond with the new development process, and to give reasons for transferring to the new agile development method.

This thesis begins with a presentation and review of the histrory and classic development methodologies, and continues with a detailed description of new agile development methodologies, especially Scrum and extreme programming. The last chapter concentrates on the merge of the two methodologies and their adaptation to the development of small companies.

Keywords

Agile methodologies, Scrum, extreme programming, merger and adaptation of agile methodologies for software development

1 Uvod

Prve metodologije razvoja programske opreme so nastale že v začetku sedemdesetih let dvajsetega stoletja in zaradi uspeha, ki so ga dosegle, tudi prepričale strokovno javnost, da je ureditev postopkov na tem področju nujnost. Od takrat naprej so se postopoma začele vpeljevati metode, postopki, tehnike in orodja, ki so zavrgla ad hoc razvoj in postavile temelje načrtovanega, dokumentiranega in vodenega razvoja. Po zgledu metodologij iz drugih panog so se razvile obsežne metodologije, ki so do potankosti opisovale vsak najmanjši korak razvoja. Za vsako aktivnost so bili predpisani številni dokumenti in formularji. Tovrstne metodologije so s časom postale preobsežne in je bilo zaradi gore dokumentacije nemogoče obvladovati razvoj in razvijati učinkovite sisteme. Zaradi sprememb na trgu je bilo treba prilagoditi razvoj tako, da se je hitro odzival na spremembe zahtev naročnikov in da je bil izdelek stabilen ter časovno in finančno nezahteven. Razvile so se agilne metodologije, ki se bolje prilagajajo spremembam.

Pri razvoju programske opreme je za obstoj na tržišču je potrebna hitra prilagodljivost, kvaliteten nadzor, natančno časovno načrtovanje, uporaba novih tehnologij ter kvaliteten in skalabilen izdelek. Zato je pri uvajanju metodologij v podjetje treba dodobra analizirati možnosti in ugotoviti, katero metodologijo bo mogoče ustrezno prilagoditi načinu razvoja, ki ga naroča trg oziroma stranka glede na tip izdelka, ki ga podjetje razvija in glede na velikost razvojnega oddelka. Vpeljava novega načina dela je v pričujoči diplomski nalogi predstavljena na primeru malega podjetja, ki na novo prevzema agilni način izgradnje programske opreme za svojo lastno uporabo, zaradi česar je najbolj ustrezna je dopolnjujoča se kombinacija aktualnih metodologij Scrum in ekstremno programiranje.

V diplomskem delu je v prvem delu na kratko predstavljen razvoj metodologij in način delovanja glavnih predstavnikov starejših, obsežnejših metodologij, pri katerih je veliko časa namenjenega načrtovanju in vodenju dokumentacije. Kasneje pa se naloga osredotoči na novejše, agilne metodologije, način uporabe in zahteve trga na odzivnejši način razvoja programske opreme. Šesto poglavje je bolj praktične narave in je posvečeno izbranim metodologijam za izboljšanje načina vodenja projektov znotraj razvoja in za izboljšanje kvalitete programske kode. Glede na to da Scrum in ekstremno programiranje vsaka zase ne pokrivata vseh prvin, potrebnih za kakovostno izgradnjo in vodenje razvoja, se tu naloga osredotoča na način, kako je mogoče obe metodologiji združiti in uporabiti tako, da se ne izključujeta.

2 Metodologije za razvoj programske opreme

Podjetja, ki razvijajo programsko opremo, imajo na razpolago veliko metodologij. V tem poglavju bo na kratko predstavljen razvoj metodologij, nekateri tipični predstavniki standardnih in agilnih metodologij, njihov namen, vrednotenje in sestava ter posledice uvedbe metodologije v podjetje.

2.1 Definicija metodologije

Definicija metodologije za razvoj programske opreme je, kot jo je izpeljal Cockburn [6] iz slovarske definicije metodologije, opredeljena kot skupek postopkov znotraj neke discipline (včasih so zraven vključeni tudi izobraževalni material, formalni učni programi, delovni programi in diagramska orodja), ki se lahko večkrat uporabijo pri gradnji programske opreme.

Tehnologija programske opreme zajema več področij, med katerimi so najbolj znane projektno vodenje, analiza, specifikacija, razvoj, kodiranje, testiranje in zagotavljanje kakovosti. Vse metodologije na področju tehnologije programske opreme predpisujejo delovanje teh področij in imajo vsaj eno skupno lastnost.

2.2 Razvoj metodologij

Zgodovino razvoja metodologij [26] lahko kronološko razdelimo na:

- obdobje pred pojavom metodologij (do 1970)
- zgodnje obdobje metodologij (od 1970 do 1980)
- obdobje uvajanja metodologij (do 1990)
- obdobje ponovne ocenitve metodologij (po 1990)

Za obdobje pred pojavom metodologij je bilo značilno, da je razvoj programske opreme temeljil na enem razvijalcu, ki je dobro poznal strojno opremo in programski jezik. Razvoj je potekal brez ustreznega vodenja in končni izdelek pogosto ni ustrezal potrebam naročnika. Razvoj je potekal ad hoc, brez začrtanih smernic.

V zgodnjem obdobju metodologij so se pojavile prve formalizacije. Proces razvoja programske opreme se je razdelil na posamezne korake in postopke. Značilen za to obdobje je pojav slapovnega razvoja programske opreme, kjer si faze razvoja sledijo postopoma in se prestop v naslednjo fazo razvoja zgodi, ko je predhodna faza popolnoma končana. Izoblikoval se je tudi strukturni pristop, ki je uvedel disciplinirano izvajanje analize in načrtovanja.

V obdobju uvajanja metodologij se začnejo razvijati obsežnejše metodologije in pojavi se komercializacija metodologij, ki se začnejo širiti tudi na strateško raven. Metodologije so se tako razvijale v svetovalnih hišah, ki so jih tržile kot izdelek. Vsak korak razvoja so opisale do podrobnosti, definirale večjo količino vlog, aktivnosti, faz in postopkov. Vse skupaj je bilo zelo obsežno in je vodilo do težavnega razumevanja metodologije. V veliki večini, predvsem na manjših projektih, so bili postopki preobsežni in neprimerni. Zaradi prevelike obsežnosti je prišlo do točke, ko so metodologije prišle do ponovne ocenitve.

2.3 Pregled znanih metodologij

Znanih je več modelov metodologij, pri čemer model predstavlja nekakšno zaporedje korakov, s katerimi se razvija programska oprema, metodologija pa vsebuje iz modela sestavljene prakse delovanja in priporočena orodja, shemo delovanja in način izobraževanja.

Najpomembnejše in najbolj znane družine metodologij razvoja programske opreme so:

- Modeli tradicionalnih metodologij:
  - Model od zgoraj navzdol (angl. top – down model)
  - Model od spodaj navzgor (angl. bottom – up model)
  - Slapovni model (angl. waterfall model)
  - Inkrementalni model (angl. incremental model)
- Modeli evolucijske metodologije:
  - Prototipni model (angl. prototyping model)
  - Spiralni model (angl. spiral model)
  - Ciklični in inkrementalni model (angl. iterative and incremental model)
- Agilne metodologije:
  - Ekstremno programiranje (angl. extreme programming)
  - Scrum
  - Vitek razvoj
  - Funkcijsko voden razvoj

2.3.1 Modeli tradicionalnih metodologij

2.3.1.1 Modela »od zgoraj navzdol« in »od spodaj navzgor«

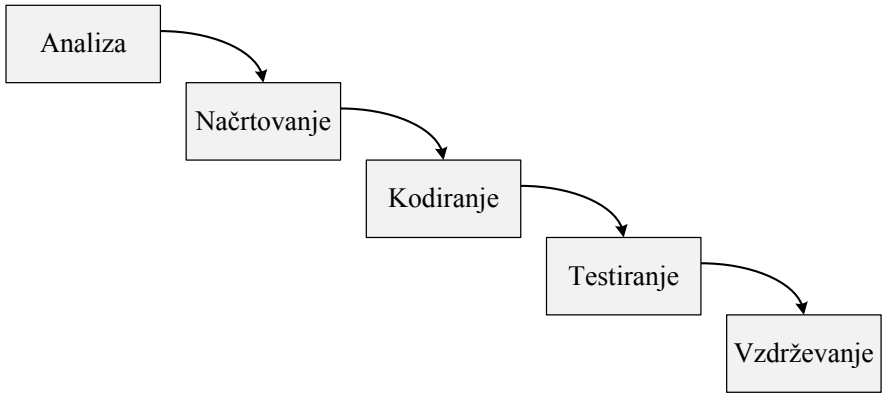
V to skupino uvrščamo družino tradicionalnih metodologij iz šestdesetih in sedemdesetih let prejšnjega stoletja. Značilno za tisto obdobje je bilo, da so se izvajali prvi veliki projekti, kar je sprožilo zahteve po vzpostavitvi pravil razvoja za koordinacijo in nadzor sistema. Pojavili so se principi razvoja 'od zgoraj navzdol' in 'od spodaj navzgor'.

2.3.1.2 Slapovni model

Slapovni model razvoja programske opreme je bil predstavljen leta 1970. Značilno za ta model je, da na začetku zahteva izčrpno analizo naročnikovih zahtev in potreb. Po dolgotrajni intenzivni komunikaciji med naročnikom, bodočimi uporabniki in razvijalci, se sestavi množica obširnih in trajnih funkcionalnih in nefunkcionalnih zahtev ter značilnosti sistema. Poteka natančno dokumentiranje, ki je potrebno za drugo fazo, načrtovanje. V tej fazi se izdela podroben načrt razvoja, sledi faza kodiranja na podlagi natančno specificirane dokumentacije. Po fazi implementacije pride na vrsto še testiranje, šele potem pa se izdelek preda naročniku v uporabo.

Najpogostejša motnja pri tem načinu razvoja, ki je sistem ne more optimalno razrešiti je, da si naročnik premisli glede zahtev. Po dolgem obdobju zbiranja zahtev in pisanja dokumentacij si uporabniki še vedno niso enotni o željah in zahtevah. V takem primeru metodologija zahteva ponovno analizo in načrtovanje oziroma vsaka sprememba zahtev ustavi razvoj in zahteva vrnitev nazaj na fazo analize. To tudi pomeni usklajevanje dokumentacije in hkrati časovni zamik.

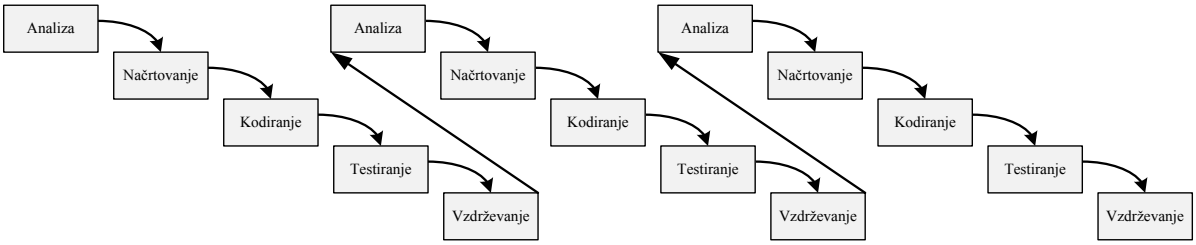
Slapovni model se je razvil z namenom, da naj bi rešil spreminjajoče se zahteve z obsežnejšo analizo in zamrznitvijo zahtev po analizi, v praksi pa je prišlo do spoznanj, da kljub razširjeni in podrobni analizi vseh zahtev sistema ni mogoče zajeti v eni fazi. [30]



Slika 1: Skica slapovnega modela

2.3.1.3 Inkrementalni model

Cilj inkrementalnega razvoja je skrajšati čas razvoja. Pojavi se večje število korakov, v katerem se izvedejo vse faze razvoja slapovnega modela. Razlika je v tem, da se ne razvija vseh funkcionalnosti naenkrat, temveč se jih razbije na več neodvisnih funkcionalnosti, kar omogoča tudi vzporedni razvoj. Po končanem posameznem sklopu se ga preda naročniku v uporabo in naprej razvija do končne različice. Vrstni red razvoja je določen tako, da se najprej razvije najpomembnejše dele oziroma tiste, ki so ključni za uspešen razvoj. [21]



Slika 2: Skica inkrementalnega razvoja

2.3.2 Modeli evolucijskih metodologij

2.3.2.1 Prototipni model

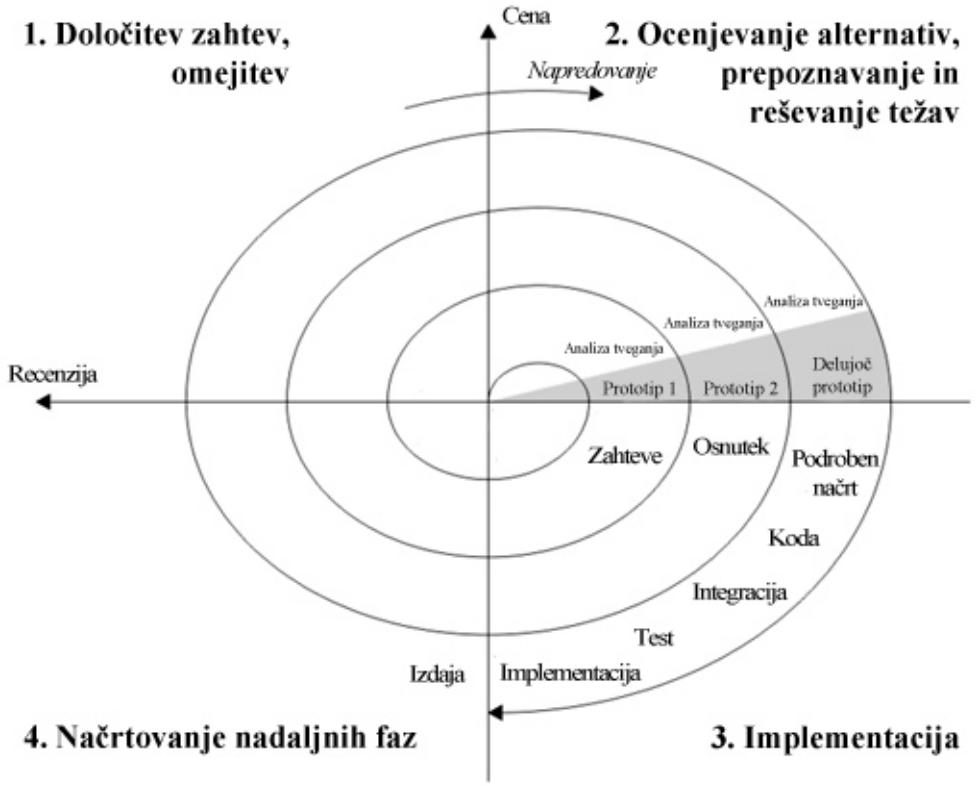
Bistvo prototipnega razvoja temelji na izdelavi prototipov, ki so nepopolno izdelane verzije z osnovnimi funkcionalnostmi sistema. To omogoča lažje komuniciranje med naročnikom in razvijalci ter najbolj pride do izraza, če zahteve niso natančno določene oziroma se razvija nov sistem, ki bo podprl proces, ki se šele uvaja. Naročnik ponavadi ni tehnično podkovan in prototipne faze so še daleč od končnega izdelka, ne ustrezajo varnostnim in zmogljivostnim zahtevam, vendar pa se na podlagi prototipa lažje in natančneje opredeli zahteve za končni produkt. [27]

Pri tem modelu ločimo dve vrsti prototipnega razvoja:

- evolucijski prototip, kjer s posameznimi fazami izboljšujemo prototip, dokler ne pokrijemo vseh funkcionalnosti, in
- prototip, ki ga po uporabi zavržemo, ker služi le kot predstavitev določenih funkcionalnosti ali za pomoč simulacije ključnega dela sistema.

2.3.2.2 Spiralni model

Spiralni model naj bi povzel najboljše lastnosti zaporednega razvoja in prototipnega razvoja. Sestavljen je iz več ciklov, ki vsebujejo vse glavne faze. Začetni cikli so prototipno usmerjeni, medtem ko so poznejši cikli namenjeni nadgrajevanju sistema in odpravljanju napak ter vzdrževanju. Če naročnikove zahteve niso dovolj jasno določene, naredimo več krogov po spirali in se z vsakim obhodom bližamo končni rešitvi. Spiralni razvoj se lahko uporablja tudi pri modularnem razvoju. [28]



Slika 3: Spiralni model

2.3.2.3 Ciklični in inkrementalni model

Ciklični in inkrementalni model je protiutež slapovnemu, ki nasprotuje zaporednemu načinu razvoja. Razvojni projekt se razdeli na več iteracij, ki so lahko različnih dolžin, in vsaka iteracija pripelje do rezultata z vso potrebno dokumentacijo. Vsak cikel je neodvisen model slapa, z analizo, načrtovanjem, kodiranjem in testiranjem. Na ta način se spremembe obravnavajo bolj učinkovito, saj se pri analizi obravnavajo le deli zahtev, ki se bodo izvedli v tekočem ciklu.

2.3.3 Agilne metodologije

Ker se razvoj programske opreme spreminja in ker so se tradicionalne metodologije izkazale za preveč okorne, se je pojavila potreba po novih, učinkovitejših metodologijah razvoja. Poglavitni razlog je bil potreba po učinkovitejšem obravnavanju sprememb med razvojem, želja po rezultatih v dogovorjenih rokih, čim bolj natančnem sledenju željam ter inovativnih rešitvah. Zaradi tega se je v devetdesetih letih prejšnjega stoletja razvilo agilno gibanje (angl. agile alliance), ki se bori za nov in bolj prožen način razvoja. Tipičen in najbolj poznan predstavnik agilnih metodologij je ekstremno programiranje, katerega oče je Kent Beck [4]. Podrobnejši opis agilnih metodologij in značilnih modelov sledi v poglavju Agilne metodologije.

2.4 Namen in koristi metodologije

Pri razvoju programske opreme metodologija zagotavlja kvaliteten zajem naročnikovih zahtev, kvalitetno komunikacijo z naročnikom in med razvijalci ter zagotavlja pregled nad stanjem projekta.

Po Cockburnu [6] se koristi metodologije pri razvoju programske opreme opazijo pri:

- **Vpeljavi novih ljudi v projekt**
  Če je vpeljan sistem, iz katerega je lahko razvidno, kako je treba delati in kam spadajo posamezniki znotraj hierarhije, se podjetja lažje in kvalitetneje vključijo v razvoj.
- **Zamenjavi ljudi v projektu**
  Izvajalca se lahko hitreje in lažje zamenja z novim.
- **Določanju delovnih obveznosti**
  Treba je določiti natančne delovne obveznosti, da se ve, za kaj je kdo odgovoren, s čimer se izognemo prelaganju odgovornosti na druge. Posamezne naloge je treba zaupati ljudem, ki so za to najbolj kvalificirani.
- **Ustvarjanju dobrega vtisa pri naročniku**
  Uporaba priznane metodologije poveča zaupanje naročnika v uspešnost projekta.
- **Pregledu stanja projekta**
  Uspešno vodenje projekta je mogoče samo ob stalnem pregledu nad opravljenim delom in s še neizvedenimi nalogami. S temi podatki se tudi naročniku omogoča stalen pregled nad napredkom projekta.
- **Določanju delovnih mest pri projektu in izobraževanju izvajalcev**
  Metodologija določa uporabo znanih tehnik in standardov razvoja, ki so potrjeni in preizkušeni. Zaposlene se na podlagi zadolžitev pošlje na določena izobraževanja, s pomočjo katerih delo napreduje hitreje in kvalitetneje.

2.5 Vrednotenje metodologije

Pred uporabo določene metodologije se je treba prepričati, ali bo dodana vrednot za projekt, na katerem jo želimo uporabiti, dovolj velika oziroma najboljša. Metodologija, ki bi bila primerna za vse tipe projektov namreč ne obstaja. Zaradi tega je treba metodologijo ovrednotiti glede na potrebe projekta. Cockburn [6] v obzir vzame sledeča vprašanja:

- Kako hitro nam omogoča izročitev izvajalcev in njihovo morebitno zamenjavo?
- Kako velik vpliv ima na razvoj in prodajo?
- Kakšno svobodo daje razvijalcem in kje jih omejuje?
- Kako hitre prilagoditve omogoča glede na nepredvidene spremembe?

- Kako dobro bo varovala organizacijo pred morebitnim neuspehom oziroma nepredvidenimi težavami?

Zgoraj našteta vprašanja so le osnova. Podrobnejša vprašanja je treba sestaviti glede na tip projekta in vizijo ter glede na to, kaj želimo s pomočjo izbrane metodologije doseči. Metodologije so narejene z namenom pokriti obširnejše število različnih tipov projektov, zato vsak projekt zahteva prilagoditev. Po Cockburnu [6] je treba pogledati predvsem:

- Kje bi bilo bolje uporabljati poglobljeno komunikacijo in kje bi poenostavljena komunikacija delovala enako ali bolje?
- Kje so ozka grla uporabe metodologije na projektu, ki jih je treba natančneje spremljati in jih poskusiti zmanjšati z razporeditvijo drugih področij?
- Kje bi se dalo zmanjšati negotovost projekta in metodologije?
- Kje bi se dalo pohitriti komunikacijo med razvijalci, z naročnikom ter razviti hiter sistem povratnih informacij?
- Kje bi se dalo kombinirati principe drugih metodologij za uspešnejši razvoj?
- Kje bi neupoštevanje metodologije povzročilo neprijetne posledice in kaj bi bilo treba narediti, da do tega ne pride?
- Kje bi se dalo zmanjšati nepotrebne notranja opravke znotraj razvoja projekta?

2.6 Osnovni pojmi metodologije

Cockburn [6] metodologijo deli na trinajst osnovnih pojmov.

- **Vloge,** ki jih prevzamejo in opravljajo zaposleni, določajo njihovo delo na projektu. Razvoj poteka znotraj skupine in v tem sklopu znotraj skupine nujno imeti različne vloge. Določitev zaposlenega glede na vlogo je treba oceniti glede na njegovo osebnost in strokovno znanje. Vsaka vloga ima uveljavljene postopke, po katerih zaposleni deluje. Vloga mora vsebovati aktivnosti, ki so koristne za projekt, sicer se jo iz projekta odstrani.
- **Strokovno znanje** je eden od osnovnih kriterijev za opravljanje določene vloge. Le usposobljena oseba lahko razvija programsko opremo znotraj določenega orodja. Minimalno potrebno znanje pri razvoju programske opreme je poznavanje programskega jezika in razvojnega okolja. Bolj specifična znanja se lahko pridobi tudi skozi specifična izobraževanja.
- **Skupina** je združba ljudi z različnimi vlogami, ki deluje na projektu. Na projektu lahko deluje tudi več skupin, kar je odvisno od velikosti projekta (skupina za razvoj, testiranje, implementacijo).

- **Postopki** navadno predstavljajo središče metodologije. Določajo kaj je potrebno za doseganje cilja. Postopki se navezujejo na osebe, vloge, skupine, način dela in izdelek.
- **Aktivnosti** predstavljajo dela, ki jih zaposleni izvajajo na projektu. Navadno so določene z vlogo na projektu. Vsebina aktivnosti je določena z zaporedjem postopkov in omejena s časovnimi roki.
- **Proces** je točno določeno zaporedje izvajanja aktivnosti, ki so omejene na različne načine (časovno, prostorsko, finančno, itd.).
- **Izdelek** je to, kar se s pomočjo metodologije poskusi ustvariti, in je tudi končni cilj. Gradnja poteka z aktivnostmi in postopki, nanj pa vplivajo kakovost izdelave (natančno izdelave) in uporabljeni standardi (način izdelave).
- **Mejniki** so dogodki, ki označujejo raven napredka oziroma konec določene aktivnosti. Nekateri mejniki so le dejstva, drugi vključujejo dejanje. Stanje projekta se določa z mejniki, saj nedoseženi mejniki kažejo na to, kaj je v okviru projekta še treba narediti.
- **Standardi** so dogovori razvojne skupine, kako se bo določeno orodje uporabljalo. Standard vpliva na delovanje in izgled izdelka in je lahko tudi širše sprejet kot le dogovor med razvijalci.
- **Kakovost** določa natančnost izdelave in natančnost pri opravljanju aktivnosti. Pogosto se izraža s številom skritih napak v programu in preprostostjo vzdrževanja. Kakovost se določa na podlagi rezultatov meritev.
- **Skupinske vrednote** so skupek splošno sprejetih vrednost razvojne skupine, ki pomembno vplivajo na vse elemente metodologije. Vrednote so od skupine do skupine različne.
- **Osebnost** predstavlja vzorec delovanja posameznika in je skupek prirojenega in priučenega. Ujemanje vloge in osebnosti omogoča boljše delovne uspehe posameznika.
- **Orodja** so katerikoli pripomoček, ki se uporablja pri razvoju. Običajno vsako orodje zahteva neko strokovno usposobljenost. Nekatere metodologije predlagajo uporabo čim bolj preprostih orodij, druge pa zagovarjajo najnovejša orodja, ki naj bi omogočala čim hitrejše delo.

2.7 Sestava metodologije

Vsem metodologijam so skupni nekateri osnovni sestavni deli [6]. Najpomembnejši so:

- Zajem zahtev in želja naročnika – zabeleži se želje uporabnikov
- Analiza – preuči se smiselnost, potrebne tehnologije in časovne okvire razvoja
- Načrtovanje – določi se vloge, aktivnosti, mejnike, stroške razvoja, …

- Razvoj – dejansko delo na projektu
- Preverjanje projekta s strani naročnika – predstavitev izdelka naročniku in njegovo mnenje
- Testiranje – preverjanje pravilnosti in stabilnosti izdelka
- Vzdrževanje – odprava napak med delovanjem izdelka že v uporabi

Bistvu metodologije je, kako so sestavni deli med seboj povezani in kako so natančneje definirani. Pri metodologijah, ki uporabljajo iteracije, je mogoče sestavne dele razbiti na manjše enote in jih večkrat ponoviti.

Glede na to, da so vse metodologije več ali manj sestavljene iz istih sestavnih delov je mogoče sklepati, da je vsaka metodologija sposobna usmerjati projekt do zaključka. Razlika med metodologijami pa se pojavlja v prilagojenosti metodologije posameznemu primeru razvoja in v hitrosti razvoja. Iz tega sledi tudi, da je najpomembnejše za uspešnost projekta izbrati primerno metodologijo.

2.8 Posledice uvedbe metodologije v podjetje

Uvedba metodologije v podjetje je pomemben del načrtovanja dela podjetja. Treba je preučiti ves izbor metodologij in glede na značilnosti projekta izbrati najprimernejšo.

Glavno težavo pri uvedbi nove metodologije v podjetje predstavlja družbeni vpliv na posameznika, v primeru, da razvojna skupina v metodologiji ne vidi pozitivnega učinka na delo. Pogost primer je, da nekatere postopke smatrajo za odvečne. Prav zaradi tega je treba pri uvajanju nove metodologije v delo zaposlene podrobno seznaniti o koristih in dodani vrednosti, ki jo metodologija prinese podjetju. [10]

2.9 Vzdrževanje metodologije

Metodologijo, ki jo uvedemo v podjetje, je treba tudi vzdrževati, kar zahteva določitev skrbnika metodologije, ki skrbi za aktivnosti povezane z uporabo metodologije. Poznati mora podrobno strukturo, postopke, aktivnosti in tudi posamezne procese podjetja. Skrbeti mora, da se na začetku projekta določi primernost metodologije ter se jo ustrezno prilagodi projektu in podjetju. Med potekom projekta je skrbnik zadolžen za to, da se vsi vključeni v projekt držijo principov in da jim je na razpolago pri morebitnih težavah. Po zaključku projekta je zadolžen za izdelavo analize, s pomočjo katere lahko dopolni in popravi postopke metodologije, če je to potrebno in smiselno [7].

Vloge skrbnika metodologije so:

- dopolnjevanje metodologije,
- usklajenost metodologije,
- izdelavo podrazličic glede na projekt,
- popolno poznavanje strukture, postopkov in aktivnosti metodologije.

3 Agilne metodologije

Agilni pristop do razvoja programske opreme predstavlja preizkus definicije procesa programske opreme, ki se razlikuje od tradicionalnih metodologij, temelječih na podrobni in obsežni dokumentaciji.

Leta 2001 se je v Združenih državah Amerike skupina strokovnjakov s področja lahkih metodologij sestala z namenom ugotoviti, kaj imajo njihove metodologije skupnega in na podlagi teh ugotovitev poenotiti osnove metodologij. Nastal je Agilni manifest (angl. Agile Manifesto) [13], listina za agilni razvoj programske opreme, ki določa temeljna načela in priporočila agilnih metodologij.

3.1 Uporaba agilnih metodologij

Highsmith [7] trdi, da čeprav se agilni pristop lahko uporabi na kateremkoli problemu, se najbolje odziva pri projektih, kjer so težave, ki nastajajo, hitre, spremenljive in nestanovitne. Agilne metodologije predvsem pridobijo na učinkovitosti, če se nivo sprememb poveča zaradi nove tehnologije, spremembe poslovnega modela ali pa je prisotna potreba po čimprejšnjem izstavitvi projekta.

3.2 Glavna načela agilnih metodologij

Namen raziskovanja metodologij je ponovno vzpostaviti zaupanje v uporabo metodologij, zato je pri agilnih metodologijah podprto modeliranje, vendar ne za ceno prevelikega števila diagramov. Agilni manifest podpira tudi dokumentiranje, vendar le v mejah uporabnosti. Tudi pri planiranju so opozorili na njegov obseg glede na spremenljivost okolja, v katerem se razvija. Agilni manifest po Becku [4] poudarja:

- ljudi bolj kot samo proces in orodja razvoja,
- delujoč program bolj kot popolno dokumentacijo,
- sodelovanje z naročnikom bolj kot pogajanja na osnovi pogodb in
- prožnost pri spremembah bolj kot slepo sledenje začrtanemu planu.

Glavna načela agilnega pristopa sledijo v naslednjih razdelkih. [1]

3.2.1 Posamezniki in medsebojno sodelovanje proti procesu in orodjem

Poudarek je na komunikaciji in povezanosti med razvijalci ter prednosti razvijalca pred procesi in orodji.

S tem se povečuje povezanost razvojne skupine, izboljšuje se delovno okolje, motivacija in občutek pripadnosti. Dober razvojni proces ne more rešiti projekta, če za njim ne stojijo sposobni ljudje. Slab proces pa kljub sposobnim razvijalcem zgreši cilje. Sposoben razvijalec ni samo strokovnjak na področju, temveč ima tudi komunikacijske sposobnosti, sposobnosti skupinskega dela in prilagajanja okolju.

3.2.2 Delujoča programska oprema proti popolni dokumentaciji

Cilj je izdelati preverjeno in delujočo programsko opremo in ne popolne dokumentacije.

Tako se nove verzije programa izdajajo v rednih intervalih, pri programiranju pa se stremi k preprostosti, ki vsebuje tehnično napredne rešitve. Pri preveč dokumentiranih projektih se pojavijo težave, ker tudi samo pisanje dokumentacije ovira razvoj, pojavijo pa se tudi potrebe po usklajevanju dokumentacije in ažuriranju. Proces prenosa znanja se tako izvaja znotraj skupine in prek kode, za dokumentacijo pa se ustvarijo le dokumenti, ki so nujni in uporabni.

3.2.3 Sodelovanje uporabnika proti podlagi pogodb

Prednost pred podrobnimi pogodbami ima komunikacija in povezava med razvijalci in naročniki.

Pogodbe so še vedno osnova, vendar so napisane po načelu, da sta v njih poudarjena sodelovanje in komunikacija med obema stranema, saj uporabnik najbolje ve kaj potrebuje in lahko s kvalitetno komunikacijo to preda izvajalcu, ki tako lažje in v rokih zagotovi, da je končni izdelek po želji naročnika.

3.2.4 Sprejemanje sprememb proti sledenju plana

Razvojna skupina je v primeru sprememb med razvojem projekta pooblaščena za spreminjanje ciljev.

Sledenje planu je dober način za organiziran razvoj, vendar pa ob tehnoloških in poslovnih spremembah, ki so stalnica, ni idealna rešitev. Zato mora biti pogodba napisana tako, da omogoča naknadno spreminjanje ciljev projekta na podlagi komunikacije z naročnikom.

3.2.5 Dvanajst osnovnih principov agilnih metodologij

Poleg štirih osnovnih načel so v agilnem manifestu zasnovali tudi dvanajst principov agilnih metodologij [13]:

1. *Najvišja prioriteta je zadovoljitev naročnika s hitrimi in nenehnimi dobavami uporabne programske opreme*

Pri vseh metodologijah je cilj zadovoljevanje naročnikovih potreb, pri agilnih pa je razlika v tem, da želijo čim prej priskrbeti delujoč program. Vse agilne metodologije temeljijo na cikličnem in inkrementalnem razvoju, na koncu vsakega cikla pa se izdelek preda v uporabo oziroma pregled naročniku, s čimer se je moč izogniti razvoju v napačno smer.

2. *Delujoč program je treba dostaviti pogosto, pri čemer imajo prednost krajši časovni intervali.*

   Ta princip nadgrajuje glavno načelo z določitvijo dolžine razvoja intervala. Bolj priporočljivi so krajši cikli, ki izločijo dolgotrajni razvoj v napačni smeri. Treba si je izbrati le delo, ki ga je razvojna ekipa sposobna realizirati.

3. *Vedno je treba sprejeti spremembe zahtev, čeprav pozno v razvoju. Agilni proces izkorišča spremembe za doseganje poslovne prednosti naročnika.*

   Glavni razlog za uveljavitev agilnih metodologij je upoštevanje sprememb pri razvoju. Pri agilnem razvoju so spremembe stalnica, ki jih je treba sprejeti in se jim ne izogiba ali jim zmanjšuje pomena. Z upoštevanjem sprememb se povečuje uporabnost programa in zadovoljitev naročnika.

4. *Naročniki in razvijalci morajo biti ves čas razvoja projekta v stiku.*

   Bistvo agilnih metodologij je sodelovanje naročnika in razvijalcev. Naročnik je predstavnik uporabnikov, ki pozna njihove zahteve in jih posreduje razvijalcem. Naročnik je tisti, ki določa, kaj je pomembno in kaj ne. Redna komunikacija razjasnjuje razvojne težave in usmeri projekt v pravo smer.

5. *V središču projekta naj bodo motivirani posamezniki, ki jim je treba zagotoviti potrebno okolje in podpora ter jim zaupati, da bodo kakovostno opravili svoje delo.*

   Agilni razvoj temelji na tem, da so razvijalci sposobni in motivirani posamezniki, ki jih organizacija podpre za opravljanje svojega dela. Metodologija prepušča notranjo organizacijo razvoja in s tem krepi motivacijo. Notranja organizacija se samoorganizira tako, da si vloge in delo razdelijo med seboj, kar pomeni, da ima posameznik večkrat možnost delati na področju, ki mu najbolj leži.

6. *Najbolj učinkovit in uspešen način prenosa informacij je neposreden pogovor.*

   To je vzrok, zakaj je pri agilnih metodologijah zelo malo dokumentacije, saj se vsaka težava rešuje s pogovori, ki pa se jih sicer pisno zabeleži in objavi, vendar le kot kratka dejstva o odločitvi. Najpogostejša komunikacija je usklajevanje razvoja med razvijalci, sledi pa ji komunikacija med naročnikom in razvijalci.

7. *Delujoč program je najpomembnejša mera napredka.*

   Agilne metodologije podpirajo hiter razvoj delujočega programa z največ mesec dni dolgimi intervali. Tako lahko razvijalci in naročniki v vsakem trenutku ocenijo opravljeno delo, s čimer ocenijo tudi, v kolikem času se bo lahko projekt zaključilo.

8. *Podpora enakomernega in stalnega razvojnega procesa. Naročniki, razvijalci in uporabniki morajo ves čas razvoja vzdrževati predviden tempo dela.*
   Agilne metodologije ne podpirajo opravljanja nadur. Predvideva se normalen delovni urnik in se nadure obravnava kot napako v planiranju oziroma nedelavnost razvijalcev. Namesto nadur se poslužuje dodajanja novih razvijalcev, realnejših ocen težavnosti in dolžin razvoja projekta ali prestavitev manj pomembnejših zahtev na kasnejši čas. Tako se ohranja konstanten tempo razvoja.
9. *Zagotoviti je treba ustrezno pozornost tehničnih odličnosti skupaj z dobrim načrtom.*
   Cilj agilnih metodologij je zadovoljiti naročnikovo željo po učinkovitem izdelku, zato se razvojne skupine ne morejo zadovoljiti z delovanjem programa, pač pa se morajo prepričati o pravilnem delovanju, ki izpolnjuje naročnikove želje. Vedno se teži k uporabi najnovejših tehnologij za dosežek večje funkcionalnosti izdelka.
10. *Ključnega pomena je spretnost, ki bi jo lahko definirali kot spretnost izločanja dela, ki ga ni treba opraviti.*
    Agilne metodologije delujejo po principu od najpomembnejšega k manj pomembnemu. Pred začetkom razvoja se tako najprej določi prioritetni seznam sklopov in se najpomembnejše izdela čim prej, nekatere manj pomembne pa se lahko tudi odstrani, če se med razvojem ugotovi, da se nepotrebni.
11. *Najboljša arhitektura, zahteve in načrti so rezultat samoorganizacije skupine, ki razvija izdelek.*
    Agilne metodologije uvajajo samoorganizacijo namesto strogega nadzora. Delo poteka na nivoju skupine. Vodstvo le zadolži skupino za izvedbo sklopa, način izvedbe pa je domena skupine. Vodstvo sodeluje le kot sestavljavec skupine in ima svetovalno vlogo. Ta princip potrebuje reorganizacijo, predvsem pa spremembo mišljenja vodstvenega kadra, ki mu ostane le svetovalna vloga.
12. *Skupina mora vedno (v ustreznih časovnih okvirih) pogledati možnost za bolj učinkovito delo in po potrebi sprejeti ustrezne sklepe.*
    Pri agilnem razvoju se po vsaki iteraciji izvaja pregled opravljenega dela in odločitev ter kako so odločitve in delo vplivali na razvoj in izboljšanje procesa. V to morajo biti vključeni in se glede sprememb strinjati vsi vpleteni, saj se tako še bolj krepi pomen samoorganizacije in motivacije razvojne skupine.

3.3 Pregled nekaterih agilnih metodologij

Najbolj znane agilne metodologije:

- Agilne tehnike podatkovnih baz (Agile Database Techniques)
- Agilno modeliranje (Agile Modeling)

- Družina metodologij Crystal
- Funkcijsko voden razvoj (Feature Driven Development)
- Dinamična metoda za razvoj sistemov (Dynamic Systems Development Methods)
- Prilagodljiv razvoj programske opreme (Adaptive Software Development)
- Vitek razvoj programske opreme (Lean Software Development)
- Scrum
- Ekstremno programiranje (extreme programming – XP)

3.3.1 Agilne tehnike podatkovnih baz

To so agilne metodologije oziroma skupek strategij, ki jih lahko uporabi skupina razvijalcev v primeru razvoja na podatkovnem delu programske opreme.

Poudarek je na postopni izgradnji podatkovne baze, sama baza pa se potem širi skupaj z razvojem komponent znotraj projekta, kar je nasprotno od klasičnih metodologij [2].

Nekaj principov:

- Podatek je eden od pomembnejših vidikov programske opreme
- Razvojna skupina upošteva vizijo podjetja
- Razvojne skupine se med sabo podpirajo
- Vsak razvojni projekt je edinstven in potrebuje poseben pristop
- Težave se rešujejo z aktivnim sodelovanjem vseh udeležencev na projektu
- Izogibanje skrajnostim pri iskanju rešitev

Tipične vloge so razvijalec (razvija in skrbi za projekt), agilni administrator podatkovne baze (administracija podatkov, razvoj podatkovnih sistemov, prenos, preverjanje), administrator projekta (dokumentira, razvija in skrbi za skupne dele projekta) in arhitekt projekta (aktivno skrbi za razvoj).

3.3.2 Agilno modeliranje

Agilno modeliranje vsebuje načela, principe in izkušnje za modeliranje in dokumentacijo programskih sistemov, za implementacijo pri razvoju na bolj fleksibilen način kot klasične metodologije. Uporablja se predvsem za modeliranje zahtev, analiz, arhitekture in za pripravo načrta izvedbe. Agilno modeliranje ni vnaprej predpisan proces, nima točno določenih navodil, ampak ponuja nasvete za učinkovitejšo izvedbo [14].

Cilji agilnega modeliranja so:

- definicija in prikaz vrednot, principov in izkušenj učinkovitega in enostavnega modeliranja,

- prikaz uporabe modeliranja pri projektih, na katerih se uporabljajo agilne metodologije razvoja (XP, Scrum),
- vpeljava vrednot, principov in izkušenj za izboljšanje modeliranja projekta znotraj drugih modelirnih procesov (RUP).

Ker agilno modeliranje ne opisuje celotnega procesa razvoja pač pa le modeliranje in dokumentiranje, je potrebna uporaba drugih metodologij, ki pokrivajo ostale segmente razvoja. Vrednote agilnega modeliranja so komunikacija, preprostost, povratne informacije, pogum in skromnost. Med principe štejemo poudarjanje enostavnosti, sprejemanje sprememb, hitro vračanje povratnih informacij, modeliranje z razlogom, uporaba večjega števila modelov za učinkovitejše delo, brisanje nepotrebnih elementov, pomen vsebine pred predstavitvijo, odprto in odkrito komuniciranje, poudarjanje kakovostnega dela in lokalno prilagajanje principov zahtevam okolja.

Izkušnje iz agilnega modeliranja: vzpostavitev večjega števila vzporednih modelov, uporaba ustreznih orodij, modeliranje v obliki prirastkov, pogosto dokazovanje pravilnosti s kodo, aktivno sodelovanje vseh zainteresiranih ljudi, ustvarjanje preprostih vsebin, preprosto opisovanje modelov, uporaba najpreprostejših orodij, uporaba standardov in skupinsko modeliranje.

Agilno modeliranje je mogoče uporabiti pri kateremkoli procesu razvoja programske opreme, ne predpisuje stvari, samo svetuje, kaj naj se upošteva. [14]

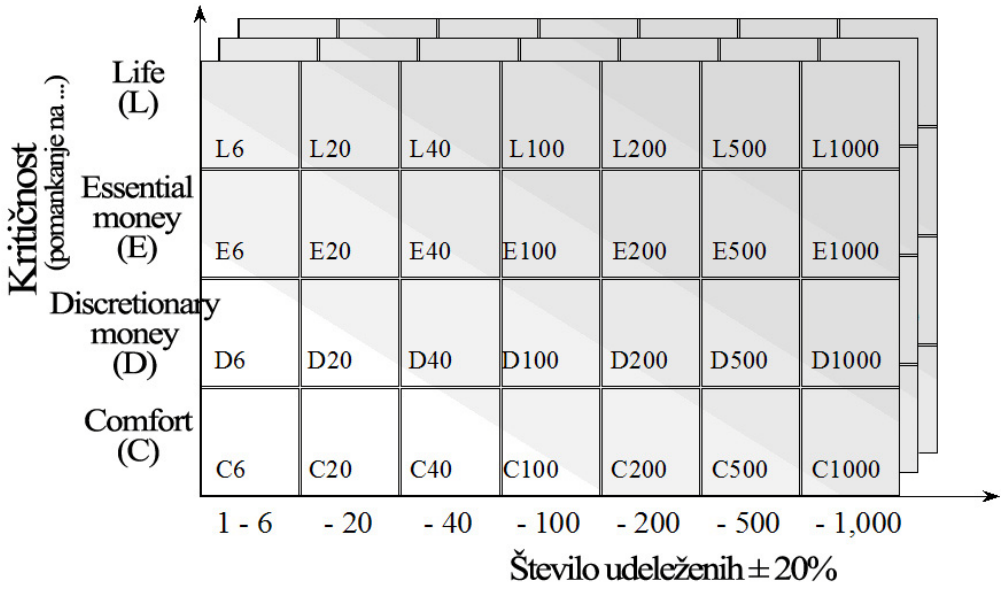
3.3.3 Družina metodologij Crystal

Avtor družine Crystal je Alistair Cockburn, znan tudi kot 'arhitekt metodologij'. Družina metodologij Crystal vključuje več različnih metodologij, ki jih lahko izberemo glede na značilnosti projekta (vse veljajo za agilne). Poleg samih metodologij so v družino vključeni tudi principi za prilagoditev potrebam projekta.

Vsaka metodologija družine je označena z drugo barvo, po kateri se tudi imenuje in predstavlja težavnost. Večji projekti potrebujejo težjo metodologijo in več koordinacije, natančnosti ter strožjo izvedbo. [6]

Drugi kriterij, po katerem se ločijo metodologije znotraj družine Crystal, je kritičnost. Meri se po tem, kaj pomeni izguba podatkov; C – pomeni izgubo udobnosti (comfort), D – izgubo denarja (discretionary), E – izgubo pomembnega dela denarja (essential money) in L – izgubo življenja (life).

Poleg delitve na težavnost in kritičnost se metodologije razlikujejo še po številu ljudi na projektu.



Slika 4: Graf kriterijev za ločitev metodologij znotraj družine Crystal

Glavne značilnosti:

- Osredotočenost na človeka – doseganje ciljev s pomočjo boljšega počutja ljudi na projektu
- Zmanjšanje nepotrebnega pisanja
- Prilagodljivost – zamenjavo dela metodologije s primernim

Temelji na naslednji lastnostih:

- Vsak projekt se vsaj delno razlikuje od drugih, zato rabi sebi prilagojeno metodologijo
- Delo na projektu je odvisno od ljudi in se spreminja glede na znanje in zmožnosti sodelovanja skupine
- Boljša komunikacija zmanjšuje potrebe po vmesnih verzijah programa

3.3.3.1 Crystal Clear metodologija

Crystal Clear metodologija je član družine Crystal in je primer agilne oziroma lahke metodologije.

Crystal Clear je aplicirana v skupine od šest do osem ljudi, ki delajo na nekritičnih sistemih. Ta metodologija je usmerjena k ljudem in ne k procesom oziroma izdelkom. [18]

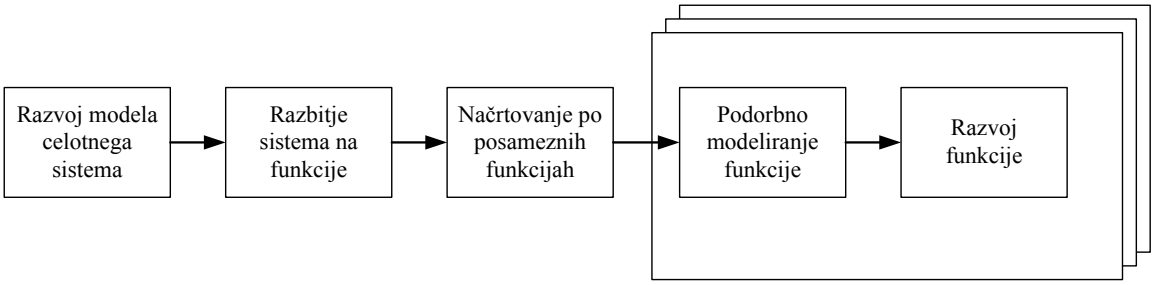
Njeno gonilo so naslednje značilnosti:

- Pogostost izdaje kode v uporabo
- Pregled in popravljanje napak
- Dobra komunikacija skupini, znotraj istega prostora
- Osebna varnost

- Fokus
- Lahko dostopno za ekspertne uporabnike
- Avtomatizirani testi, pogosta integracija, upravljanje sestave (Configuration Management)

3.3.4 Funkcijsko voden razvoj

Funkcijsko voden razvoj (angl. feature driven development - FDD), katerega avtor je Jeff de Luca, predvideva cikličen razvoj izdelka z najučinkovitejšo prakso iz industrije. Ne podpira celotnega razvoja, ampak je usmerjen predvsem v faze načrtovanja in gradnje sistema, za ostalo pa dopušča uporabo drugih metodologij. Skozi cel proces se poudarja kakovosti, hitre in jasne rešitve. Objektni model in plan razvoja se spreminja med razvojem, vendar to ni natančno poudarjeno. Metodologija je sestavljena iz petih zaporednih procesov, v katerih se načrtuje in razvija. Iterativni del podpira agilni razvoj, prilagojen na pozne spremembe in poslovne zahteve. Iteracija traja od enega do treh tednov. [19]



Slika 5: Procesi funkcijsko vodenega razvoja

- *Razvoj modela celotnega sistema* – iz zahtev se določi vsebino razvoja
- *Razbitje sistema na funkcije* – določa se jih iz vsebine razvoja in želja naročnika
- *Načrtovanje po posameznih funkcijah* – določitev prioritete in dodelitev razvojnim skupinam
- *Podrobno modeliranje funkcije* – podroben pregled posebnosti posamezne funkcije
- *Razvoj funkcije* – programiranje posamezne funkcije

Za funkcijsko voden razvoj velja osem principov:

- Objektni model domene – vsebuje osnovne značilnost o funkcijah, izdela se ga na začetku izvajanja projekta in dopolnjuje po vsakem ciklu
- Razvoj po funkcijah – ves razvoj je podrejen eni funkciji naenkrat (majhna funkcionalnost, ki je za uporabnika pomembna)
- Lastništvo kode – obstaja le en lastnik objekta/kode. Ta oseba je tudi vzdrževalec tega objekta.

- Razvojna skupina funkcij – sestavljena iz glavnega razvijalca in lastnikov kode
- Nadziranje – poskrbi, da se upošteva dogovorjene standarde in smernice
- Redno izdajanje nadgradenj – časovni okvir, v katerem se uporabi novo napisana koda
- Vodstvena politika – ni natančno določena, določuje le zagotovljen nadzor
- Pregled stanja projekta – pregled na ravni funkcij (nedokončana funkcija ima vrednost 0, dokončana pa vrednost 1). Vsota vseh funkcij deljeno s seštevkom vseh funkcij pove stopnjo dokončanosti projekta

3.3.5 Vitek razvoj programske opreme

Ideja vitkega razvoja programske opreme je prenesena iz 'vitkega' razmišljanja iz drugih industrijskih panog, predvsem iz japonske avtomobilske industrije pri Toyoti. To v bistvu ni metodologija, ampak skupek navodil, kako zagotoviti učinkovito in kakovostno izdelavo. Drži se deset glavnih principov, ki se lahko z manjšimi popravki uporabijo v kateremkoli tipu industrije [22]:

- Zmanjševanje odpadkov
- Zmanjševanje obsega skladiščenja
- Povečevanje toka
- Delo na zahtevo
- Prenos odločanja na delavce
- Doseganje zahtev naročnika
- Princip 'naredi pravilno v prvem poizkusu'
- Odpravljanje lokalne optimizacije
- Vzdrževanje partnerskega odnosa z dobavitelji
- Ustvarjanje ozračja stalnega izboljševanja

3.4 Pregled agilnih oblik v uporabi pri različnih metodologijah

Agilne oblike razvoja programske opreme, ki pa niso cele metodologije, pač pa dobre prakse, ki se uporabljajo v različnih metodologijah. [11]

- **Testno voden razvoj** (angl. test driven development – TDD) je agilna oblika razvoja programske opreme, ki temelji na kratkih ponavljajočih se ciklih. Razvijalec najprej napiše avtomatiziran test, ki določa novo funkcijo, potem pa okrog tega še ustrezen program.

- **Programiranje v paru** je tehnika agilnega razvoja, kjer dva programerja delata za istim računalnikom. Medtem ko eden piše kodo, jo drugi sproti pregleduje. Vlogi se večkrat dnevno zamenjata. Med pregledovanjem kode drugi programer načrtuje tudi nadaljnjo strategijo programiranja in poskusi iskati načine izboljšanja, medtem ko je prvi osredotočen na programiranje.

- **Planning poker** oziroma načrtovanje pokra je zasnovano na soglasju pri tehniki ocenjevanja, predvsem uporabljeno za ocenjevanje truda ali relativne zahtevnosti nalog.

- **Kontinuirana integracija** je način zgodnje in pogoste integracije, da se izognemo zapletom. Cilj take integracije je zmanjšati obseg dela, stroškov in časa. Pri tem se stremi k čim večji avtomatizaciji integracije.

- **RITE oziroma hitro iterativno testiranje in vrednotenje** (angl. rapid iterative testing and evaluation) je metoda, ki jo je razvil pri Microsoftu. Za razliko od testiranja uporabe programske opreme so podatki analizirani po vsaki uporabi ali po koncu cikla testiranja. Spremembe se uvede čim se odkrije napako in je rešitev jasna. Potem se popravljeno obliko preda v nadaljnje testiranje. [24]

- **Vedenjsko voden razvoj** (angl. behavior driven development – BDD) je tehnika pri agilnem razvoju programske opreme, ki spodbuja sodelovanje med razvijalci, netehničnimi oziroma poslovnimi udeleženci pri razvoju. Ta tehnika je bila razvita kot odgovor na TDD in se je skozi leta še nadalje razvila. BDD se osredotoči na to, da udeleženci uporabljajo laičen jezik za opis zahtev, kar pomaga razvijalcem, da se osredotočijo na kodo in ne tehnične podrobnosti. To pomaga tudi pri boljši komunikaciji vseh udeležencev. [16]

4 Ekstremno programiranje

Ekstremno programiranje je po besedah Kenta Becka [4] lahek, učinkovit, prilagodljiv, napovedljiv, znanstven in zabaven način razvoja programske opreme brez tveganja.

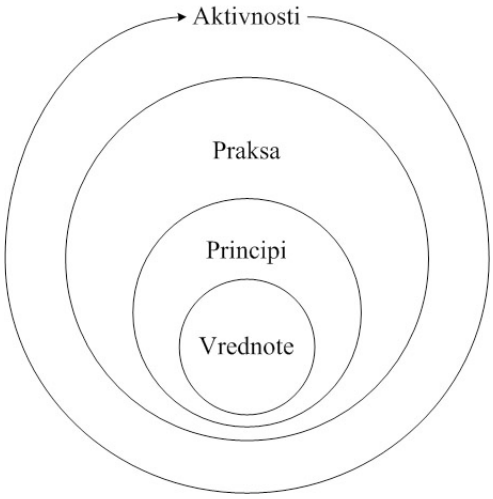
Od drugih metodologij se razlikuje po:

- ustvarjanju hitrih rezultatov, stvarnem in stalnem toku povratnih informacij iz kratkih razvojnih ciklov,
- po svojem inkrementalnem pristopu, ki omogoča hiter razvoj delujočega ogrodja sistema, ki je podlaga za nadgradnjo skozi celotni razvojni cikel projekta,
- po svoji sposobnosti agilnega razvoja funkcionalnosti glede na spreminjajoče se zahteve naročnikov,
- po zanašanju na avtomatizirane teste, narejene s strani razvijalcev in uporabnikov za nadzorovanje napredka razvoja sistema in dovolj hitro odkrivanje napak,
- po dejstvu, da se zanaša na ustno komunikacijo, teste in izvorno kodo pri prenosu strukture sistema in njegovega namena,
- po zanašanju na evolucijski razvojni proces, ki traja toliko časa kot projekt,
- po zanašanju na učinkovito sodelovanje razvijalcev z običajnimi sposobnostmi in
- po zanašanju na dobrih praksah, ki delujejo tako s kratkotrajnimi instinkti razvijalcev kot z dolgoročnimi interesi projekta.

Ekstremno programiranje je organizirano tako, da najbolje deluje na projektih, ki se jih razvija v sklopu skupine velikosti od dveh do največ desetih razvijalcev, ki niso omejeni na določeno razvojno okolje. Pogoj je, da to okolje omogoča zagon avtomatskih testov v razmeroma kratkem času. [11]

Ekstremno programiranje je v bistvu skupek konservativnih in znanih starejših in preizkušenih idej, ki so prvič povezane in uporabljene skupaj, so čim temeljiteje uporabljene in se med seboj čim bolj dopolnjujejo.

Bolj podrobno je ekstremno programiranje, kot ga vidimo na Sliki 6, sestavljeno iz štirih delov, ki so med seboj povezani in delujejo v sinergiji za skupne cilje.



Slika 6: Skica sestave ekstremnega programiranja

Središče ekstremnega programiranja zavzemajo vrednote oziroma glavne usmeritve, ki se morajo držati razvijalci. Širši krog so principi dela, ki so vrednote, razširjene na področje razvoja programske opreme. Nad vrednotami in principi dela so uveljavljene delovne prakse, ki so v bistvu principi dela preneseni na dejansko delo na projektu. Delovne prakse tudi določajo način razvoja in so sestavljene iz aktivnosti, ki zajemajo posamezne sklope dela, ki jih razvijalci opravljajo.

4.1 Vrednote ekstremnega programiranja

Glavne vrednote ekstremnega programiranja so [11]:

- komunikacija,
- preprostost,
- povratne informacije in
- pogum.
- spoštovanje

4.1.1 Komunikacija

Komunikacija je najpomembnejša vrednota znotraj ekstremnega programiranja. Nujno je potrebna za razvoj sistema. V formalnih metodologijah se to doseže prek podrobne dokumentacije, ekstremno programiranje pa se lahko predstavi kot tehnika razvoja za hitro izgradnjo sistemov in razpršenega znanja znotraj članov razvojnih skupin. Ekstremno programiranje tudi vsebuje prakse, ki pospešujejo komunikacijo. Ena osnovnih praks je zahteva po stalni prisotnosti naročnika med razvojem. Tako se poskrbi za stalno komunikacijo

med razvijalci in uporabniki, kar omogoča sprotno reševanje težav. Cilj je, da naročnik in uporabniki vsem razvijalcem predstavi skupinsko videnje sistema. Ekstremno programiranje daje prednost preprostosti, sodelovanju med uporabniki in razvijalci ter pogosto verbalno komunikacijo. Vseskozi poteka tudi komunikacija razvijalcev z vodstvom, kjer se prenašajo informacije o poteku razvoja in težav med razvojem.

Potreben je tudi nadzor nivoja komunikacije in ustrezno ukrepanje v primeru primanjkljaja ali neustrezne komunikacije. [4]

4.1.2 Preprostost

Ekstremno programiranje spodbuja začetek z najpreprostejšo rešitvijo, dodatna funkcionalnost pa se dodaja postopoma pozneje. Razlika med tem in konvencionalnim razvojem je, da se osredotoča na razvoj in kodiranje za trenutne potrebe. Čeprav to lahko pripelje do več dela pri spreminjanju sistema, je utemeljitev metodologije v tem, da je prednost v tem, d a se v trenutnem času ne raziskuje podrobnejših zahtev in izgublja preveč časa na stvareh, ki še niso bistvene. Načrtovanje in razvijanje negotovih prihodnjih zahtev predstavlja tveganje prevelike porabe virov na nečem, kar mogoče sploh ne bo potrebno.

Preprostost je neposredno povezana s komunikacijo, tako da lahko preprostost kodiranja poenostavi komunikacijo in izboljša njeno kakovost. Prednosti, ki jih prinese preprostost, so poleg manj trenutnega dela v preprostejšem in bolj fleksibilnem sistemu, ki je lažje razumljiv s strani vseh razvijalcev v skupini, lažja komunikacija in vključitev v celoto sistema. [4]

4.1.3 Povratne informacije

Pri ekstremnem programiranju so povratne informacije ključne za uspeh projekta. Poznamo več vrst povratnih informacij:

- Povratne informacije od sistema – pridobimo jih s testi modulov ali s periodičnimi testi. Tako imajo razvijalci neposredne povratne informacije o stanju sistema po izvršitvi sprememb
- Povratne informacije od naročnika – funkcijski testi (odobritveni testi), ki so napisani s pomočjo uporabnikov. Prek njih naročnik dobi stvarne podatke o trenutnem stanju sistema. Tak pregled je zaradi lažjega nadzora razvoja s strani naročnika načrtovan vsake dva do tri tedne.
- Povratne informacije od razvojne skupine – ko naročnik zahteva novo funkcionalnost sistema, razvojna skupina sporoči oceno časa potrebnega za izvedbo

Povratne informacije so močno povezane s komunikacijo in preprostostjo. Več povratnih informacij obstaja, lažje je komunicirati. Napake na sistemu se preprosto opišejo prek testov,

ki pokažejo na težavo oziroma napako v kodi. Neposredne povratne informacije od sistema povedo razvijalcem, da je treba določen del ponovno napisati. Prav tako lahko naročnik sistem periodično testira glede na funkcionalne zahteve. [4]

4.1.4 Pogum

Pogum skupaj z ostalimi tremi vrednotami zagotavlja nenehno spremembo sistema na bolje. Pogum pooseblja več praks, med njimi najpomembnejša sta razvoj in načrtovanje za trenutno situacijo. Pogum omogoča razvijalcem, da lahko zavržejo delujočo kodo, ki je preveč zapletena ter sestavijo preprostejšo rešitev. Vse vrednote tako tudi povečujejo samozavest razvijalcev, da se počutijo motivirane in zadovoljne ter verjamejo v nadaljnji razvoj. Pogum pa kljub temu brez vrednot preprostosti, komunikacije in povratnih informacij pripelje razvoj na nivo iskanja rešitev, ne da bi se vedelo, kaj se sploh poskuša rešiti. [4]

4.1.5 Spoštovanje

Peta vrednota, spoštovanje, se odraža na več načinov. Pri ekstremnem programiranju člani skupine spoštujejo drug drugega. Razvijalci si ne smejo privoščiti, da porušijo delujočo kodo oziroma podrejo teste svojih kolegov, saj to pripelje do zamud. Člani skupine spoštujejo delo kolegov in stremijo k višji kvaliteti in najboljšim rešitvam.

Prevzem prvih štirih vrednot pripelje do spoštovanja posameznikov v skupini. Nihče v skupini se ne sme počutiti nepomembnega ali prezrtega. To prinese visoko stopnjo motivacije in spodbuja k zvestobi skupini in ciljem projekta. Tudi ta vrednota je močno odvisna od ostalih in je usmerjena k ljudem. [4]

4.2 Principi

Ekstremno programiranje vsebuje pet glavnih principov dela, ki usmerjajo razvoj projektov [4]:

- Hitre povratne informacije
- Princip enostavnosti
- Inkrement sprememb
- Vključitev sprememb
- Kakovost dela
- Stranski principi

4.2.1 Hitre povratne informacije

Hitre povratne informacije so zelo uporabne in pomembne pri učenju in prenosu znanja na sistem. V nasprotju s klasičnimi metodologijami, je kontaktiranje naročnika pogosteje, zato se odzivi na informacije merijo v dneh ali tednih, in nikakor ne v mesecih.

Zadovoljstvo naročnika poveča tudi jasen vpogled v sistem in hitro upoštevanje sprememb.

4.2.2 Princip enostavnosti

Princip enostavnosti govori o obravnavanju vsakega problema, kot da je rešitev izjemno preprosta. Po ocenah Kenta Becka [4] preprosta rešitev zadošča pri 98 odstotkih težav. Ekstremno programiranje zavrača idejo tradicionalnih metodologij, da je treba predvideti prihodnje potrebe in ponovno uporabiti staro kodo.

4.2.3 Inkrement sprememb

Inkrement sprememb vodi v postopen razvoj aplikacije in je bistvo vseh faz razvoja. Poanta je tudi v tem, da se vse obdeluje v najmanjših in obvladljivih delcih, nikoli v celoti, kar poveča nadzor nad razvojem.

4.2.4 Vključitev sprememb

Bistvu vključitve sprememb je sprejemanje sprememb in prisiliti razvijalce, da rešujejo le najpomembnejše težave. Nenatančno definirane in manj pomembne zahteve se pušča odprte. Spremembe je tako treba ob nastanku sprejeti, obravnavati in določiti prioriteto izvedbe.

4.2.5 Kakovost dela

Kakovost dela je princip, ki je nepogrešljiv in se ga ne sme zanemarjati. Ekstremno programiranje postavlja kakovost kode na prvo mesto in prav tako skrb zanjo s testi in programiranjem v parih.

4.2.6 Stranski principi

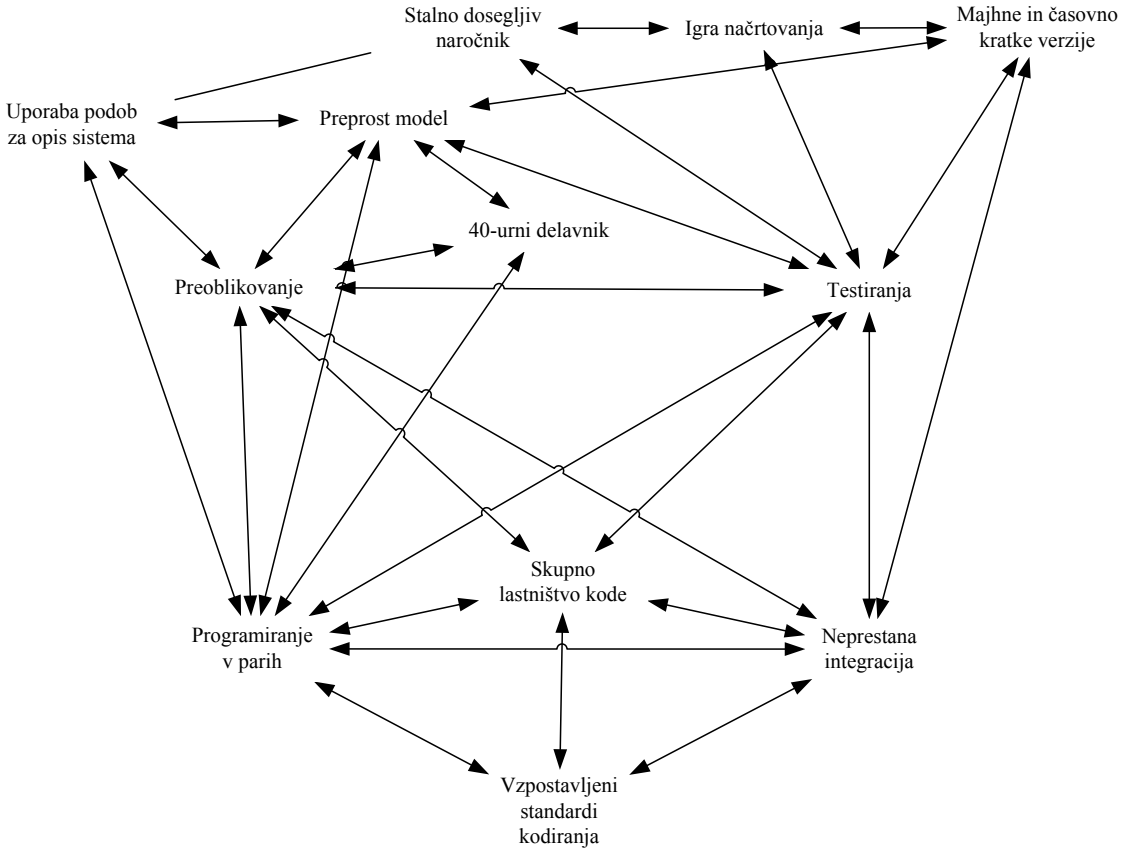
Poleg glavnih principov pri ekstremnem programiranju poznamo še stranske, ki so manj pomembni, vendar imajo kljub temu pomembno vlogo pri učinkovitem delovanju metodologije.

- Učenje razmišljanja
- Uporaba majhnega začetnega vložka
- Igranje na zmago
- Konkretni preizkusi

- Odprta in poštena komunikacija
- Upoštevanje človeških instinktov
- Sprejemanje odgovornosti
- Lokalna prilagoditev pravil
- Čim manj prtljage
- Uporaba pravil meril

4.3 Prakse

Vrednote, principi in aktivnosti predstavljajo osnovo za prakse ekstremnega programiranja. Znanih je dvanajst praks in vsaka oblikuje in določa izvajanje aktivnosti v skladu z vrednotami in principi. Vse te prakse so med seboj povezane na tak način, da je slaba lastnost ene prakse podprta z dobro lastnostjo druge. [4]



Slika 7: Prikaz povezav med praksami ekstremnega programiranja

4.3.1 Igra načrtovanja

Osnovni proces znotraj ekstremnega programiranja se imenuje igra načrtovanja (angl. Planning game). To je v bistvu sestanek, ki se zgodi enkrat na vsako iteracijo (tipično enkrat tedensko). Razdeljen je v tri dele:

- *Raziskovanje* – naročnik razvijalcem predstavi napisane uporabniške zgodbe, ki jih ti časovno in tehnično ocenijo ter po potrebi razdelijo na manjše zgodbe.
- *Sprejetje načrta razvoja* – prioretizacija in izbira najpomembnejših zgodb za razvoj nove verzije sistema. Naročnik glede na trajanje razvoja izberejo zgodbe, ki so potrebne za vključitev v novo verzijo sistema.
- *Vodenje razvoja* – popravljanje načrta glede na stanje med razvojem. Zgodbe iz prvega razvojnega cikla potrebujejo delujoč sistem, ki ga zgodbe v nadaljnjih ciklih dopolnjujejo.

Celotne igre načrtovanja ni mogoče izvesti brez aktivnega sodelovanja med naročniki in razvijalci. Bistvo igre načrtovanja je v tem, da zgodbo o uporabniški izkušnji, ki jo napiše naročnik, razvijalec oceni glede na časovne in tehnične zahteve ter povezanost z ostalimi zgodbami za izvedbo nove verzije sistema. Naročnik nato določi še prioritete zgodbe. [4]

4.3.2 Programiranje v parih

Programiranje v parih je ena najbolj prepoznavnih praks ekstremnega programiranja in tudi prva asociacija. Predstavlja delo dveh razvijalcev na isti nalogi za enim računalnikom. Medtem ko eden od razvijalcev tipka kodo, drugi istočasno preverja pravilnost kode in strateško razmišlja o posledicah, možnostih preoblikovanja, nadaljnjem razvoju, testih in podobnem. Vlogi se po določenem času redno menjati. Pari razvijalcev se formirajo na prostovoljni bazi, na podlagi izkušenj in skupnih interesov. Povezanost parov je lahko kratkotrajna ali stalna. Medtem pa dinamično zamenjevanje parov omogoča prenos komunikacije in znanja med vsemi razvijalci. Če nek par razvije novo funkcionalnost, se ta par razcepi, da jo lahko predstavi še ostalim razvijalcem v novih parih. Bistvo programiranja v parih pa je predvsem komunikacija in skupen razvoj. V primeru, da pride do motenj ali nerazumevanja določenih parov, je te treba odstraniti iz razvojne skupine. [4, 23]

4.3.3 Skupno lastništvo kode

Skupno lastništvo kode je pristop, ko lahko kdorkoli v skupini pregleduje in popravlja kode kateregakoli razvijalca. Pri tem se ne zgodi, da en razvijalec čaka na drugega, ampak se lahko kar odloči in popravi kodo tako, da mu ustreza v okviru pravilnega delovanja.

Skupno lastništvo kode in programiranje v parih ustvarjata okolje, kjer tudi razvijalci, ki niso razvijali določene kode, približno vedo, kako in za kaj se uporablja.

4.3.4 Majhne in časovno kratke verzije

Potrebno je usklajevanje med hitrostjo in vsebinsko bogato izdajo verzije. Cilj je imeti čim krajše verzije, ki vsebujejo vsebinsko čim več pomembnih zahtev. Prednost kratkih verzij je tudi v manjši potrebi po planiranju vnaprej. Izdaja verzij tako sledi na največ dva meseca.

4.3.5 Uporaba podob za opis sistema

Metodologija priporoča uporabo podob za opis sistema in tako odpravlja potrebo po ustvarjanju arhitekture sistema. Prikaz zgradbe s pomočjo kock in povezav je pomembna za preprostejše razumevanje sestave sistema. Tako s pomočjo podob dosežemo prikaz sistema kot nek realen objekt, ki ga vsak dobro pozna. Na ta način je olajšano tudi komuniciranje o projektu, saj lahko uporabljamo kar lastnosti in opise izbrane podobe.

4.3.6 Preprost model

Pri razvoju uporabljamo najenostavnejši model, ki pokriva izbrane zahteve po pričakovanju naročnika, odvečne dele pa odstranimo. S tem vzdržujemo enostavnost in preprostost programske kode ter posledično lažje vzdrževanje.

Štiri zahteve za preprost model po pomembnosti:

- Sistem mora pravilno delovati
- Ne sme biti podvajanja kode
- Sistem naj ima čim manj razredov
- Sistem naj ima čim manj metod

4.3.7 Testiranje

Testiranje je pomemben dejavnik pri razvoju. Ekstremno programiranje predvideva uvedbo medsebojno neodvisnih in avtomatiziranih testov. Tak pristop olajša razvoj, saj so vhodni podatki in pričakovani rezultati točno določeni. V primeru, da pride do nepričakovanega rezultata je treba napisati nov test. Teste pišejo razvijalci in naročniki. Medtem ko razvijalci pišejo teste notranjosti sistema (delovanje metod), naročniki testirajo širšo implementacijo zgodb (delovanje pri določenih vnosih, odzivnost). Razvijalci pišejo teste pred pisanjem kode, ki se prilagaja tako, da zadovolji testu, naročnik pa piše teste za preverjanje implementacije zgodb, ki vsebujejo realne primere.

Testiranje je prva faza pri uvajanju metodologije in jo lahko uporabljamo tudi samostojno.

4.3.8 Preoblikovanje kode

V razvoju pogosto prihaja do sprememb zahtev, kar povzroči spreminjanje kode. Z dopolnjevanjem kode lahko ta postane preobsežna in neberljiva, zato je treba dele kode ponovno napisati. Kodo se ponavadi izboljša tudi tako, da se odstrani odvečne dele. Zaradi uporabe testov je vse, kar je potrebno, to, da nova koda ustreza testom.

4.3.9 Neprestana integracija

Integracija nove delujoče kode se izvaja večkrat dnevno, tako se preverja konflikte z ostalimi razvijalci. Še posebej je to pomembno za novo kodo, ki lahko podre že delujoče funkcionalnosti sistema. Za morebitne napake, ki se jih odkrije, je zadolžen par, ki je te napake povzročil, saj mora vedno zagotoviti popolno delovanje testov. Za to je potrebna tudi uporaba orodij, ki omogočajo hitro integracijo nove kode in poganjanje avtomatiziranih testov. Integracija mora biti preprosta in hitra, s hitrim odgovorom o delovanju sistema.

4.3.10 40-urni delavnik

Metodologija ekstremnega programiranja odsvetuje opravljanje nadur, vsaj ne v dveh zaporednih tednih. Če pride do tega, to pomeni, da obstaja težava pri vodenju projekta in da je prišlo do napačne časovne ocene trajanja projekta oziroma količine razvijalcev na projektu.

Kratkoročno opravljanje nadur naj bi sicer povečalo produktivnost na projektu, vendar pa dolgoročno opravljanje nadur poveča razdraženost ter slabša komunikacijo, to pa povzroči manjšo produktivnost in posledično tudi zmanjša kakovost rešitve. V primeru preutrujenosti se namreč zmanjša učinkovitost in poveča število napak.

4.3.11 Stalna dosegljivost naročnika

Že večkrat sem omenil stalno dosegljivost naročnika v času razvoja, kar pomeni, da je naročnik z razvijalci skupaj v istem prostoru, vendar pa ne nujno zmeraj in skozi celoten proces.

Predstavnika naročnika je treba izbrati med bodočimi uporabniki sistema. Poznati mora celotno področje sistema, njegovo glavno delo pa je sestavljanje uporabniških zgodb, določati prioriteto razvoja, reševati nastale težave, usmerjati razvoj ter pisati in izvajati funkcijske teste. Težave, ki jih odpravlja, so navadno vsebinske in ne tehnične narave.

4.3.12 Vzpostavljeni standardi kodiranja

Treba je postaviti standard kodiranja, nek enoten stil, ki se ga držijo vsi razvijalci. S tem se olajša razumevanje kode in lažji razvoj, preoblikovanje in vzdrževanje.

Vzpostavljeni standardi se lahko nanašajo na stil pisanja, strukturo kode, principe imenovanja, lovljenja napak, komentarje ter na karkoli bi pripomoglo k lažji komunikaciji.

4.4 Aktivnosti

Aktivnosti zajemajo sklope dela, ki jih opravljajo razvijalci na projektu in se vrtijo skozi cikle ekstremnega programiranja [4]:

- Poslušanje – zajema aktivno poslušanje in postavljanje vprašanj. S tem se lahko poslušalec približa bistvu zadeve, da si ustvari predstavo, kako je treba zadevo graditi.
- Preverjanje – med razvojem in tik pred predajo naročniku.
- Kodiranje – aktivna gradnja programa in enolično zapisan potek programa. Vsak razvijalec mora poznati osnove programskega jezika in logike programiranja ter upoštevati standarde kodiranja.
- Načrtovanje – načrtovati pomeni ustvarjati strukturo, ki organizira logiko sistema. Načrtovanje je pomembno za predvidevanje rasti sistema, ki se ne poruši.

4.5 Vloge

Kot pri vsaki metodologiji tudi pri ekstremnem programiranju vloge določajo pomen, pravice in odgovornosti. Vse vloge so podrejene skupnemu cilju in zaradi tega je pomembno, da vse vloge sodelujejo med seboj.

Ekstremno programiranje pozna sledeče vloge [4]:

- Razvijalec – programira, kodira, piše teste, preizkuša tehnologije, sodeluje pri načrtovalnih sestankih.
- Naročnik – piše uporabniške zgodbe in odloča o smeri razvoja.
- Preizkuševalec – piše funkcionalne teste in poganja teste na sistemu
- Slednik – odgovoren je za zajem, obdelavo in shranjevanje informacij o razvoju sistema. Drži tudi pregled na hitrostjo in učinkovitostjo razvoja.
- Vodja razvoja – zadolžen je za potek procesa razvoja in vodenje, usmerjanje ter učenje vključenih v projekt. Sposoben mora biti komunicirati učinkovito, ker le tako lahko projekt uspešno pripelje do zaključka.
- Svetovalec – je zunanji človek, ki najde točno določeno rešitev težave, ki nastane med razvojem.
- Glavni vodja (angl. big boss) – zadolžen je za nadzor celotnega procesa, spremlja razvoj in odstranjuje večje ovire za razbremenitev razvijalcev

4.6 Proces ekstremnega programiranja

Proces oziroma življenjski cikel ekstremnega programiranja je razdeljen na šest faz [4].

Faza raziskovanja

V fazi raziskovanja naročnik napiše uporabniške zgodbe, ki jih želi imeti vključene v prvi izdaji. Vsaka opisuje neko funkcionalnost. Razvojna skupina se v tej fazi seznani s tehnologijo in orodji, ki jih bo uporabljala. Ta faza ponavadi traja od nekaj tednov do nekaj mesecev, kar je odvisno od zahtevnosti.

Faza načrtovanja

Faza načrtovanja, ki traja nekaj dni, vključuje določitev prioritetnega seznama zgodb in vsebine prve izdaje. Razvijalci glede na zgodbe ocenijo zahtevnost in določijo terminski načrt.

Faza razvoja

Razvijalci opravljajo naloge, ki jim jih predvidevajo njihove vloge. Med razvojem se spremlja napredek in v primeru neskladja z načrti se primerno spremeni obseg nalog, jih doda ali vzame.

Faza delovanja

Faza delovanja zahteva dodatno testiranje ter preverjanje učinkovitosti in stabilnosti sistema. Morebitne napake se odpravijo v razvojnih ciklih. Tveganje se v tej fazi zmanjša in vsi bolj tvegani posegi se prenesejo v naslednjo verzijo.

Faza vzdrževanja

Po predaji prve verzije v uporabo naročniku nastopi faza vzdrževanja. Posledično lahko upočasni razvoj in zahteva vključitev dodatnih razvijalcev na projektu, saj se začne vzporedno ob razvoju vzdrževanje sistema.

Faza smrti

Faza smrti nastopi, ko na spisku za razvoj ni več nobene uporabniške zahteve. Predčasna smrt pa lahko nastopi, če naročnik prekine razvoj ali pa je nadaljnji razvoj nesprejemljiv. Za zaključek življenjskega cikla projekta se tako napiše kratko dokumentacijo, ki vsebuje opis ključnih dejavnikov.

4.7 Uvajanje ekstremnega programiranja v razvoj programske opreme

Za uporabo ekstremnega programiranja v razvojnem okolju je najbolj primerna velikost razvojne skupine, ki ne presega dvanajst članov in je pretežno sestavljena iz izkušenih

razvijalcev. Razvojna ekipa mora biti znotraj istega prostora, da je komunikacija neposredna in takojšnja. Od razvijalcev je tudi pričakovano, da so odprti za spremembe in se lahko hitro prilagodijo na nov način dela.

Za ekstremno programiranje so primerni manjši projekti, kjer se zahteve in specifikacije hitro spreminjajo. Pogoj je tudi uporaba orodij in tehnologij, ki omogočajo hiter razvoj in enostavne spremembe ter skalabilnost. Zaradi načina razvoja pri ekstremnem programiranju ta metodologija tudi ni najbolj primerna za kritične sisteme, saj lahko pride do nepravilnega delovanja začetnih verzij.

4.8 Kdaj uporabiti ekstremno programiranje

Za učinkovito uvedbo metodologije ekstremnega programiranja v proces razvoja programske opreme so najbolj primerne majhne ali srednje velike ekipe, sestavljene iz sposobnih razvijalcev, ki dobro sodelujejo. Primerna je tudi pri projektih, katerih zahteve se pogosto spreminjajo in je mogoče tudi izdajanje pogostih verzij.

V primeru, da naročnik ne ve, kaj potrebuje, je najbolje projekt nadaljevati z razvojem prototipov, iz katerih naročnik dobi vizijo za razvoj projekta. Če ima naročnik neprimerne in neizvedljive zahteve je treba začeti z bolj aktivnim komuniciranjem, kjer se naročniku pojasnijo dejstva in posledice, da potem ponovno pretehta zahteve in se odloči.

V primeru neupoštevanja želj naročnika se lahko projekt razvleče v večkratno popravljanje in dopolnjevanje sistema. Tudi ta problem je rešljiv z boljšo komunikacijo, kjer je naročniku treba predstaviti predloge in poglede na izvedbo. Odločitev o razvoju ostane v rokah naročnika.

Ekstremno programiranje ni idealna metodologija za razvoj programske opreme, saj ne odgovori na to, kako se obnaša na projektih, ki imajo stalen obseg, ceno in roke izdelave, kako vpliva spreminjanje vmesnikov na soprojekte, kako se upoštevajo nefunkcionalne zahteve, kot so zmogljivost in varnost, kako se uporablja pri razpršenem razvoju na različnih lokacijah, kako se vključi v razvoj že razvite komponente in kako se vodi večje projekte. [11]

5 Scrum

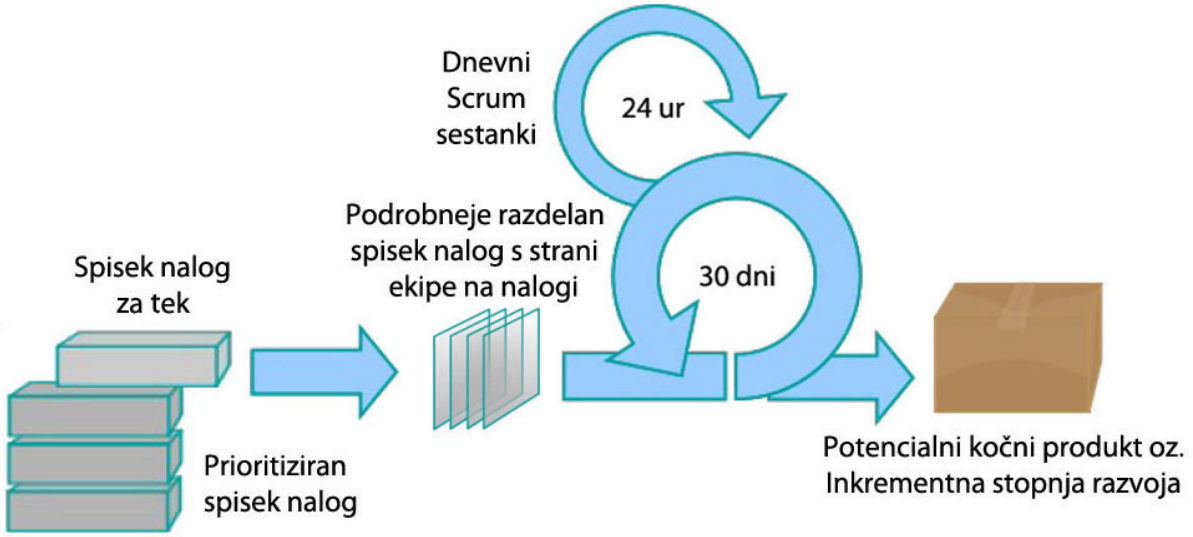
Metodologijo Scrum so razvili za uspešnejše upravljanje projektnega dela razvojnega procesa. Razvojni proces programske opreme po tej metodologiji je prišel iz industrije na podlagi izkustvenega pristopa. Scrum poudarja prožnost, prilagodljivost, ustvarjalnost in ne določa uporabe specifične tehnike pri razvoju, ampak se omejuje le na delo posameznika v skupini za dosego nekega cilja. Cilj je agilni razvoj v stalno spreminjajočem se okolju zahtev.

Zgodovina razvoja te metodologije sega že v leto 1986, ko so razvili nov holistični pristop za hiter in prilagodljiv razvoj novih izdelkov. Ta pristop so na začetku primerjali s taktiko v igri ameriškega nogometa, ko celotna ekipa poskuša s kombinacijo podaj doseči točko, leta 1991 pa so ga v nekem članku poimenovali Scrum.

Istočasno sta tak pristop vsak v svojem podjetju uporabila Ken Schwaber in Jeff Sutherland. Skupaj sta ga predstavila na konferenci leta 1995, svoje izkušnje in najboljše prakse pa sta združila v metodologijo Scrum. Leta 2001 je Schwaber skupaj z Mikeom Beedleom metodologijo zapisal v knjigi Agile Software Development with Scrum [10].

5.1 Ogrodje metodologije Scrum

Razvoj pri metodologiji Scrum poteka v več ciklih, pri katerih vsak povečuje funkcionalnost izdelka (angl. interactive incremental process). Izvedba cikla se imenuje *tek* (angl. sprint) in traja ponavadi od dva do štiri tedne. Znotraj vsakega teka se na dnevnem sestanku, ki se imenuje *dnevni Scrum* (angl. daily Scrum) preverja stanje pri vsakem razvijalcu in se po potrebi prilagaja razvoj glede na nastale spremembe. Naloge, ki morajo biti izvedene, se predhodno zapiše v *seznam zahtev izdelka* (angl. product backlog).



Slika 8: Prikaz ogrodja metodologije Scrum

Pri metodologiji Scrum razvojna skupina sama določi, koliko je sposobna narediti glede na svoje znanje in sposobnosti razvijalcev. Razvoj se spremlja dnevno in tako tudi lahko prihaja do dnevnih sprememb v primeru nepredvidenih zapletov in težav. Na ta način se spodbuja kreativno delo razvojne ekipe in povečuje učinkovitost.

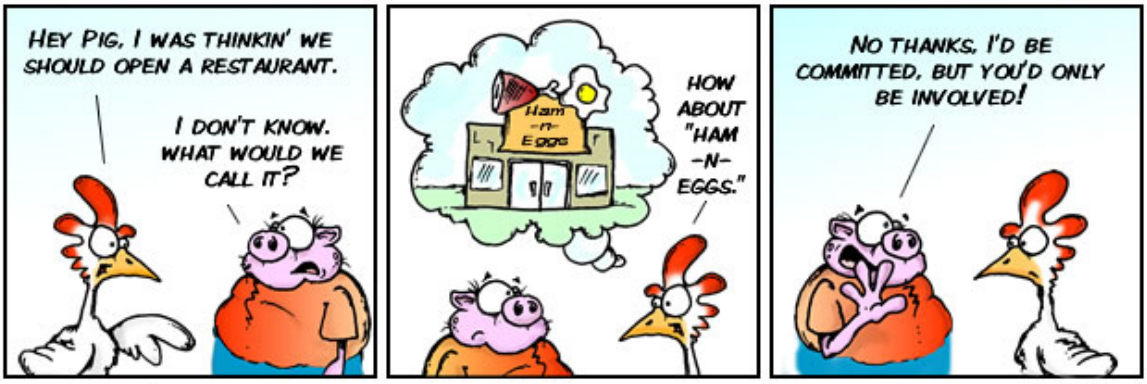
5.2 Vloge metodologije Scrum

Pri metodologiji Scrum imamo tri vloge, ki skupaj sestavljajo Scrum ekipo (angl. Scrum team) [10]:

- Vodja Scruma (angl. Scrum master)
- Produktni vodja (angl. Product owner)
- Razvojna ekipa (angl. Team)

Metodologija Scrum strogo loči med ljudmi, ki so na projekt vezani in ljudmi, ki so pri projektu samo prisotni. Avtorja Ken Schwaber in Mike Beedle [10] za prispodobo vlog ljudi pri Scrum razvoju predstavita vlogi prašičev in piščancev.

Prispodoba izhaja iz šaljive zgodbe o piščancu in prašiču, ki odpirata skupno restavracijo z imenom "Šunka in jajce". Iz imena restavracije je razvidno, da pride do različnih vlog, saj se mora prašič popolnoma žrtvovati za projekt, piščanec pa je samo prisoten pri projektu.



Slika 9: Vlogi piščanca in prašiča [20]

5.2.1 Vodja Scruma

Vodja Scruma ali Scrum gospodar je zadolžen za pravilno uporabo metodologije Scrum na projektu. Njegova naloga je poskrbeti za to, da so člani razvojne skupine podučeni o metodologiji, ki jo izvajajo. Prav tako se ukvarja z implementacijo metodologije Scrum v organizacijo in skrbi za dosledno sledenje pravil metodologije.

5.2.2 Produktni vodja

Produktni vodja, ki ga imenujejo naročnik, uprava ter vodja Scruma, je zadolžen za preverjanje in usmerjanje projekta in sestavljanje dnevnika zahtev. Podrobno je obveščen o vseh zahtevah naročnika. Vsebina njegovega dela vključuje določanje spreminjanja seznama zahtev izdelka projekta, določanje zaporedja in prioritete razvoja oziroma izgradnje funkcionalnosti.

5.2.3 Razvojna ekipa

Razvojno ekipo sestavljajo razvijalci z različnih področij in z raznolikim znanjem. Je notranje neodvisna in samostojna. Vsaka razvojna skupina se samoorganizira in odloči o tem, kako se bo funkcionalnost razvijala skozi teke. Tako tudi za izvedbo funkcionalnosti projekta odgovarja ekipa kot celota.

5.3 Dokumenti pri metodologiji Scrum

Scrum je agilna metodologija in se drži posebnosti drugih agilnih metodologij, ki ne predpisujejo velikega števila dokumentacije. Pomembna dokumenta pri Scrum sta dva, in sicer seznam zahtev izdelka (angl. Product Backlog) in seznam zahtev teka (angl. Sprint Backlog). [10]

5.3.1 Seznam zahtev izdelka

Seznam zahtev izdelka (angl. Product Backlog) je prioritiziran seznam vseh funkcionalnosti, ki se jih naj bi v projektu izvedlo. Ta seznam sestavi in ima nad njim tudi popoln nadzor produktni vodja. Seznam zahtev izdelka vsebuje izdelave funkcij in omogoča enostaven pregled v dokončanost projekta.

Seznam zahtev lahko vključuje različne stvari od razvoja posameznih funkcij, odprave napak, pohitritve delovanja, nadgradenj pa do spremembe uporabniškega vmesnika. Seznam zahtev izdelka je aktiven, dokler projekt ni zaključen. Glede na primer je razvidno, da prvi stolpec seznama določa naziv zahtev, v drugem stolpcu je predvideno število dni za izvedbo, tretji pa vsebuje korekcijski faktor. Popravljeno oceno števila dni za izvedbo, ki je rezultat drugega in tretjega stolpca, vsebuje četrti stolpec. Korekcijski faktor vsebuje tudi morebitno odsotnost člana razvojne ekipe.

5.3.2 Seznam zahtev teka

Seznam zahtev teka izdelka sestavi razvojna ekipa skupaj, in sicer iz določenih zahtev s seznama zahtev izdelka, ki se bodo realizirale v trenutnem teku. Ta seznam je na vpogled vsem, ki sodelujejo na projektu, vendar ga spreminja le razvojna ekipa.

Vsebuje podrobne naloge, ki so na programu za izdelavo v trenutnem teku. To so zahteve, vzete s seznama zahtev izdelka in razdeljene na posamezne naloge in opravila. Ena naloga traja od štiri do največ šestnajst ur delovnega časa, vse daljše naloge pa se razdelijo na krajše enote.

Poleg samega spiska nalog in opravil, je ob seznamu zahtev teka zapisano še, kdo je sprožil zahtevek, kdo je zanj odgovoren, kakšen je status naloge in koliko ur razvoja je še do izpolnitve zahteve. Ta zadnji podatek je ocena razvijalca, ki na nalogi dela. Ocene delovnih ur se tako dnevno zmanjšujejo, razen če razvijalec naleti na nepričakovano oviro.

Vsebina seznama zahtev se spreminja in dopolnjuje na obveznem dnevnem Scrum sestanku.

5.4 Proces razvoja s Scrum

Razvoj projekta se začne z vizijo delovanja sistema, ki se ga bo razvilo. Vizija se skozi ponovitve razvoja dopolnjuje in natančno opredeli. Zaradi tega tudi nekatere izmed na začetku določenih zahtev postanejo odvečne in se jih zavrže ter doda druge.

V prvem koraku produktni vodja skupaj z vodjo Scruma sestavi seznam zahtev izdelka, v katerem so po prioriteti zabeležene vse zahteve, ki jih je treba izvršiti. Pri gradnji sistema se

tako najprej izdela zahteve, ki se jih lahko najhitreje vgradi v končno verzijo in tiste ki imajo višjo prioriteto. [10]

5.4.1 Načrtovalni sestanek

Razvoj poteka v več razvojnih ciklih oziroma tekih, ki so dolgi po trideset koledarskih dni. Vsak tek se prične z načrtovalnim sestankom (angl. sprint planning meeting). Na njem se izbere zahteve za trenutni tek, določi potek zahtev in razporeditev v seznam zahtev teka. Oba sestanka, tj. izbira zahtev in določitev podrobne izvedbe, sta omejena na štiri ure:

- Na prvem delu sestanka produktni vodja, vodja Scruma in razvojna ekipa zastavijo načrt razvoja za tekoči tek. Potem ko produktni vodja predstavi glavno prioritetno zahtevo, se razvojna ekipa dodobra seznani z zahtevo, predstavi lasten pogled nanjo in predviden časovni interval ter koliko je mogoče opraviti v trenutnem teku. Na koncu se izbere skupek takih zahtev, ki tvorijo celoto po končanem teku in bodo do neke mere pripravljene za uporabo. Namen sestanka je določitev dela, zato se tehničnih vprašanj na samem sestanku ne loteva.
- Drugi del sestanka je usmerjen v razvojno ekipo, ki si izdela podroben načrt teka. Skupina se samoorganizira, razdeli naloge in izda seznam zahtev teka (angl. sprint backlog). Po koncu tega dela se tek začne.

Med tekom se seznam zahtev teka naj ne bi spreminjal, če pa pride do sprememb seznama, mora biti ob koncu teka še vedno dosežen zadani cilj. V primeru, da pride do izrednega stanja, se lahko celoten tek prekine in začne nov tek z novim sestankom in novimi zahtevami.

5.4.2 Dnevni Scrum sestanek

Razvojna ekipa in vodja Scruma se srečata vsak dan na dnevnem Scrum sestanku (angl. daily Scrum), ki traja največ petnajst minut. Običajno poteka stoje v krogu, da se vsi udeleženci gledajo v oči. Na tem sestanku vsak član razvojne ekipe jedrnato odgovori na tri vprašanja:

- Kaj je naredil od prejšnjega dnevnega Scrum sestanka?
- Kaj namerava delati do naslednjega dnevnega Scrum sestanka?
- Na katere težave je naletele pri delu?

Dnevni Scrum sestanek je namenjen dnevni sinhronizaciji razvojne ekipe in sprotnemu obveščanju o težavah.

5.4.3 Sestanek pregleda rezultatov teka

Na koncu teka se vedno izvede sestanek za pregled rezultatov teka (angl. sprint review meeting). Vsak sestanek pri metodologiji ima določeno maksimalno dolžino; pregledni

sestanek teka je lahko dolg največ štiri ure. Kar se tiče njegove vsebine, razvojna ekipa predstavi rezultate teka produktnemu vodji s poudarkom na funkcionalnostih, pripravljenih za uporabo. Produktni vodja tako pregleda izvedene rešitve in se seznani z možnostmi za nadaljnji razvoj. Na sestanku lahko pride tudi do sprememb prioritet določenih funkcionalnosti in zahtev.

5.5 Uporaba in dobre lastnosti metodologije Scrum

Uporaba metodologije Scrum pride najbolj do izraza pri projektih, kjer je razvoj negotov in kjer so projekti manjšega obsega. Pri teh pogojih najbolj v ospredje pridejo hiter razvojni cikel, stalna prisotnost naročnika, delujoča rešitev na koncu vsakega cikla (teka) in stalna prilagodljivost razvojne ekipe za doseg cilja teka.

Ker so razvojni cikli oziroma teki razmeroma kratke narave in ker je naročnik (produktni vodja) vedno prisoten, to omogoča kvalitetno sprotno preverjanje izbranih odločitev med samim razvojem. Ko na koncu teka produktni vodja pregleda izdelavo in funkcionalnost izdelka, poda tudi oceno rešitve, ki jo razvojna ekipa kasneje upošteva pri novem teku. Ker je glavna zahteva Scrum na koncu teka delujoča koda, je lažje tudi testiranje in preverjanje kvalitete kode. Zaradi kratkih razvojnih ciklov pri Scrum je treba razvoj razdeliti na manjše in lažje vodljive dele, ki se jih lahko izvaja in testira tudi, če celotni sistem še ni zgrajen.

Scrum kot agilna metodologija pri razvoju predpisuje nujno prisotnost naročnika, predvsem na začetku in na koncu teka, ko se tek načrtuje in ocenjuje. Naročnikove želje in mnenje določajo smer razvoja, vendar pa po začetku teka naročnik ne more več posegati v seznam zahtev in ima razvojna ekipa, oziroma Scrum skupina, mir za razvijanje in doseganje zastavljenih ciljev z začetnega sestanka.

Ker ima Scrum predpisan dnevni Scrum sestanek se kontrola razvoja izvaja že med samim tekom prek treh ključnih vprašanj. V primeru, da pride do težav ali nepredvidenih zapletov, se te razrešuje takoj tudi s prilagajanjem zahtev, vendar pod pogojem, da se na koncu teka še vedno doseže glavne funkcionalnosti in delovanje zastavljenega dela sistema. Sestanek je koristen tudi, ker se vsi razvijalci seznanijo z delom v ekipi, s čimer se poveča motiviranost in skupino bolj poveže. Zaradi omejitve trajanja sestanka na največ petnajst minut in jedrnatih odgovorov na tri bistvena vprašanja ta sestanek poteka stoje, tako da so vsi obrnjeni proti sredini kroga.

Ko se tek začne, se razvojna ekipa sama organizira in razdeli naloge glede na izkušnje in znanja v skupini. V primeru konfliktov se te rešuje med člani ekipe in, samo če razvojna ekipa sama ne more razrešiti konflikta, vmes poseže vodja Scruma ter po potrebi odstrani motečega člana. [10]

6 Prilagoditev in združitev Scrum in ekstremnega programiranja

Metodologija Scrum se osredotoča predvsem na sam proces vodenja projekta in ne predpisuje dobrih inženirskih oziroma programerskih praks, kar lahko povzroči padec produktivnosti. Zato tu pride do izraza ekstremno programiranje, ki je sestavljeno ravno iz najboljših inženirskih praks in je osredotočeno na gradnjo boljše kode. Pomanjkanje določitev vodenja metodologije ekstremnega programiranja se nadomesti z ogrodjem metodologije Scrum.

6.1 Način prilagoditve metodologij

V raziskavi sem se osredotočil na sam okvir za vodenje projekta, ki je del Scrum metodologije (teki, dnevni Scrum sestanki, seznami zahtev, produktni vodja, vodja Scruma), ekstremno programiranje pa je bila osnova pri uporabi dobrih praks programiranja kot nadomestek samoorganiziranja kot ga zagovarja Scrum. Združitev Scrum in ekstremnega programiranja je logična rešitev za učinkovit razvoj programske opreme. Skupaj sta zelo učinkoviti metodologiji, ki dajeta dobre rezultate in zagotavljata agilnost, saj se dopolnjujeta. [8]

V nekaterih metodah se ti dve metodologiji prekrivata, obe rešujeta isto težavo razvoja. V tem pogledu smo večinoma uporabili metodologijo Scrum, ekstremno programiranje pa za pri Scrum nedefiniranih področjih in pri uporabi inženirskih praks kodiranja in testiranju.

Glavni deli metodologije ekstremnega programiranja, ki sem jih vključil so:

- **Preprost model** – posvečali smo se načrtovanju in izgradnji samo tistega, kar je nujno potrebno za podporo zahtevani funkcionalnosti s seznama zahtev
- **Najprej testiraj** – že predhodno so bili za vsak korak napisani testi modulov. To je prisililo razvijalce, da so razumeli pričakovan rezultat posameznega modula. Skozi razvoj smo teste zbirali in si tako gradili knjižnico regresijskih testov.
- **Neprestana/kontinuirana integracija** – za doseganje te zahteve iz metodologije ekstremnega programiranja smo si izbrali orodje, ki samodejno integrira kodo v repozitorij ter zažene knjižnico testov, ki preverjajo pravilno delovanje
- **Preurejanje kode** – preurejanje kode omogoča stalno izboljšavo sistema po korakih
- **Programiranje v parih** – skozi teke je potekalo tudi programiranje v parih pri vseh aktivnostih
- **Skupno lastništvo kode** – s tem smo zagotovili lažje preoblikovanje kode in pregledovanje, popravljanje napak

- **Enotni standard kodiranja** – ena od zahtev za lažje skupno lastništvo kode in dodana vrednost, ki pripomore k preprostejšemu modelu
- **40 urni delavnik** – razvijalci ne opravljajo nadur, ki se jim izognemo z natančnim načrtovanjem na začetku posameznega teka v dogovoru s produktnim vodjo, izločanjem nenujnih zahtev ter reorganizacijo prioritetnega seznama zahtev

Ostale zahteve metodologije ekstremnega programiranja so pokrite v ogrodju metodologije Scrum. Časovno kratke verzije so zagotovljene s principi tritedenskih tekov, opis sistema je narejen s pomočjo seznama zahtev izdelka in časovnega ocenjevanja ter opisom posameznih nalog. Vlogo produktnega vodje, ki je aktivno vključen v razvoj, sem prevzel iz metodologije Scrum, od koder izhaja tudi v združeni metodologiji uporabljena organizacijska plat. Za enako vlogo metodologija ekstremnega programiranja uporablja izraz naročnik. Princip igre načrtovanja iz ekstremnega programiranja sem v združeni metodologiji nadomestil s kvalitetnim vodenjem (graf realizacije zahtev iz metodologije Scrum) in vodjo Scruma.

Zaradi uporabe tehnik ekstremnega programiranja se je močno povečala kvaliteta kode v primerjavi s starejšimi projekti pred uporabo nove metodologije.

Način organizacije razvoja v podjetju temelji na principu metodologije Scrum. Nove funkcionalnosti se razvijajo na podlagi vpisa v seznam zahtev, ki se potem ustrezno prilagodi za posamezne verzije in teke. Razvojna skupina še vedno samostojno odloča, kako bo potekal razvoj, ampak se tudi drži inženirskih praks ekstremnega programiranja. Razvita koda je vselej preprosta; že med programiranjem se za pravilnost kode skrbi z integracijskimi testi, vzdržuje pa se jo s preoblikovanjem, kar omogoča enoten standard kodiranja. Delujočo kodo se večkrat dnevno prenese v delujoč testni sistem in sproti tudi opravlja predpisane teste. Dnevni Scrum sestanek zagotavlja, da se razvoj odvija po zastavljenem časovnem načrtu.

Tabela 1 predstavlja prikaz v prilagojeni združeni metodologiji uporabljenih elementov, ki izhajajo iz metodologij Scrum in ekstremnega programiranja.

Delo	Scrum	XP
Programiranje v parih		X
Prizadevnost (40-urni delavnik)	X	X
Poučno delovno okolje		X
Globinsko reševanje problemov (root-cause analysis)		X
Pregled opravljenega dela	X	

Sodelovanje	Scrum	XP
Zaupanje v ekipo	X	
Skupno reševanje (odprto delovno okolje s tihimi sobami)		X
Ekipa	X	
Vključevanje naročnika v razvoj (produktni vodja)	X	

Razumljiv jezik za opisovanje	X	
Dnevni Scrum sestanki	X	
Enoten standard kodiranja		X
Predstavitev teka	X	
Pregled teka, poročanje	X	

Izdaje	Scrum	XP
Brez hroščev	X	X
Pregled verzij (enoten repozitorij)		X
Izgradnja in testiranje vsako uro		X
Neprestana integracija		X
Skupno lastništvo kode		X
Baza znanja in dokumentacija verzij na internem Wikiju		X

Načrtovanje	Scrum	XP
Vizija		X
Načrtovanje izdaj in seznam zahtev	X	
Upravljanje s tveganjem		X
Teki (iterativni razvoj)	X	
Prosti čas med teki	X	
Zgodbe	X	
Ocenjevanje	X	

Razvoj	Scrum	XP
Neprestane potrebe	X	
Testiranje	X	X
TDD		X
Preurejanje kode		X
Preprost model		X
Inkrementalen model in arhitektura	X	
Tehnološko napredne rešitve		X
Optimizacija delovanja		X

Vodenje projektov	Scrum	XP
Vodja Scruma	X	
Produktni vodja	X	
Prekinitev teka	X	
Cilj teka	X	

Tabela 1: Pregled združevanja Scrum in ekstremnega programiranja

6.2 Predstavitev podjetja

Podjetje se ukvarja predvsem z digitalnim oglaševanjem in je za svoje potrebe programsko opremo razvijalo z zunanjimi sodelavci. Z začetkom razvoja nove generacije programske podpore, ki je nujna za prilagoditev trgu, je ustanovilo oddelek razvoja programske opreme, kjer so zaposleni predvsem izkušeni mladi programerji, ki se dobro prilagajajo spremembam. V oddelku za razvoj je trenutno devet ljudi z različnimi znanji. S pomočjo primerov uporabe ogrodja Scrum in ekstremnega programiranja [8] smo za primer razvoja v podjetju prilagodili organizacijo procesa razvoja, izdajanja novih verzij in kvalitetnih praks pisanja programske kode, do končnega stanja izdelka, ki se uporablja v produkciji.

Zaradi zahtev trga mora biti razvoj programske opreme zelo odziven, saj se smernice razmeroma hitro spreminjajo, zato je pomembno tudi sledenje najnovejšim tehnologijam. V procesu razvoja uporabljamo take, ki so primerne za posamezen del programa, in pa sistem za kontrolo verzij. Bazo znanj znotraj podjetja gradimo na internem Trac[1] in Wiki[2] portalu. V nadaljevanju je predstavljen proces razvoja.

6.3 Izdelava seznama zahtev izdelka

Začetek razvoja novega izdelka je izdelava seznama zahtev izdelka, ki je sestavljen iz prioritetnega seznama zahtev, zgodb ali značilnosti, opisanih laično s strani uporabnika.

Zgodbe razdelimo na več področij:

- ID – enolični identifikator
- Naziv – kratko opisno ime zgodbe, ki je dovolj jasno, da ga razume tako razvijalec kot produktni vodja
- Pomembnost – prioritetni seznam produktnega vodje za zgodbo. Uporabljamo številčne ocene v območju 0 do 100, vendar pa v primeru dodatne še pomembnejše zahteve dovoljujemo tudi višje številke
- Začetna ocena – prva ocena razvojne ekipe o obsegu dela za implementacijo zgodbe. Ocena se navadno poda v enoti človek-dan.
- Tip predstavitve – podroben opis, kako se bo predstavilo izdelano rešitev na koncu teka (preproste značilnosti testa, ki ga bo morala rešitev prestati)
- Opombe – vse ostale in dodatne informacije oziroma opombe na kratko

[1] Trac - Spletni aplikacija za sledenje napakam in vodenje projektov.
[2] Wiki - spletna aplikacija, ki omogoča enostavno izdelavo in urejanje spletnih strani.

Sezam zahtev izdelka					
ID	Naziv	Pomembnost	Ocena	Tip predstavitve	Opombe
11	*Predvajanje videa*	*90*	*6*	*Prijava v sistem, s seznama ponujenih izdelanih videov izbor enega pripravljenega in opcija predvajanja v novem oknu ter celozaslonsko*	*Baza uporabnikov, pravice uporabnikov, varnost podatkov še ni treba zagotoviti*
12	*Možnost dodajanja*	*30*	*2*	*Prijava v sistem, pregled trenutnega stanja, poljubno dodajanje datoteke in potrjevanje pošiljanja*	*Uporaba seznama in atributov brez dodatnih informacij, zgolj le testne*

Tabela 2: Primer seznama zahtev izdelka

Seznam z naštetimi atributi se je izkazal kot dovolj obsežen in zadovoljiv, saj pokrije vse potrebne informacije o zahtevi oziroma o zahtevani funkcionalnosti. Dokument, v katerem se nahajajo seznami zahtev izdelka, je trenutno v tabelarični obliki na interni Wiki strani. Lastnik datoteke je produktni vodja, vendar je vpogled na voljo vsej razvojni ekipi. Sistem verzij ni potreben, saj glavne spremembe opravlja le produktni vodja, ostali dokument le pregledujejo.

V podjetju je produktni vodja navadno predstavnik marketinškega oddelka, zato tudi ti dokumenti niso preveč tehnološko opredeljeni. V primeru, da je seznam opredeljen s preveč tehničnimi podrobnostmi, je produktni vodja opozorjen in se z njim pogovori o razlogih za take podrobnosti.

6.4 Načrtovanje teka

6.4.1 Priprava na načrtovanje teka

Prva stvar, ki jo je treba zagotoviti pred sestankom za načrtovanje teka, je popoln seznam zahtev. To pomeni, da je:

- Seznam zahtev izdelka izpopolnjen;
- En izdelek ima le en seznam, za katerega skrbi en produktni vodja;
- Vse pomembne zahteve morajo imeti označen nivo pomembnosti;
    - Manj pomembne zahteve imajo lahko enako vrednost, pomembne pa morajo nujno biti prioritizirane
    - Označene morajo biti tudi zahteve, ki bodo najverjetneje vključene v naslednji verziji
    - Večja pomembnost pomeni večjo prioriteto. Četudi le za eno stopnjo, ima enak pomen, kot če je ta vrednost dosti večja

    - Ko se zahteve prioritizira, se pusti nekaj prostora pri dodeljevanju prioritete, da lahko vmes vključimo še dodatno zahtevo, ki se pojavi med razvojem
- Produktni vodja podrobno pozna vsako zgodbo, saj je avtor in odgovoren za seznam. Ni treba, da ve vse, kar je treba implementirati, mora pa poznati posamezne zgodbe zahtev;
- Če kdo drug dodaja zgodbe na seznam, jih najprej pregleda produktni vodja in jim tudi dodeli pomembnost izvedbe.

6.4.2 Sestanek za načrtovanja teka

Sestanek za načrtovanje teka je ključen sestanek, saj lahko slabo načrtovanje podre celoten tek. Bistvo sestanka je posredovati ekipi dovolj informacij za delo in od tega sestanka dalje ekipi pustiti tudi dovolj miru za kvalitetno delo na teku. [9]

Konkretni rezultati, ki jih da sestanek so:

- Določitev cilja teka
- Seznam članov ekipe in njihova vpletenost
- Seznam zahtev teka
- Določitev datuma predstavitve izdelka teka
- Določitev kraja in časa za dnevni Scrum sestanek

6.4.3 Obseg sestanka

V primeru, da se sestanek za načrtovanje teka bliža časovni omejitvi, je treba nekatere stvari izpustiti. Sestanek se prekine in nadaljuje naslednji dan le, če niso bile dogovorjene osnovne stvari, sicer se nerešene zadeve pusti za obravnavo med tekom. Obstaja prioritetni seznam stvari, ki morajo biti obdelane in tiste, ki niso najnujnejše za začetek teka.

Najnižjo prioriteto ima razčlemba zgodb na naloge, saj se to lahko izvaja dnevno, prav tako se lahko dnevni Scrum sestanek določi po sestanku s strani vodje Scruma. Če ne gre drugače, se izpusti tudi računanje hitrosti in virov.

Sestanek se ne more končati brez zaključkov glede ciljev teka in datuma predstavitve verzije, seznama zgodb izbranih za tek, ocene za vsako zgodbo in opisa preverjanja delovanja posamezne zahteve oziroma zgodbe.

6.4.4 Udeležba na sestanku za načrtovanje teka

Prisotnost tako produktnega vodje kot tudi vseh članov razvojne ekipe je bistvena, saj vsaka zgodba vsebuje tri spremenljivke, ki so med sabo globoko povezane in jih določi ekipa skupaj. To so ocena, pomembnost in obseg. Medtem ko začetno oceno in obseg določi

produktni vodja, dokončno oceno poda razvojna ekipa. Na sestanku se te tri spremenljivke dokončno izoblikujejo z dialogom med razvojno ekipo in produktnim vodjo.

Sestanek se začne s predstavitvijo cilja bodočega teka, kot si ga je zamislil produktni vodja in predstavitvijo najpomembnejših zahtev oziroma zgodb. Nato ekipa oceni časovno zahtevnost posameznih zahtev, ki so že prioritizirane. Pogosto se zgodi, da časovne ocene izvedbe zahteve močno odstopajo od pričakovanj produktnega vodje, zaradi česar se pomembnost zahtev prerazporedi in ponovno oceni. Tak način načrtovanja, z neposrednim stikom, je ključen za agilni razvoj programske opreme.

6.4.5 Pri kvaliteti se ne popušča

Kvaliteto razdelimo na notranjo in zunanjo kvaliteto. Medtem ko notranja kvaliteta uporabniku ni vidna in se kaže v lažjem upravljanju in vzdrževanju sistema, se zunanja kvaliteta izraža v tem kako na izdelek gledajo uporabniki. [8]

V nekaterih primerih je povsem razumljivo, da se daje več poudarka na notranjo kvaliteto, sam vmesnik pa se dogradi v naslednjih verzijah. Obratno to ni dopustno.

Če se produktnemu vodji zdi časovna ocena zahteve preobsežna in želi, da se naredi nekakšen polizdelek v polovičnem času, je to nedopustno. Vsako popuščanje pri notranji kvaliteti prinese le nadaljnje delo v naslednjih verzijah, kar lahko časovno spremenljivko le še podaljša, kljub hitri izdelavi polizdelka. Zato je v takem primeru potrebno dodatno pogajanje o globini in pomembnosti nekaterih funkcionalnosti, ki se jih lahko za trenutno verzijo omeji in razvija v naslednjih verzijah, vendar ne na račun zmanjšanja notranje kvalitete.

6.4.6 Sestava sestanka za načrtovanje teka

Sestanek za načrtovanje teka ima točno določen časovni okvir (štiri ure). V primeru, da v tem času ne pride do kvalitetnega načrta za tek, je sestanek najbolje prekiniti. Glede na določila za uvajanje metodologije se na začetku tek lahko začne s slabšim načrtom, da se ne bi prekoračilo časovnega intervala sestanka, s čimer se ekipa nauči časovnih omejitev za načrtovanje. Na začetku uvajanja je sprejemljivo tudi to, da se sestanek po preteku dodeljenega časa prekine in zaključi naslednji dan. [9]

Uvedli smo ustaljeno agendo teh sestankov. Sestanek traja od 12. do 16. ure z deset minutnimi odmori vsako uro.

Okvirni dnevni red sestanka za načrtovanje teka:

- 12:00 – 12:30 – Produktni vodja predstavi cilje teka in povzame seznam zahtev izdelka. Postavljeni so datumi predstavitve.

- 12:30 – 14:00 – Ekipa oceni časovno zahtevnost dela in po potrebi razbije zahteve na manjše enote, ki jim produktni vodja ponovno dodeli pomembnost. Zahteve so pojasnjene vsem in opiše se način predstavitve izdelka na koncu teka.
- 14:00 – 15:00 – Ekipa izbere zgodbe, ki bodo vključene v tek in postavi realno oceno hitrosti razvoja.
- 15:00 – 16:00 – Dogovor o času in kraju dnevnega Scrum sestanka in dodatna razčlemba zgodb na naloge.

6.4.7 Določitev dolžine teka in ciljev teka

Eden od proizvodov sestanka za načrtovanje teka je datum predstavitve izdelka teka, kar avtomatsko opredeli dolžino trajanja teka.

Kratki teki zagotavljajo agilnost in večkratno spremembo smeri razvoja, pogostejše izdajanje verzij in manj časa porabljenega za razvoj v napačno smer. Pri daljših tekih ima ekipa več časa, da pridobi zagon in več časa za odpravo napak in zapletov. [9]

Razvijalcem je bolj po godu, da so teki daljši, medtem ko produktni vodja želi čim krajše. Zato je tu nujna sklenitev kompromisa. Glede na priporočila smo se odločili za tritedenske teke, kar se je izkazalo za dobro. Tako so dovolj dolgi, da se razvojna ekipa kvalitetno vključi v razvoj in še dovolj kratki za morebitne spremembe in razvoj v drugo smer. Po treh mesecih razvoja se držimo dolžine tekov treh tednov, čeprav se kdaj zdijo predolgi glede na ideje in smeri razvoja, ki nastajajo pri produktnem vodji. Če je nujno se odločimo tudi za krajši tek, ki ima ponavadi poseben namen.

Cilj teka je lahko tudi samo nekaj, kar pritegne vodstvo, ko se jim ga predstavi, in zapiše v laičnem jeziku. Tako je cilj teka lahko tudi 'izboljšati delovanje programa' ali 'izboljšati našo prepoznavnost', mora le odgovoriti na to, kakšni so razlogi za načrtovanje tega teka. Med načrtovanjem določitev cilja teka to sicer nima velikega učinka na ekipo, vendar se izkaže za dobro motivacijo v sredini teka, če pride do zmede in nejasnosti. [8]

6.4.8 Vključitev izbranih zgodb v tek

Glavna aktivnost sestanka za načrtovanje je odločitev, katere zgodbe s seznama zahtev izdelka se vključi v trenutni tek.

Ko imamo zahteve urejene po prioriteti in so časovno ocenjene, razvojna ekipa pa v časovnem obdobju enega teka ne more izvršiti vseh zahtev, se s seznama zahtev izdelka izbere tiste, ki se jih bo izvedlo v naslednjem teku.

6.4.8.1 Vpliv produktnega vodje na vključitev zgodb v tek

V primeru, da se zahteva, ki si jo produktni vodja zamisli v trenutnem teku, ne uvrsti na seznam, ki ga sestavi ekipa na sestanku, lahko ta zahteve na novo prioritizira in tudi želeno uvrsti na seznam. Druga možnost je, da se zahtevo zmanjša, tretja, pa da se zgodbo razdeli na več delov in izvede tistega, ko ga produktni vodja določi za ključnega.

6.4.8.2 Kako se ekipa odloči, katere zgodbe vključi v tek

Ekipa se odloča za vključitev zgodb v tek na podlagi izkušenj in občutka ter izračunov trajanja. Izračun trajanja je izveden v dveh korakih. V prvem se oceni hitrost izvedbe, nato pa se preračuna, koliko zahtev se lahko doda, da se ocene ne preseže.

6.4.8.3 Računanje ocene hitrosti izvedbe

Hitrost izvedbe pove, koliko dela je že opravljenega. Osnovana je na oceni, ki je bila postavljena na sestanku za načrtovanje teka na začetku. Poznejše spremembe ocene se ne upoštevajo. Hitrost izvedbe se izračuna z oceno virov glede na število razvijalcev in število dni ocenjenih za izvedbo. Ko dobimo podatek o številu človek/dni za tek, lahko nadaljujemo tako, da ta podatek pomnožimo z znanim faktorjem osredotočenosti, ki ga dobimo iz prejšnjih tekov s koeficientom dejanske hitrosti in razpoložljivih virov (človek/dan). [9]

$$\textit{razpoložljivi človek/dan} \times \textit{faktor osredotočenosti} = \textit{ocena hitrosti}$$

Slika 10: Izračun hitrosti izvedbe

6.4.9 Uporaba samolepilnih listkov

Na sestanku se večji del časa porabi za zgodbe, ki so na seznamu zahtev izdelka; se jih podrobneje opisuje, ocenjuje trajanje izvedbe, prioritizira, deli na manjše dele in podobno. Ravno zato sestanek veliko bolje teče ob uporabi table, pisala in samolepilnih listkov z zahtevami, kot pa z uporabo računalnika. Kdorkoli želi narediti spremembo, pristopi k tabli in dopiše oziroma premakne posamezen listek glede na prioriteto in izvedbo.

Tabelo v Excelu se izpolni šele po koncu sestanka, ko so vse prioritete in trajanja točno določena.

Slika 11: Razporeditev zgodb na table med sestankom

6.4.10 Zgodbe s seznama izdelka in razčlemba zgodb na naloge

Če zgodbe na uvodnem sestanku niso dovolj razložene oziroma pride do napačnega razumevanja pri razvijalcih, se lahko zgodi, da se bo produktnemu vodji nova funkcionalnost na koncu teka zdela popolnoma nepotrebna oziroma nezaželena. Zato je na sestanku, ko se načrtuje tek, treba podrobno predelati zgodbe in zahteve tako, da ni nič dvoumnega.

Razlaga zgodbe je lahko zelo kratka, vendar pa morajo biti zgodba in zahteve natančno opredeljene, da gre razvoj v pravo smer in ne pride do nepotrebnih nesporazumov.

Zgodbe prav tako ne smejo biti preobsežne ali prekratke. Obsežnost zgodb je ponavadi odvisna tudi od zahtevnosti in pomembnosti. Ena zgodba je ponavadi dolga do deset človek/dni, kar pomeni, da je glede na velikost ekipe do deset zgodb na tek, kar je dovolj majhna številka za lažje načrtovanje s samolepilnimi listki.

Slika 12: Razdelitev zgodbe na manjše zgodbe

Zgodbe se na sestanku razdeli še na manjše enote oziroma naloge. Glavna razlika med zgodbo in nalogo je v tem, da je zgodba na koncu teka nek izdelek, medtem ko je naloga le vmesna pot do izdelka in se ne tiče samega seznama zahtev oziroma produktnega vodje.

Slika 13: Razdelitev zgodbe na naloge

6.4.11 Določitev dnevnega Scrum sestanka

Dnevni Scrum sestanek je bistvo metodologije Scrum. Brez tega sestanka je tek brezpredmeten. Še posebej pomemben je prvi dnevni Scrum sestanek, na katerem vsi razvijalci predstavijo, s čim bodo začeli in kako. [8]

Za dnevne sestanke smo določili dopoldanski čas, in sicer 10. uro. Vsak dan se vsi iz ekipe razvijalcev v sredini prostora postavijo v krog, obrnjeni v notranjost kroga. Vsak odgovori na tri omenjena osnovna vprašanja, ki so jedro sestanka. Sestanek se zaključi v desetih minutah. Dopoldanska ura je bila izbrana zato, da si razvijalec vnaprej lahko načrtuje nadaljnje delo in ima zjutraj do desetih še dovolj časa, da pregleda delo prejšnjega dne.

6.4.12 Tehnične zahteve

Tehnične zahteve so ponavadi nujno potrebne za izvajanje teka. Sem spadajo predvsem tehnologije, v katerih se bo razvoj odvijal. Gre za nujnosti, ki pa jih produktni vodja ne določi v seznamu zahtev za izdelek in se jih na koncu teka ne predstavi. Kljub temu je nujno, da se z njim lahko rešuje ostale zahteve in zgodbe s seznama.

Da ne pride do zmešnjave oziroma da to delo ne izgubi pomena in se ga ne uvrsti v računanje in ocenjevanje časa razvoja, se tehnične zahteve preoblikuje v zgodbe, ki so lahko merljive s strani poslovanja. Produktni vodja lahko tako oceni njihovo vrednost in prioriteto in jih uvrsti na seznam zahtev izdelka. Če preoblikovanje ni mogoče, se tehnično zahtevo poskuša uvrstiti v drugo zgodbo kot nalogo. V najslabšem primeru je treba narediti še en seznam zahtev in

skupaj z produktnim vodjo določiti čas v razvoju, ki bo namenjen reševanju teh tehničnih zahtev in potreb.

6.4.13 Uspešen sestanek za načrtovanje teka

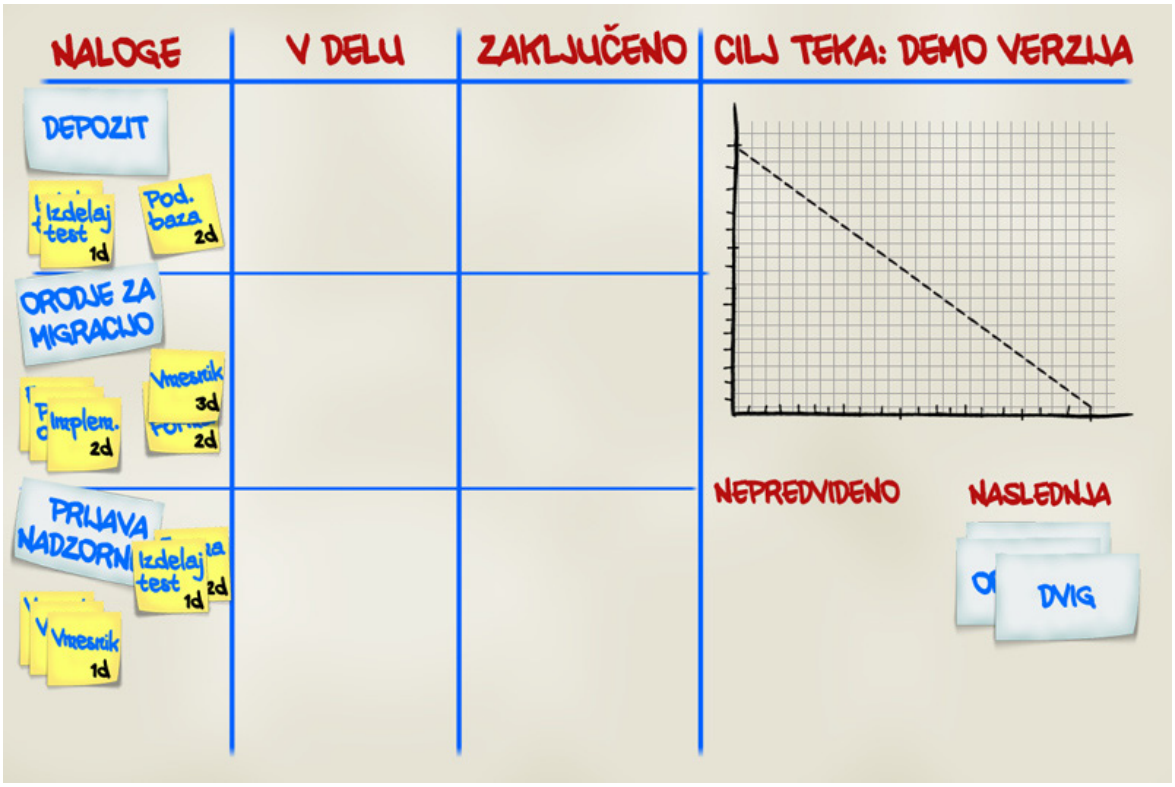
Sestanek je uspešen, če vsi člani ekipe, ki prisostvujejo sestanku, sobo po štirih urah zapustijo dobro razpoloženi in motivirani za čim prejšnji začetek in prvi dnevni Scrum sestanek.

6.5 Izdelava seznama zahtev teka

Seznam zahtev za tek izdela vodja Scruma po zaključenem načrtovalnem sestanku in še pred prvim dnevnim Scrum sestankom.

6.5.1 Oblika seznama zahtev za tek

Za predstavitev seznama zahtev za tek in potek teka se je za najbolj pregleden, preprost in učinkovit način izkazala velika steklena stena, ki se jo uporablja skupaj s samolepilnimi listki, ki označujejo posamezne naloge in ki se jih premika glede na stanje naloge. Na steno je mogoče pisati tudi opombe, ki se jih potem zbriše. Tablo uporabljamo kot pomoč pri skiciranju konceptov in načrtov zasnove ter za razlago posameznih delov.



Slika 14: Primer stene opravil za potek teka

Kot je razvidno iz Slike 14 je stena opravil sestavljena iz več stolpcev in vrst. Po vrstah so razdeljene posamezne zgodbe, ki so tudi urejene po prioriteti od zgoraj navzdol, listki pa predstavljajo naloge, ki so potrebne za zagotovitev te zgodbe.

V prvem stolpcu so naloge, ki jih še nihče ni prevzel oziroma na njih ta dan nihče ne dela. V drugem stolpcu so naloge, na katerih poteka trenutno delo, tretji pa prikazuje že uspešno zaključene naloge.

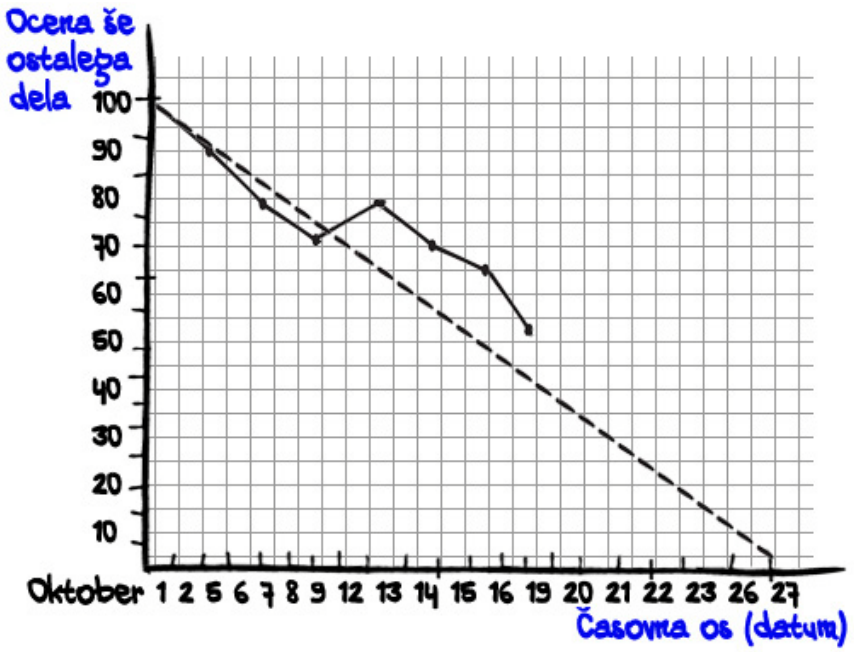
Na desni strani je še graf realizacije zahtev [10] oziroma stopnja opravljenih nalog, ki se dnevno spreminja in dopolnjuje glede na stanje razvoja in zaključenih nalog oziroma opravljenega dela. Pod grafom se nahajata še dva prostora. V prvi prostor se odlaga naloge, ki niso bile načrtovane, a so nujne za izvedbo celotnega teka. Zaradi takih nepredvidenih nalog lahko pride do morebitnega odvzemanja nalog uvrščenih v trenutni tek, zato da se lovi določen časovni rok. V drugem prostoru pa se nahajajo dodatne naloge oziroma naloge, ki se bodo izvajale v naslednjem teku, ker niso bile uvrščene v trenutnega. Če so vse obvezne naloge že uspešno zaključene pred koncem teka, imajo razvijalci čas za reševanje dodatnih.

Vsak razvijalec, ki vzame nalogo iz predala nalog in prestavi v drugi stolpec, se na listek podpiše oziroma ga tudi barvno označi s svojo barvo, da je jasno, kdo dela na kateri nalogi, kar omogoča lažji nadzor in obveščenost ostalih članov razvojne ekipe.

Slika 15: Primer zapolnjene stene opravil

6.5.2 Graf realizacije zahtev

Graf realizacije zahtev prikazuje delo, ki je že uspešno opravljeno glede na začetne ocene, koliko človek/dni bo tek potekal. Na levi koordinati je število opravil, ki jih je treba še zaključiti glede na časovno oceno, spodaj pa je mera, koliko koledarskih dni je še do konca teka.

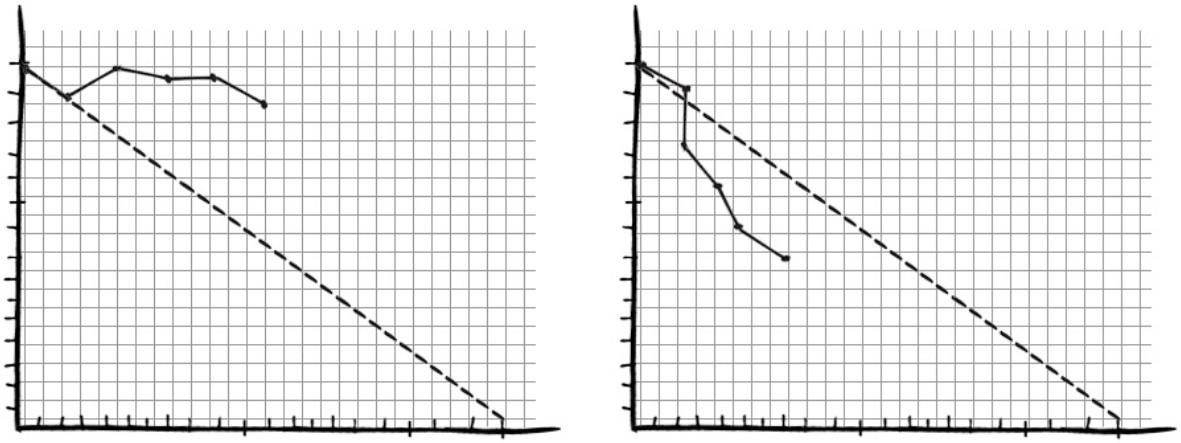


Slika 16: Primer grafa realizacije zahtev

Optimalni potek realizacije je ravna povezava med vsem opravljenim delom pred začetkom teka in vsem opravljenim delom na zadnji dan teka. Vendar pa je to v realnosti krivulja, ki od dneva do dneva niha in zato je treba naloge prirejati, da se krivulja kar najbolj drži optimalne poti. To vključuje dodajanje novih nalog in odstranjevanje nalog, ki so prezahtevne oziroma ponovno oceniti in skrčiti brez odvzemanja glavnih funkcionalnosti izdelka na koncu teka.

6.5.3 Pravilno branje stene opravil

Že na prvi pogled je določljivo trenutno stanje teka, za katerega je odgovoren vodja Scruma. Če je pri grafu realizacije zahtev krivulja nad optimalno potjo poteka, je jasno, da se mora nekaj nalog odstraniti, če pa je krivulja pod optimalno potjo, pa je treba dodajati nove naloge. Tako se tek konča na določen datum.



Slika 17: Primera nepravilne realizacije zahtev

Prvi primer se lahko zgodi, če je bilo na nalogah delo napačno ocenjeno ali pa je prišlo do nepričakovanih in nenačrtovanih opravil skozi tek. To se na steni opravil vidi na grafu realizacije zahtev ali pa na veliki količini listkov na nepravilnih mestih (desno spodaj).

Stena opravil je tudi odličen način, kako slediti poteku teka tako, da se steno na koncu vsakega dneva fotografira, kar omogoča pregled na dnevnim dogajanjem.

6.5.4 Ocenjevanje človek/dan

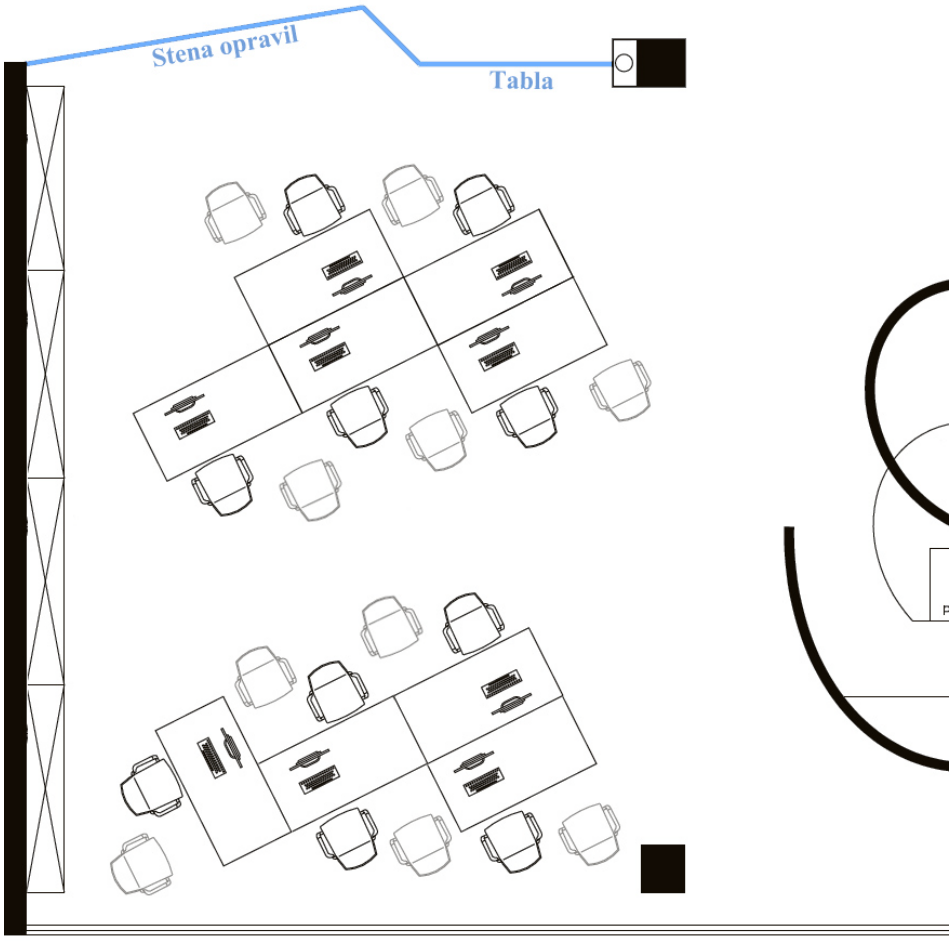
V podjetju smo se odločili za oceno človek/dan zaradi izkušenj s časovnim upravljanjem in upravljanjem virov v tej enoti. Ta enota se je izkazala za dovolj majhno glede na trajanje teka in hkrati ne premajhno, da bi prihajalo do preveč dela z organizacijo ur in ljudi na projektih. Ocen manjših od polovice človek/dni ne uporabljamo.

6.6 Razvojna ekipa

V podjetju imamo prostore organizirane tako, da so celotni oddelki v skupnem prostoru. Komunikacija znotraj razvojne ekipe zaradi tega poteka brez večjih težav; z delovnega mesta iz oči v oči in brez pomoči drugih medijev.

Večino najbolj zanimivih in odločilnih pogovorov o arhitekturi se zgodi v debati pred tablo in steno opravil. Ureditev prostora in sobe je temu prilagojena in pušča dovolj prostora okoli stene opravil in table. Na istem mestu potekajo tudi dnevni sestanki.

Produktni vodja ni v istem prostoru kot razvojna ekipa, ampak je na voljo ekipi za morebitna pojasnila in dodatne informacije. V neposredni bližini razvojne ekipe je tudi vodja Scruma, ki nadzira in skrbi za tekoč potek teka glede na zastavljeno metodologijo in cilje.



Slika 18: Tloris prostorov oddelka razvoja

6.6.1 Prostorska razdelitev ekipe, delo od doma

Trenutno se še nismo soočili s prostorsko razdelitvijo ekipe, saj so vsi razvijalci vedno v istem prostoru. V primeru, da katerim članov razvojne ekipe bolj ustreza delo od doma ali v poznejših urah, to dovoljujemo, vendar ne več dni zaporedoma in le ob predhodnem dogovoru, da ne pride do prevelike odtujitve od ostale razvojne skupine. Kljub delu od doma mora biti razvijalec dosegljiv za vodjo Scruma, predvsem v času za Scrum dnevni sestanek. Komunikacijski kanal, ki se uporablja znotraj ekipe med razvojem je v tem primeru, ko niso vsi v istem prostoru, eden izmed internetnih pogovornih odjemalcev. V našem primeru je to program Skype[3].

Ekipa se je sčasoma prilagodila načinu dela in tudi tisti, ki so bili vajeni delati ob drugih urah in od doma, so že po nekajkratni ponovitvi tekov pristopili k ostali ekipi in delali po enakem načelu.

---

[3] Skype - program za medmrežno komuniciranje med uporabniki. Omogoča tekstovne, glasovne in video medsebojne ali konferenčne pogovore.

6.7 Dnevni Scrum

Dnevni Scrum sestanki potekajo, kot omenjeno, vsako dopoldne ob 10. uri na sredi sobe razvojne ekipe ne več kot deset minut. Ekipa se je zelo dobro privadila na način izvedbe sestankov in na to, kako dobro poročati.

6.7.1 Posodabljanje stene opravil in seznama zahtev teka

Med dnevnim sestankom pride do vnašanja sprememb na steno opravil. Ko vsak opiše, kaj je opravil pretekli dan in kaj ima v načrtu za delo v tekočem dnevu, vodja Scruma, ki vodi sestanek, premakne listek na steni opravil in posodobi stanje. Hkrati tudi ažurira časovno oceno in napiše nov časovni interval ter ga dopiše na listek in prečrta starega. Vodja Scruma na koncu vsakega dnevnega Scrum sestanka povzame vse nove časovne ocene in na novo nariše graf realizacije zahtev.

Tudi seznam zahtev se posodobi vsakodnevno, v kar je prav tako vključena celotna ekipa. Vsak popravlja in dopolnjuje seznam sam.

Slika 19: Spreminjanje ocen na nalogah

6.8 Predstavitev teka

Predstavitev verzije po koncu teka je ena od pomembnejših prvin Scrum [10]. V predstavitvi vsi ostali izvedo, na čem je ekipa delala. Pozitiven dodatek so tudi pohvale za opravljeno delo.

Glede na naše izkušnje dobi ekipa na predstavitvi ključne povratne informacije in tudi ostali razvijalci, ki niso nujno sodelovali na tem projektu, imajo priložnost diskutirati o verziji in razvoju. Za predstavitev mora razvojna ekipa do datuma dejansko opraviti vse potrebno in v določeni meri zaključiti verzijo, ki je tudi primerna za testiranje.

Če ekipa v preteklem teku ni bila dobro organizirana in se predstavitev ne izide najbolje, v naslednjem teku motivacija naraste, da se napaka ne bi ponovila. Ekipa naredi vse, da popravi vtis s predhodne predstavitve.

Uspešna predstavitev vsebuje:

- Jasno predstavljene cilje teka
- Mora biti jedrnata in povzetek bistvenega, brez olepšav
- Mora biti kvalitetna, vendar se za to ne porabi preveč časa
- Opisi zagotovljenih funkcionalnosti so na poslovnem in ne preveč tehničnem nivoju
- Verzijo testirajo vsi vključeni poslušalci predstavitve

6.8.1 Kako drugače predstaviti rezultate teka

V našem primeru, zlasti na začetku gradnje sistema, so cilji teka večinoma postavljeni tako, da jih ni mogoče jasno predstaviti. V takem primeru se dosežene cilje pregleda skozi rezultate testov, ki se jih je opravljalo skozi tek in pa, ko je bila dosežena funkcionalnost verzije oziroma zastavljen cilj teka.

V primeru, da je bil cilj teka ugotoviti ali bo razviti prototip oziroma tehnologija uporabljena v teku prava izbira za nadaljnji razvoj, je na predstavitvi treba predstaviti samo razloge za in proti ter morebitne zaključke in pridobljena znanja.

6.9 Izvajanje pregleda opravljenega teka

Pregled izvajanja teka je obvezen. To opravilo je ponavadi nezaželeno, a se izkaže za učinkovito. Pregled opravljenega dela ni nujno razlog za to, da se doseže boljše rezultate v naslednjem teku. Cilj sestanka je dosežen že s tem, da lahko vsak od udeleženih nekaj vključi in dopolni. Pregled opravljenega teka je kontrola in zagotovitev, da se ekipa znova ne ujame na enakih napakah, saj se nastale napake pregleda in poskusi odpraviti.

6.9.1 Sestanek za pregleda teka

Sestanek za pregled opravljenega teka traja največ tri ure. Prisotni so celotna razvojna ekipa, produktni vodja in vodja Scruma. Lokacija sestanka je sejna soba, kjer se izvršijo tudi sestanki za načrtovanje teka. Vodja Scruma na sestanku s pomočjo razvojne ekipe na kratko povzame tek, dogodke in odločitve. Vsak član ima priložnost, da pove kaj je bilo po njegovo dobro in kaj se lahko še izboljša v naslednjem teku. V primeru, da je ocena močno odstopala od dejanskega dela, se to analizira, da se v prihodnje to prepreči. Pred koncem sestanka vodja Scruma povzame odločitve, ki se jih bo preneslo v naslednji tek kot izboljšava.

Zapisnik sestanka oziroma pregled teka je potem na voljo vsem na interni Wiki strani.

Slika 20: Tabla na sestanku pregleda teka

Med sestankom prav tako uporabljamo samolepilne listke in pa tablo, ki je razporejena na tri stolpce oziroma kriterije. V prvem so opravila, ki so potekala dobro in brez opomb, v drugi so opravila, ki jih je treba izboljšati, v tretji pa so predlogi kako ta opravila izboljšati v naslednjem teku. Ker je idej za izboljšanje več, saj lahko vsak izmed prisotnih predlaga svojo, se izvede glasovanje oziroma argumentirano prioritiziranje in najboljše ideje se potem izvede v naslednjem teku.

6.9.2 Izpostavljeni problemi pri pregledu teka na katere smo naleteli

- *Več časa je treba nameniti razbitju zgodb v naloge in opravila*
  Ekipa se je iz prvega teka naučila iz te napake in je naprej ni več ponavljala. Že na uvodnem sestanku smo dovolj časa posvetili razbitju zgodb na manjše naloge.
- *Preveč motenj od zunaj*
  Iz ostalih oddelkov so stalno prihajale motnje zaradi katerih so morali razvijalci prekiniti svoje delo in priskočiti na pomoč. Rešitev je bila beleženje motenj od zunaj, vodja Scruma pa je tako dobil večji vpogled in je lahko zagotavljal, da ne pride do motečih elementov. Sledila izdelava bolj realnega načrta izvedbe naslednjega teka.
- *Prehitro spreminjanje zahtev*
  Zaradi hitrega spreminjanja zahtev na področju razvoja programske opreme, ki jo razvojna ekipa razvija, prihaja do prehitrega spreminjanja zahtev. Rešitev za to je skrajšanje časovnega obdobja teka s štirih na tri tedne. V obdobju trajanja teka do sprememb ne sme prihajati, ampak morajo počakati na naslednji sestanek za načrtovanje teka.

6.10 Obdobje med teki

Razvojni cikli oziroma teki so zelo intenzivni in razvijalci potrebujejo oddih od vsakodnevnega programiranja in poročanja na dnevnih sestankih, o tem kaj delajo in kaj bodo delali.

Po zaključenem in pregledanem teku je celotna ekipa zagnana in polna novih idej, kako kaj izboljšati. Če se nov tek začne prehitro, se teh idej ne da predelati v glavi. Poleg tega produktni vodja takoj po predstavitvenem sestanku nima takoj dovolj časa za razporeditev novih prioritet zgodb za nov tek. Tudi sicer je kratek oddih pred naslednjim intenzivnim tekom več kot dobrodošel.

Po koncu teka in pred začetkom novega izvedemo bolj ohlapen časovni interval. Vsekakor se ne sme pregled teka in načrtovanje novega izvesti še isti dan. Sestanek za načrtovanje teka je na vrsti nekaj dni po pregledu preteklega teka, vmes pa se dneve zapolni s prostimi dnevi, ko razvijalci lahko raziskujejo svoje ideje in projekte brez pritiska. To je lahko razvijanje hobi projektov, razglabljanje o tehnoloških novostih s kolegi, izobraževanje in podobno. V tem času pride do zasnove novih idej, ki se jih lahko kasneje razvije v rednem procesu. Glede na načrtovanje tekov je ponavadi tak en dan po vsakem tritedenskem teku oziroma se ta čas prilagodi glede na časovno opredeljenost projektov in dni v tednu, torej da se npr., kot je razvidno s Slike 21, s sestankom za načrtovanje naslednjega teka začne v ponedeljek.

Četrtek	Petek	Sobota	Nedelja	Ponedeljek
09h-10h: Predstavitev rezultatov teka 10h-11h: Pregled teka	Prost dan za nerazvrščene samoiniciativne projekte			08h-12h: Načrtovanje naslednjega teka

Slika 21: Izkoriščanje dni med teki

6.11 Vključene prakse ekstremnega programiranja

6.11.1 Programiranje v parih

V samem začetku uvajanja metodologije v podjetje smo naleteli na zavračanje te metode, zato smo se tudi odločili, da ga bomo uvajali postopoma in da se nikoli ne bo izvajalo stalno, pač pa le pri določenih nalogah, kjer sta zaradi zapletenosti razvoja potrebna dva izkušena razvijalca oziroma kjer izkušenejši uvaja novega. Kolikor je mogoče, se držimo priporočil iz praks ekstremnega programiranja. V primeru, da se uporablja ta praksa je treba pri ocenjevanju trajanja naloge upoštevati zmanjšanje faktorja osredotočenosti.

6.11.2 Testiranje (Test-Driven Development)

Test-driven development (TDD) pomeni, da se za vsako zaključeno celoto kode (funkcijo), napiše tudi avtomatiziran test, ki se ga mora opraviti uspešno, oziroma se že pred pisanjem funkcije napiše ustrezen test ter nato napiše kodo, ki testu ustreza. Glede na test se potem to funkcijo izboljšuje z večkratnim pregledovanjem in predelovanjem kode. [29]

Testiranje na podlagi TDD je zahtevno in zahteva nekaj časa, da se ga privadiš. Učinki, ki jih prinese, ko je to utečen proces, so pa zato zelo pozitivni za gradnjo sistema. Na začetku gradnje novega sistema je težava v tem, katere teste uvajati. Za to se porabi več časa, vendar je izgubljeni čas na začetku razvoja vseeno bolje izrabljen kot naknadno iskanje napak in hroščev v programski kodi.

6.11.3 Inkrementalno načrtovanje

Inkrementalno načrtovanje je logična lastnost agilne metodologije. Začne se s preprostimi stvarmi, ki se jih v poznejših verzijah dopolnjuje, vendar se že na začetku zgradi delujočo verzijo, ki potem stoji in čaka ostale komponente.

Ta način izgradnje in načrtovanja se je zelo dobro prijel, saj zaradi stalnega spreminjanja zahtev in novih dognanj zelo dobro funkcionira, da se komponente gradi postopoma do končne verzije skozi več tekov. Hkrati pa je inkrementalna gradnja tudi posledica načrtovanja in programiranja s pomočjo izvajanja testov (TDD).

6.11.4 Neprestana integracija

Neprestane integracije smo se lotili tako, da smo predpisali princip, po katerem je razvijalec dolžan novo napisano kodo vsakodnevno oziroma nujno po koncu opravljene naloge integrirati v testni ritem, in s tem preveriti morebitne konflikte z ostalimi razvijalci. Vzpostavljen je bil enoten repozitorij za kodo, ki ga uporabljajo vsi razvijalci. Po integriranju nove verzije v sistem se ta koda tudi zgradi in poženejo se osnovni avtomatizirani testi, ki preverijo ustreznost nove kode.

6.11.5 Enoten standard kodiranja

Od samega začetka smo si zastavili pravila kodiranja in komentiranja kode. Za standardizacijo kodiranja smo porabili en dan, in sicer da smo vse razvijalce obvestili o načinu, ki ga bomo uporabljali. Prišlo je do izmenjave različnih idej, od katerih so bile najboljše zapisane v pravilnik o pisanju kode, ki se ga sedaj držijo vsi razvijalci. Nismo se spuščali v zadeve, ki so bile samoumevne za vse, definirali smo samo bolj kočljiva področja in pravila oziroma čemu se je treba pri kodiranju izogibati in kaj je natančneje treba komentirati.

6.11.6 Skupno lastništvo kode

Skupno lastništvo kode spodbujamo, vendar pa do tega pride le v primerih, ko ekipa deluje po principu programiranja v parih. Najboljša stvar pri skupnem lastništvu kode je predvsem to, da je ekipa kljub nepredvideni odsotnosti pomembnega člana lahko še vedno solidna, saj tudi ostali dovolj poznajo kodo, da lahko težavo hitro najdejo in razrešijo. Zaradi dela v istem prostoru, dobre komunikacije in sodelovanja med razvijalci velikokrat prihaja do skupnega lastništva samoumevno brez predhodnega načrtovanja.

6.11.7 Delovno okolje

Večina agilnih metodologij zatrjuje, da so nadure kontraproduktivne, kar se je izkazalo kot resnica tudi v našem primeru, saj je zaradi nadur prišlo do psihične izčrpanosti in zmanjšanja motivacije ekipe. Zaradi tega že pri načrtovanju teka predvidimo, da nadur ne opravljamo in v primeru, da graf realizacije zahtev preveč naraste ustrezno skrčimo funkcionalnosti, ki bodo predstavljene na koncu teka. Prav zaradi zmanjšanja nadur je prišlo do večje produktivnosti v ekipi in tudi kvaliteta kode se je vidno izboljšala.

Kot zelo produktivni so se izkazali prosti dnevi med dvema tekoma, v katerih imajo razvijalci na voljo vse vire za razvijanje svojih idej, ki se morebiti vključijo v enega izmed kasnejših tekov oziroma med funkcionalnosti izdelka, ki ga razvijamo.

6.12 Uvajanje nove združene metodologije

Pred odločitvijo za uvedbo nove organizacije v oddelek za razvoj programske opreme se znotraj oddelka ni uporabljala nobena metodologija, ampak je razvoj potekal bolj ad hoc, s popisom zahtev na začetku projekta. Zaradi zahteve vodstva po boljši organizaciji in časovnemu vodenju projektov se je uvedlo ogrodje metodologije Scrum za organizacijo vodenja projekta. Skupaj z novim projektom, ki je bil zasnovan na Scrum, se je postopoma uvedlo še inženirske prakse iz ekstremnega programiranja s katerimi skrbimo za boljšo kvaliteto samega programiranja.

Sprememba načina dela in delovnih navad vseh vezanih na razvoj projekta je lahko zelo težavna, posledice pa se lahko poznajo predvsem na neučinkovitosti. Začetno uvajanje samo osnov metodologije Scrum in vodje Scrum omogoči lažje prehajanje. Kot je bilo omenjeno pri opisu Scrum metodologije, vodja Scruma ugotavlja in odpravlja nepravilnosti pri uporabi, odpravlja komunikacijske pregrade in skrbi za nemoteno delo razvojne ekipe. Predvsem na začetku uvajanja je bila v našem primeru njegova vloga še toliko bolj zahtevna, saj je bilo treba vodstvu in vsem, ki niso vključeni v projekt, preprečiti vmešavanje v razvojno ekipo, hkrati pa ekipo prisiliti v nujnosti, kot so sestanki in uporaba inženirskih praks.

6.12.1 Uvajanje ogrodja Scrum

Na metodologijo Scrum smo prešli po 'metodi velikega poka' z uvedbo nove metodologije na novem projektu. Ekipa je od prvega dneva uvedbe projekta delala po navodilih vodje Scruma in se držala vseh predpisanih pravil in zahtev metodologije.

Pred samim začetkov novega projekta, ki je temeljil na novi metodologiji, je bila ekipi v enodnevnem izobraževanju predstavljena nova oblika organizacije, za vsa nadaljnja vprašanja pa so se še skozi celoten prvi tek obračali na vodjo Scruma.

6.12.2 Uvajanje prakse testiranja

Glede na strukturo razvojnega kadra v ekipi ni bilo težav z vpeljavo testiranja v proces pisanja kode. Kot je določeno v praksi ekstremnega programiranja, se pisanje testov načrtuje še pred pisanjem kode, tako da se koda prilagaja potrebam testa. Tudi zaradi koncepta je še bolj očitno, da je za kvaliteten razvoj treba komponente razvijati čim bolj neodvisno med seboj. Neodvisnost zagotavlja tudi lažjo ponovno uporabo kode in preoblikovanje obstoječe kode za nove primere.

Testi morajo biti zelo natančno določeni, saj se nadaljnji razvoj odvija okrog zagotavljanja pravilnih rezultatov testov. Testi so strukturirani tako, da predvidevajo uporabo razredov in metod, od katerih se pričakuje določene rezultate. V primeru odsotnosti razredov in metod, je treba teste napisati tako, da prinašajo pričakovane rezultate. Preverjajo se vse mejne vrednosti, ki se lahko oziroma se ne smejo pojaviti. Iz nekaterih testov je tudi takoj razvidno, kakšna je namembnost kode.

Osvojitev prakse testiranja je bila preprosta in že v sklopu prvega teka so jo vsi razvijalci brez težav uporabljali. Čeprav se kar nekaj časa na začetku porabi za pisanje primernih testov, pa se končen rezultat izkaže za kakovostnejšega, saj je končna koda bolj stabilna in vsebuje manj napak in hroščev, zaradi katerih se lahko razvoj nepredvideno podaljšuje.

6.12.3 Uvajanje prakse preoblikovanja kode

Čisto razumljivo je, da je najtežji del pri preoblikovanju kode popravljanje že delujoče kode. Nevarno je namreč, da z majhnim popravkom silimo v nadaljnje popravke, ki pa so lahko bolj obsežni in časovno zahtevni. Zato je smiselno izvajati to prakso samo v primeru, ko že obstoječa koda, ki jo želimo implementirati, ne dopušča enostavnega dodajanja novosti. Preoblikovanje kode brez ustreznega testiranja ni priporočljivo, saj se hitro lahko podre funkcionalnost kode. Preoblikovanje kode poteka tako, da se ob dodajanju nove kode le te še ne spreminja. To se naredi šele pri tretjem preverjanju kode.

6.12.4 Uvajanje prakse neprestane integracije

Neprestana integracija je bila tretja stopnička pri uvajanju praks iz ekstremnega programiranja v okvir razvoja. Neprestana integracija je mogoča, če so deli oziroma moduli programa točno opredeljeni in preprosto zastavljeni. Ker je testiranje že dovolj ukoreninjeno v delo in razvijalci uporabljajo tudi prakso preoblikovanja, je neprestana integracija samoumevno naslednja stopnica. Razvijalec mora probleme še podrobneje razdeliti na najpreprostejše sestavne dele, ki skupaj gradijo končno celoto in ki jih je moč testirati in integrirati brez težav.

6.12.5 Uvajanje ostalih praks

Ko so se prijele glavne tri prakse iz metodologije ekstremnega programiranja in ko smo usvojili vodstveno organizacijo po metodologiji Scrum, se je postopoma uvajalo še ostale inženirske prakse glede na zahtevnost v vsakem novem teku. Za to skrbi vodja Scruma, na katerega se vsi ostali obračajo v primeru kakršnegakoli problema ali nejasnosti v zvezi z organizacijo.

6.13 Načrtovanje

Načrtovanje verzij izdajanja do končnega izdelka je zahtevno delo, vendar zaradi podatkov, ki jih s tem načrtovanjem dobimo, izredno pomembno. To je časovna ocena izdaje prve delujoče verzije novega sistema.

6.13.1 Določitev praga odobritve

Produktni vodja na začetku določi pragove pomembnosti zahtev na seznamu. V praksi morajo biti zahteve, ki so označene s pomembnostjo nad 100 brezkompromisno vključene že v prvo verzijo. Naslednja stopnja je namenjena tistim zahtevam, ki morajo tudi že biti vključene v prvo verzijo, vendar je sprejemljivo, če se jih premakne v prvo naslednjo stopnjo.

Glede na obseg projekta je teh stopnic lahko še več. V zadnji so vedno zahteve, ki so predvidene za vključitev šele v zadnjo verzijo ali pa se jih lahko tudi izključi iz razvoja. Glede na območje prioritet zahtev po verzijah seznam zahtev dopolnimo z barvami, pri kateri je rdeča najpomembnejša, zelena pa pomeni najnižje prioritete, ki so nekakšne dodatne funkcije.

Torej, če do konca teka izpolnimo vse rdeče in rumene zahteve, je tek uspešno zaključen. Če nam zmanjka časa, je sprejemljivo odstraniti zahteve iz rumenega seznama, nikakor pa ne iz rdečega. V primeru, da se lotimo tudi zelenih zahtev, je to dodatno delo, ki ni bilo načrtovano.

Prioriteta	Naziv
130	Naloga Ena
120	Naloga Dve
115	Naloga Tri
110	Naloga Štiri
100	Naloga Pet
90	Naloga Šest
80	Naloga Sedem
70	Naloga Osem
60	Naloga Devet
40	Naloga Deset
35	Naloga Enajst
10	Naloga Dvanajst
10	Naloga Trinajst

Rdeča – *mora biti vključeno v verziji 1.0 (od najpomembnejše naloge Ena do naloge Pet)*

Rumena – *naj bi bilo vključeno v verzijo 1.0 (od naloge Šest do naloge Devet)*

Zelena – *lahko so implementirane kasneje (od naloge Deset do naloge Trinajst)*

Slika 22: Primer prioritetnega seznama

6.13.2 Časovne ocene

Med ocenjevanjem izvršitve zahtev za tek je ključno časovno ocenjevanje za izdajanje verzij, ki poteka po enakem principu. Produktni vodja kot tudi razvojna ekipa podata oceni za izvedbo. Ocena produktnega vodje so ponavadi preveč optimistične, zato upoštevamo predvsem oceno razvojne ekipe, ki poda bolj smiselno okvirno oceno, ki pa ni strogo obvezujoča.

Prioriteta	Naziv	Ocena dela
130	Naloga Ena	12
120	Naloga Dve	8
115	Naloga Tri	18
110	Naloga Štiri	2
100	Naloga Pet	20
90	Naloga Šest	12
80	Naloga Sedem	10
70	Naloga Osem	8
60	Naloga Devet	10
40	Naloga Deset	14
35	Naloga Enajst	8
10	Naloga Dvanajst	3
10	Naloga Trinajst	2

Slika 23: Primer dokončnega seznama s časovnimi ocenami

6.13.3 Izdelava načrta izdaje verzij

Ko je narejena ocena hitrosti razvoja se seznam zahtev izdelka razdeli na teke. Vsak tek ima na seznamu za izdelavo toliko zgodb, da ne preseže meje ocene hitrosti ekipe. Glede na

primer so to trije teki, v časovnem okviru pa sta to dva meseca. Na koncu vsakega teka produktni vodja vidi do neke stopnje zaključen, delujoč izdelek.

Prioriteta	Naziv	Ocena dela
		Tek 1
130	Naloga Ena	12
120	Naloga Dve	8
115	Naloga Tri	18
110	Naloga Štiri	2
		Tek 2
100	Naloga Pet	20
90	Naloga Šest	12
80	Naloga Sedem	10
		Tek 3
70	Naloga Osem	8
60	Naloga Devet	10
40	Naloga Deset	14
35	Naloga Enajst	8
10	Naloga Dvanajst	3
10	Naloga Trinajst	2

Slika 24: Razdelitev na teke

Na koncu vsakega teka se izvede pregled, ali se je dejanska hitrost razvoja močno razlikovala od ocenjene. Če pride do prevelikega odstopanja, je treba ocenjevalne kriterije za nadaljnje teke ponovno pregledati in spremeniti. Napačna ocena pripelje do povečanja števila tekov oziroma zmanjšanja zahtev za izdajo končne verzije.

6.14 Končno testiranje

Testiranje je ključno opravilo pri agilnem razvoju. Poleg uporabe integracijskih testov na koncu vsakega teka, če je le mogoče, izvajamo skupaj s uporabniki sistema še končno testiranje, ki lahko prinese spremembe v nadaljnjem razvoju.

6.14.1 Sprejemljivost verzije

Po koncu vsakega teka naj bi bila po navodilih Scrum verzija pripravljena za zagon oziroma uporabo, vendar pa je to preveč idealiziran model. Ponavadi je treba po končanem teku verzijo testirati. Za to se odredi novo ekipo, ki izvaja testiranje, sporoči razvojni ekipi napake, ki so bile ugotovljene, ta jih odpravi in ponovno preda v testiranje, dokler ni verzija zadovoljiva za zaključek teka.

6.14.2 Skrajšanje zaključne testne faze in povečanje kvalitete testiranja

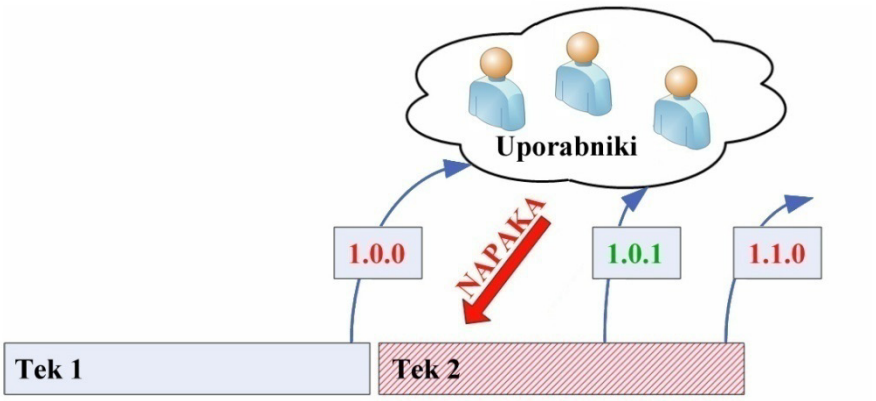
Zaključno testiranje po končanem teku je največja šibka točka, ki lahko nadaljnji razvoj časovno poveča. Da bi to fazo testiranja čim bolj skrajšali, smo se osredotočili na povečevanje učinkovitosti ročnih testov in izdajanje kakovostnejše kode.

Do kakovostnejše kode smo prišli s pomočjo uvajanja enega člana v razvojno ekipo, ki je zadolžen izključno za testiranje in preverjanje kode ter zaključenih opravil. V ekipo je vključen kot ostali člani in zadolžen za razvoj istega izdelka, ima pa glavno besedo pri tem, kdaj je neka naloga zaključena in je izdan uradni dokument zaključka. Naloga tega člana ekipe je, da organizira in pripravi vse potrebno za predstavitev teka.

Član, odgovoren za testiranje, je vključen v ekipo od prvega dne in se pridružuje posameznim razvijalcem, medtem ko pišejo teste, jih usmerja, da bodo testi učinkoviti in skrbi za knjižnico testov. Pred začetkom teka in v času načrtovalnega sestanka je ta oseba zadolžena za pripravo testnega okolja, opredelitev zahtev, razbitje zgodb na naloge in opravila ter odgovarjanje na ključna vprašanja razvijalcev.

6.14.3 Testiranje se nadaljuje po zaključku teka

Testiranje, ponovna izdaja in razhroščevanje je zelo težko časovno oceniti. Glede na zahteve metodologije Scrum in praks ekstremnega programiranja sicer ne bi smelo priti do napak na koncu teka, saj ekipa preda že zaključen izdelek pripravljen za v uporabo, pa tudi razvoj je sproti potekal s pomočjo vnaprej napisanih testov. V praksi vseeno ni vedno vse popolno, zato je testiranje obvezno tudi po zaključku teka.

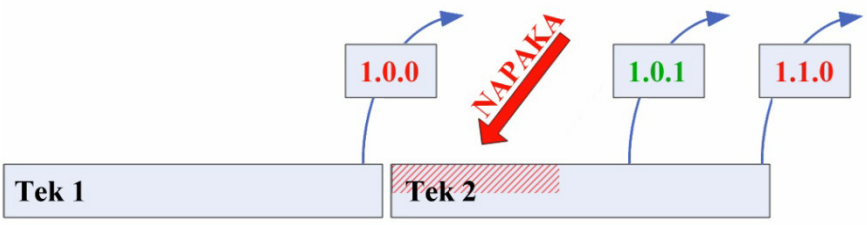


Slika 25: Prikaz realne situacije pojavitve napake prejšnje verzije med novim tekom

Po prvem teku je izdana verzija, ki se jo preda v uporabo oziroma testiranje v produkcijskem okolju. Med potekom naslednjega teka se začnejo izdajati poročila o napakah in hroščih iz prejšnje verzije. Lahko pride do situacije, ko je nujnejše reševati napake iz prejšnjega teka,

kot pa da se po načrtu opravlja tekoči tek. Možnosti napake smo zmanjšali že med gradnjo verzije s pomočjo testiranja, vendar pa do tega lahko vseeno pride.

V podjetju te težave rešujemo tako, da kljub morebitnim napakam nadaljujemo z naslednjim tekom. Vseeno imamo nekaj časa v naslednjem teku predvidenega za reševanje hroščev iz predhodne verzije, ki je v testiranju in tako ustrezno prilagodimo tudi faktor osredotočenosti pri računanju hitrosti izvedbe. O tem presodimo že na načrtovalnem sestanku. V primeru, da odpravljati napak ni treba, se glede na nivo grafa realizacije zahtev v tek dodaja nove zahteve.



Slika 26: Prikaz, kako se rešuje napake prejšnjih verzij med novim tekom

V enem od začetnih tekov smo zaradi napačnega načrtovanja na koncu ugotovili večje število hroščev, zato smo na naslednjem sestanku za načrtovanje teka celoten, časovno krajši tek namenili odpravljanju hroščev in izpopolnjevanju programske kode. V najslabšem primeru bomo tako še naprej reševali sorodne težave.

6.15 Upravljanje več ekip

Zaradi načina dela Scrum ni učinkovit, če je število ljudi v ekipi preveliko [10]. Ker pri nas v razvoju na istem projektu ponavadi dela deset ljudi, je bilo treba organizirati dve ekipi. Zaradi tega je treba ustrezno prilagoditi tudi načrt razvoja na dve različni poti. Prvi pogoj za več ekip je seveda, da ima vsaka ekipa svojo steno opravil.

V primeru več razvojnih ekip se pojavi vprašanje, kakšen način razvoja je najbolj primeren. Teki ekip bi lahko potekali časovno zamaknjeno, tako da bi imel produktni vodja delo enakomerno porazdeljeno. Avtor Scruma Ken Schwaber [9] priporoča, da vse ekipe tek začnejo in končajo istočasno. Do zdaj smo se držali slednjega, saj imamo največ dve ekipi. Prednosti tega so, da se lahko ekipe po koncu teka na novo razdeli brez večjih ovir in motenj teka. Prav tako lahko vse ekipe hkrati stremijo k istem cilju, kar poveča povezanost in sodelovanje med razvijalci. Navsezadnje je tudi manj administrativnega dela, manj skupnih sestankov, predstavitev in izdaj.

Zaradi majhnega števila razvijalcev in največ dveh razvojnih Scrum ekip je en vodja Scruma za obe ekipi dovolj. V primeru, da se naš razvoj razširi na večje število ljudi pa je v načrtu, da

trenutni vodja Scruma prenese svoje naloge na nove vodje Scruma. Med njimi je eden, ki razporeja ljudi v ekipe, nekdo pa vodi vse vodje Scruma.

Med teki ne spreminjamo razvojnih ekip, razen ko pride do zaključne verzije ali pa neke mejne točke v razvoju. V takih primerih lahko pride tudi do tega, da se neki ekipi dodeli začasnega člana, vendar res samo v izjemnih primerih, saj nočemo porušiti sistema odgovornosti v ekipi.

6.15.1 Specializirane ekipe

Tip programske rešitve, ki jo razvijamo v podjetju je sestavljen iz treh delov (odjemalec, strežnik in podatkovna baza) in večina tekov obsega delo na vseh treh področjih. Zato razvojne ekipe v večini tekov niso specializirane oziroma usmerjene samo v delo na enem področju, saj večina zgodb, ki jih predstavi produktni vodja vedno vključuje vse tri nivoje. Eno ekipo smo v nekaj tekih vendarle specializirali, saj so določene funkcionalnosti sistema vezane samo na odjemalca. Prav tako v primeru, ko so potrebne specifične dopolnitve, organiziramo specializirano ekipo, ki dela samo na določenem nivoju celotnega produkta.

6.15.2 Upravljanje seznama zahtev za več ekip

Možnih je več načinov za upravljanje seznama zahtev v primeru več ekip. Glede na velikost podjetja, specifiko produkta, ki ga razvijamo, in število ekip nam zadošča, da uporabljamo samo en seznam zahtev, ki si ga potem ekipi na prvem sestanku pred začetkom teka razdelita glede na zgodbe, ki jih bosta razvijali.

6.16 Predstavitev kontrolnega seznama vodje Scruma

Vodja Scruma je posebna vloga vodje projektov, ki je ob združitvi Scrum z ekstremnim programiranjem dobila dodatne administrativne naloge. Poleg spodaj naštetih nalog ima vodja Scruma tudi glavno besedo pri poteku, organizaciji virov in morebitni prekinitvi teka iz nepredvidenih razlogov. Kontrolni seznam vodje Scruma je sledeč.

Ob začetku teka:

- Po prvem načrtovalnem sestanku ustvariti informativno stran v Wiki:
    - Enake podatke natisniti in prilepiti ob steno opravil
- Obvestiti vse, da se je začel nov tek, z informacijami in povezavo na Wiki stran
- Ažurirati glavni dokument teka z oceno trajanja, hitrosti, velikosti ekipe, dolžino teka, itd.
- Zagotoviti je treba prostor in določiti čas za dnevni Scrum sestanek

Vsakodnevno:

- Zagotoviti začetek in konec Scrum sestanka glede na časovna določila
- Zagotoviti, da so zgodbe dodane/odvzete iz seznama zahtev tako, da bo tek v skladu s časovnimi roki
    - Ustrezno obveščati o tem produktnega vodje
- Zagotoviti ažurnost podatkov na steni opravil in seznamu zahtev
- Zagotoviti, da so težave in ovire rešene oziroma sporočene produktnemu vodji oziroma vodji razvoja

Po koncu teka:

- Izvesti je treba predstavitev rezultatov teka
    - Obvestiti vse, ki bodo prisostvovali predstavitvi
- Izvesti je treba pregled teka s celotno ekipo in produktnim vodjo ter vanj vključiti tudi vodjo razvoja
- Ažurirati je treba glavni dokument teka in dodati dejansko hitrost in glavne točke iz pregleda teka

7 Sklepne ugotovitve

Metodičnost in sistematičnost delovanja kakršnekoli dejavnosti je dandanes zaradi učinkovitosti in predvsem časovne optimizacije dela pomemben dejavnik v uspešnosti podjetja. V povezavi s tem je pomembna tudi izbira primerne metodologije glede na delovno področje določenega podjetja.

Pričujoče delo predstavlja poskus čim bolj natančne prilagoditve najprimernejših metodoloških praks na primeru dejavnosti realnega podjetja. Teoretični del predstavlja pregled tradicionalnih in novih metodologij za razvoj programske opreme, medtem ko je v drugem delu zajet predlog načina združitve dveh ustrezno povezanih in potrebam realnega podjetja prilagojenih agilnih metodologij. Združitev vključuje kompatibilni metodologiji ekstremno programiranje in Scrum, pri katerih ogrodje Scrum zagotavlja izdelano vodenja projekta, medtem ko ekstremno programiranje prispeva dobre inženirske prakse oziroma kvaliteto izgradnje programske kode.

Na podlagi skoraj polletne izkušnje z uvajanjem prilagojene metodologije v razvoj programske opreme in njene uporabe, lahko zaključim, da se je agilni princip v tej obliki izkazal za pravega. Vodstvo je dobilo več in boljše informacije o časovnem poteku projekta za načrtovanje finančnega bremena, razvojne ekipe pa so se na nov način dela postopoma navadile, medtem ko so usvajale drugačen način programiranja po novih principih.

Preskok iz ad hoc organiziranosti k organiziranju po metodah Scrum v enem koraku torej ni povzročal večjih problemov. Nekaj več prilagajanja na nov sistem je zahtevala vpeljava inženirskih praks, ki pa so se pokazale kot nujnost za kvalitetno dopolnitev gradnje programske kode. Kljub uspešnemu prevzemu nove organizacije in postopkov je v načrtu podjetja postopke digitalizirati, kolikor je to mogoče, in preiti na popolno elektronsko vodenje in organizacijo. S pomočjo odprtokodnih orodij, ki jih bomo poskusili vključili v interni Trac portal, bomo digitalizirali tudi steno opravil.

Poskus bi za večjo verodostojnost rezultatov oziroma morebitno kategorizacijo vrste obravnavane metodologije na specifičen tip podjetja moral potekati dlje časa oziroma bi se v analizo moralo vpeljati več podjetij. Morda bi se izkazalo, da je aplikacija možna samo v primeru obravnavanega malega podjetja, kljub temu pa z vso gotovostjo lahko rečem, da je naloga dosegla vse na začetku zastavljene cilje.

Slike

Tabele

Literatura in viri

[1] Abrahamsson, P., Saio, O., Ronkainen, J., Warsta, J., »Agile Software Development methods, Review and analysis«, VTT Publications, 2002

[2] Ambler S., »Agile Database Techniques: Effective Strategies for the Agile Software Developer«. Wiley, 2003

[3] Bajec M., Krisper M., »Agilne metodologije razvoja informacijskih sistemov«, Uporabna informatika. april, maj, junij 2003, 11(2), str. 68-76

[4] Beck, Kent, »Extreme Programming Explained: Embrace Change«, Addison-Wesley, 1999

[5] Cecil M. R., »Agile software development: principles, patterns, and practices«, Prentice Hall, cop., 2003

[6] Cockburn, A., »Agile Software Development«, Pearson Education Inc., 2002

[7] Highsmith J., Cockburn A: »Agile software development ecosystems«, Addison-Wesley, 2002

[8] Kniberg H., »Scrum And XP From The Trenches«, 2nd edition, Lulu.com, 2007

[9] Schwaber K., »Agile Project Management with Scrum«, Microsoft Press, cop., 2004

[10] Schwaber K., Beedle M., »Agile Software Development with Scrum«, Prentice Hall, cop., 2002

[11] Warden S., Shore J., »The Art of Agile Development«, O'Reilly, 2007

[12] (2009) Agile Alliance,
Dostopno na: http://www.agilealliance.org/

[13] (2009) Agile manifesto,
Dostopno na: http://www.agilemanifesto.org/

[14] (2009) Agile modeling,
Dostopno na: http://agilemodeling.com/

[15] (2009) Agile Software Development,
Dostopno na: http://en.wikipedia.org/wiki/Agile_software_development

[16] (2009) Behavior Driven Development,
Dostopno na: http://en.wikipedia.org/wiki/Behavior_Driven_Development

[17] (2009) Continuous integration,
Dostopno na: http://en.wikipedia.org/wiki/Continuous_integration

[18] (2009) Crystal Clear,
Dostopno na: http://en.wikipedia.org/wiki/Crystal_Clear_software_development

[19] (2009) – Feature driven development,
Dostopno na: http://en.wikipedia.org/wiki/Feature_Driven_Development

[20] (2009) Implementing Scrum,

Dostopno na: http://www.implementingscrum.com/
[21] (2009) Iterative and incremental development,
Dostopno na: http://en.wikipedia.org/wiki/Iterative_and_incremental_development
[22] (2009) Lean software development,
Dostopno na: http://en.wikipedia.org/wiki/Lean_software_development
[23] (2009) Pair programming,
Dostopno na: http://en.wikipedia.org/wiki/Pair_programming
[24] (2009) RITE METHOD,
Dostopno na: http://en.wikipedia.org/wiki/RITE_Method
[25] (2009) Software development methodology,
Dostopno na: http://en.wikipedia.org/wiki/Software_development_methodology
[26] (2009) Software development methodologies,
Dostopno na: http://users.encs.concordia.ca/~b_zamani/Methodologies.pdf
[27] (2009) Software prototyping,
Dostopno na: http://en.wikipedia.org/wiki/Software_prototyping
[28] (2009) Spiral model,
Dostopno na: http://en.wikipedia.org/wiki/Spiral_model
[29] (2009) Test-driven development,
Dostopno na: http://en.wikipedia.org/wiki/Test-driven_development
[30] (2009) Waterfall model,
Dostopno na: http://en.wikipedia.org/wiki/Waterfall_model