Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Miha Gruden

Izboljšanje razvojnega procesa programske opreme na primeru sistema za bančno poslovanje

magistrsko delo
Magistrski program Informacijski sistemi in odločanje

Mentor: prof. dr. Franc Solina

Ljubljana, 2014

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št.: 129-MAG-ISO/2014
Datum: 16. 05. 2014

Miha GRUDEN, univ. dipl. inž. rač. in inf.

L j u b l j a n a

Fakulteta za računalništvo in informatiko Univerze v Ljubljani izdaja naslednjo magistrsko nalogo

Naslov naloge: **Izboljšanje razvojnega procesa programske opreme na primeru sistema za bančno poslovanje**

Information system development process improvement in the case of banking system

Tematika naloge:
V eni izmed slovenskih bank lasten razvoj programske opreme predstavlja pomemben delež v celotnem informacijskem sistemu banke. Eden izmed večjih in uspešnejših projektov je tudi razvoj sistema za bančno poslovanje, katerega razvojni proces in rezultati bodo v nalogi tudi podrobno predstavljeni. Podrobno bodo opisane aktivnosti obstoječega razvojnega procesa in izdelki (dokumentacija, programska koda), ki nastajajo tekom procesa.

Kljub zelo dobrim rezultatom projekta in zadovoljstvu končnih uporabnikov se je tekom let razvoja v procesu dela identificiralo tudi pomanjkljivosti in možnosti izboljšav. Veliko se jih je sproti vključevalo in stalno izboljševalo razvojni proces.
V zadnjem času pa nas v spremembe razvojnega procesa sili predvsem želja po čim hitrejši implementaciji novih poslovnih zahtev (angl. time to market), čemur bo posvečen glavni del magistrske naloge. Predlagane spremembe razvojnega procesa bodo osredotočene na agilne in vitke principe razvoja programske opreme.
Cilj je, v obstoječem razvojnem procesu identificirati aktivnosti, ki jih je mogoče izboljšati in predlagati natančne postopke izboljšav teh aktivnosti s prej omenjenimi modernimi principi razvoja.

Mentor:
prof. dr. Franc Solina

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Dekan:
prof. dr. Nikolaj Zimic

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V eni izmed slovenskih bank lasten razvoj programske opreme predstavlja pomemben delež v celotnem informacijskem sistemu. Eden izmed večjih in uspešnih projektov je tudi razvoj sistema za bančno poslovanje, katerega razvojni proces in rezultati bodo v nalogi tudi podrobno predstavljeni. Podrobno bodo opisane aktivnosti obstoječega razvojnega procesa in izdelki (dokumentacija, programska koda), ki nastajajo tekom procesa.

Kljub zelo dobrim rezultatom projekta in zadovoljstvu končnih uporabnikov se je tekom let razvoja v procesu dela identificiralo tudi pomanjkljivosti in možnosti izboljšav. Veliko se jih je sproti vključevalo in stalno izboljševalo razvojni proces. V zadnjem času pa nas v spremembe razvojnega procesa sili predvsem želja po čim hitrejši implementaciji novih poslovnih zahtev (angl. time to market), čemur bo posvečen glavni del magistrske naloge. Predlagane spremembe razvojnega procesa bodo osredotočene na agilne in vitke principe razvoja programske opreme. Cilj je, v obstoječem razvojnem procesu identificirati aktivnosti, ki jih je mogoče izboljšati in predlagati natančne postopke izboljšav teh aktivnosti s prej omenjenimi modernimi principi razvoja.

Izjava o avtorstvu magistrskega dela

Spodaj podpisani Miha Gruden, z vpisno številko **63960039**, sem avtor magistrskega dela z naslovom:

Izboljšanje razvojnega procesa programske opreme na primeru sistema za bančno poslovanje

S svojim podpisom zagotavljam, da:

- sem magistrsko delo izdelal samostojno pod mentorstvom prof. dr. Franca Soline,
- so elektronska oblika magistrskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko magistrskega dela,
- soglašam z javno objavo elektronske oblike magistrskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, september 2014 Podpis avtorja:

Najprej se moram zahvaliti svoji ženi za spodbudo in podporo pri izdelavi naloge. Predvsem za prikrajšane večere in kak dopustniški dan. Potem so tu sodelavci v razvoju sistema za bančno poslovanje s katerimi smo skupaj veliko naredili in sproti izboljševali svoje delovne procese. Posebej še Urošu s katerim sva veliko potencialnih izboljšav večkrat prediskutirala in jih potem preizkusila v praksi. Zahvaljujem se prof. Solini za strokovno podporo, usmerjanje, ažurnost in prijaznost. Pa seveda Alešu za pomoč v banki, pri pregledu naloge in podanih zelo dobrih predlogih. Brez vas ne bi šlo, ali pa bi se še zavleklo. Zato iskreno Hvala!

Za Žiga, ki je šel večkrat spat z besedami,
da gre pa oči še pisat tole knjigico:)

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
IT	information technology	informacijska tehnologija
UML	unified modeling language	jezik za enotno modeliranje
DML	data manipulation language	jezik za ažuriranje podatkov
FE	front-end	razvoj uporabniškega vmesnika
BE	back-end	razvoj zalednihi sistemov
ASD	adaptive software development	prilagodljivi razvoj programske opreme
AM	agile modeling	agilno modeliranje
AUP	agile unified process	agilno procesno ogrodje
RUP	rational unified process	objektno procesno ogrodje
DAD	disciplined agile delivery	ogordje urejene agilne dobave
DSDM	dynamic systems development method	metoda razvoja dinamičnih sistemov
XP	extreme programming	ekstremno programiranje
FDD	feature driven development	funkcionalno voden razvoj
CRM	customer relationship management	celovita obravnava in skrb za stranke
SOA	service oriented architecture	storitveno orientirana arhitektura
JAD	joint appliaction development	skupni razvoj programske opreme
JIT	just in time	razvoj v realnem času
TDD	test driven development	testno voden razvoj

Povzetek

Magistrska naloga temelji na primeru razvoja sistema za bančno poslovanje, ki je plod lastnega znanja banke v kateri sem zaposlen. Temelji za obstoječi proces razvoja so se postavili leta 2005, ko se je začel razvoj sistema. Proces temelji na slapovnem in spiralnem modelu, principih klasičnega projektnega vodenja in UML tehnikah modeliranja.

V prvem delu naloge so podrobno predstavljene vse faze obstoječega procesa razvoja, vloge udeležencev, ki v procesu nastopajo, tehnike dela ter ključni izdelki, ki tekom procesa nastajajo in tvorijo temelj projektne metodologije razvoja informacijskega sistema banke. Trenutna finančna situacija in trendi razvoja v IT nas vodijo v spremembe v načinu dela, če želimo ohraniti obseg dela in izboljšati kakovost sistema za bančno poslovanje. Zato so v drugem delu naloge predstavljene metodologije, ki nam to omogočajo. Med predstavljenimi metodologijami so bistveni vitki principi razvoja ter različne agilne metodologije kot so Scrum, ekstremno programiranje (XP), agilno procesno ogrodje (AUP), prilagodljivi razvoj programske opreme (ASD), ogrodje urejene agilne dobave (DAD).

Predvsem vitki principi so tisti, ki za celotno organizacijo prinašajo velike koristi, vendar zahtevajo spremembo v načinu razmišljanja na vseh ravneh, kot je tudi opisano v poglavju 4 kar pomeni, da je tudi uvedba bolj dolgoročna. Glavni cilj naloge je aplicirati moderne agilne in vitke pristope razvoja na obstoječi proces razvoja sistema za bančno poslovanje. Temu je posvečen tretji del naloge, kjer je predlagan nov proces razvoja sistema ter podan tudi predlog za širitev načina dela na celoten IT v banki.

Za uspešnost novega procesa dela je pomembno sodelovanje in sprememba načina razmišljanja vseh vključenih v razvoj informacijskega sistema banke in široka podpora vodstva. Kot smo ugotovili pa so ključni naslednji dejavniki oziroma gradniki novega procesa razvoja: priprava in vzdrževanje prioritetno urejenega enotnega seznama zahtev, oblikovanje heterogenih ekip za izvajanje razvoja ter kratki, fiksno časovno določeni razvojni cikli.

V nalogi sem si postavil štiri teze, katere lahko vse potrdim, saj predstavljajo temelj modernih načinov dela. Teze pravijo, da je proces razvoja potrebno čim manj ločiti po fazah, potrebno je avtomatizirati testiranje, planiranje vsebin razvoja pa je potrebno izvajati v ločenem procesu, ki skrbi za pripravo enotnega, prioritetno urejenega seznama zahtev.

Ključne besede: vitki razvoj, agilni razvoj, sistem za bančno poslovanje.

Abstract

The thesis is based on the case of developing banking information system in the bank where I am working. The banking information system is a result of our own knowledge and is based on the existing development process that was set in 2005. The process is based on the waterfall and spiral model, the classical principles of project management and UML modeling techniques.

All phases of the existing development process are presented in the first part of the thesis. The participant roles, tecniques and key products generated during the process form the basis of the methodology of information system development in the bank. The current financial situation, software development trends lead us to change the way we work if we want to maintain the workload and improve the quality of the banking system. Therefore, the second part of the paper presents methodologies that enable us to achieve those goals. Among presented methodologies the most important are lean software development and agile methodologies like Scrum, extreme programming (XP), agile unified process (AUP), adaptive software development (ASD), disciplined agile delivery (DAD).

Above all, lean principles are those who bring great benefits to entire organization, but require a change in mindset at all levels. As described in section 4 this means that the implementation is more of a long-term issue.

The main objective of the thesis is to apply a modern agile and lean development approaches to the existing process of developing banking information system. The third part describes proposed new development process and suggests ways to apply the process to the entire IT in the bank. The

success of the new development process depends on cooperation and changing the mindset of all involved in the software development process. The key factors of new development process are: preparation and maintenance of product backlog, creating of heterogeneous development teams and short, fixed-time development cycles.

In the thesis we have set ourselves four thesis, which can all be confirmed as being the foundation of modern software development approach. They say that we have to minimize the number of phases in the new development lifecycle, use automated testing methods and maintain unique prioritized requirements backlog.

Keywords: lean, agile, banking information system.

Poglavje 1

Uvod

Bančništvo je eno izmed poslovnih segmentov, kjer je informacijska podpora ključnega pomena. Že v zgodovini informacijske tehnologije je bilo to področje eno prvih, kjer so se izvajale avtomatizirane obdelave podatkov in je predstavljalo gonilno silo razvoja računalništva. Prav to velja tudi za bančništvo in informacijsko tehnologijo v Sloveniji.

V banki, kjer sem zaposlen, se je informacijska tehnologija začela razvijati že v poznih šestdesetih letih 20. stoletja. Od takrat pa vse do danes je informacijska podpora ključna za delovanje banke in tudi drugih bank za katere izvajamo informacijsko podporo. Informacijski sistem banke je zelo kompleksen in ga sestavlja preko 300 aplikacij, katerih velik delež je plod lastnega znanja in razvoja.

V magistrski nalogi bom prikazal primer večjega projekta, ki se v času hitrejših sprememb in željah po še učinkovitejšem načinu delovanja sooča z izzivi, kako obstoječe načine dela še izboljšati. Prikazal bom obstoječe načine dela na projektu ter rezultate projekta skozi leta. Na podlagi mojih izkušenj na projektu bom v nadaljevanju magistrske naloge predlagal izboljšave v procesu razvoja ter jih poskušal implementirati na delu novega razvoja in nato predstaviti rezultate izboljšav.

Naloga uvodoma podaja karakteristike projekta razvoja sistema za bančno poslovanje, ozadje nastanka in dosedanje rezultate projekta. Osrednji del

naloge se osredotoča na izvedbeni del projekta, to je faza poslovne analize, načrtovanja, razvoja in testiranja [60] informacijskega sistema za bančno poslovanje, ki je ključni izdelek oziroma rezultat projekta.

Prvi cilj magistrske naloge je predstaviti dobre prakse na projektu in jih tudi nazorno prikazati na primerih. Dobre prakse namreč izvirajo iz več teoretičnih modelov razvoja, a so bile v različnih fazah projekta skozi leta izpopolnjene oziroma prilagojene lastnim načinom dela.

Drugi, glavni cilj magistrske naloge pa je, predlagati izboljšave procesa razvoja sistema za bančno poslovanje in tudi celotnega razvoja v banki. Del izboljšav bomo skušali tudi realizirati oziroma navesti konkretne postopke, kako predloge izboljšanega procesa tudi implementirati. Izboljšave procesa razvoja temeljijo predvsem na principih agilnih in vitkih metodologij. Ključno pri vseh predlaganih izboljšavah pa je, da skušamo podpreti osnovno izhodišče naloge, in sicer da bo razvoj po izboljšanem procesu:

- hitrejši,
- bolj obvladljiv,
- širši po vsebini in
- cenejši.

Pod hitrejši imamo v mislih, da bomo lahko stvari razvili v krajšem času.

Bolj obvladljiv pomeni, da bomo z deljenjem obsega projekta po vsebini (danes gre po fazah za vse vsebine hkrati - "slapovni pristop", nov pristop pa predvideva razvoj po vsebinah v kratkih razvojnih ciklih: najprej se razvije v celoti ena vsebina (uporabniška zahteva), potem druga, itd; oziroma se več vsebin dela vzporedno glede na razpoložljive kadre) dosegli večji nadzor in lažje sledenje začrtanim časovnim planom.

Širši po vsebini (v povezavi s hitrostjo) pomeni, da bomo lahko v enem ciklu razvoja (1 verziji) razvili več funkcionalnosti.

Cenejši pa pomeni, da bo na koncu strošek razvoja enega produkcijskega cikla manjši, predvsem zaradi večje kakovosti končnega produkta, večjega

obsega vsebin, izpolnjevanja časovnih rokov in manjše količine kasnejšega vzdrževanja.

Poglavje 2

Informacijska tehnologija v bančništvu

Bančništva si danes brez informacijske tehnologije (v nadaljevanju IT) ne znamo več predstavljali. Tudi stopnja avtomatizacije delovnih procesov in informacijske podpore za izvajanje vseh procesov sta na zelo visoki ravni. Posebnost informatike v bankah je tudi ta, da ima veliko aplikacij in sistemov razvitih po naročilu. To pomeni, da so jih razvili v banki sami ali pa so jih po naročilu razvili zunanji dobavitelji. Naročeni in lastni razvoj sta namreč odgovor na to, da na trgu ne obstaja dosti programske opreme za bančništvo. Vsa informatika v bankah temelji bolj ali manj na zelo zanesljivih in performančno zmogljivih strežniških sistemih in arhitekturah (angl. legacy sistemi). Zaradi velikega števila aplikacij in sistemov ter dejstva, da je zgodovina IT v bankah zelo dolga, je temu primerna tudi kompleksnost bančnih informacijskih sistemov. Stanje je ponavadi takšno, da imamo par ključnih sistemov – za vsako poslovno področje, okrog njega pa množico manjših aplikacij in sistemov. Glavnina teh je namenjena svojemu koščku bančnega poslovanja, potrebam zakonodaje in regulatorja bančnega sistema.

Na trgu je mogoče najti nekaj večjih ponudnikov osnovnih (angl. core) bančnih informacijskih sistemov. Vsak nakup takšnega novega sistema sam po sebi še ni problematičen in tudi cenovno še ne največji zalogaj.

Prvi problem in z njim povezani veliki stroški nastanejo pri integraciji novega sistema v obstoječe IT okolje [14]. Integrirati oziroma povezati ga je potrebno:

- na eni strani z obstoječimi ključnimi sistemi. Tu imam v mislih predvsem sisteme, kjer vodimo temeljne storitve strank kot so osebni računi, depoziti, krediti. To lahko naredimo na nivoju podatkov (migracija) ali na nivoju vmesnikov,

- z ostalimi sistemi v banki, ki skrbijo za ključne podporne funkcije (šifrant strank, šifrant storitev, računovodstvo, plačilni sistemi, sistemi za poročanje po zahtevah regulatorjev in zakonodaje, kadrovske evidence, organizacijska struktura, idr.).

Drug problem pa se pojavi pri prilagoditvah sistema lokalnim zahtevam. Te prilagoditve so pri nas po lastnih izkušnjah sodeč zelo velike in se dogajajo pri večini, če ne celo vseh kupljenih rešitvah. Sam lahko potrdim, da se še ni zgodilo, da bi kupili neko aplikacijo oziroma sistem in ga uvedli takšnega kot je bil kupljen. Vedno se zahteva prilagoditve sistemov, namesto da bi lastne poslovne procese prilagodili sistemu. Tu je še smiselno dodati, da ne gre za sisteme, ki tečejo v nekih majhnih neprimerljivih okoljih, temveč sisteme, ki tečejo v bankah, ki so za red velikosti večje od naših bank. Pri zahtevanih prilagoditvah gre poleg dodatnih uporabniških zahtev po spremembah delovanja še za zahteve nacionalne zakonodaje pri računovodstvu, poročanju in plačilnih sistemih. Posledica prilagajanja je [57]:

- visoka cena prilagoditev in vzdrževanja sistema,

- veliko nadgradenj kupljenega sistema, kar povzroča težave dobavitelju,

- kupljen sistem postane sistem izdelan po naročilu, kar pomeni, da ga ne moremo več nadgrajevati z rednimi nadgradnjami sistema s strani dobavitelja, temveč nadgradnjami izključno za našo banko.

Podobno kot sem opisal zgoraj, je tudi stanje v banki, kjer sem zaposlen. Prav zaradi nakupov sistemov, prilagoditev teh sistemov in na koncu

nezadovoljstva končnih uporabnikov ter strank [2], je bila v banki pred leti sprejeta odločitev, da se sama loti razvoja integracijskega sistema za bančno poslovanje fizičnih oseb (v nadaljevanju sistem za bančno poslovanje). Zaradi slabega nadzora nad sistemi, je bila banka primorana v razvoj lastnega integracijskega in uporabniškega vmesnika.

Do takšnega stanja smo prišli predvsem zaradi obdobja "debelih krav", ki je omogočalo nakupe produktov na trgu in potem neskončno prilagajanje lastnim potrebam. Drugi razlog pa je slabo upravljanje informatike, ki ni imelo strateških usmeritev in vizije o ciljnem stanju informacijskega sistema banke.

Za potrebe razvoja so se ob ustanovitvi projekta razvoja tega sistema, pridružili še zunanji partnerji, vendar je celoten nadzor nad projektom in izvajanjem projekta bil v rokah banke.

Poglavje 3

Razvoj sistema za bančno poslovanje

3.1 Nastanek projekta za razvoj

Projekt za razvoj sistema za bančno poslovanje [51] je predstavljal odgovor na nezadovoljstvo s kupljenimi sistemi za bančno poslovanje. Ostali ključni povodi za lasten razvoj pa so bili:

- nezadovoljstvo uporabnikov s kupljenim sistemom, predvsem uporabniškim vmesnikom,
- kupljen sistem je transakcijsko naravnan in ne procesno,
- zakonske spremembe pri vodenju računov strank - uvedba transakcijskih - osebnih računov,
- ročno spremljanje gotovinskega poslovanja,
- spremembe v plačilnem prometu,
- uporabnik pri svojem osnovnem delu uporablja preveč aplikacij in se zato ukvarja s tehniko izvedbe posla namesto z vsebino posla,
- dolgotrajen zaključek na koncu delovnega dne v vsaki poslovni enoti.

Ideja za nov sistem za bančno poslovanje je bila ta, da bi se vsi obstoječi sistemi nadgradili s sistemom, ki bi uporabnikom ponudil enoten uporabniški vmesnik (za uporabnika to pomeni uporabo ene aplikacije) ter bi podpiral izvajanje poslovnih procesov. Na ta način bi se bančni delavci lahko ukvarjali z vsebino posla in ne z razmišljanjem, kako bodo v vseh sistemih naredili vse kar je potrebno, da bo na koncu izvedba posla ustrezna. Stanje IT v banki pred začetkom projekta za razvoj sistema za bančno poslovanje je bilo sledeče:

- informacijski sistem za podporo poslovanja fizičnih in pravnih oseb je sestavljalo preko 140 različnih aplikacij, od katerih je vsaka podpirala del poslovanja,
- delavec na bančnem okencu je uporabljal 25 aplikacij pri svojem rednem delu,
- aplikacije so tekle na zelo različnih platformah, kar je oteževalo integracijo in višalo stroške vzdrževanja,
- podatki v banki so bili razdrobljeni in nekonsistentni med seboj, kar je zopet onemogočalo poenotenje,
- manjkal je enoten katalog produktov, skupni šifranti,
- v banki so se podatki zbirali v podatkovnem skladišču, vendar se iz njih ni izvajalo posebnih analitik, ampak so služili zgolj za poročanje,
- v razvoju in pri nakupih rešitev ni bilo enotnega standarda in tehničnih zahtev, kar spet onemogoča povezovanje aplikacij,
- povezovanje med aplikacijami je potekalo vsak z vsakim in ni bilo enotnega storitvenega nivoja, ki bi omogočal lažjo integracijo. Število vmesnikov med aplikacijami je zato veliko, kar povečuje kompleksnost,
- razvoj informacijskega sistema je bil zaradi zgoraj opisane kompleksnosti usmerjen predvsem v vzdrževanje in ne v pravi razvoj – niti tehnološko, niti vsebinsko.

Obstoječa informacijska podpora je bila usmerjena transakcijsko v izvajanje posameznih aktivnosti in je v zelo omejenem obsegu omogočala izvajanja poslovnih procesov. Produktivnost v poslovni mreži ni bila optimalna in še vedno je bilo potrebno množico podatkov beležiti ročno.

Temeljni cilji projekta za razvoj sistema za bančno poslovanje so bili:

1. Racionalizacija dela v poslovalnicah s podporo izvajanja poslovnih procesov - na ta način bodo bančni delavci več časa lahko namenili delu s strankami in manj administrativnim postopkom.

2. Zagotoviti enotno informacijsko podporo delu v poslovalnicah z enim uporabniškim vmesnikom.

3. Poenotiti, poenostaviti in skrajšati zaključek dneva v poslovalnicah.

4. Omogočiti celovit pogled na stranko z vsemi potrebnimi informacijami o stranki na enem mestu (ekranu).

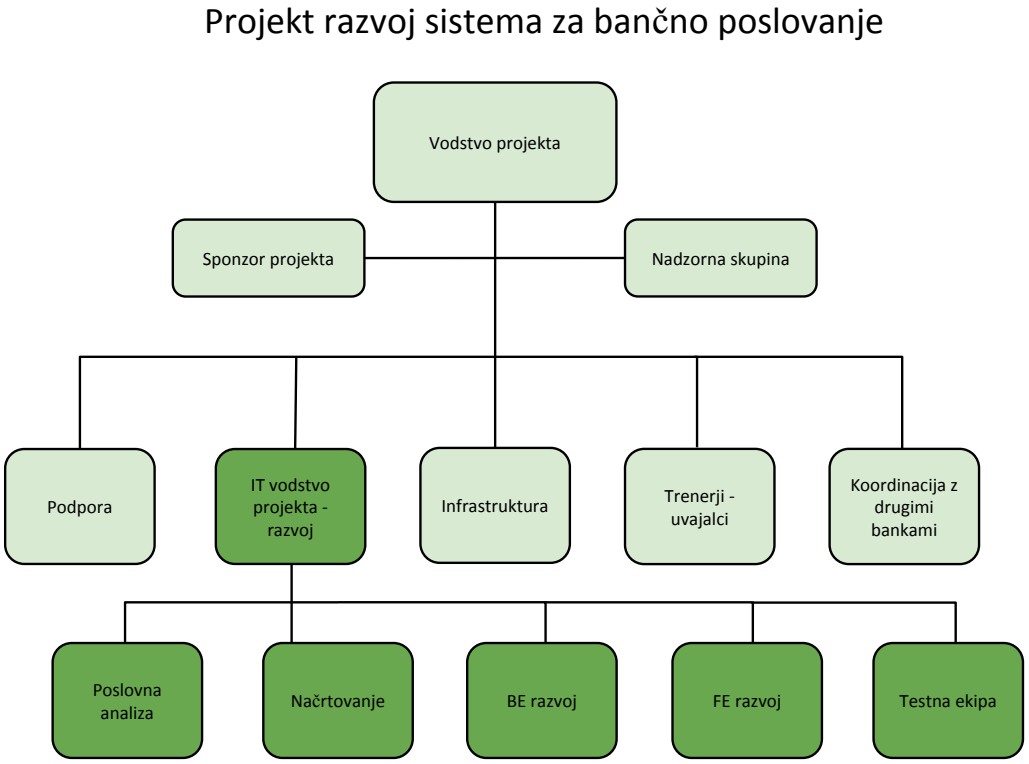
5. Nov informacijski sistem mora biti zasnovan na moderni spletni zasnovi z ločitvijo poslovne logike in predstavitvenega nivoja (uporabniškega vmesnika).

6. Nova podpora mora podpirati delo za več bank (angl. multibanking), da bomo storitev lahko tržili.

7. Sistem mora biti visoko razpoložljiv in odziven.

Projekt je imel krovne cilje, katerih uresničitev pa je bila že od začetka zamišljena s faznim pristopom.

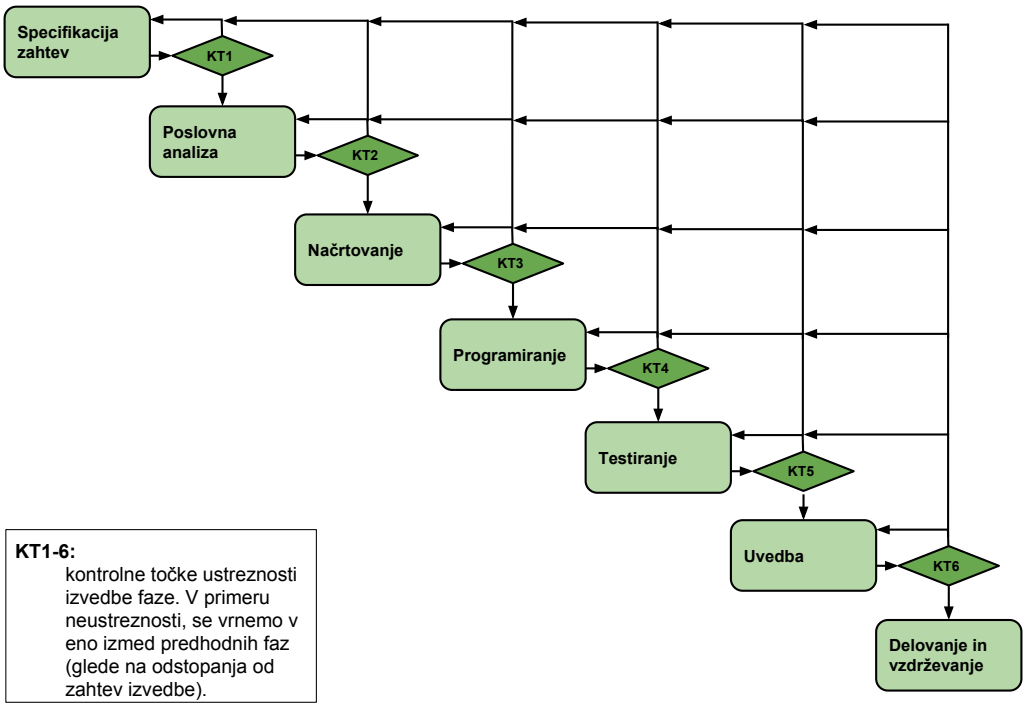
Projekt se je tako uradno začel sredi leta 2004 in prva uvedba v produkcijsko okolje je bila načrtovana za začetek leta 2006. Od takrat pa vse do konca leta 2013, se je projekt izvajal z dvoletnimi planskimi cikli in kot glavni rezultat ponudil dve verziji sistema letno. V projekt je od samega začetka pa vse do danes vključenih med 50 in 100 posameznikov v različnih vlogah na projektu. Od tega je izvedbeni oziroma razvojni del projekta (poslovna



Slika 3.1: Organizacijska struktura projekta

analiza, načrtovanje, kodiranje) v celotnem obdobju opravljalo med 15 in 40 posameznikov, ki so se skozi leta razvoja sicer tudi menjavali, ampak ne glede na to, se je število IT strokovnjakov več ali manj gibalo v navedenih številkah. Prav tako na projektu vseskozi sodeluje več podjetij (člani iz banke in partnerskih podjetij), kar dodatno omogoča dobro izmenjavo izkušenj in prenos oziroma ohranjanje znanj znotraj banke.

Na shemi 3.1 je prikazana organizacija projekta, ki se je sčasoma rahlo spreminjala, vendar je v temelju ves čas izvajanja dela na projektu ostala enaka. V magistrskem delu se bom v nadaljevanju osredotočil predvsem na izvedbeni del projekta, katerega rezultat je izdelava nove verzije sistema za bančno poslovanje dvakrat letno. Faze izvedbenega dela in postopke dela bom podrobno opisal in v tem delu predlagal tudi izboljšave, ki bodo prinesle



Slika 3.2: Slapovni model izvajanja projektov

večjo učinkovitost.

Izvajanje projekta temelji na projektni in razvojni metodologiji banke. Ta predpisuje slapovni (angl. waterfall) model razvoja [66] za posamezni razvojni cikel projektov kot je prikazano na sliki 3.2. V konkretnem primeru, kjer gre za projekt, ki traja več let, je ta razvojni cikel ena verzija oziroma pol leta. Zaradi dolgotrajnosti projekta se za planiranje verzij uporablja še spiralni model [50], znotraj katerega, kot sem že omenil, pa se vsak razvojni cikel izvaja po slapovnem modelu.

Po metodologiji razvoja ima vsaka faza v slapovnem modelu tudi svojo zaključno kontrolno točko. V tej točki se pregleda ali je izdelava posamezne faze ustrezna ali ne, s tem je posledično omogočeno ali onemogočeno tudi nadaljevanje z naslednjo fazo.

Kontrolna točka 1 (KT1) predstavlja kontrolo faze specifikacije zahtev uporabnika. Če zahteve niso dovolj natančno definirane, se poslovna analiza ne začne izvajati. V kontrolni točki 2 (KT2) se preveri ustreznost pripravljene poslovne analize. Če ta ni dovolj dobro pripravljena, se faza načrtovanja ne bi smela pričeti. Tako naprej velja za vse kontrolne točke, tudi do uvedbe in dejanskega prenosa v produkcijsko delovanje in vzdrževanje. Vsa ta pravila imamo definirana, vendar se velikokrat zgodi, da so kontrolne točke bolj postopkovne kot vsebinske. Zaradi tega pride do težav pri izvajanju posameznih ključnih vmesnih faz za razvoj, kar bo podrobneje predstavljeno v nadaljevanju.

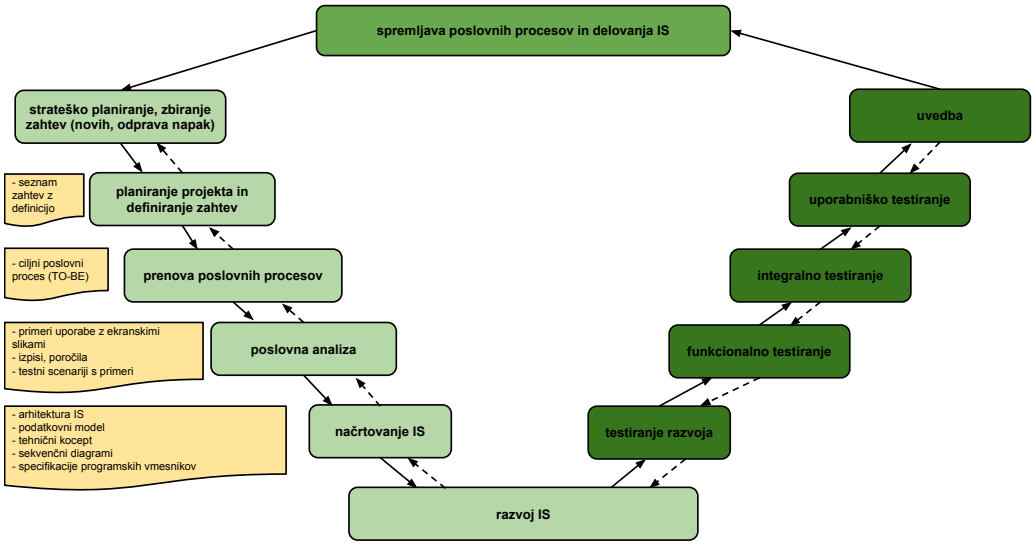
3.2 Obstoječi proces dela

Kot sem že omenil zgoraj, sta rezultat letnega cikla projekta dve verziji sistema za bančno poslovanje. Letni plan je tako vezan na produkcijska datuma, ki sta glede na bančni koledar prehodov v produkcijsko okolje naslednja: prvi pomladanski je v maju oziroma juniju in drugi jesenski v novembru.

Celoten potek faz posameznega polletnega produkcijskega cikla je nazorno prikazan na V-modelu. Na modelu 3.3 so prikazane faze, dokumenti, ki v posameznih fazah nastajajo in vloge posameznikov na projektu, ki so nosilci posameznih aktivnosti.

Skozi leta razvoja na projektu se je število sodelujočih v posamezni vlogi spreminjalo. Tudi vloge, ki so prisotne danes, so se tekom let spreminjale in prilagajale evoluciji razvoja sistema. Ključne vloge, ki sovpadajo s fazami razvoja danes, so naslednje:

- uporabnik oziroma predstavnik uporabnikov,
- poslovni analitik,
- načrtovalec,
- razvijalec poslovne logike (razvijalec BE (angl. Back-End)),



Slika 3.3: Življenjski cikel razvoja (V-model)

- razvijalec uporabniškega vmesnika (razvijalec FE (angl. Front-End)),

- tester.

V prvih fazah vloge poslovnega analitika in načrtovalca sploh ni bilo, ampak je bila samo vloga tehnologa, ki je bila prilagojena delu s starimi tehnologijami. Tehnolog je bil namreč bolj poslovni analitik, ki je natančno definiral oziroma analiziral poslovno zahtevo in to direktno predal v programiranje. V času arhitekture več-nivojskega (angl. multi-tier) razvoja pa se je kot nujna izkazala vloga načrtovalca (angl. software designer) oziroma arhitekta razvoja posameznega sistema. Tako sta se izmed tehnologov in izkušenih razvijalcev tekom prvih faz razvoja na projektu izoblikovali vlogi analitika in načrtovalca. Iz izkušenj se je med drugim tudi izkazalo, da mora biti število sodelujočih v posamezni vlogi čimbolj uravnoteženo. To pomeni, da mora biti v celotni ekipi npr. 5 poslovnih analitikov, 5 načrtovalcev, 5 razvijalcev BE, 5 razvijalcev FE, če želimo, da faze razvoja posameznega produkcijskega cikla tečejo čimbolj nemoteno in usklajeno. Sam sem se pro-

jektu priključil leta 2008 in sem bil najprej načrtovalec, od leta 2010 naprej pa sem poleg načrtovanja vodil tudi skupino načrtovalcev in bil namestnik vodje razvoja sistema.

Za podporo skupinskem delu na projektu smo definirali skupen repozitorij vse dokumentacije, ki nastane na projektu. Ta repozitorij je del sistema za vodenje verzij dokumentov (angl. CVS). Na ta način lahko spremljamo spremembe vse dokumentacije na projektu ter pogledamo v zgodovino sprememb.

Vsa dokumentacija je urejena po fazah projektnega vodenja oziroma izvajanju produkcijskih ciklov. Na sliki 3.4 je posebej razširjen del na katerega se osredotočam v nalogi - to je dokumentacija poslovne analize (na sliki je to 03.Funkcionalnosti) in načrtovanja (na sliki pod 02.Specifikacije). Na prvem nivoju so imeniki, ki v grobem sovpadajo s fazami projektnega vodenja:

- 01.Vodenje - tu se nahaja vsa dokumentacija o poslovnem in IT vodenju projekta,

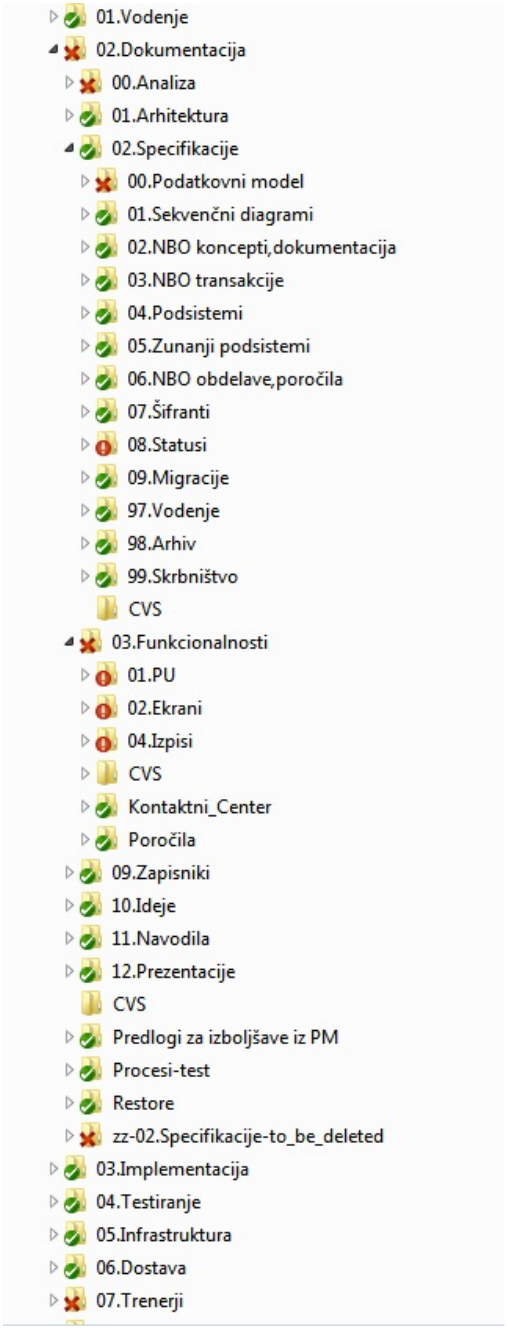
- 02.Dokumentacija - vsebuje vso vsebinsko dokumentacijo projekta, predvsem izvedbenih faz,

- 03.Implementacija - vsebuje tehnične dokumente o izvedbi, razvoju projekta. Tu so predvsem dokumenti razvoja, ki so še pomembni poleg izvorne kode sistema, ki se nahaja v drugih sistemih za vodenje verzij,

- 04.Testiranje - tu se nahaja vsa dokumentacija o testnih scenarijih za posamezne funkcionalnosti in tudi o izvedenem testiranju posameznih produkcijskih ciklov,

- 05.Infrastruktura - vsebuje vse tehnične dokumente o namestitvi sistema, posebnih komponentah za namestitev in dokumentacijo o infrastrukturi na kateri sistem teče,

- 06.Dostava - tu se nahaja vsa dokumentacija o namestitvah verzij sistema. Predvsem so pomembni dokumenti sistema kakovosti, ki ga iz-



Slika 3.4: Struktura organiziranosti dokumentacije

vajamo ter skripte za nadgradnje strukture in vsebine podatkovnih baz ter podatkov v šifrantih, parameterskih tabelah,

- 07.Trenerji - vsebuje vso dokumentacijo o uvedbi posameznih verzij sistema. Tu gre predvsem za posamezna navodila za uvajanje ter izvedbo uvajanja. Pomembni pa so tudi zapisniki uvajanja oziroma zabeležke o opažanjih ter vtisih končnih uporabnikov pri uvedbi.

Skozi evolucijo projekta za razvoj sistema za bančno poslovanje od leta 2005 pa do leta 2013 so se postopki dela v podrobnostih spreminjali in izpopolnjevali. V podrobnejšem opisovanju vsake faze bomo tako posebej izpostavili že izvedene izboljšave v teku let projekta, ki so se zgodile na podlagi izkušenj (na podlagi storjenih napak ter pridobljenih novih izkušenj, dobrih praks v svetu, nadgradenj standardov) ter so prinesle večjo dodano vrednost in spremembo v načinu dela.

Ključni princip dela je ta, da vsako poslovno spremembo (zahtevo) izvedemo po vrsti v vseh izvedbenih fazah, kot je tudi prikazano na sliki 3.3 (življenjski cikel razvoja) in sproti tudi ažuriramo vso potrebno dokumentacijo:

- najprej se izvede podrobna poslovna analiza, v kateri poslovni analitik dopolni dokumente poslovne analize,
- potem načrtovalec delno dopolni dokumentacijo poslovne analize in pripravi ter dopolni še vse tehnične specifikacije,
- za njim razvijalci BE in FE vse specificirano v dokumentaciji razvijejo in
- na koncu poslovni analitik preveri ali nadgradnja deluje v skladu z zahtevano spremembo.

Na ta način je zagotovljeno, da je vsa dokumentacija vedno ažurna. Pri ažurnosti vse dokumentacije, ki jo bomo podrobneje opisali pri posameznih fazah razvoja, velja poudariti še sistem označevanja vsebine dokumentov, ki

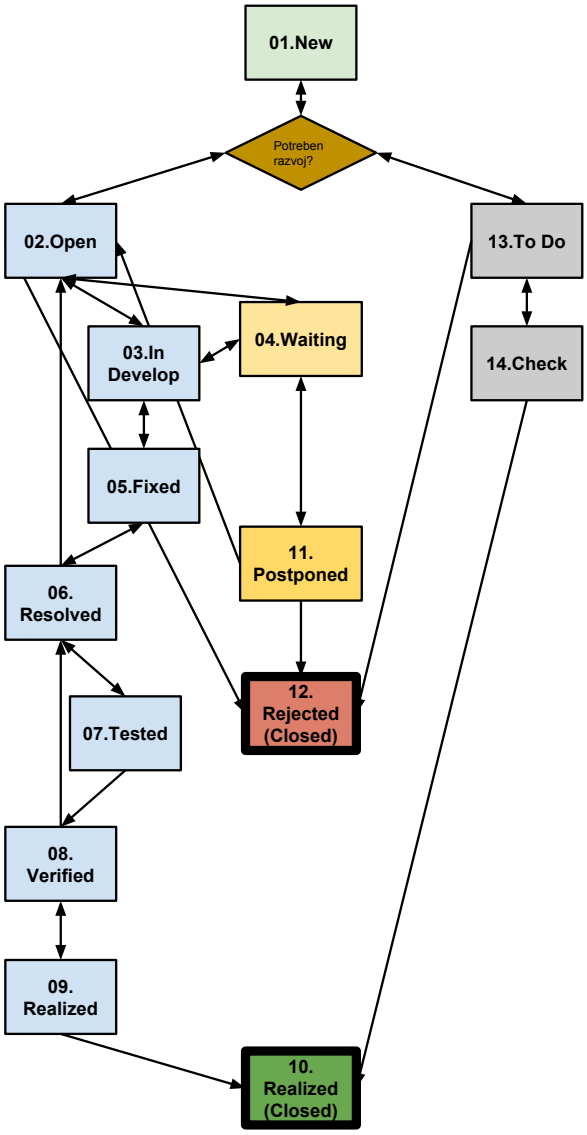
omogoča lažje sledenje spremembam v dokumentaciji. Z različnimi barvami namreč osenčimo vsebino v dokumentu, ki se je spremenila in s tem na enostaven način zagotovimo preglednost in hitro informacijo o tem, kaj se je dejansko spremenilo. Dogovor na projektu je sledeč:

- zelena barva – vse kar je v dokumentu na novo v zadnji spremembi glede na prejšnje stanje,
- rdeča barva – vse kar smo v dokumentu brisali in ni več aktualno glede na prejšnje stanje dokumenta,
- rumena barva – vse kar je v dokumentu spremenjeno, vendar še ni dokončno potrjeno (je še v fazi dogovarjanja in usklajevanja) glede na predhodno verzijo dokumenta.

Za spremljanje in izvajanje razvoja se uporablja orodje za vodenje zahtev Techexcel DevTrack. Gre za klasično orodje za spremljanje zahtev (angl. task tracking, issue tracking) v katerem so vnesene vse zahteve na katerih se izvaja razvoj v banki. Zahteve ločimo na naslednje vrste:

- BR – nove uporabniške zahteve (angl. Business Request),
- CR – spremembe uporabniških zahtev (angl. Change Request),
- OR – operativne zahteve (angl. Operational Request),
- IR – napake (angl. Incident Request).

Vse poslovne zahteve posameznega produkcijskega cikla se definirajo kot BR. Če med razvojem zaznamo spremembo uporabniške zahteve, potem se generira CR zahteva. Pod OR zahteve se smatrajo redne vzdrževalne aktivnosti oziroma posamezni posegi po nastavitvah šifrantov ali pripravi določenih podatkov. Vse napake, ki nastanejo v fazi testiranja in se do konca produkcijskega cikla odpravljajo, pa se definirajo kot IR. V orodju ima vsak zahtevek svoj življenjski tok, ki je definiran na podlagi faz razvoja in prikazan na sliki



Slika 3.5: Življenjski tok zahteve v orodju DevTrack

3.5. V vsakem trenutku tako lahko spremljamo kje (pri kateri osebi, v kateri fazi) se zahtevek nahaja in lahko ustrezno ukrepamo v primeru zamud.

Za lažje delo in razporejanje znotraj posameznih faz razvoja, imamo definirane tudi skupine. Privzeto se zahtevki pošiljajo na skupino in vodja posamezne skupine ima nalogo te zahtevke razporejati med člane skupine, ti pa poskrbijo za nadaljnje korake prehoda zahtevka v naslednjo fazo razvoja.

Vse nove zahteve (BR, CR) pridejo v prvi fazi (v statusu 01.New) najprej na poslovno analizo (skupina +Poslovna analiza v orodju DevTrack). Tu ga najprej vodja skupine poslovne analize glede na plan razporedi in pošlje na posameznega poslovnega analitika (v statusu 02.Open). Ko se analiza zaključi, potem analitik, ki je analizo opravil in pripravil dokumentacijo, zahtevek pošlje na skupino načrtovalcev (še vedno v statusu 02.Open, vendar preneseno na skupino +Načrtovalci).

Poleg novih zahtev, ki pridejo na skupino načrtovalcev (skupina +Načrtovalci v orodju), na to skupino pridejo direktno vsi novi OR zahtevki za nastavitve šifrantov ali vpoglede v produkcijske podatke ter IR zahtevki v fazi testiranja. Proces vsakega zahtevka izmed teh dveh tipov se tako kot za vse začne s statusom 01.New. Vse zahteve skupaj potem vodja skupine načrtovalcev razporeja med člane svoje skupine (še vedno v status 02.Open), da se izvede načrt rešitve. Razporejanje se izvaja glede na plan in dodeljenega posameznika za posamezno vsebino in glede na skrbništvo področij že delujočih rešitev (za vsako področje, ki je podprto v sistemu imamo namreč definirane skrbnike in njihove namestnike, tako da vemo kdo najbolj pozna posamezno vsebino). Ko načrtovalec pripravi vso potrebno dokumentacijo, zahtevek pošlje v razvoj (v statusu 03.In Develop). V primeru, da gre za večjo spremembo ali nov razvoj, zahtevek razdeli na dva dela in enega pošlje na razvoj FE, drugega na razvoj BE. Na ta način zagotovimo, da se vzporedno pripravi nadgradnja uporabniškega vmesnika in priprava poslovne logike. Na koncu pa se potem poslovna logika integrira v uporabniški vmesnik. Ko se razvoj posameznega zahtevka zaključi in naredi osnovni razvojni test, se status zahtevka prestavi v 05. Fixed. Ko se zaključi del funkcionalnosti in se

jih namesti v paket za testiranje, se status zahtevkov spremeni v 06.Resolved. Takrat gre zahtevek na skupino za funkcionalno testiranje, ki ga izvajajo poslovni analitiki. Vodja skupine ga glede na vsebino in razpoložljive kadre razporedi med sodelavce. V primeru uspešnega testa se zahtevek prestavi v status 07.Tested, v nasprotnem primeru pa nazaj v 02.Open ter ga usmeri na skupino +Načrtovalci. Če se zahtevek zavrne, potem spet naredi nov cikel po istih korakih. V primeru uspešnega funkcionalnega testiranja pa se zahtevki pošljejo v statusu 07.Tested na skupino +IUAT, ki izvaja uporabniško testiranje. To skupino sestavljajo predstavniki uporabnikov ter končni uporabniki iz poslovne mreže. Ko ta skupina začne uporabniško testiranje, vodja testiranja razporedi zahtevke med uporabnike. Če je uporabniški test uspešen, se prestavi zahtevek v status 08.Verified na skupino +Produkcija, v nasprotnem primeru pa nazaj na skupino načrtovalcev +Načrtovalci v statusu 06.Resolved.

Zakaj gre zahtevek v primeru neuspešnega uporabniškega testa na načrtovalce? Skozi leta na razvoju sistema se je namreč izkazalo, da imamo v tej skupini največji pregled nad vsebino in načinom implementacije ter tako najbolj celovito pregledamo, razrešimo in usmerimo napako naprej v odpravo ustreznim posameznikom oziroma skupinam razvijalcev.

Status 08.Verified pa pomeni, da je test zahtevka uspešen in gre popravek ali nova zahteva, realizirana v okviru zahtevka, lahko v produkcijsko delovanje. Ko v končni fazi to naredimo – prenesemo novo verzijo sistema v produkcijsko delovanje, se status spremeni v 09.Realized. Iz tega statusa potem v določenih časovnih intervalih, ko se izvaja zaključno poročilo produkcijskega cikla, zahtevke končno zapremo.

3.2.1 Projektno vodenje

Sama vsebina letnega cikla se oceni, definira in planira do konca tekočega leta za naslednje leto. Letni plan se potem razdeli na dva dela in tako določi vsebino obeh glavnih verzij sistema. Ves ta del se izvaja pod vodstvom vodje projekta v tesnem sodelovanju z IT vodjo projekta in vodji posameznih

skupin, ki priskrbijo ustrezne ocene dela za planiranje.

Ko je plan določen, koordinacijo in izvedbo prevzame IT vodja projekta in izvedbena ekipa, katere delo bom podrobno opisoval v nadaljevanju naloge.

Projektni plan za izvedbo posamezne verzije sistema je narejen s pomočjo orodja za planiranje projektov in je definiran po slapovnem modelu s fazami po projektni metodologiji. Primer je prikazan na sliki 3.6. Na projektu razvoja sistema za bančno poslovanje smo planiranje delno že izboljšali, tako da se faze razvoja delno prekrivajo med seboj, s čimer pridobimo nekaj časa na račun prekrivanja posameznih faz in dela, ki se lahko izvaja vzporedno.

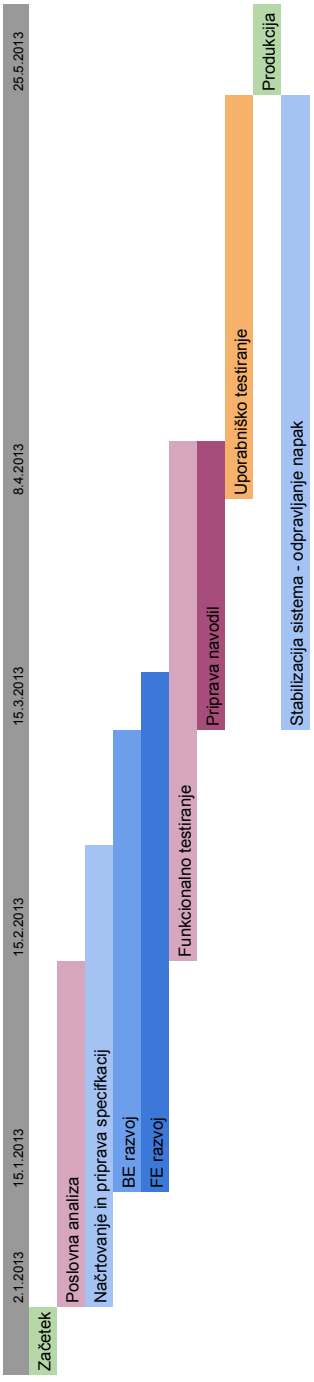
Za primere večjih novih funkcionalnih zahtev, ki se razvijajo v posamezni verziji, ponavadi pripravimo še podroben projektni plan za razvoj tiste funkcionalnosti. Ta načrt je bolj podrobno specificiran, na njega so določeni posamezniki in se v teku razvoja spremlja tudi podrobna izvedba tega načrta.

Na sliki 3.7 je predstavljen izsek iz podrobnega projektnega plana načrtovanja za razvoj izbrane funkcionalnosti.

Za ostale manjše zahteve pa se samo v seznamu določi nosilce posameznih faz razvoja in se potem na enostaven način spremlja izvedba.

Pri izdelavi vsake verzije sistema ima glavno koordinacijsko vlogo IT vodja projekta (v nadaljevanju bom o njem govoril kot o vodji projekta, ker bom prikazoval izvedbeni del celotnega projekta). Ta opravlja vse naloge po projektni metodologiji banke (ta temelji na PMI metodologiji [1]) s ciljem, da se razvoj opravi v predvidenih časovnih in stroškovnih okvirih ter nova verzija sistema uvede v produkcijsko okolje. Pri tem mu glavno podporno vlogo nudijo vodje posameznih skupin, ki skrbijo za svojo fazo v razvojnem ciklu:

- poslovne analize,
- načrtovanja,
- razvoja uporabniškega vmesnika (v nadaljevanju FE (Front-End) razvoja),
- razvoja poslovne logike (v nadaljevanju BE (Back-End) razvoja),



Slika 3.6: Primer projektnega plana razvoja sistema

Naziv vsebine	Trajanje	Začetni datum	Končni datum	Predhodniki	Izvajalci
Informativni izračun		29.8.2011	7.10.2011		
PU Informativni izračun		29.8.2011	16.9.2011		
Pravila	5 dni				*Načrtovalec1*
Shranjevanje/ažuriranje	5 dni			4	*Načrtovalec1*
Pregled	5 dni			5	*Načrtovalec1*
PU Kreditna sposobnost		16.9.2011	30.9.2011		
Pravila	5 dni				*Načrtovalec2*
Shranjevanje/ažuriranje	5 dni			8	*Načrtovalec2*
Pregled	5 dni			9	*Načrtovalec2*
PU Informativni izračun - premostitveni kredit	4 dni	19.9.2011	22.9.2011	3	*Načrtovalec1*
PU Pošiljanje epošte	1 dan	7.10.2011	7.10.2011		*Načrtovalec3*
Izračunavanje kreditov		1.8.2011	24.9.2011		
EOM	10 dni	1.8.2011	12.8.2011		*Načrtovalec2*
Amortizacijski načrt	10 dni	15.8.2011	26.8.2011	14	*Načrtovalec2*
Kreditna sposobnost	5 dni	29.8.2011	2.9.2011	15	*Načrtovalec2*
Faktor varnosti 1.del zavarovanj	10 dni	5.9.2011	16.9.2011	16	*Načrtovalec2*
Faktor varnosti 2.del zavarovanj	5 dni	3.10.2011	8.10.2011	17	*Načrtovalec2*
Parametrizacija kreditov		15.8.2011	20.1.2012		
Premijski stavek	10 dni	15.8.2011	26.8.2011		*Načrtovalec1*
Parametrizacija nastavitev		19.9.2011	14.10.2011		
Osnovne nastavitve	10 dni			3	*Načrtovalec3*
Pooblastila tarifa	5 dni			22	*Načrtovalec1*
Pooblastila obrestne mere	5 dni			23	*Načrtovalec1*
Poslovna pravila	10 dni	11.10.2011	22.10.2011	1	*Načrtovalec2*
Polnjenje podatkov za parametrizacijo	35 dni	17.10.2011	27.11.2011	21	*Načrtovalec1*
Polnjenje poslovnih pravil	10 dni	25.10.2011	5.11.2011	25	*Načrtovalec2; Načrtovalec3*
Podpora poslovnemu procesu		29.8.2011	16.9.2011		
Postavitev sistema	5 dni				*Načrtovalec3*
Ažuriranje	5 dni			29	*Načrtovalec3*
Pregled	5 dni			30	*Načrtovalec3*
Odpiranje kredita		8.12.1011	17.4.2012		
PU Vloga za odobritev	12 dni	8.12.2011	19.12.2011	50	*Načrtovalec2*
PU Obdelava in odpiranje kredita		8.12.2011	13.1.2012		
Postavitev poslovnega procesa in pravil	25 dni				*Načrtovalec3*
PU Kreditna mapa	15 dni	23.1.2012	10.2.2012	21	*Načrtovalec3*
PU Analiza poslovanja	20 dni	23.1.2012	17.2.2012		*Načrtovalec2*
PU Preverjanje za vse vrste zavarovanj	94 dni	8.12.2011	17.4.2012	50	*Načrtovalec4; Načrtovalec5*
PU Odobravanje kredita	24 dni	13.2.2012	15.3.2012	36	*Načrtovlec3*
PU Odpiranje kredita v ciljnem sistemu	17 dni	8.12.2011	4.1.2012	50	*Načrtovalec1*
PU Dodatni podatki za pogodbo	6 dni	5.3.2012	12.3.2012	37	*Načrtovalec2*
PU Tiskanje dokumentov in obrazcev	5 dni	13.3.2012	19.3.2012	41	*Načrtovalec2*
PU Sklepanje pogodbe	5 dni	20.3.2012	26.3.2012	42	*Načrtovalec5*
PU Črpanje kredita	16 dni	5.3.2012	26.3.2012	43	*Načrtovalec1*
PU Spremljava	15 dni	20.3.2012	9.4.2012	42	*Načrtovalec2*
PU Zaključek kreditne naloge	8 dni	3.2.2012	14.2.2012	40	*Načrtovalec4*
Pregled kredita iz ciljnega sistema		14.3.2012	4.4.2012		
PU Podatki o kreditu	15 dni	14.3.2012	4.4.2012		*Načrtovalec5*
Prenos poslovnih pravil za informativni izračun v sistem za pravila	10 dni	5.4.2012	17.4.2012	47	*Načrtovalec1*
Produkcija informativni izračun	0 dni	8.12.2011	8.12.2011		
Produkcija odpiran je kredita	0 dni	26.5.2012	26.5.2012		

Slika 3.7: Podrobni plan načrtovanja izbrane funkcionalnosti

- testiranja.

Vodja projekta se z vodji skupin sestaja tedensko, kjer se pregleda statuse izvedbe po posameznih fazah in se seznani z morebitnimi spremembami plana in vsebin za določeno verzijo. Na ta način se po stalnem dnevnem redu:

- pregleda odprte sklepe,
- stanje produkcije,
- stanje razvoja,
- razno in
- tako spremlja ključne dejavnike na projektu.

Naloga vodje projekta je potem spremljanje časovne usklajenosti izvedbe projekta s planom (ali prihaja do zamikov (ponavadi zakasnitev)) in ustrezno ukrepanje v primeru odstopanj. Vodja projekta poleg rednih tedenskih sestankov opravlja seveda nadzor nad razvojem projekta in s svojim znanjem in izkušnjami tudi sodeluje pri odločanju v primeru:

- konceptualnih dilem za izvedbo podpore določeni funkcionalnosti,
- ugotovljenih napak v testiranju, kjer se tudi odloči, če se določena nepomembna napaka odpravlja v naslednji verziji ali popravku po redni produkciji,
- manjših konfliktov pri razhajanju mnenj med člani projekta.

Za ustrezno podporo vodji projekta imajo vodje skupin prav tako redne tedenske sestanke skupin, katerih se pogosto udeleži tudi vodja projekta. Na teh sestankih vodja poda informacije iz skupnega sestanka vodij projekta o statusu, odprtih in nujnih stvareh, pregleda pa se:

- status razvoja po funkcionalnostih znotraj posamezne faze (npr. načrtovanja),

- razreši morebitne konceptualne odprte zadeve na posameznih funkcionalnostih,

- izmenjava mnenj in dobrih praks pri razvoju posamezne funkcionalnosti,

- eventualne odsotnosti članov skupine.

3.2.2 Poslovna analiza

Po opravljenem planiranju in določitvi vsebine obeh letnih verzij, se najprej izvaja poslovna analiza, katere prvi cilj je popis obstoječega delovanja (angl. AS-IS) [55]. V celotni fazi poslovne analize sodeluje več posameznikov v različnih vlogah na projektu: uporabnik oziroma predstavnik uporabnikov, procesnik, poslovni analitik, v večini primerov načrtovalec in pri večjih sklopih s konceptualno novimi vsebinami, tudi razvijalci.

Ko je popisano obstoječe stanje, se naredi predlog izboljšav na poslovnem procesu (angl. TO-BE) oziroma ciljni proces, ki se ga dopolni še z osnutki novih primerov uporabe (angl. use-case diagram, v nadaljevanju PU) [17] in ekranskih slik za lažje razumevanje in usklajevanje prenovljenega procesa. Ko je TO-BE poslovni proces usklajen, se pripravi podrobne specifikacije primerov uporabe, podprtih z ekranskimi slikami in izpisi dokumentov, ki morajo v procesu nastati.

Faza poslovne analize posamezne funkcionalnosti poteka pod koordinacijo poslovnega analitika, ki je nosilec analize določene nove funkcionalnosti. On je tisti, ki mora z vseh strani (procesnik, uporabnik, razvojni tehnolog produktov) pridobiti prave informacije in jih uporabiti za pripravo celovite poslovne analize. Tako se v fazi analize, predvsem v prvem delu, organizira več delavnic, na katerih se vsi seznanjajo z vsebino funkcionalnosti in kjer se usklajuje mogoče načine prenovljene informacijske podpore (implementacije) te funkcionalnosti. V drugi fazi potem analitiki pripravijo svoje izdelke (dokumente) in po potrebi še organizirajo potrditvene delavnice ali pa sami stopijo v stik z ljudmi v ustreznih vlogah. Na ta način pridobijo vse potrebne

informacije za pripravo analize in na koncu pridobijo potrditev ustreznosti (skladnosti z zahtevo) pripravljenih izdelkov.

Ključni rezultat poslovne analize posamezne funkcionalnosti je primer uporabe, ki podrobno opisuje postopek izvajanja funkcionalnosti v novi informacijski podpori. Natančno opredeljuje osnovni tok, alternativne tokove in variacije osnovnega toka postopka izvajanja funkcionalnosti v novi podpori. PU podrobno opisuje vse ekranske maske, akcije, ki se izvajajo na maskah, način obravnave napak, zaloge vrednosti posameznih polj na ekranu. Sama definicija PU je bila zasnovana v začetku projekta in se je kasneje (v letu 2009) standardizirala tudi za vse ostale nove projekte v banki. PU kot ga poznamo na projektu je v osnovi povzet po UML metodologiji [15], a je malenkost dopolnjen glede na potrebe projekta, ki so se izkazale ključne skozi pretekle razvojne cikle.

Kot priloga PU, ki ga dopolnjuje in dodatno oplemeniti za lažjo predstavo, služijo ekranske slike v katerih so natančno definirane vse ekranske maske, ki morajo biti realizirane v delujočem sistemu v okviru PU.

Poleg ekranskih mask so ključen rezultat poslovne analize definicije izpisov, ki po teoriji izvirajo že iz poslovnega procesa [9]. Izpisi nastanejo tekom priprave PU. Ti so definirani kot predloge in natančno določajo statični tekst in dele, kjer je potrebno umestiti spremenljive podatke (podatke o poslu, stranki, itd.).

PU in izpisi so narejeni v obliki tekstovnega dokumenta, ekranske slike pa se rišejo v orodju za risanje ekranskih mask.

Vsi izdelki poslovne analize (PU, ekranske slike, izpisi) so temelj za načrtovanje in razvoj. Predvsem so ti dokumenti prvi vhod za FE razvoj, kjer na osnovi PU in ekranskih mask že lahko začnejo razvijati uporabniški vmesnik.

Drugi pomen dokumentacije poslovne analize pa je za zaključno fazo v razvoju – testiranje. Iz primerov uporabe se namreč direktno pripravijo testni scenariji in testni podatki. Testni scenariji zajemajo osnovni tok in vse alternativne tokove, različni testni podatki (primeri) pa morajo preveriti še vse variacije v izvajanju PU.

Pri poslovnih analitikih moramo opozoriti na posebnost, da v večini poslovni analitiki banke po izobrazbi niso informatiki, ampak prihajajo bolj iz poslovnega okolja banke (poslovalnice, zaledne službe). Ker nimajo dovolj tehničnega znanja razvoja programske opreme, ki so za pripravo zasnov izvedbe rešitve ključna, ponavadi že v fazi poslovne analize sodelujejo tudi načrtovalci. Predvsem pri razvoju povsem novih funkcionalnosti se je to dejansko izkazalo kot nujno. V začetnih letih projekta poslovne analize kot smo jo poznali kasneje ni bilo. Nastala je s časom, ko se je izvajala podpora temeljnim procesom v poslovanju na bančnem okencu in je bilo potrebno poslovne zahteve dobro analizirati in definirati ciljni poslovni proces ter način informacijske podpore tega procesa. Skozi evolucijo dela in rezultatov pa se je izkazala potreba po sodelovanju načrtovalcev, če smo želeli, da so PU dovolj dobro napisani in da je tudi končna izvedba čimbolj skladna z obstoječimi smernicami in standardi razvoja informacijskih sistemov.

Je pa tu težava, ker se ponavadi poslovna analiza novih razvojnih vsebin dela, ko so vsebine prejšnje verzije še v razvoju ali v testiranju in je stalno prisotnost načrtovalcev v obstoječem načinu dela težko zagotoviti. Tako se nam prepogosto zgodi, da se potem v fazi načrtovanja spreminjajo koncepti zasnove iz poslovne analize in se moramo vračati v predhodno fazo razvoja ter s tem podaljšujemo celoten cikel razvoja. Na ta način potem zamujamo z zaključkom faze načrtovanja ter začetkom faze razvoja (kodiranja, programiranja), s čimer ogrozimo uspešnost izvedbe celotne verzije. Ta težava se je v praksi večkrat zgodila in je ena izmed ključnih, ki jih je potrebno poglobljeno analizirati in s spremenjenim načinom dela tudi rešiti.

Prva teza: *predlog izboljšave je do neke mere združiti fazo poslovne analize in načrtovanja posamezne funkcionalnosti, da se izognemo prepoznim konceptualnim spremembam ter da se ne (oziroma se nam število teh vrnitev zmanjša) vračamo v predhodne faze razvoja.*

3.2.3 Načrtovanje

V praksi se je na projektu izkazalo, da strogo ločevanje med fazo poslovne analize in načrtovanja ni mogoče ter dejansko ne sme biti ločeno. V določeni meri se morata fazi prekrivati, do česar smo prišli iz spoznanja, da morajo v fazi analize zelo dejavno sodelovati načrtovalci.

Profil načrtovalca se je na projektu izkazal kot ključen za uspešno izvedbo projekta. Oni so namreč tisti, ki predstavljajo ključno vez med poslovno vsebino funkcionalnosti in tehnično implementacijo – razvojem podpore – tej funkcionalnosti. Skratka povezuje poslovni svet z IT svetom. Kot profil načrtovalca so se izkazali za zelo dobre:

- bivši izkušeni programerji ali pa
- informatiki s širokim spektrom znanj in odprtega razmišljanja ter izkušnjami iz različnih poslovnih področij.

Prav zaradi obsega dela, ki ga načrtovalci opravijo pa je ta faza ključna za uspešno izvedbo projekta. V določenih kritičnih situacijah je celo predstavljala ozko grlo v procesu razvoja.

V fazi načrtovanja [37] se najprej pogleda poslovni proces, PU, ekranske slike in pripravi načrt izvedbe informacijske podpore te funkcionalnosti. Ker skušajo biti načrtovalci čimbolj prisotni že v fazi analize, je ponavadi poznavanje poslovnega področja že zelo dobro. Celoten načrt izvedbe posamezne nove funkcionalnosti zajema naslednje ključne aktivnosti in izdelke, ki v načrtovanju nastanejo:

- *arhitektura nove informacijske podpore (zasnova podsistema),*
- *tehnični koncept*,
- razširitev podatkovnega modela,
- *sekvenčni diagram*,
- *specifikacije programskih vmesnikov*,

- ažuriranje šifrantov,

- skripte sprememb na podatkih in šifrantih za prehod med okolji.

Izbrane izdelke (navedene v poševnicah), ki so ključen del in predstavljajo hrbtenico razvoja sistema za bančno poslovanje, bomo v nadaljevanju podrobneje opisali. Podali bomo tudi teoretično osnovo izdelka, način prilagoditve izdelka za potrebe dela na projektu in poudarili pomen posameznega izdelka za izvedbo razvoja programske kode ter vzdrževanja celotnega sistema. Tu se mi zdi še pomembno predstaviti način, ki ga uporabljamo za vzdrževanje šifrantov. Vse šifrante namreč primarno vodimo v datotekah v obliki tabel, kjer spremljamo tudi zgodovino sprememb šifrantov in celotno zalogo vrednosti. Ta način je zelo uporaben predvsem zaradi različnih razvojnih in testnih okolij, kjer je potrebno spremembe v različnih fazah izvajati in lažjega prehajanja med okolji v razvojnem ciklu. S pomočjo orodja si iz tabel v datotekah generiramo DML stavke (v glavnem INSERT, UPDATE SQL stavke), ki jih potem za razvojno, testno in funkcionalno testno okolje izvajamo ročno. Za okolje uporabniškega testiranja (iuat) in produkcijsko okolje pa DML stavke shranimo v skupen dokument sprememb na podatkih v podatkovni bazi za posamezen razvojni cikel. Ta skupni dokument nastaja v fazi načrtovanja in razvoja. Ko se izvaja prehod v uporabniško testiranje, se celotna skripta sprememb izvede na iuat okolju, ko pa gremo z verzijo v produkcijsko okolje, se skripta izvede še tam. Na ta način zagotovimo, da pri pripravi šifrantov in sprememb v šifrantih ničesar ne pozabimo, ko prehajamo iz razvoja v različna testna okolja in produkcijo.

Arhitektura sistema za bančno poslovanje

Arhitekturna slika sistema za bančno poslovanje ponazarja logično več-nivojsko zasnovo sistema. Na najvišjem nivoju je predstavitveni nivo, ki se izvaja v spletnem brskalniku na delovni postaji. Nižje v shemi je aplikacijski strežnik, na katerem se izvaja predstavitveni del poslovne logike. Oba nivoja sta v domeni FE razvoja projekta. Iz aplikacijskega strežnika se kliče ve-

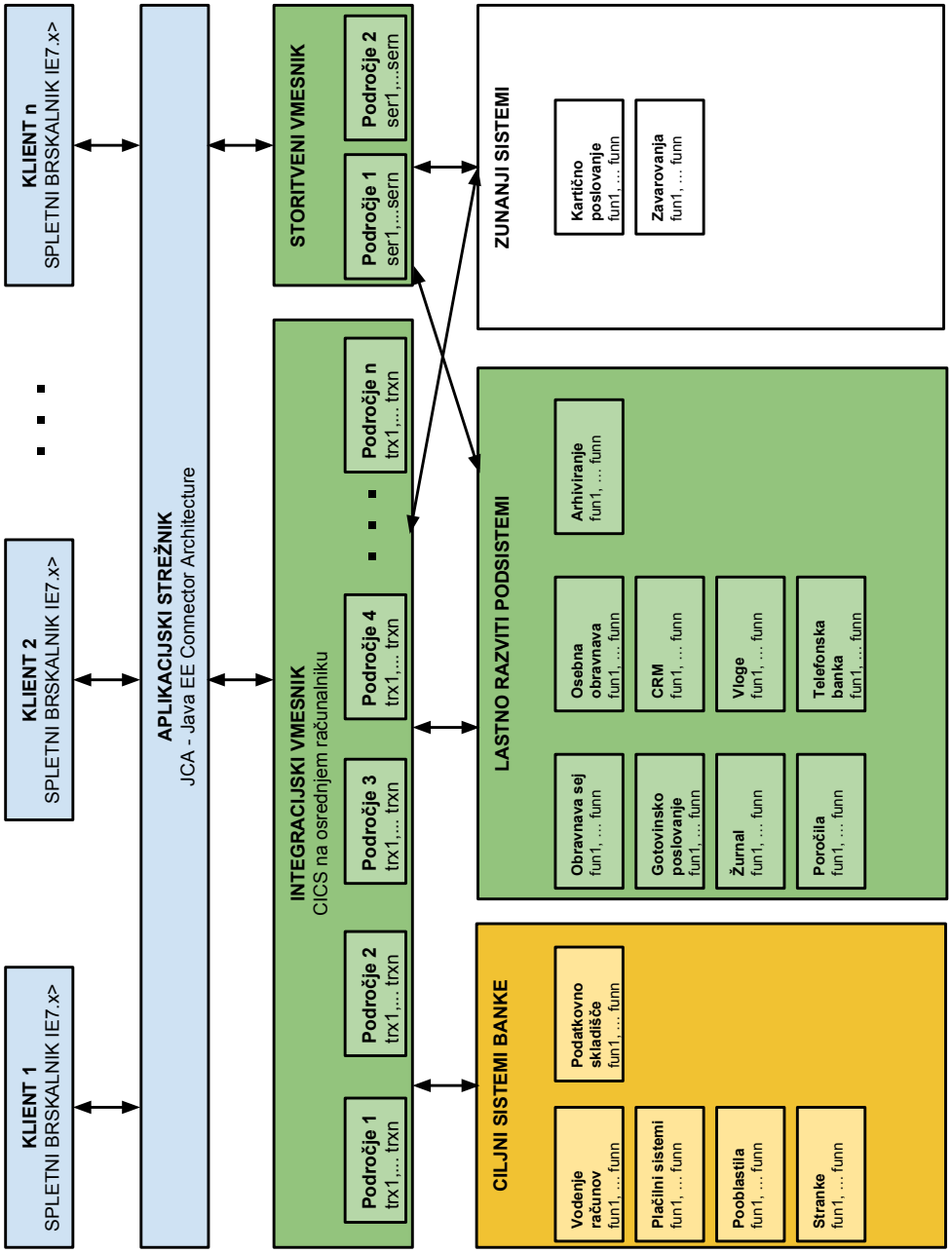
dno integracijski vmesnik sistema, ki mu rečemo tudi transakcijski nivo. Iz transakcijskega nivoja pa se potem klic usmeri na zadnji nivo poslovne logike določenega funkcionalnega sklopa, ki mu rečemo podsistem. Na zadnjem nivoju ločimo bančno razvite podsisteme, zunanje podsisteme ter podsisteme, ki smo jih razvili v sklopu razvoja sistema za bančno poslovanje. Z vsemi temi podsistemi se naš sistem integrira na transakcijskem ali storitvenem nivoju, ki smo ga tudi razvili sami.

Storitveni nivo [35] je namenjen temu, da podsisteme razvite v sklopu razvoja sistema za bančno poslovanje, lahko uporabljajo tudi drugi sistemi v banki, npr. elektronsko bančništvo ter internetni portal za stranke. To nam omogoča centralizacijo podatkov za vsebino nekega skupnega podsistema ter zmanjšuje kompleksnosti sistemov in podvajanje funkcionalnosti v različnih sistemih.

Na nivoju integracijskega vmesnika ter na najnižjem nivoju podsistemov se vsebine ločijo po vsebinsko sorodnih sklopih. Oba ta dela sta v domeni BE razvoja. Razdelitev v smiselne zaključene sklope ter definicija poslovne logike na obeh nivojih, pa je v domeni načrtovalcev.

Na sliki 3.8 je ponazorjena arhitekturna shema sistema za bančno poslovanje z glavnimi funkcionalnimi sklopi, ki so del sistema. Barve na shemi ponazarjajo, kje se je sistem oziroma podsistem razvil in se vzdržuje:

- modra barva ponazarja FE razvoj v sklopu sistema za bančno poslovanje,
- zelena barva predstavlja BE razvoj, prav tako v sklopu sistema za bančno poslovanje,
- rumena barva prikazuje vse podsisteme, ki so plod razvoja znotraj informacijskega sistema banke, vendar niso bili razviti v sklopu razvoja sistema za bančno poslovanje, ampak so ga razvijale druge ekipe,
- z belo pa so prikazani sistemi izven informacijskega sistema banke, s katerimi pa je sistem za bančno poslovanje povezan.



Slika 3.8: Arhitektura sistema za bančno poslovanje

Tehnični koncept

Tehnični koncept je krovni dokument načrtovanja določenega vsebinskega sklopa. Celoten sistem za bančno poslovanje smo namreč razdelili na vsebinske sklope, ki so vidni tudi na arhitekturni shemi sistema na sliki 3.8, v obliki področij na integracijskem vmesniku ali med posameznimi podsistemi na najnižjem nivoju. Za vsak tak vsebinski sklop se napiše tehnični koncept, ki se v fazah nadgradenj ali sprememb na posameznih sklopih tudi dopolnjuje.

Tehnični koncept začne nastajati že v fazi poslovne analize, ko se sploh predlaga idejno tehnično rešitev in se potem dopolnjuje do zaključka načrtovanja, ko dokument vsebuje dejansko celotna tehnična navodila za vzdrževanje vsebinskega sklopa. Vsebuje:

- opis procesov, ki se izvajajo v vsebinskem sklopu ter na kakšen način so ti procesi implementirani v sistemu,
- seznam šifrantov, namenjenih tem sklopu, s podrobnimi opisi namena uporabe,
- seznam parametrov, ki krmilijo izvajanje procesov, z opisom nastavitev in načina krmiljenja procesa,
- seznam vseh programskih vmesnikov, z opisom katero aktivnost v procesih se z njimi implementira,
- seznam tabel v podatkovni bazi, z opisom vsebine tabele ter razlago ključnih stolpcev v tabelah.

Po koncu razvoja posameznega vsebinskega sklopa ta dokument zares služi kot osnovno tehnično navodilo in referenčni dokument za podrobne specifikacije delovanja informacijske podpore za ta funkcionalni sklop. Ta dokument je namreč prvi, ki ga preberemo za obnovo znanja, ko se po nekaj razvojnih ciklih spet pojavi zahteva po spremembi na določenem sklopu. Za predajo

znanja oziroma učenje novih članov načrtovalske ekipe je to tudi osnovni dokument, s katerim se dobi občutek o vsebini funkcionalnega sklopa in načinu delovanja obstoječe podpore v sistemu za bančno poslovanje.

Sekvenčni diagram

Tehnika sekvenčnih diagramov [58] izhaja iz UML metodologije. Za potrebe razvoja sistema za bančno poslovanje smo tehniko sekvenčnih diagramov poenostavili in jo prilagodili lastnim potrebam. Po naši metodologiji dela vsakemu primeru uporabe (PU, angl. Use-Case) pripada en dokument s sekvenčnimi diagrami. Ker je v sklopu enega PU implementiranih več podprocesov in jim tudi pripada več ekranskih mask, ima zato vsak podproces pripravljen pripadajoči sekvenčni diagram.

S sekvenčnim diagramov ponazorimo zaporedje izvajanj klicev na različnih logičnih arhitekturnih nivojih sistema:

- nakažemo uporabnikovo (tudi vlogo uporabnika s katero nastopa v sistemu) izbiro akcije (ukaza) na uporabniškem vmesniku,
- ponazorimo klic ustrezne transakcije/storitve na integracijskem/storitvenem nivoju,
- prikažemo klice ustreznih ciljnih podsistemov in
- ponazorimo še vračanje rezultatov vseh zgornjih klicev vse do rezultata, ki se zgodi pri uporabniku na podlagi izbire akcije (ukaza) na uporabniškem vmesniku.

Najprej na sekvenčnem diagramu ponazorimo akterja (uporabnika v določeni vlogi), ki sproža izvedbo podprocesa v okviru primera uporabe. Klic se vedno začne na določenem ekranu z izbiro določene akcije (povezava v brskalniku), ki jo ponazorimo z nazivom na puščici. Z objekti sekvenčnega diagrama, ki jih predstavimo kot stolpce v diagramu, predstavimo logične nivoje arhitekture sistema za bančno poslovanje, ki so potrebni za ustrezno izvedbo podprocesa v posameznem PU. Na vsakem sekvenčnem diagramu so

vedno definirani nivoji aplikacijskega strežnika, integracijskega vmesnika in temeljnih podsistemov, ki zagotavljajo konsistenco delovanja sistema (npr. žurnal).

Prvi klic, na podlagi izbrane akcije na uporabniškem vmesniku, gre najprej na aplikacijski strežnik, kjer potem z imenom na puščici točno določimo katera transakcija se mora ob izbiri akcije na ekranu klicati na integracijskem nivoju. Naprej se transakcija razveja v klice funkcij oziroma storitev enega ali več podsistemov. Vsak klic ima po izvedbi ustrezen rezultat izvedbe, katere vračamo nazaj nivo po nivo v obratnem vrstnem redu kot so se izvajali klici po logičnih arhitekturnih nivojih. Pri klicih transakcij in funkcij je poleg imena naveden tudi način klicanja, mapiranje podatkov na vhod funkcije oziroma storitve in definicija vrednosti ključnih vhodnih atributov.

Sekvenčni diagrami so ključni za FE razvijalce, saj jasno ponazarjajo klice transakcij na integracijskem vmesniku pri izbiri akcije na uporabniškem vmesniku. Tako omogočajo FE razvijalcem pravo informacijo katero transakcijo morajo klicati. Na drugi strani sekvenčni diagram služi tudi BE razvijalcem, saj točno vedo, katere funkcije se kličejo iz posamezne transakcije.

Velik pomen imajo sekvenčni diagrami na koncu predvsem pri testiranju in kasnejšem vzdrževanju sistema, saj lahko zelo hitro ugotavljamo napake zaradi napačnih klicev transakcij ali funkcij oziroma storitev.

Prav tako nam sekvenčni diagrami zelo pripomorejo v primerih optimizacij delovanja sistema. S pregledom sekvenčnih diagramov namreč zelo hitro ugotovimo na kaj vse ima lahko vpliv sprememba na določeni transakciji/funkciji/storitvi. Na ta način se potem lažje odločamo za optimizacije, ker lahko zelo dobro ocenimo obseg spremembe in potencialen vpliv izboljšave.

Specifikacija programskega vmesnika

S pripravo sekvenčnega diagrama se ponavadi identificira vse potrebne transakcije, funkcije ter storitve na različnih podsistemih, ki jih je potrebno specificirati in kasneje sprogramirati. Vsaka transakcija, funkcija, storitev, ki jih

združimo pod pojmom programski vmesnik, se potem natančno vsebinsko definira s ključnimi informacijami vsakega programskega vmesnika:

- vhodnimi parametri,
- izhodnimi parametri,
- psevdo kodo programske logike z logiko upravljanja s podatki (SQL stavki),
- klici drugih programskih vmesnikov.

Poleg naštetih ključnih informacij se v specifikaciji vmesnika definira še vse tabele (podatkovne objekte), ki nastopajo v poslovni logiki vmesnika ter odvisne vmesnike, ki se v novem vmesniku uporabljajo.

Vhodni/izhodni parametri so definirani z imenom in programskim tipom, ter opisom pomena parametra. Pri opisu je ključna tudi zaloga vrednosti parametra in informacija, če se ta zaloga vrednosti polni iz določenega šifranta.

Za psevdo kodo se uporablja ogrodje nekega temeljnega programskega jezika (podobno kot basic ali pascal jezik) z jasnimi komentarji v psevdo kodi, da vsakdo razume pomen napisane poslovne logike. V glavnem je psevdo koda sestavljena iz neke "if-then-else" logike, vsebuje vključene elemente podatkovne baze (tabele s pripadajočimi SQL stavki) ter klice drugih vmesnikov. Zelo pomembno ali kar ključno pravilo pri pisanju psevdo kode je, da ta vsebuje čim več opisnih komentarjev poslovne logike. Samo na ta način namreč zagotavljamo, da lahko po letih brez sprememb na določenem vmesniku, ob branju specifikacije programskega vmesnika, zelo hitro izluščimo vsebino in identificiramo potencialno napako v delovanju ali mesto, kjer bi želeli implementirati spremembo.

Tako kot vsi prej opisani elementi dokumentacije sistema za bančno poslovanje, so tudi specifikacije programskih vmesnikov redno vzdrževane in odražajo res pravo stanje delovanja sistema. To zagotavljamo z načinom dela oziroma sprememb, ki se vedno naredijo na celotni razvojni poti. V primeru novega razvoja, nadgradenj, sprememb posameznih funkcionalnosti se vedno popravi vse dokumente po fazah razvoja:

- najprej se popravi dokumentacija poslovne analize (PU, ekranske maske, definicije izpisov),
- potem vsa načrtovalska dokumentacija (arhitektura, tehnični koncept, sekvenčni diagrami, podatkovni model, specifikacije programskih vmesnikov),
- šele na koncu se popravlja programska koda po prej dopolnjenih specifikacijah.

Na prvih straneh ta specifikacija vsebuje tabelo s ključnimi podatki specifikacije. To strukturo smo v okviru razvoja sistema za bančno poslovanje definirali zato, da lahko na enostaven način:

- zgeneriramo seznam vseh specifikacij,
- pridobimo soodvisnosti med programskimi vmesniki, s čimer si olajšamo identifikacijo potencialnega vpliva sprememb na določenem vmesniku,
- pridobimo seznam elementov podatkovne baze (v glavnem tabel), ki se v določenih vmesnikih uporabljajo,
- generiramo seznam vmesnikov na katere vpliva sprememba na tabeli v podatkovni bazi,
- vodimo skrbništva in enostavno izvajamo spremembe skrbništev.

3.2.4 Razvoj

Pri pripravi vseh novih funkcionalnih sklopov ali konceptualnih sprememb v sistemu za bančno poslovanje oziroma enem izmed podsistemov celotnega sistema, že v fazi načrtovanja sodelujejo tudi razvijalci, sistemski inženirji in administratorji podatkovnih baz. Na ta način se trudimo zagotoviti ustrezno načrtovano končno rešitev in se izogniti kasnejšim neskladnostim. Tako se skušamo v največji meri izogniti tehničnim oviram in preprečimo spreminjanje konceptov v fazah razvoja ali še kasnejših fazah, ko to lahko postane časovno in stroškovno neučinkovito [66].

Kot smo že v podpoglavju projektnega vodenja razvoja sistema za bančno poslovanje zapisali, se razvojna faza deli na dva dela:

- razvoj zalednega dela z glavnino poslovne logike – Back-End (BE) razvoj,

- razvoj uporabniškega vmesnika – Front-End (FE) razvoj.

FE razvoj začne svoje delo že takoj po zaključku poslovne analize. Najprej pripravijo surove ekranske maske (brez zalednega dela) po logiki napisani v PU s prilogo ekranskih slik. Razvoj uporabniškega vmesnika in dela poslovne logike na aplikacijskem strežniku temelji na programskem jeziku java in arhitekturi JCA (angl. Java Enterprise Edition Connector Architecture). FE razvijalci si iz pripravljenih primerov uporabe zgradijo tudi že objektni model [58], ki služi kot osnova nadaljnjemu razvoju funkcionalnosti v okviru vsakega PU.

Vzporedno s pripravo ekranskih mask na FE, se v načrtovanju pripravlja sekvenčne diagrame in specifikacije programskih vmesnikov, ki so ključen vir za pripravo programov na BE razvoju. V teh specifikacijah je definirana glavna poslovna logika sistema bančno poslovanje. BE razvoj temelji na zakonitostih CICS [29] transakcijskega sistema na osrednjem računalniku in se izvaja v programskem jeziku COBOL.

Ko je vmesniški del (angl. input/output) programa po specifikaciji na BE pripravljen, se ta po navodilih v sekvenčnem diagramu že začne implementirati na FE. V tem času se na BE programira poslovna logika in ko je ta zaključena in interno testirana, se prenese v FE razvojno okolje. Tam potem FE razvijalci še dokončajo razvoj po PU in naredijo svoj interni test (angl. unit test) preden predajo program v funkcionalno testiranje.

Ko BE sprogramira program po specifikaciji programskega vmesnika, tudi dopolni specifikacijo z imenom programa, da lahko zagotovimo popolno sledljivost od specifikacije programskega vmesnika, ki so jo pripravili načrtovalci, do izvajalnega programa in olajšamo vzdrževanje sistema v njegovih zrelih fazah.

Za uspešno izvedbo obeh delov razvoja je pomembno, da obe ekipi dobro sodelujeta in komunicirata. Pri vsem tem jim v času razvoja in kasneje testiranja nudijo podporo tudi načrtovalci s svojim poznavanjem vsebine uporabniške zahteve in zasnove rešitve. Prav tako jim načrtovalci pomagajo z dodatnimi pojasnili pri dokumentaciji (konceptu, sekvenčnih diagramih, specifikacijah vmesnikov) in dopolnjevanjem dokumentacije v primeru odkritih pomanjkljivosti.

Za spremljanje izvedbe pa se tako, kot v vseh prejšnjih in tudi kasnejših fazah, uporablja orodje za spremljanje zahtev. Ta nam z lastništvom zahtevka in ustreznim statusom zahtevka natančno pove, kje se zahtevek nahaja in v kakšni fazi razvoja je. Na ta način ima tudi vodja projekta in vodje posameznih skupin pregled nad trenutnim stanjem razvoja posameznih zahtevkov in vzvode za ustrezno ukrepanje v primeru zakasnitev.

3.2.5 Testiranje

Zaradi več-nivojske arhitekture sistema za bančno poslovanje se tudi testiranje izvaja na različnih nivojih. Skladno s tem potrebujemo več okolij za razvoj in testiranje novih funkcionalnosti v sklopu posameznega razvojnega cikla. Tako moramo za potrebe razvoja ter ustreznega testiranja imeti vzpostavljenih več razvojnih in testnih okolij. To pomeni, da moramo v vsakem okolju imeti povsem ločen del izvajalne programske opreme za uporabniški vmesnik in aplikacijski strežnik, ločeno izvajalno programsko opremo za integracijski vmesnik in podsisteme ter ločene podatkovne baze. Zato imamo v banki vzpostavljena naslednja okolja:

- razvojno okolje za BE razvoj (okolje rzv),
- razvojno okolje za FE razvoj (kjer je BE koda že stabilna – v testu, okolje tst),
- funkcionalno testno okolje (okolje fet),
- uporabniško testno okolje (okolje iuat) in

- produkcijsko okolje (okolje prod).

Testiranje se zaradi narave dela izvaja na vseh nivojih razvoja. Vsaka razvojna ekipa mora namreč pred zaključkom svojega dela preveriti, ali razvita podpora ustrezno deluje. Hkrati z ustreznostjo nove rešitve je nujno preveriti tudi potencialni vpliv nove rešitve na obstoječe delovanje (regresijski test), s čimer zagotovimo minimalno število napak pri prehodu v funkcionalno testiranje.

Pravo testiranje, ki ga izvajajo poslovni analitiki in delno načrtovalci, se začne s funkcionalnim testiranjem, ko se preveri delovanje funkcionalnosti sistema po testnih scenarijih. Testiranje se izvaja po pripravljenem PU oziroma izvedenih testnih scenarijih, ki izhajajo iz PU.

Funkcionalnem testiranju sledi še uporabniško testiranje, ki se prav tako izvaja po pripravljenih testnih scenarijih izvedenih iz PU in ga izvajajo končni uporabniki iz poslovne mreže ter služb, kjer se sistem za bančno poslovanje uporablja.

V fazi testiranja se nam velikokrat zgodi, da je udeležba končnih uporabnikov iz poslovne mreže premajhna, zato poslovni analitiki sodelujejo tudi pri izvajanju uporabniškega testiranja. Ker je sistem za bančno poslovanje funkcionalno zelo obsežen in se obseg funkcionalnosti z vsako novo verzijo še povečuje, se temu sorazmerno povečuje tudi obseg potrebnega testiranja. Predvsem je zaradi količine obstoječih funkcionalnosti problematično regresijsko testiranje, ki ga še vedno v celoti izvajamo ročno. Celoten sistem za bančno poslovanje pokriva namreč okrog 250 primerov uporabe za katere je definiranih več kot 5000 testnih scenarijev, kar zahteva veliko časa in je zato težko izvedljivo v celoti.

Druga teza: *predlog izboljšave je avtomatizirati regresijsko testiranje. Vso obstoječo funkcionalnost bi morali testirati avtomatsko in s tem sprostiti končne uporabnike s tega dela testiranja. Tako bi se lahko posvetili testiranju predvsem novih vsebin in skrajšali čas uporabniškega testiranja ter zmanjšali število potrebnih testerjev.*

3.3 Trenutno stanje razvoja sistema za bančno poslovanje

Projekt razvoja sistema za bančno poslovanje je od leta 2005 pa do danes prinesel veliko izboljšav pri delu v bančnih poslovalnicah in tudi drugih organizacijskih delih banke. Prvi razvojni cikli so bili usmerjeni v temeljne poslovne procese v bančnih poslovalnicah, katerih podpora ni bila ustrezna in se je izvajala ročno ali zelo razpršeno v več sistemih. Z razvojem novega sistema se je šele zares začelo razvijati pravo informacijsko podporo poslovnim procesom. Osnovna zahteva je namreč bila, da je potrebno uporabnike razbremeniti razmišljanja o tehniki izvedbe posla (kje – v kateri aplikaciji – morajo kaj narediti) in jim omogočiti osredotočanje na vsebino posla.

Razvoj sistema za bančno poslovanje se je od leta 2005 pa do leta 2014 vsako leto nadgradil z dvema verzijama. Tako smo od verzije 1, ki je šla v produkcijsko okolje v začetku leta 2006, danes prišli do verzije 17, katere prehod v produkcijo je bil izveden v začetku junija 2014. Zelo pomembno je poudariti, da so bili razvojni cikli v principu vedno uspešni, saj se nikoli ni zgodilo, da redne polletne verzije nadgradnje ne bi razvili in pripravili za uporabo.

Projekt razvoja sistema za bančno poslovanje je bil že v začetku zasnovan v sodelovanju z več partnerskimi podjetji in v tesnem sodelovanju s takratnimi strateškimi poslovnimi partnerji banke. V začetnih razvojnih ciklih je bil vpliv zunanjih partnerjev zelo velik. Poudariti moramo tudi, da je bila kohezija članov projekta ter zagnanost že od samega začetka na zelo visoki ravni. Zagotovo je to tudi posledica izbora članov projekta v okviru banke, kjer so se zbirali res najboljši kadri, kadri z vizijo in željo po tem, da naredijo veliko korist za banko. Tu želimo povedati, da je projekt deloval kot eno in so bili zares vsi člani projekta, ne glede na partnersko podjetje iz katerega so prihajali, enakopravni. To se skozi vsa leta izkazuje tudi na druženju članov projekta izven delovnega časa, saj še danes na neformalna srečanja projekta prihajajo posamezniki, ki so bili zraven na začetku in že več let niso člani

projekta ali zaposleni v katerem od partnerskih podjetij.

V nadaljevanju bomo prikazali kronologijo in najpomembnejše vsebine razvoja sistema za bančno poslovanje skozi leta. Prikazali bomo tudi ključne spremembe, izboljšave v postopkih (procesu) dela, ki so se v letih razvoja sistema že zgodile.

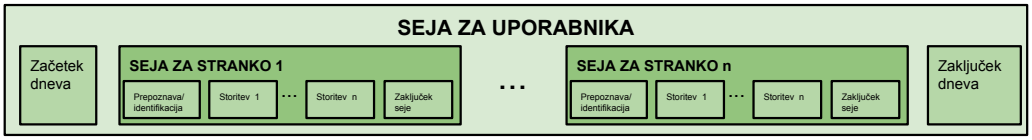
Ključno za začetek razvoja sistema za bančno poslovanje je bilo odpraviti pomanjkljivosti obstoječe podpore pri delu v bančnih poslovalnicah. Predvsem so bile težave pri vodenju gotovinskega poslovanja in kontrol pri zaključkih dneva. Zato je bilo v prvih verzijah nujno urediti:

- gotovinsko in blagajniško poslovanje,
- poenotiti zaključke dneva ter
- poenostaviti izvajanje finančnih transakcij za uporabnike v poslovni mreži.

Za podporo tega je bilo potrebno razviti še par ključnih konceptov sistema, ki so tudi omogočili spremembe in izboljšave procesov dela v poslovalnici. Ključna sta koncepta:

- seje za uporabnika in
- seje za stranko.

Oba koncepta smo združili v podsistem za upravljanje sej (angl. Session Management), ki je prikazan na 3.9. Po novem tako uporabnik oziroma delavec v poslovni mreži vsak dan začne z novo sejo ter temeljnimi začetnimi aktivnostmi v dnevu (npr. za blagajnika to pomeni prevzem kasete iz trezorja in aktiviranje svoje blagajne). Ko ima uporabnik svojo sejo aktivno, lahko začne z izvajanjem sej za stranke. Seja za stranko se vsakič začne s prepoznavo in identifikacijo stranke, ki pride k bančnemu delavcu. Potem se za to stranko izvajajo storitve in na koncu (ob zaključku seje) se vse storitve, narejene za stranko v okviru seje, prikažejo na ekranu. Na ta način ima



Slika 3.9: Koncept obravnave sej

bančni delavec celovit pregled nad vsemi storitvami, ki jih je za stranko opravil v okviru seje. Tudi potrdilo o opravljenih storitvah je pregledno in ko ga stranka podpiše, je seja za stranko zaključena, bančni delavec pa lahko začne s prepoznavo nove stranke, kar pomeni z začetkom nove seje za stranko.

Ob koncu svojega delovnega dne, bančni delavec samo izvede postopek zaključka dneva, kjer se vsi relevantni dogodki (finančni in nefinančni) pregledno prikažejo na ekranu. Delavec mora še preveriti:

- ali se stanje dokumentov vodenih v sistemu, ujema s fizičnim stanje ter
- ali se stanje v njegovi blagajni ujema s stanjem v sistemu.

Če se vsi podatki ujemajo, to pomeni da ni prišlo do nobenih napak in se seja za uporabnika lahko uspešno zaključi. V primeru odstopanj med dejanskim stanjem in stanjem v sistemu imamo tudi definirane postopke kontroliranja in dodatnega odkrivanja napak, ki so prav tako informacijsko podprti. Ko vsi uporabniki v poslovni enoti naredijo svoj zaključek, je samo še na vodji poslovalnice, da naredi zaključek enote.

Drugi ključni koncept je princip beleženja vseh dogodkov, ki je pomemben za hitro izvajanje zaključka dneva ter odkrivanja potencialnih napak pri delu. Ta podsistem smo poimenovali žurnal (angl. Journal). To je dnevnik vseh aktivnosti, ki se izvajajo v sistemu za bančno poslovanje. Vsako aktivnost v sistemu zato enolično označimo z dogodkom. Vsi dogodki so vezani na sejo za uporabnika, kjer so v storitve vključene tudi stranke, pa tudi na sejo za stranko. Za vsako aktivnost v sistemu se v žurnal zapisujejo vsi ključni podatki, ki zagotavljajo revizijsko sled delovanja sistema ter omogočajo različne kontrole:

- šifra dogodka,
- časovni žig,
- uporabnik,
- delovno mesto,
- poslovna enota,
- status izvedbe in
- ostale ključne podatke, ki se razlikujejo glede na vrsto dogodka.

Beleženje vseh dogodkov na enem mestu nam tako na zelo enostaven način omogoča zaključke, možnost raznih kontrolnih obdelav, izdelavo poročil, izvoz podatkov za knjigovodstvo.

V času implementacije prvih verzij sistema za bančno poslovanje je bila glavnina dela v poslovalnicah vezana na izvajanje finančnih transakcij za stranke. Z enotnim uporabniškim vmesnikom za izvajanje finančnih transakcij, ne glede na to, v katerih sistemih so bili vodeni računi strank, se je izvajanje zelo poenostavilo, saj je za tehniko finančne transakcije skrbel sistem, uporabnik pa je lahko bil osredotočen samo na vnos vrednosti in zagotavljanje storitve stranki. Da bi se finančne transakcije celovito lahko izvajale je bilo potrebno še zagotoviti nadzor nad gotovinskim poslovanjem. Ključno za to je bilo, da v vsakem trenutku vemo koliko gotovine je v poslovalnici. To smo omogočili z vodenjem gotovine v trezorju poslovalnice in vseh blagajnah na blagajniških delovnih mestih. Ob vsaki seji za stranko, v sklopu katere so bile gotovinske transakcije, sistem blagajniku sam ponudi podatke koliko gotovine mora vplačati ali izplačati. Ko to uporabnik naredi, se ustrezno ažurira stanje njegove blagajne v sistemu. Ker to velja za vse uporabnike in vse blagajne v poslovni enoti, imamo na ta način natančen nadzor nad gotovino v tej enoti in na enak način v vseh ostalih enotah.

Z vsemi temi osnovnimi izboljšavami delovnih procesov v prvi verziji sistema za bančno poslovanje (v letu 2006) smo zaposlene v poslovni mreži

predvsem razbremenili stalnega razmišljanja o tehniki izvajanja storitev za stranko. Ta del smo prenesli na sistem za bančno poslovanje, ki na podlagi vhodnih podatkov ve, kako je potrebno transakcijo izvesti. Zaposleni so se tako lahko osredotočili bolj na delo s stranko in pravo bančno svetovanje.

Poleg tega smo zaključke dneva v povprečju skrajšali za več kot polovico in tako omogočili podaljšanje delovnega časa v poslovalnicah. Pred uvedbo novega sistema je bilo po zaprtju poslovalnice za stranke potrebno izvajati vse zaključke ročno. Ročno je bilo potrebno iz vseh sistemov zbrati podatke in preveriti ustreznost ter prešteti vso gotovino v enoti in jo primerjati z jutranjim stanjem in evidenco vseh izvedenih transakcij. Ta zaključek je tako trajal tudi dve uri in več. Z uvedbo sistema za bančno poslovanje se je ta zaključek skrajšal na dobre pol ure.

Razvoj sistema za bančno poslovanje se je po prvih uspehih in velikem zadovoljstvu končnih uporabnikov nadaljeval v dvoletnih ciklih do konca leta 2013. Od leta 2014 naprej pa razvoj sistema ne poteka več projektno ampak linijsko. V nadaljevanju so navedene ključne vsebine razvoja, ki so se tekom let implementirale v sistemu za bančno poslovanje:

- verzija 2 (2006):
  - integracija s centralnim trezorjem,
  - uvedba več-bančne podpore in implementacija v drugih bankah,
- verzija 3 (2006):
  - uvedba evra,
  - podpora DNT (dnevno nočni trezor),
  - odpiranje in pregled stanja na računih pravnih oseb,
  - integracija s plačilnim prometom,
- verzija 4 (2007):
  - podpora procesu sklepanja in obravnave depozitov in varčevanj,

  - optimizacija žurnala,
  - celovit pogled na stranko – 1.del,

- verzija 5 (2007):
  - podpora trženjskim akcijam,
  - vzpostavitev sistema za podporo kontaktov s strankami,
  - administracija tržnih poti za stranko,
  - prenova sistema vlog za plačilne kartice,
  - plačilni nalog SEPA,

- verzija 6 (2008)
  - podpora nakazil v tujino,
  - vpogled in tiskanje E-izpiskov,
  - trajni nalog SEPA,
  - varnostno SMS obveščanje za plačilne kartice,

- verzija 7 (2008):
  - podpora vseh storitev na osebnih računih,
  - depozitna enota,

- verzija 7.5 (2009):
  - upravljanje s kontakti strank,
  - on-line povezava s sistemom kartičnega poslovanja,
  - nadgradnja SEPA pravil,

- verzija 8 (2009):
  - sistem za upravljanje s kontakti strank,
  - podpora za kontaktni center,

  - prenos trženjskih akcij v sistem za upravljanje s kontakti strank,
  - osebno bančništvo,

- verzija 9 (2010):
  - telefonska banka,
  - nadgradnja sistema za upravljanje s kontakti strank,
  - prodajni portal,

- verzija 10 (2010):
  - prenova knjigovodstva,
  - nadgradnja celovitega pogleda za stranko,

- verzija 11 (2011):
  - analiza kreditnega procesa,
  - omejitev poslovanja,
  - zapiranje računa,

- verzija 12 (2011):
  - informativni izračun kredita,

- verzija 13 (2012):
  - kreditni proces za enostavne kredite,
  - podpora izvajanju izterjav,

- verzija 14 (2012);
  - celoten kreditni proces,

- verzija 15 (2013)
  - finančno svetovanje,

  - procesi sprememb na kreditih,

- verzija 16 (2013)

  - procesi sprememb na kreditih,
  - elektronska dokumentacija, skeniranje na delovnem mestu,

- verzija 17 (2014):

  - avtomatske blagajne,
  - prilagoditve potrebam ameriške zakonodaje.

Sistem za bančno poslovanje je s prvimi verzijami prinesel zares revolucionarne spremembe zaposlenim v poslovni mreži. Predvsem je olajšal njihovo delo in jih razbremenil velikega števila ročnih postopkov in kontrol.

Ključne prelomnice v razvoju sistema so po prvih verzijah v letu 2006 sledile še v naslednjih letih. V letu 2007 smo uporabnikom ponudili celovit pogled na prepoznano stranko, ki je na enem ekranu prikazal vse ključne podatke (celovito finančno stanje, segmentacijo stranke, profil stranke, ključne osebne podatke stranke kot je datum rojstva, zaposlitev) o stranki za bolj celovito obravnavo stranke v bančnih poslovalnicah. To je bil prvi del sistema za celovito upravljanje s strankami (angl. CRM – Customer Relationship Management) [36], ki smo ga v naslednjih letih še dograjevali.

Naslednja prelomnica se je zgodila v letu 2009, ko smo razvili nov sistem za upravljanje s kontakti strank in tako omogočili enotno obravnavo stranke ne glede na to, kje v banki pridemo v stik z njo. To je bil drugi ključni del do celovitega CRM sistema. V istem letu se je uporaba sistema za bančno poslovanje razširila še na kontaktni center in telefonsko banko. Podpora telefonski banki je prinesla nov proces dela s strankami zaradi posebnosti dela po telefonu. Zato smo morali širiti osnovni koncept seje za stranko.

Sistem za upravljanje s kontakti je bil eden prvih podsistemov, ki je bil razvit storitveno po konceptu SOA (Storitveno Orientirane Arhitekture) [35, 33]. Na ta način je na voljo za integracijo tudi drugim sistemom in aplikacijam s ciljem celovitega vodenja stikov s stranko ne glede na tržno pot.

Z razvojem tega podsistema smo zato v arhitekturni shemi sistema morali poleg integracijskega vmesnika dodati še storitveni vmesnik. Na tem nivoju so tako vsem sistemom v banki na voljo storitve kot so kreiranje naloge, dodajanje kontakta, pošiljanje/prevzem naloge. Na ta način je v naslednjih letih zelo enostavno potekala integracija upravljanja s kontakti v vse tržne poti in zaledne sisteme za podporo delu s strankami.

V naslednjem letu, letu 2010, je bila v banki ključna zamenjava sistema za računovodstvo, kar je vplivalo na vse sisteme, ki izvajajo kakršnekoli storitve in te poročajo v glavno knjigo. Implementacija je bila izvedena brez večjih težav.

3.3.1 Prekrivanje razvojnih ciklov

V letu 2010 smo začeli z največjim še nepodprtim sklopom storitev v sistemu za bančno poslovanje - kreditnim procesom. Najprej smo, kot je definirano v razvojnem ciklu, izvajali poslovno analizo, ki se je zelo zavlekla in ni prinesla najboljših rezultatov. Razlog za to je bil, da uporabniške zahteve niso bile dovolj dobro definirane in usklajene med vsemi udeleženimi uporabniki. Kreditni proces je bil razdrobljen na več delov in poenotenje je bilo zelo težavno:

- na eni strani smo imeli procesnike, ki so imeli zelo grobo videnje kreditnega procesa,

- potem so bili predstavniki uporabnikov, ki so imeli zelo podrobno poznavanje kreditov, a so manjkala pravila za izračunavanje,

- na tretji strani pa je bila razdrobljena obstoječa podpora, ki je bila med seboj neusklajena,

- kar je bilo najhuje pa je bilo to, da so se vzporedno s poslovno analizo spreminjala pravila (uporabniški priročnik, zakonodaja) za izvajanje kreditov.

Poslovna analiza se je tako iz predvidenega pol leta zavlekla na leto in pol in v celoti še vedno ni zaključena.

Pri implementaciji kreditnega procesa smo na projektu naredili spremembo, da smo poslovno analizo izvajali v enem razvojnem ciklu prej kot se je začelo potem načrtovanje in razvoj. Tako so v enakem obdobju poslovni analitiki izvajali analizo kreditov, načrtovalci in razvijalci pa razvijali nekaj povsem drugega. In ko smo v naslednjem oziroma po dveh razvojnih ciklih načrtovalci začeli z delom na kreditnem procesu, smo ugotovili še ogromno nedorečenosti in nekonsistentnosti v sami analizi. Prav zato smo potem še prvi razvojni cikel del načrtovalske ekipe in poslovnih analitikov izvajali skupaj v fazi poslovne analize in načrtovanja, da smo na koncu prišli do načrta, ki smo ga lahko zares začeli implementirati. Tako smo se naučili, da zamikanje razvojnih ciklov ni bilo najbolj učinkovito in primerno.

Iz tega tako lahko postavimo **tretjo tezo** naloge: *za povsem nov sklop funkcionalnosti (novo poslovno področje) je nujno, da v fazi poslovne analize stalno sodelujejo načrtovalci in po potrebi tudi razvijalci.*

Največ težav pri kreditnem procesu je bilo s popisovanjem poslovnega procesa in aktivnosti. Tudi pravila izračunavanja parametrov kredita niso bila nikjer jasno definirana, zato je bilo tudi to potrebno dokumentirati. Po vseh različnih zapletih smo šele v drugi verziji leta 2011 naredili podporo za informativne izračune kreditov, v letu 2012 pa dejansko izvedli podporo za celoten kreditni proces. Je pa potrebno poudariti veliko dodano vrednost nove podpore:

- pred uvedbo podpore v sistemu za bančno poslovanje se je namreč kreditni proces izvajal v treh do desetih sistemih oziroma aplikacijah (odvisno od vrste zavarovanj za kredit),

- z uvedbo podpore v sistemu za bančno poslovanje pa se proces od začetka do konca izvaja v enem sistemu, ki se v zaledju integrira z vsemi ciljnimi sistemi.

Na ta način je uporabnikom delo zelo olajšano in odobravanje kredita je v veliki meri avtomatizirano, kar tudi pomeni velik časovni prihranek pri sklepanju kredita. Proces smo skrajšali iz nekaj ur za najenostavnejše kredite, na manj kot uro. Na ta način smo omogočili, da je banka na trgu ponudila nove kreditne storitve.

Dodatno moramo poudariti, da so določeni specifični in zelo redki krediti še vedno ostali nepodprti in se bodo implementirali v letu 2014. V letu 2013 pa smo se lotili procesov sprememb na kreditih, ki se lahko zgodijo v življenju kredita do njegovega dokončnega poplačila. Tu so se ponovno izkazale pomanjkljivosti v procesu razvoja na enak način kot pri analizi procesov odpiranja kredita. Poslovni proces smo morali ohraniti enak in zato je bil ta razvoj spet zelo kompleksen. Prav zaradi tega in množice vrst kreditov, ki niso več v aktualni ponudbi, a so še aktivni, še nismo podpore razvili v celoti, a je v planu da jo v letu 2014 ali prvi polovici 2015 zaključimo.

Poleg težav in izzivov s podporo kreditnega procesa, smo v letu 2013 izvedli še eno večjo nadgradnjo, ki se nanaša na spremembo pri tiskanju dokumentov (internih in za stranko). Uvedli smo skeniranje na delovnem mestu in avtomatsko pošiljanje dokumentov v arhivski dokumentni sistem. Za skeniranje smo se odločili zaradi velikega števila vhodne dokumentacije (dokumenti, ki jih stranka prinese v banko kot potrdila za izvedbo posla). Če te vhodne dokumentacije ne bi bilo toliko, bi bilo bolj smiselno razmišljanje o podpisnih tablicah. Na ta način smo spet skrajšali zaključek dneva v poslovalnici, zmanjšali število tiskanj, olajšali dokumentno poslovanje ter zmanjšali možnost napak pri dokumentaciji.

Sistem za bančno poslovanje je danes ključen sistem za delo v bančnih poslovalnicah, kontaktnem centru, telefonski banki in zalednih službah, ki izvajajo podporo delu v poslovalnicah. To je eden izmed ključnih razlogov, zakaj je razvoj sistema še vedno zelo aktualen. Sistem se namreč, kljub prehodu v redno linijsko delo, še zelo intenzivno nadgrajuje z novimi funkcionalnostmi in tudi stalno vzdržuje in prilagaja novim potrebam uporabnikov.

3.3.2 Način planiranja razvoja

Če povzamem zgoraj napisane ključne razvite funkcionalnosti po posameznih razvojnih ciklih sistema za bančno poslovanje in jih primerjam s planiranimi funkcionalnostmi, moram poudariti, da je bilo razmerje med planiranim in realiziranim iz verzije v verzijo boljše. Na začetku projekta je bilo planiranje namenjeno predvsem zagotovitvi sredstev za razvoj. Funkcionalnosti, ki jih je bilo potrebno podpreti, se je glede na prioriteto uvrstilo potem v ustrezne razvojne cikle. Ko se je razvoj dejansko začel s podrobno procesno in poslovno analizo, se je šele ocenil obseg dela za razvoj posamezne funkcionalnosti. Če se je ugotovilo, da je dela za en razvojni cikel preveč, se je zahtevo prestavilo v naslednji cikel ali pa se je delno realizirala v prvem ciklu, delno v drugem.

Realizacija, glede na prvotni plan, v teh primerih zato ni bila ustrezna. V nadaljnjih razvojnih ciklih razvoja sistema se je planiranje izboljševalo in tudi oceno potrebnega časa za razvoj se je skušalo dovolj dobro določiti že v fazi planiranja. V praksi pa se je pri vseh večjih vsebinskih sklopih (uporabniških zahtevah) dejansko zgodilo, da so bile ocene še vedno preskromne. V času od mojega prihoda na projekt konec leta 2008 (razvoj verzije 7 sistema) pa do danes, se je pri večini verzij zgodilo, da se je vsebina verzije od plana pa do izvedbe spremenila. Po navadi se je kak večji vsebinski sklop razpotegnil čez več verzij, v posamezne verzije pa se je dodajalo vsebine, ki so se kot uporabniške zahteve porodile med samim razvojnim ciklom. Eden izmed razlogov za spreminjanje vsebin posameznih verzij v času razvoja je bil, da so bili vsebine (poslovne zahteve) razvojnih ciklov preveč optimistično planirane in so planirane verzije zajemale preveč uporabniških zahtev. Ključni razlog za neizpolnjevanje prvotnega plana in spreminjanje plana so bile preohlapno definirane poslovne zahteve. V veliki meri se je za vse glavne vsebinske sklope (uporabniške zahteve, ki so pomenilo večje konceptualne spremembe ali novo podporo večjega poslovnega sklopa) v fazah poslovne analize in načrtovanja šele začela podrobna definicija uporabniške zahteve in izkazalo se je, da je potrebnega več dela kot je bilo planiranega ali pa da uporabniška zahteva

sploh še ni zrela za razvoj. To je pomenilo, da imamo preveč alternativ in neznank v uporabniški zahtevi in se razvoj ni mogel nadaljevati brez redefinicije zahteve.

Takšne zadeve so se vedno sproti usklajevale med vodjo projekta in nadzorno skupino. Ker je bila ekipa projekta zelo utečena in zaupanja vredna predvsem zaradi rezultatov, ki jih je skozi leta izkazala, se je praktično vedno zaupalo presoji in ocenam vodje projekta (vodja IT izvedbenega dela projekta) in razvojne ekipe ter se naredilo spremembo plana.

Praksa je bila, da se je planiran čas za razvoj neke funkcionalnosti, ki je bila prioritetna, lahko podaljšal, drugo ali več manj prioritetnih funkcionalnosti pa se je prestavilo v naslednjo verzijo.

Druga možnost je bila še, da se je prioritetna funkcionalnost razdelila na več delov – faz. Tako so se najprej (v prvi fazi) razvijale funkcionalnosti, ki so bile prioritetne, manj prioritetne funkcionalnosti pa so se razvijale v naslednjih verzijah. Zgodilo se je tudi, da v času, ko je manj prioritetna funkcionalnost prišla na vrsto, ta ni bila več aktualna zaradi novih okoliščin in se posledično sploh ni razvijala.

Po koncu vsakega leta razvoja (dveh letnih razvojnih ciklov) se je za tisto leto pripravilo tudi zaključno poročilo projekta razvoja sistema za bančno poslovanje. V tem poročilu se je podrobno opisalo kakšen je bil prvotni plan (katere vsebine naj bi se razvijale) in kakšna je bila končna realizacija (kaj se je dejansko razvilo). Glede na prej opisane spremembe vsebin tekom produkcijskega cikla in sprotnega usklajevanja sprememb med uporabniki, nadzorno skupino in vodstvom projekta, je bilo leto praktično vedno zaključeno kot uspešno. Razlogi za spremembe vsebin in zamik v naslednje verzije so bili argumentirano predstavljeni s strani razvojne ekipe, razumljeni s strani uporabnikov in tako usklajeni, da so bili uporabniki kljub spremembam vseeno zadovoljni z rezultati posamične verzije.

Spreminjanje planov oziroma preveč optimistično planiranje se je v času od mojega prihoda na projekt najbolj izkazalo v naslednjih primerih:

- v letu 2009 smo planirali nov podsistem za upravljanje s kontakti, pod-

poro telefonske banke in še manjše zahteve. Zaradi obsega vzpostavitve sistema za upravljanje s kontakti, se je podpora telefonski banki premaknila v leto 2010. Tako se je dejansko planirana vsebina za leto 2009 raztegnila še na leto 2010,

- podobno se je zgodilo leta 2011. Takrat smo med drugim planirali podporo celotnega kreditnega procesa za odpiranje novih kreditov (kot zanimivost še to, da je bila podpora kreditom prvotno planirana že za leto 2008). Zaradi velikega obsega vsebine in spreminjanja uporabniških zahtev smo v letu 2011 omogočili samo izdelavo informativnih izračunov, šele v letu 2012 pa smo v celoti podprli kreditni proces z izjemami nekaj posebnih vrst kreditov, ki jih bomo podprli šele v letu 2014,

- v letu 2013 smo planirali podporo procesom v teku kredita, vendar smo morali del razvoja prestaviti še v leto 2014. Funkcionalnost pa zaradi težav pri testiranju in posledično dvoma uporabnikov o ustreznosti sploh še ni omogočena v produkciji.

Planiranje smo delno izboljšali od leta 2012 naprej, ko smo v plan vsake verzije vključili veliko kvoto razvojnih kapacitet za vzdrževanje delovanja sistema, ki je bila namenjena odpravi odkritih napak prejšnjih verzij in dodatnim zahtevam ali spremembam zahtev tekom razvoja verzije. Vendar za optimalno razporejanje virov ta način ni najboljši, ker se potem to kvoto za vzdrževanje zlorablja za dodajanje novih zahtev tekom razvojnega cikla.

Glede na napisano lahko rečemo, da obstoječi način planiranja ni najbolj primeren, saj se spreminjanje in le delno uresničevanje prvotnega plana redno dogaja. Zakaj je temu tako? Uporabniki oziroma naročniki razvoja novih funkcionalnosti sistema za bančno poslovanje prihajajo iz različnih delov banke. Ker je interes vsakega za svoje vsebine velik in ker je zaradi interesov ustrezna prioritizacija zahtev praktično nemogoča, se vse do faze izvedbe (razvoj v IT) ne filtrira vsebine po prioritetah globalno (v okviru cele banke) in ustrezno razporedi (planira) na daljše časovno obdobje (več verzij). Ko pa

pride do izvedbene faze v IT, se pač ne da narediti vsega in takrat se potem predlaga naročnikom, da se uskladijo o prioritetah in ponovnem planiranju.

V nedorečenih in spreminjajočih se uporabniških zahtevah se skriva tudi ena glavnih slabosti lastnega razvoja (angl. in-house development), saj interno ceno razvoja in sprememb v času razvoja le redki lahko enačijo z računom, ki ga morajo plačati v primeru naročenega razvoja ali prilagoditve kupljene rešitve. Težave pri usklajevanju različnih uporabnikov so se še povečale z ukinitvijo projekta za razvoj sistema za bančno poslovanje in prehodom razvoja v linijsko delo. Globalno (skupna, enotna) določanje pomembnosti zahtev (prioritizacija) se je na ta način še otežilo in posledično povzročilo napetosti med različnimi uporabniki.

Ravno zaradi teh razlogov je bila v letu 2013 v banki izvedena reorganizacija, katere glavna dodana vrednost je bila centralizirano upravljanje z zahtevami uporabnikov. Za to je bilo zasnovano posebno področje za upravljanje z zahtevami v banki (angl. demand management) [39]. Osnovni namen nove organizacijske enote naj bi bil centralizirano vodenje zahtev vseh uporabnikov, prioritizacija teh zahtev, analiza upravičenosti zahtev in umestitev v ustrezni plan izvedbe (razvoja). Ključno pri tem je, da se pred kakršnimkoli planiranjem razvoja izvede analiza upravičenosti uporabniške zahteve (CBA – angl. Cost Benefit Analysis) [44]. Za pripravo ustrezne analize upravičenosti mora uporabniška zahteva biti že zelo dobro razdelana, da se lahko ocena potrebnega razvoja tudi primerno dobro oceni. Pri pripravi analize upravičenosti je za oceno časa razvoja potrebno sodelovanje IT kadra, saj v področju za upravljanje z zahtevami ni kadrov z ustreznimi IT znanji in poznavanjem razvoja informacijskih sistemov. Pri pripravah ocen sodelujejo predvsem načrtovalci, ki pa so v času, ko prihaja do teh analiz, že zasedeni z delom v rednem produkcijskem ciklu in to ocenjevanje zanje pomeni moteč faktor v fazi načrtovanja tekočega razvojnega cikla.

Ideja in uveljavitev vloge področja za enotno upravljanje z zahtevami je po mojem mnenju zelo smiselna, vendar se mora vloga te organizacijske enote še ustrezno umestiti v praksi. Moj predlog je, da bi v tej enoti bili ali vsaj z

njo redno sodelovali ključni vsebinski IT kadri – področni arhitekti oziroma ključni načrtovalci. Ti bi lahko potem samostojno, brez interakcij s sodelavci iz razvoja, pripravili ustrezne ocene razvoja za analize upravičenosti.

Na ta način bi lahko zgradili enoten seznam zahtev za razvoj. Ta bi moral biti urejen po prioritetnem vrstnem redu, katerega osnovni kriteriji so v strateških usmeritvah vodstva banke, zakonskih zahtevah in uporabniških zahtevah, ki sledijo strateškim usmeritvam banke. Nanj bi se uvrstile zahteve po analizi upravičenost na ustrezni vrstni red glede na opisane kriterije.

Iz tega bi postavil še eno tezo, ki dobesedno kliče po agilnih in vitkih pristopih.

Četrta teza: *ali je glede na slabe izkušnje izvedbe prvotnih planov in spremembe uporabniških zahtev, planiranje verzij sploh potrebno? Trdim da ne in je potrebno samo po prioritetnem vrstnem redu in glede na rezultat analize upravičenosti realizirati uporabniške zahteve eno za drugo.*

3.4 Slabosti obstoječega razvojnega procesa

Kot glavna pomanjkljivost v obstoječem načinu dela se je skozi faze projekta razvoja sistema za bančno poslovanje izkazalo:

- planiranje in
- slapovni način razvoja v okviru posameznega produkcijskega cikla (verzije).

Ta način, ko se delo in spremljanje dela izvaja po fazah v razvojnem ciklu (analiza, načrtovanje, kodiranje, testiranje) in ne po posameznih uporabniških zahtevah (vsebinah), onemogoča hitrejši razvoj in posledično možnost, da bi uporabnik funkcionalnost dobil v uporabo kar se da hitro (hitreje kot šele v naslednji verziji, kar pomeni čez pol leta). Drugo kar slapovni način povzroči, so preveliki preskoki med posameznimi fazami v razvojnem ciklu. Ker v vsaki fazi sodelujejo v glavnem specialisti za izvedbo posamezne faze, se predaja informacij in znanj izvaja predvsem v obliki formalno predpisane

dokumentacije, kar posledično lahko povzroči tudi miselne prepade v razumevanju vsebin in v skrajnem primeru celo napačne interpretacije ter posledično vračanje v prejšnje faze razvoja.

Prva težava se pojavi v fazi poslovne analize, ko ugotovimo, da je bila poslovna zahteva pregrobo definirana in posledično podcenjena (v smislu potrebnega časa za implementacijo) in zahteva večji obseg dela, kot je bil planiran. Ko se to ugotovi, se je izjemno težko dogovoriti z naročniki (uporabniki), da se neka poslovna zahteva umakne iz plana trenutne verzije v naslednjo ali da se jo razdeli v več faz in implementira po delih.

Druga težava je, da se zahteve spreminjajo in širijo. Ključni razlog za to, po mojem mnenju je, da fazo poslovne analize v preveliki meri narekuje uporabnik in ne analitik. Zahteve se posledično spreminjajo tudi še takrat, ko smo že globoko v načrtovanju in razvoju. Razumemo, da je to življenjsko, saj v tisti fazi uporabniki že začutijo kakšno rešitev bodo dobili ali celo dobijo prvo verzijo v test. Če se to zgodi, so lahko težave velike, ker veliko že opravljenega dela dobesedno vržemo proč in začnemo znova. To dejstvo kar kliče po izboljšavi in agilnejšem, iterativnem načinu dela z večjo vključenostjo uporabnika in hitrejšega razvoja prvega prototipa, s čimer uporabnik šele dobi pravi občutek kakšna bo rešitev njegove zahteve.

Tretja težava je prenos informacij med fazami razvoja, kar sem napisal že delno v uvodu. Posamezne faze razvoja izvajajo različni posamezniki (v različnih vlogah) in zato prihaja do slabega prenosa informacij. V glavnini se informacije prenašajo preko formalne dokumentacije in premalo neposredno. Posledica tega je enako kot pri drugi težavi, da se prevečkrat vračamo v prejšnje faze razvoja, ker ugotovimo, da vse ni dobro definirano ali da se je razumelo narobe.

Četrta težava oziroma zadeva, ki se je skozi leta razvoja izkazala kot slabost je, da se v fazi načrtovanja pogosto ugotovi, da poslovna analiza ni pregledala vseh vidikov poslovne zahteve. Zgodi se tudi, da bi boljša analiza in malo drugačen predlog izvedbe (sprememba TO-BE procesa) in rešitve zahteve, omogočil boljšo končno rešitev za uporabnika in tudi lažjo izvedbo

(razvoj). In tako spet pridemo do dejstva, da šele v fazi načrtovanja uporabniku predlagamo boljšo rešitev in v primeru soglasja, je potrebno poslovno analizo ponoviti oziroma popraviti, kar posledično še podaljša planirani cikel. V praksi zato stremimo k temu, da v poslovno analizo v čim večji meri vključimo načrtovalce, vendar je to zaradi drugih zadolžitev včasih težko. Tudi tu ugotavljamo, da bi agilnejši način dela, čim več skupnih interakcij uporabnika, poslovnega analitika, načrtovalca in tudi razvijalcev prinesel veliko dodano vrednost, saj bi skupaj predstavili vsak svoj vidik in hitreje prišli do najboljše rešitve.

Na obstoječi način dela tako v fazo načrtovanja pade največji in ključen obseg dela v razvojnem ciklu. S tem tudi pride do preobremenjenosti načrtovalcev in prevelike odvisnosti od njihovega dela. Če bi njihovo delo v določeni meri izvedli že poslovni analitiki in na drugi strani del tudi razvijalci, bi se ta faza razbremenila in bi tako dosegli večjo uravnoteženost faz. Če v prihodnosti, kot enem izmed predlogov sprememb načina dela, govorimo o agilnem razvoju, bo tu potrebno nekaj spremeniti, saj morajo vsi udeleženi v enem ciklu imeti uravnoteženo količino dela, sicer lahko prihaja do zamikov pri pripravi izdelkov (rezultatov) in posledično zamud pri izvajanju posamičnih ciklov.

Peta zaznana **težava** je, da zaradi podaljšanja faze analize in načrtovanja skrajšujemo čas razvoja oziroma fazo razvoja že prehiteva testiranje. Dejstvo je, da celotna verzija obsega veliko vsebin kar se v fazi testiranja (funkcionalnega in uporabniškega) izkaže kot preobsežno in težko obvladljivo. Če tu dodamo še, da se obseg obstoječih funkcionalnosti sistema za bančno poslovanje stalno povečuje in tudi zvišuje kompleksnost sistema, kar vse skupaj povečuje obseg regresijskega testiranja [67], ki ga v uporabniškem testnem ciklu izvajamo ročno, to pomeni, da je testiranje res obsežno in dolgotrajno. Zaradi zakasnitev, ki nastanejo v prejšnjih fazah, se tudi razvoj podaljšuje v fazo testiranja. Ker pa testiranja ne moremo prestavljati zaradi vnaprej definiranih produkcijskih datumov, pride do težav. Ko se testiranje začne, določeni deli novih funkcionalnosti še niso do konca razviti. Pa tudi ko so

razviti, se potem pokaže dosti pomanjkljivosti med posameznimi moduli in funkcionalnostmi.

Če bi testiranje izvajali po delih prej, takoj ko je funkcionalnost razvita in v to vključili še avtomatsko regresijsko testiranje, bi na koncu zmanjšali kompleksnost testiranja, ker bi posamični funkcionalni sklopi že bili testirani.

Šesta težava oziroma del pri katerem bomo predlagali izboljšave, pa so same vsebine oziroma uporabniške zahteve, ki se razvijajo. Ta težava najbrž ne bi smela biti na šestem mestu ampak na prvem, ker bi morala za kakršnokoli izboljšanje načina dela to biti osnova. Vendar je zaradi vpliva in osredotočenosti naloge predvsem na izvedbeni del projekta (ne tudi na planski del projekta in razporejanje vsebin in prioritizacije uporabniških zahtev) umeščena na konec. Gre pa tu za vprašanje ali vse stvari, ki se razvijajo in so med uporabniškimi zahtevami, sploh prinašajo kakršnokoli (ali vsaj zadosti veliko) dodano vrednost in jo je posledično sploh smiselno razvijati. Vprašamo se namreč o smislu ali ne bi bilo mogoče bolje delati kaj drugega, kar res sovpada s strateškimi smernicami banke[40]. Ta težava se je v resnici že nekoliko omilila z uvedbo področja za upravljanje z zahtevami, ampak za dejansko uskladitev vseh interesov različnih uporabnikov potrebuje še nekaj časa za ustrezno umestitev v proces dela in organizacijo.

Poglavje 4

Vitki in agilni razvoj programske opreme

4.1 Vitki principi razvoja

Ideja oziroma koncept vitkega (angl. lean) poslovnega procesa izvira iz Toyotinega proizvodnega procesa[45]. V Toyoti so začeli izdelovati avtomobile (štirideseta in petdeseta leta 20. stoletja), vendar zaradi omejenega trga (Japonska) masovna proizvodnja ni bila smiselna. Ideja je bila, da naredijo avto šele, ko dobijo naročilo. Da pa bi na ta način bili ustrezno odzivni in cenovno konkurenčni, je bilo potrebno izločiti vse nepotrebne dejavnike (ključni koncept izločitve nepotrebnega – angl. eliminate waste).

Koncept proizvodnega procesa so za tem aplicirali tudi na razvojni proces. Temeljna ideja v razvojnem procesu je, da se mora razvoj čim prej zaključiti. In kdaj je razvoj zaključen? Ko prvi izdelek (v Toyotinem primeru gre za avto) zapusti proizvodno linijo. Gledano v tem smislu razvoj predstavlja samo strošek, ki ne prinaša nobene vrednosti dokler izdelek ni narejen. Enako velja tudi za skladiščenje v transportni verigi, kjer je samo skladiščenje strošek, ki ga želimo čimbolj zmanjšati za optimalen potek procesa transportne verige.

Iz teze o strošku razvoja tako lahko izpeljemo pravila, da :

- mora biti razvoj čim hitrejši, s čimer nastanejo manjši stroški,

- z majhnimi razvojnimi stroški že z izdelavo in prodajo majhnega števila izdelkov ustvarimo dodano vrednost (dobiček),

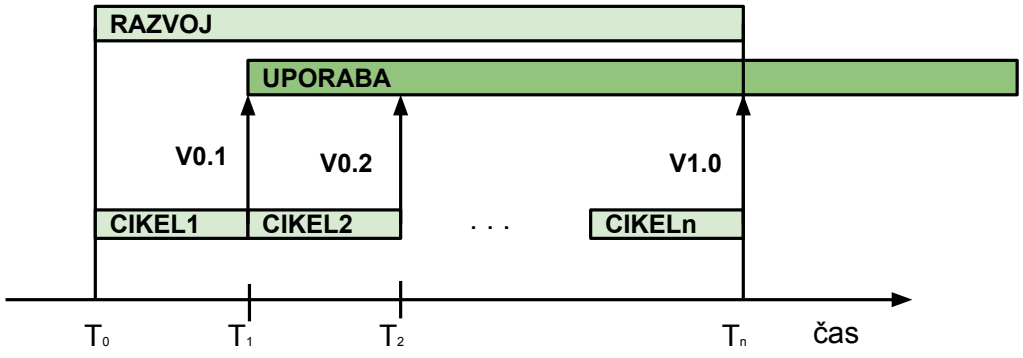
- v primeru daljšega (tudi dražjega) razvoja ustvarimo dodano vrednost, ki pokrije razvojne stroške šele po daljšem obdobju proizvodnje oziroma z izdelavo več izdelkov. Kar pa seveda glede na konkurenco na trgu lahko predstavlja težavo [52].

Prav iz slednje alineje lahko enostavno apliciramo vitke principe tudi na razvoj programske opreme. Razmišljanj o uvajanju vitkih principov iz industrije v razvoj programske opreme sta se prva lotila zakonca Poppendieck z izdajo knjige o vitkem razvoju programske opreme leta 2003 [53]. S tem se je začela oblikovati nova smer v okviru agilnih metodologij, ki naj bi v praksi pomenila še korak višje od agilnih konceptov. Tako so že znane uvedbe vitkih pristopov v velikih podjetjih za razvoj informacijske tehnologije [32]. Prav tako pa se tudi v drugih IT podjetjih v zadnjih letih posveča veliko energije prehodu na vitko razmišljanje pri informacijskih rešitvah kot kažejo študije [46, 63].

Razvoj programske opreme po tradicionalnih metodah, kot smo pokazali tudi na primeru razvoja sistema za bančno poslovanje, poteka preveč togo in zahteva preveč vložka, za katerega se na koncu lahko izkaže, da ne prinaša dodane vrednosti. Od trenutka, ko uporabnik specificira zahtevo, ko se ta analizira, načrtuje, razvije in ponudi uporabniku v testiranje, preteče namreč preveč časa. V času razvoja je uporabnik premalo vključen in zato ni povsem prepričan, če bo delovanje funkcionalnosti enako želenemu. Z vitkimi principi pa se tem težavam izognemo, s tem da želimo:

- razvoj čimbolj skrajšati oziroma ga razdeliti na več kratkih razvojnih ciklov,

- po vsakem kratkem razvojnem ciklu programsko opremo že ponuditi v uporabo in tako začeti čim prej ustvarjati dodano vrednost,



Slika 4.1: Shematski prikaz principa vitkega razvoja

- v kratkih razvojnih ciklih po principu inkrementalnega razvoja [24] čimprej priti do ciljno želene programske opreme.

Razvoj na ta način ves čas vključuje tudi uporabnika. Začne se tako, da najprej razvijemo grobi okvir rešitve s čimer potrdimo osnovno zamisel. Nato se doreče prvi nivo podrobnosti in se ga razvije ter ponudi v uporabo kot prikazujemo na sliki 4.1. To ponavljamo do podrobnosti, ki so še smiselne, kar pomeni, da je strošek razvoja tega nivoja podrobnosti dovolj nizek, da še vedno ustvarjamo dodano vrednost.

Da pa lahko izvajamo kratke razvojne cikle (vanje vključimo uporabnika (naročnika) in že v času razvoja začnemo ustvarjati dodano vrednost), je nujna sprememba iz tradicionalnega pristopa v vitki princip. Za lažje usmerjanje vitkega razvoja pa nam pomagajo temeljni principi vitkega razvoja, ki skupaj določajo:

- kaj v določenem ciklu narediti,
- kdaj narediti naslednji cikel,
- kdaj je izveden zadnji cikel, ko ni več smiselno razvijati nadaljnjih podrobnosti,
- kako naj deluje razvojna ekipa in

- kako naj se v razvoj vključijo uporabniki.

Temeljni principi vitkega razvoja [53], od katerih bomo vsakega v nadaljevanju še podrobno razdelali, so :

1. izločanje nepotrebnega in nepomembnega (angl. eliminate waste),
2. spodbujanje učenja in iterativno delo (angl. amplify learning),
3. sprejemanje odločitev čim kasneje (angl. decide as late as possible),
4. prikaz rezultatov razvoja kar se da hitro (angl. deliver as fast as possible),
5. zaupanje v razvojno ekipo (angl. empower the team),
6. gradnja integritete skozi razvoj (angl. build integrity in),
7. celovit pregled (angl. see the whole) in delovanje v skupno dobro.

Prvi princip – izločanje nepotrebnega – je predvsem pomemben na začetku razvoja, ko določimo prioritetni vrsti red uporabniških zahtev. Vsako zahtevo potem razdelimo na manjše dele, izmed katerih potem izločimo tiste nepomembne, katerih se ne lotimo, ker ne prinašajo dodano vrednosti.

Naslednji trije – razširjanje znanja, sprejemanje odločitev čim kasneje in prikaz rezultatov kar se da hitro – predstavljajo temelj modernega načina razvoja programske opreme. Govorijo o načinih stalnega učenja, prototipnega in iterativnega razvoja ter čim večji vključenosti uporabnika v vse faze razvojnega procesa.

Sledijo pa še zadnji trije principi – zaupanje v ekipo, gradnja integritete skozi razvoj in celovit pregled, ki predvsem poudarjajo pomen ekipe in timskega dela vseh vlog potrebnih za razvoj. Tu govorimo tako o najožji razvojni ekipi, vodjih in koordinatorjih kot tudi uporabnikih. Posebej pomembno za dobro delo ekipe je vzpostaviti zaupanje med vsemi člani in tako ustvariti pravi pretok informacij, s čimer si lahko vsi ustvarijo celovito sliko razvojne naloge in kar največ pripomorejo k skupnim ciljem.

Ko vse principe povežemo skupaj lahko rečemo, da je temelj vitkega razvoja ta, da resnično smatra razvoj kot ustvarjalen proces. To pomeni, da si je potrebno vzeti čas za razmislek, za pregled vseh možnih vidikov v zgodnjih fazah razvoja. V vseh pogledih si ne smemo zapirati vrat s prezgodnjimi odločitvami v eno smer, saj na ta način na koncu ne pridemo do najboljše rešitve. Spodbujati je potrebno inovativnost, nekonvencionalno razmišljanje in čimbolj odprt pretok vseh informacij, saj le to na koncu vodi do najboljših rezultatov.

4.2 Izločanje nepotrebnega in nepomembnega

Prvi korak za doseganje uspehov z vitkimi metodologijami je, da se naučimo zaznavati nepomembno in nepotrebno ter to izločiti iz nadaljnje obravnave (angl. eliminate waste). Kako pa to zaznamo? Lahko si postavimo definicijo, da je nepomembno vse, kar za naše uporabnike (stranke) ne prinaša dodane vrednosti.

Pri razvoju programske opreme so najpogostejši viri nepotrebnega in nepomembnega naslednji:

- nedokončana programska oprema,
- dodatni podporni procesi razvoja,
- dodatne funkcionalnosti,
- preklapljanje med različnimi področji dela,
- čakanje,
- prenos/predaja informacij,
- napake v delovanju.

Pri razvoju programske opreme, predvsem novih produktov, se pogosto zgodi, da nikoli ne preidejo v produkcijsko delovanje. Bistveno pri razvoju

namreč je, da je končni produkt na voljo uporabnikom, na podlagi česar lahko ugotavljamo dodano vrednost. Sam konec razvoja v življenjski dobi produkta še ne pomeni nič, saj mora biti produkt še testiran in integriran v okolje, s čimer so šele omogočeni predpogoji za uporabo.

Podporni procesi razvoja so dostikrat sami sebi v namen. Eden izmed takih je dokumentiranje, pri katerem so zelo pogoste postopkovne zahteve, ki pa ne prinašajo dodane vrednosti. Dokumentacija prinaša dodano vrednost, če jo je nujno napisati in predati, da naslednji v procesu lahko sploh nadaljuje z delom. V nasprotnem primeru je dokumentacija nepotrebna. Pri programiranju je v smislu vitkih principov zelo pomembno, da naredimo natančno tisto, kar bo omogočilo informacijsko podporo uporabniški zahtevi. Karkoli naredimo več kot zahtevajo uporabniške zahteve, se smatra kot nepotrebno.

Zelo pogost pojav pri razvoju je, da posamezniki sodelujejo v več razvojnih ekipah, ki se ukvarjajo z zelo različnimi uporabniškimi zahtevami. Posledica tega je veliko "preklapljanja" med vsebinami kar pomeni izgubo časa. Za vitke principe je bistveno, da so posamezniki člani le ene skupine za razvoj izbranega produkta. Ko se ta razvoj konča, pa postanejo lahko del druge skupine in razvoja drugega produkta.

Čakanje je temelj nepotrebnega in izgube. Vsako čakanje v razvojnem ciklu pomeni, da razvojni cikel ni optimalen. Optimalen razvojni cikel je namreč takšen, v katerem so vse informacije za nadaljnji korak na voljo natanko takrat, ko je eden izmed članov skupine prišel v procesu do tega koraka. Vsemu čakanju v razvojnem procesu se je skoraj nemogoče izogniti, a ga je potrebno minimizirati. Pretok informacij je ponavadi zelo povezan s čakanjem. Če posamezen član skupine potrebuje za nadaljnje delo informacije s strani uporabnika, je zelo priporočljivo, da jih lahko dobi takoj. Najlažji način za zagotovitev tega je, da je tudi uporabnik član skupine in je lociran v pisarni skupaj z vsemi ostalimi člani. To tudi omogoča veliko neformalne komunikacije in stalne izmenjave mnenj, kar tudi preprečuje izgubo določenih informacij pri predaji formalne dokumentacije med koraki v procesu.

Napakam pri delovanju se je nemogoče izogniti v celoti. V smislu principa nepotrebnega ni problem v napaki sami, ampak je težava v pretečenem času od takrat, ko se napaka pojavi, pa do trenutka, ko jo odkrijemo (identificiramo). Če je ta čas predolg, je lahko vir velike škode in posledično manjše dodane vrednosti. V razvojnem procesu lahko velik del nepotrebnega predstavljajo tudi vodstvene in koordinacijske aktivnosti. Če je proces zasnovan tako, da se v določenih točkah zahteva avtorizacija posameznega vodje, je to že potencialen razlog zamude in posledično nepomembnega. Vzrok za to so pogosto tudi razna orodja za vodenje projektov in zahtev ter razni predpisi o avtorizacijah vsebin v posameznih procesnih korakih.

Zato, da se nepomembnemu delu v kar največji meri izognemo, si lahko pomagamo z raznimi tehnikami. Zelo priporočljiva je tehnika ugotavljanje toka vrednosti (angl. Value Stream Mapping) [27]. Tehnika temelji na predstavitvi celotnega razvojnega procesa, od uporabniške zahteve do izvedbe, in tudi faze uporabe informacijske podpore za izbrano uporabniško zahtevo. Vsak korak v procesu je potrebno oceniti s časom potrebnim za izvedbo, pri čemer si pomagamo s preteklimi izkušnjami (statistikami) razvoja. Na ta način ocenimo koliko časa je potrebnega, da informacijska podpora začne ustvarjati vrednost. Na ta način pridemo tudi do tehnik razvoja s kratkimi cikli, ki omogočajo, da del zahteve že hitro realiziramo in ponudimo v uporabo, s čimer že začnemo ustvarjati dodano vrednost za uporabnika. Tako se tudi izognemo tveganjem in izločimo nepotreben razvoj detajlov zahteve že v prvem razvojnem ciklu.

4.3 Spodbujanje učenja

Vsak razvoj, tudi razvoj programske opreme, je ustvarjalni proces in ga je potrebno na ta način strogo ločevati od industrijskega proizvodnega procesa. Ravno zaradi tega je potrebno biti pozoren pri aplikaciji vitkih principov v razvoj. Ključ do rezultatov razvoja je namreč poskušanje in učenje na podlagi narejenih prototipov (poskusov).

To načelo, spodbujanje učenja (angl. amplify learning), je za proces razvoja ključno. Pravi, da se ne smemo takoj lotiti rešitve celotne zahteve ali problema, temveč da ga moramo najprej razbiti na manjše dele. Če želimo namreč doseči najboljše rezultate razvoja in se iz tega tudi veliko naučiti za nadaljnji razvoj, moramo zahtevo (problem) najprej razdeliti na manjše ter obvladljive dele. Vsak del problema skušamo potem rešiti na različne načine. Za rešitev celotnega problema moramo rešitve posameznih delov povezati skupaj in izbrati tiste rešitve delov, ki skupaj najbolj optimalno rešujejo celoten problem oziroma zahtevo.

Nazorno se ta princip prikaže iz dveh različnih principov razvoja programske opreme:

- prvi pravi, da moraš v prvem poskusu narediti pravi izdelek, rešitev v vseh fazah – in načrtovanju in razvoju – v nasprotnem primeru, je to neuspeh,

- drugi pa pravi, da se veliko več naučiš in pridobiš več izkušenj za naprej ter za nadaljni razvoj z manjšimi poskusi, testiranjem in popravljanjem/prilagajanjem poskusov.

Poudarimo, da je prvi način zelo dober pri dobro definiranih zahtevah oziroma problemih. Drugi pa je veliko bolj uspešen v problemskih domenah, kjer še iščemo najbolj primerno rešitev in sploh ugotavljamo, kakšna bi optimalna oziroma najbolj primerna rešitev bila. Drugi princip je temelj vitkega razvoja in je primeren pri povsem novih uporabniških zahtevah oziroma snovanju nove informacijske podpore.

Če zahtevo razdelimo na več delov, to pomeni da naredimo več iteracij – ciklov razvoja. V principu se v vsaki iteraciji lotimo enega dela. Na koncu cikla je ključno, da preverimo ali smo del zahteve rešili na najboljši način in lahko začnemo za naslednjo iteracijo. Povratne zanke (vračanje v prejšnjo iteracijo) so seveda dovoljene in so ključ, da na koncu razvoja pridemo do najboljših rešitev.

Težava kratkih iterativnih ciklov je lahko, da ne vemo kdaj je rešitev

dovolj dobra. Zato pa je tu uporabnik, ki na koncu vsakega cikla preveri ustreznost razvoja in skupaj z razvojno ekipo definira vsebino naslednjega cikla. Če se na koncu cikla ugotovi neskladje z zahtevami na začetku cikla, se v novem ciklu vsebina ponovi in odpravi neskladnosti. Ker število ciklov ponavadi ni definirano, mora razvoj zagotoviti sprotno testiranje rešitve, da se potem lahko čim hitreje delovanje prenese v produkcijsko delovanje.

Za hitrejše doseganje ciklov in čim manj vračanja v ciklih je pomembno, kako se lotimo iskanja rešitev. Obstaja točkovni pristop (angl. point based approach) [59], ki določa, da se najprej natančno definira zahteva in se jo potem realizira ter izpopolnjuje tako, da je na koncu rešitev točno taka kot je bila zahtevana. Ta pristop zahteva več ponovitev, da pridemo do končnega cilja, kot intervalni pristop (angl. set based approach). Intervalni pristop pa temelji na podanih omejitvah zahteve. Tako močno spodbuja inovativnost končne rešitve. S postavitvijo omejitev se določi okvirje za razvoj, s čimer se doseže, da do končne rešitve pridemo zelo hitro (v malo ponovitvah). Je pa tu druga težava, ki se lahko zgodi v primeru, ko je rešitev preveč inovativna in se zanjo s strani uporabnika spet zahteva ponovitev cikla. Vse je seveda odvisno od okolja v katerem nastajajo rešitve.

4.4 Sprejemanje odločitev čim kasneje

Že pri principu spodbujanja učenja smo rekli, da je potrebno zahteve razdeliti na manjše dele in najti rešitev za vsak del zahteve. Ta rešitev ne sme biti ena sama, ampak je potrebno del zahteve pogledati z vseh zornih kotov in pripraviti čim več predlogov rešitev – prototipov. Ko rešitve delov zahteve namreč sestavljamo skupaj za rešitev celotne zahteve, se nekatere rešitve delov zahteve pokažejo skupaj boljše kot druge.

Pristop imenujemo tudi vzporedni razvoj (angl. concurrent development) [59, 53], ki pravi, da najprej pregledamo vse možnosti rešitve zahteve (angl. breadth first). Šele ko je na tem nivoju izbrana in potrjena ustrezna konceptualna rešitev, se začnemo ukvarjati z nadaljnjim nivojem podrobnosti

rešitve. Na ta način si puščamo odprta vrata za izbiro najboljših možnih rešitev zahtev in si s prezgodnjimi odločitvami v kakšni izmed podrobnosti ne otežimo nadaljnjega razvoja. Če se namreč prezgodaj odločimo za določeno smer, se nam kasneje v primeru težave lahko zgodi, da moramo spreminjati temeljne koncepte, da bi končna rešitve bila sprejemljiva s strani uporabnika oziroma naročnika.

Temelj načela, da se odločitve sprejemajo kar se da pozno (angl. decide as late as possible) je, da si moramo določeno potencialno možnost rešitve zapreti čim kasneje, ko zares ugotovimo, da ne more pripeljati do boljših rešitev kot katera izmed drugih alternativ. Pri tem odločanju je potrebno poudariti dve zadevi, ki se tičeta stroškov:

- strošek poznega odločanja in dolgotrajnega vzdrževanja več alternativ je visok in s časom, v katerem ne sprejmemo odločitev in večamo število alternativ, narašča,

- strošek spremembe odločitve v poznih fazah razvoja prav tako narašča s časom preteklim od izbire alternative.

Drugi strošek je zelo škodljiv in zato je zelo pomembno, da se za izbiro alternativ odločamo v pravem času in po ustreznem tehtanju možnosti. Prvi strošek pa dejansko ne predstavlja le tega, saj se z iskanjem (učenjem) alternativnih rešitev tudi pridobiva znanje in izkušnje za nadaljne razvojne vsebine oziroma nove uporabniške zahteve.

Za princip čim kasnejšega sprejemanja odločitev je bistveno pretehtati oba prej zapisana stroška in ju z ustreznim odločanjem držati na sprejemljivem nivoju. Kdaj pa je zadnji, še sprejemljiv, trenutek za sprejem odločitev? Za določitev tega trenutka v razvojnem ciklu je potrebno upoštevati naslednja priporočila:

- vse delno zaključene arhitekturne in načrtovalske informacije je potrebno deliti z vsemi člani razvojne ekipe,

- čim več neposredne komunikacije in sodelovanja,

- zgraditi je potrebno kulturo sprejemanja sprememb uporabniških zahtev,

- zgraditi je potrebno občutek, kaj je ključnega v določeni zahtevi,

- zgraditi je potrebno občutek, kdaj se morajo sprejeti odločitve,

- na vse se je potrebno hitro odzvati.

Odločanje v pravem trenutku je določeno predvsem s stroški vzdrževanja alternativ, kot smo že zapisali in intuicijo posameznika, ki odločitev sprejema. Ta mora za sprejem odločitev v pravem trenutku imeti širok nabor znanj ter odlične sposobnosti sklepanja in simuliranja.

4.5 Prikaz rezultatov razvoja kar se da hitro

Princip prikaza rezultatov razvoja kar se da hitro (angl. deliver as fast as possible) gre z roko v roki s predhodnim, da odločitve sprejemamo čim kasneje. Povezana sta s tem, da oba govorita o razgradnji zahteve na manjše dele. Pri tem prvi čim dlje pušča odprte vse možnosti, drugi pa skrbi da postopoma dodajamo nivoje podrobnosti v gradnjo rešitve uporabniške zahteve. Bistvo principa hitrega prikaza rezultatov razvoja je v tem, da čim prej prikažemo prvi prototip delujoče rešitve. Na ta način že takoj vključimo uporabnika, ki z delujočo rešitvijo dobi občutek o končni izvedbi. Prav tako lahko že takoj poda svoje mnenje in usmerja nadaljnji razvoj v pravo smer, da je na koncu rešitev res takšna kot si jo je uporabnik zamislil.

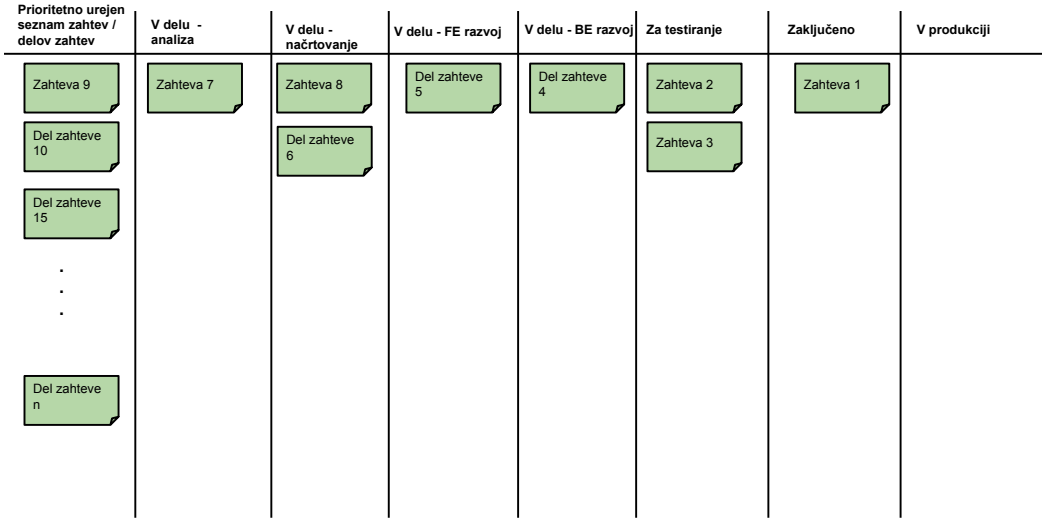
Drugi pomemben vidik tega principa je, da s hitrim prikazom delujoče rešitve razbremenimo razvojno ekipo skrbi o ustreznosti rešitve. Pri tradicionalnih razvojnih metodah je skrb o ustreznosti rešitve prisotna ves čas razvoja do začetka testiranja ali celo povsem do prehoda v produkcijo. V vsem tem času se napetost v razvojni ekipi samo stopnjuje. Da to napetost čim prej sprostimo, se v vitkih in agilnih metodah uporabnika vključi čim prej in se mu prikaže delujočo rešitev. Po potrebi se naredi korekcije, s čimer

se poenotimo z uporabnikovim pogledom in potem razviti prototip (v okviru kratkega razvojnega cikla) potrdimo, kar pomeni da je rešitev ustrezna.

Principu v praksi rečemo tudi princip hitrih dostav (angl. rapid delivery). V praksi to pomeni, da lahko zahtevo razvijemo tako hitro, da se naročnik ne more premisliti oziroma spremeniti osnovne zahteve. Za vzpostavitev takega načina dela je potrebno zagotoviti, da vsakdo natančno ve, kaj mora narediti danes in kaj ga čaka jutri. Se pravi, da mora vedno biti dovolj dela na zalogi in da se tudi vzpostavi kultura v ekipi, kjer vsakdo ve kaj mora delati, ne da čaka na direktna navodila vodje. Temu načinu dela pravimo tudi vlečni sistem (angl. pull system). Na ta način se optimizira oziroma maksimira izkoristek razvojne ekipe oziroma celotnega razvojnega procesa.

Cilj takšnega načina dela je tudi, da ekipa postane samo-organizirana in samo-vodena. Vodje takih ekip se tako ukvarjajo z mentorstvom in usmerjanjem (angl. coaching) ekipe in ne z razporejanjem in nadzorom dela.

Ko je ekipa tako uigrana kot smo napisali (da je samo-organizirana in deluje po vlečnem principu) smo samo še korak do razvoja/proizvodnje v realnem času (angl. just in time manufacturing). Pri tem si lahko pomagamo tudi s kanban mehanizmom[28], ki temelji na osnovi kanban kartic [42]. Na začetku vsakega razvojnega cikla se definirajo deli uporabniške zahteve, ki se vsak zapiše na kanban kartico z približno oceno časa razvoja. Vse kanban kartice se postavijo na tablo v inicialno stanje in posamezne vloge v skupini jih jemljejo po vrsti. Ko svoj del naredijo, kartico prestavijo v drug status, kjer jo prevzame član skupine v drugi vlogi. Na ta način v vsakem trenutku lahko vidimo kako poteka razvoj in v kakšni fazi je vsak izmed delov zahteve. Za hitro koordinacijo ekipe je smiselno imeti dnevna kratka srečanja, ki ne smejo trajati več kot pol ure. Na njih se samo pogleda status posameznih kartic kot je prikazano na 4.2 in pogovori o eventualnih težavah [53, str.75].



Slika 4.2: Prikaz kanban table z lepljivimi listki v posameznih statusih

4.6 Zaupanje v razvojno ekipo

Za uspešnost vsakega razvojnega procesa je ključna ekipa (skupina) in ekipni duh. V vitkih principih je še posebej potrebno vzpostaviti zaupanje v razvojne ekipe (angl. empower the team) in jim tudi pustiti svobodo. Najvišja zrelostna stopnja vsake skupine je namreč samo-organiziranost, do katere pa lahko pride le v svobodnem in dereguliranem okolju.

Ekipi je potrebno dajati občutek vrednosti, vključenosti, občutek odločanja in pomembnosti, kar so osnove pozitivne klime za razvoj ekipe. V praksi obstaja veliko postopkovnih standardov kot so ISO9000 ali CMM (angl. Capability Maturity Model), ki s svojo implementacijo v praksi prepogosto vzbudijo dvome pri razvojnih ekipah. Ti standardi so po mnenju razvojnih ekip prepogosto preveč birokratski, togi, administrativni in posledično omejujejo kreativnost ekip [53, str.96]. Težava takšnih standardov je v tem, da razvijalci nimajo možnosti odločanja in se zahteva princip, da v prvem poskusu pripravimo pravo rešitev (angl. do it right the first time). To pa je ravno v nasprotju z vitkimi principi, zato temeljni predpostavki pravita [53,

str.98,99]:

1. Zrela organizacija gleda na celoten sistem in skrbi za optimalno delovanje celotnega sistema. Ne osredotoča se na optimiziranje sestavnih delov sistema.

2. Zrela organizacija se osredotoča na učinkovito učenje in predajo znanja ter spodbuja ljudi, ki izvajajo temeljne procese ustvarjanja vrednosti k sprejemanju odločitev.

Bistvo ekip v vitki metodologiji je, da vodstvo ne uvaja nekih programov vodenja, koordiniranja, nadzora in dela od drugje oziroma iz teorije. V vsakem podjetju se morajo delovni postopki, sistem odločanja in sprejemanja odločitev zgraditi iz podjetja samega. To pomeni, da je treba poslušati izvajalce procesov ustvarjanja vrednosti in dovoliti, da opravljajo svoje delo tako kot mislijo, da je najbolj prav.

Za uspešnost ekipe, mentorstvo in usmerjanje je poleg svobode temeljna še ustrezna motivacija. Osnova za motivacijo pri delu je, da vsak pozna namen svojega dela. Pomembno je namreč začutiti dodano vrednost, ki jo s svojim delovanjem ustvarjamo za uporabnika. Vsakdo, ki čuti in ve, da dela nekaj za dobrobit nekoga, s tem občutkom dobi dodatno vnemo za delo (motivacijo) in se v svojem delu še bolj angažira [53, st.105]. Najtežje se je namreč motivirati, če je vsaka zadolžitev delegirana s strani vodje in niti ni podrobno predstavljen namen dela. V gradnji stopnje motivacije so zato ključna:

- pripadnost,

- varnost,

- kompetence,

- napredek.

O načinu motivacije zgovorno govori primer iz literature [53, str.104], kjer so postavili pravila za motivacijo:

1. zaposliti je potrebno kvalitetne kadre in jim dati svobodo,
2. če ljudi, predvsem to velja za razvoj, omejiš, potem kreativni duh počasi izgine,
3. ljudi je potrebno spodbujati, da lahko razvijejo svoje ideje,
4. vsak naj čimprej preizkusi svoje ideje in načine motiviranja.

Pri tem so predlagali tudi, da se vsaj 15% časa vsakega razvijalca (tehnično znanstvenega kadra) nameni času za raziskovanje novih idej in izvajanju projektov po lastnih željah. Pri ekipi v vitkih principih poudarjamo predvsem to, da ekipa ne sme biti nadzorovana ampak usmerjana. To vpliva tudi na sami princip vodenja skupin, ki se v vitkih principih mora razlikovati od tradicionalnih pristopov. Zelo pomembno je, da se res skuša ekipe samoorganizirati. Na ta način se vodilni člani ekipe (pa naj bodo to razvijalci, načrtovalci, analitiki) sami izluščijo kot primerni za vodje izmed vseh članov. To ponavadi postanejo posamezniki z močnim vsebinskim in tehničnim znanjem ter širokim pogledom, ki s svojim načinom dela povežejo vse člane skupine med seboj in tako omogočajo klimo za doseganje najboljših rezultatov ekipe.

4.7 Gradnja integritete skozi razvoj

Integriteta pri razvoju programske opreme pomeni, da razvojna ekipa ve in razume uporabnikove zahteve in bo zato tudi rezultat razvojne ekipe ustrezno stabilna, uglašena ter za uporabnika najbolj optimalna programska oprema. O gradnji ustrezne integritete pri razvoju pa govorijo načelo tega principa - gradnja integritete skozi razvoj (angl. build integrity in).

Ločimo dve vrsti integritete:

- zaznavna (angl. perceived) integriteta predstavlja celovitost produkta, ki ustrezno uravnoteži funkcionalnost, uporabnost, zanesljivost in ekonomičnost za potrebe stranke,

- konceptualna (angl. conceptual) integriteta pa pomeni, da vsi sestavni deli (koncepti) produkta oziroma sistema delujejo brezhibno in uravnoteženo kot celota.

Kot odličen primer zaznavne integritete lahko vzamemo "Google" z vsemi svojimi storitvami, ki so res intuitivne, uporabne, zanesljive in redko se zgodi, da pri njihovih storitvah kaj bistvenega manjka. Primer slabe konceptualne integritete je uporaba dveh programskih rešitev za enako storitev (npr. dva različna sistema za elektronsko bančništvo znotraj iste banke – eden za pravne osebe, drugi za fizične osebe).

Konceptualna integriteta je predpogoj za zaznavno integriteto, saj brez ustreznega poenotenega koncepta sistema, ne moremo pričakovati zadovoljstva strank z uporabo naših rešitev.

Ključno za doseganje obeh vrst integritete je zagotavljanje pravega toka informacij med vsemi deležniki (uporabniki, analitiki, načrtovalci, razvijalci) v razvojnem procesu [53, str.128]. To pomeni, da morajo vsi udeleženci v razvojnem procesu razpolagati z vsemi informacijami. Informacije se ne smejo zakrivati oziroma prilagajati za različne vloge v procesu. Vsak razvijalec mora imeti pred očmi informacije o željah stranke, sicer lahko zaide v prevelike tehnične podrobnosti in optimizacije, ki končnemu uporabniku ne prinesejo dodane vrednosti. Za takšen tok informacij do razvijalcev ponavadi skrbijo glavni inženirji (samo-organizirani vodje skupin), ki imajo poleg tehničnega znanja še veliko vodstvenih sposobnosti, da želje strank prenesejo v tehnični koncept in jih ustrezno predstavijo razvijalcem.

Problem ustreznega pretoka informacij v zaporednem razvoju programske opreme (npr. slapovni model) je v tem, da se po zaključku vsake faze v pripravljeni dokumentaciji in pri predaji teh dokumentov, izgubi veliko pomembnih informacij iz zahtev uporabnika. Zato je ključno za izboljšanje pretoka informacij, da se z agilnejšim razvojem, kratkimi cikli ter vključitvi uporabnika po preverjanju rezultatov vsakega cikla, vzpostavi neko dokaj stalno komunikacijo z uporabnikom in na ta način pogost odziv (angl. feedback), da gre razvoj v pravo smer.

Za zagotavljanje ustreznega nivoja integritete razvojnega procesa si lahko pomagamo z več orodji, kot so:

- načrtovanje na podlagi modela (angl. model driven design) [11],
- izboljševanje in prilagajanje (angl. refactoring) [22],
- avtomatizirano testiranje [16].

Načrtovanje na podlagi modela pomeni, da se razvoj programske opreme izvaja direktno iz domenskega modela. Domenski model je model, ki ga vzpostavijo skupaj vsi člani ekipe (uporabnik (naročnik), analitik, načrtovalec in razvijalci) z namenom lažjega razumevanja zahteve oziroma poslovnega področja in ustrezno enostavnejšega pretoka informacij. S pomočjo takšnega modela si pod izbranim pojmom vedno vsi predstavljajo enako stvar.

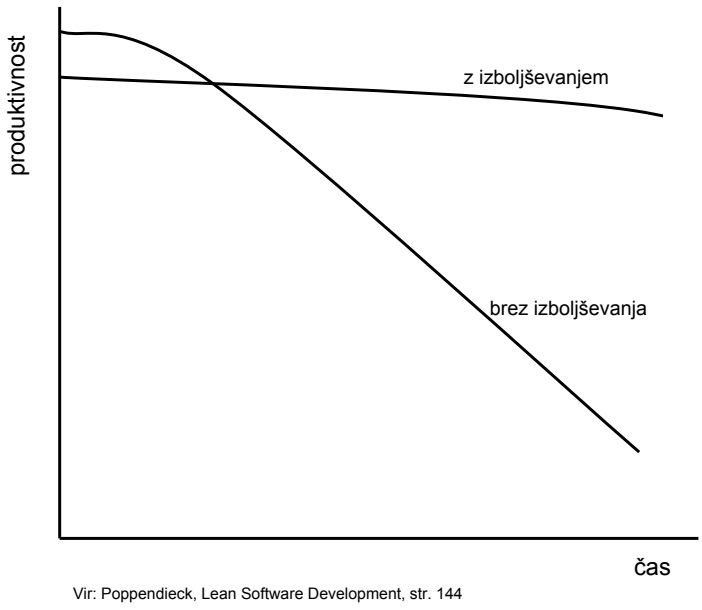
Koncept stalnega izboljševanja in prilagajanja pomeni stalno izboljševanje načrtovanja/zasnove v času razvojnega cikla.

Dobra načrtovalska praksa je, da se načrt lahko ves čas razvoja še spreminja in prilagaja boljšim rešitvam razvijalcev. V nasprotnem primeru so posledice takšne kot so vidne na sliki 4.3. Če se ugotovi večja sprememba v načrtu v času kodiranja, je smiselno ustaviti se in izboljšati načrt preden se nadaljuje s kodiranjem.

Avtomatsko testiranje [30] je ključen koncept, če želimo ohranjati konceptualno integriteto kot tudi vzbujati upanje (z veliko verjetnostjo) na visoko zaznavno integriteto naših strank – končnih uporabnikov. Avtomatsko testiranje gre z roko v roki s prilagajanjem načrta končne rešitve, saj nam zagotavlja odgovor na vprašanje o ustreznosti načrta (ali je naš načrt dober, ali bo performančno ustrezen,...) in celotnega izvedenega razvoja.

Če na koncu še enkrat povzamemo, so ključne karakteristike sistema za zagotavljanje konceptualne integritete naslednje:

- enostavnost – načrt in programska koda morata biti pregledna in enostavna,



Slika 4.3: Izboljševanje ter rast/padec produktivnosti v času

- nazornost, čistost (angl. clarity) – programska koda mora biti lahko razumljiva in pregledna,

- primernost za uporabo - načrt mora ustrezati osnovnemu namenu in uporabniški zahtevi,

- brez ponavljanj - naj se določeni deli programske kode ne ponavljajo, vse naj bo razvito ustrezno modularno. Na ta način lahko s spremembo na pravem mestu uredimo vse in se ni potrebno prebijati po programski kodi in izvajati spremembe na veliko delih,

- brez nepotrebnih funkcionalnosti - sistem naj podpira samo kar je zahtevano. Funkcionalnosti, ki jih nihče ne uporablja, ne smejo biti del sistema in morajo biti izločene že na začetku kot nepotrebne (angl. waste).

4.8 Celovit pregled

Zadnji izmed temeljev vitke metodologije je celovit pregled (angl. see the whole). Bistveno za uspešnost podjetja ali ekipe je, da ima vse potrebne informacije za svoje delo, s čimer si zagotovi celovit pregled nad stanjem v katerem deluje. Princip sistemskega razmišljanja (angl. systems thinking) pravi, da sistem deluje najbolje, ko so vsi sestavni deli sistema med seboj najbolj utečeni. Ni nujno, da je vsak sestavni del posamezno najboljši. Bistveno je, da v interakciji v sklopu celotnega sistema delujejo med seboj najbolj uglašeno. In to je bistvo celovitega pregleda – potrebno je vse sile usmeriti na optimalno delovanje celotnega sistema, ne na njegove sestavne dele.

V vitkem principu se za ustvarjanje celovitega pregleda poslužujemo tudi metode "petih zakaj" (angl. five whys). Metoda temelji na tem, da se za vsako zahtevo petkrat zapored vprašamo "Zakaj?" in tako pridemo do izvora težave oziroma do bistva zahteve. Ko podrobnosti poznamo, pogledamo vse skupaj in si naredimo načrt kako podrobnosti celotne zahteve skupaj najbolj optimalno povezati, da bo izdelek (informacijska podpora) najboljši za končnega uporabnika (naročnika).

Ker je bistvo iskanje najboljših rešitev za celotno zahtevo, se je nesmiselno posluževati metod lokalne optimizacije. Ta namreč rešuje zahteve po delih in se osredotoča na optimizacijo vsakega dela posebej. Na ta način se nam namreč lahko zgodi, da posamezne dele zahteve optimiziramo, izboljšamo, a celotne zahteve ne rešimo najbolje ali pa podporo zahtevi celo poslabšamo in podaljšamo delovni proces. Zato je pri merjenju in postavljanju različnih metrik za uspešnost (angl. KPI – key performance indicator) potrebna velika previdnost. Pravilo naj bo, da ne merimo raje ničesar, če ne moremo meriti oziroma spremljati vsega [53, str.159]. Da pa bi zgradili metrike za merjenje vsega, se poslužujemo naslednjih metod:

- standardizacija,

- specifikacija,

- dekompozicija.

Pravilen pristop pri reševanju napak razvoja tako ni, da vsako potencialno napako pripišemo točno določeni osebi, ampak da napake spremljamo informativno in jih rešujemo znotraj organizacije na način, da poiščemo pravi izvor napake. Če bi napake pripisovali individualno, bi bil to tipičen primer prelaganja bremena na pleče posameznika in lahko si predstavljamo, kaj bi to pomenilo za njegovo nadaljne delo in motivacijo.

Pri implementaciji agilnih in vitkih metodologij v proces razvoja se velikokrat zgodi, da se pojavi težava v pogodbah o sodelovanju med partnerskimi podjetji. Ponavadi so v pogodbah definirani roki, zato morajo biti pogodbe z roki prilagojene agilnemu načinu dela. Predvsem pa morajo biti pogodbe zasnovane tako, da partnerji med seboj sodelujejo in si izmenjujejo informacije, ki so potrebne za dosego skupnega cilja. Nujno je, da si informacij med seboj ne prikrivajo. Delovanje mora res biti partnersko in dolgoročno. Partnersko predvsem z vidika, da močnejši partner ne izsiljuje šibkejšega. Dolgoročno pa v smislu, da ne pritiskamo preveč na uspeh v kratkem roku, ker se pravo partnersko sodelovanje izkaže kot izjemno šele na dolgi rok.

Z vidika vitkih metodologij v pogodbah ne definiramo rokov in podrobnega obsega rešitve, ki jo želimo razviti. Pogodba mora biti napisana v partnerskem (enakovrednem) duhu in mora določati le okvirni namen sodelovanja, cilj sodelovanja in kriterije za merjenje uspešnosti izvajanja razvoja.

4.9 Agilni razvoj

Agilni razvoj programske opreme je sestavljen iz več metod razvoja programske opreme, ki temeljijo na iterativnem in inkrementalnem razvoju [38, 19]. Bistvo agilnega razvoja je, da se izvaja v samo-organiziranih heterogenih skupinah, v katerih nastopajo vse potrebne vloge za izvedbo določene uporabniške zahteve (uporabnik, analitik, načrtovalec, razvijalci). Drugi temelj je evolucijski razvoj in časovno omejeni iterativni pristop, ki zagotavlja hitre in redne dobave izdelka (delujoče programske rešitve) naročniku oziroma

uporabniku.

Principi agilnega razvoja so znani dobrih 25 let [66], bolj formalizirani in združeni pod okriljem agilnih metod pa so postali šele leta 2001 z manifestom agilnega razvoja [23]. Ta je zelo kratek in jedrnat. V njem je 17 sopodpisnikov manifesta zapisalo naslednje:

"Odkrivamo boljše načine razvoja programske opreme tako, da jo razvijamo, in pri tem pomagamo tudi drugim. Naše vrednote so ob tem postale:

- *posamezniki in interakcije pred procesi in orodji,*
- *delujoča programska oprema pred vseobsežno dokumentacijo,*
- *sodelovanje s stranko pred pogodbenimi pogajanji,*
- *odziv na spremembe pred togim sledenjem načrtom.*

Z drugimi besedami, četudi cenimo dejavnike na desni, vseeno bolj cenimo tiste na levi."

Prva pomeni, da želimo čim več neposredne komunikacije in interakcije, saj bo na ta način prenos informacij najboljši in najhitrejši. Čim prej želimo uporabniku prikazati delujočo programsko rešitev (prvi prototip), namesto da se dolgotrajno najprej usklajujemo prek dokumentacije. Tako kot interakcije s člani razvojne ekipe, je potrebno tudi uporabnike (stranko) vključiti kot del ekipe in omogočiti odprto komunikacijo z njimi. Do sprememb osnovnih zahtev vemo, da bo vedno prišlo. Bistvo agilnega pristopa je, da spremembe dopuščamo in se nanje čim hitreje odzovemo, ne da se pogajamo ali je sprememba smiselna ali ne, ali je bila v planu, itd.

Filozofija agilnega razvoja temelji na nasprotju tradicionalnih pristopov, da je vse potrebno najprej analizirati in definirati, šele nato razvijati. Pravi, da se je potrebno pogovoriti o vsebini, se o tem poenotiti in čim prej pripraviti delujočo rešitev. Potem se ta prva rešitev pregleda, potrdi in nadgrajuje do končne podobe in izvedbe, ko zajema vse ali večji del uporabniških zahtev in je zrela za produkcijsko uporabo. Pri tem je treba poudariti, da je agilni pristop predvsem primeren pri povsem novem razvoju (novega poslovnega

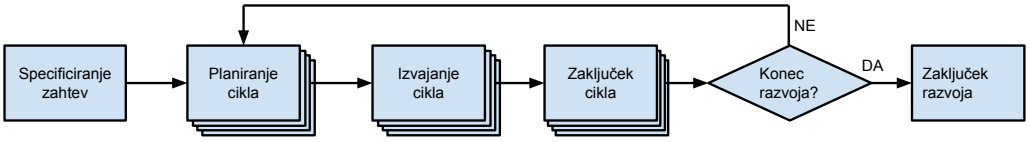
področja, funkcionalnosti), kjer je največ neznank v načinu implementacije. S prototipnim, inkrementalnim razvojem na začetku začrtamo smernice in jih potem samo korigiramo k pravi končni rešitvi.

Za primerjavo v tradicionalnem pristopu izvajamo najprej poslovno analizo. Ko pridemo do načrtovanja, že ugotovimo konceptualna razhajanja z zmožnostjo optimalnega načrta, enako se zgodi v fazi kodiranja, kjer so potencialne napačne analize in načrti ter spremembe na njih zelo potratne in pomenijo večje zakasnitve v razvoju. Tako v agilnem principu razvoja spodbujamo prilagodljiv (angl. adaptive) pristop, kjer se spremembam sproti prilagajamo. Ta je nasproten od tradicionalnega, kjer skušamo vse analizirati in načrtovati vnaprej in so spremembe zelo moteč dejavnik. V agilnem pristopu smatramo spremembe kot izziv in gonilo nadaljnjega razvoja [66, str.398-406]. Spodbuja se iterativni razvoj in ne slapovni načini. Prednost pred preobsežnim dokumentiranjem ima programiranje. Dokumentacija mora biti prava in količina ravno ustrezna za kasnejše vzdrževanje sistema. Ta predlog temelji na tem, da je vedno najbolje pogledati v programsko kodo, kako stvari delujejo, kot pa po nekih specifikacijah, ki so ponavadi zastarele in neusklajene z zadnjimi nadgradnjami v programski kodi. To lahko tudi sam potrdim, saj imamo pri našem delu dokumentacijo res zgledno, a ugotavljam, da velik del nje redko uporabljamo pri reševanju napak, ampak se raje poslužujemo vpogledov direktno v programsko kodo, saj smo le na ta način stoodstotni, da gledamo delujočo kodo in ne kakšno dokumentacijo, ki ni več aktualna. Sicer po mojem mnenju to velja do trenutka, ko ljudje, ki so razvili sistem, le-tega tudi vzdržujejo. Ko se vzdrževanju priključujejo novi posamezniki, pa je dokumentacija ključnega pomena za določanje mesta napake, za natančno diagnozo pa se pogleda v programsko kodo.

Na shemi 4.4 je prikazan generični proces agilnega razvoja nekega sistema, ki pravi:

- zahteve so definirane v prioritetno urejenem seznamu zahtev,

- za vsak cikel oziroma iteracijo se vzame zahteve za cikel,



Slika 4.4: Konceptualna shema agilnega razvoja

- izvaja se cikel,

- spremlja se izvajanje cikla in na koncu preveri ustreznost izvedbe,

- glede na rezultate cikla se začne nov cikel ali pa zaključi razvoj sistema.

Danes lahko tako vse tehnike razvoja (tako stare kot nove), ki izpolnjujejo vrednote agilnega, štejemo med agilne metodologije in principe. Med drugim so danes najbolj znane in razširjene naslednje:

- prilagodljivi razvoj programske opreme (angl. Adaptive Software Development – ASD),

- agilno modeliranje (angl. Agile Modeling),

- agilno procesno ogrodje (angl. Agile Unified Process),

- metode Crystal (angl. Crystal Methods),

- ogrodje urejene agilne dobave (angl. Disciplined Agile Delivery – DAD),

- metoda razvoja dinamičnih sistemov (angl. Dynamic Systems Development Method - DSDM),

- ekstremno programiranje (angl. Extreme Programming – XP),

- funkcionalno voden razvoj (angl. Feature Driven Development – FDD),

- Scrum.

Vse naštete metodologije temeljijo na osnovi agilnega principa – to je kratkih razvojnih ciklih in hitri dobavi rezultatov razvoja uporabnikom. Velik poudarek dajejo tudi na čim večji/stalni vključenosti naročnika – uporabnika. Razlikujejo pa se v:

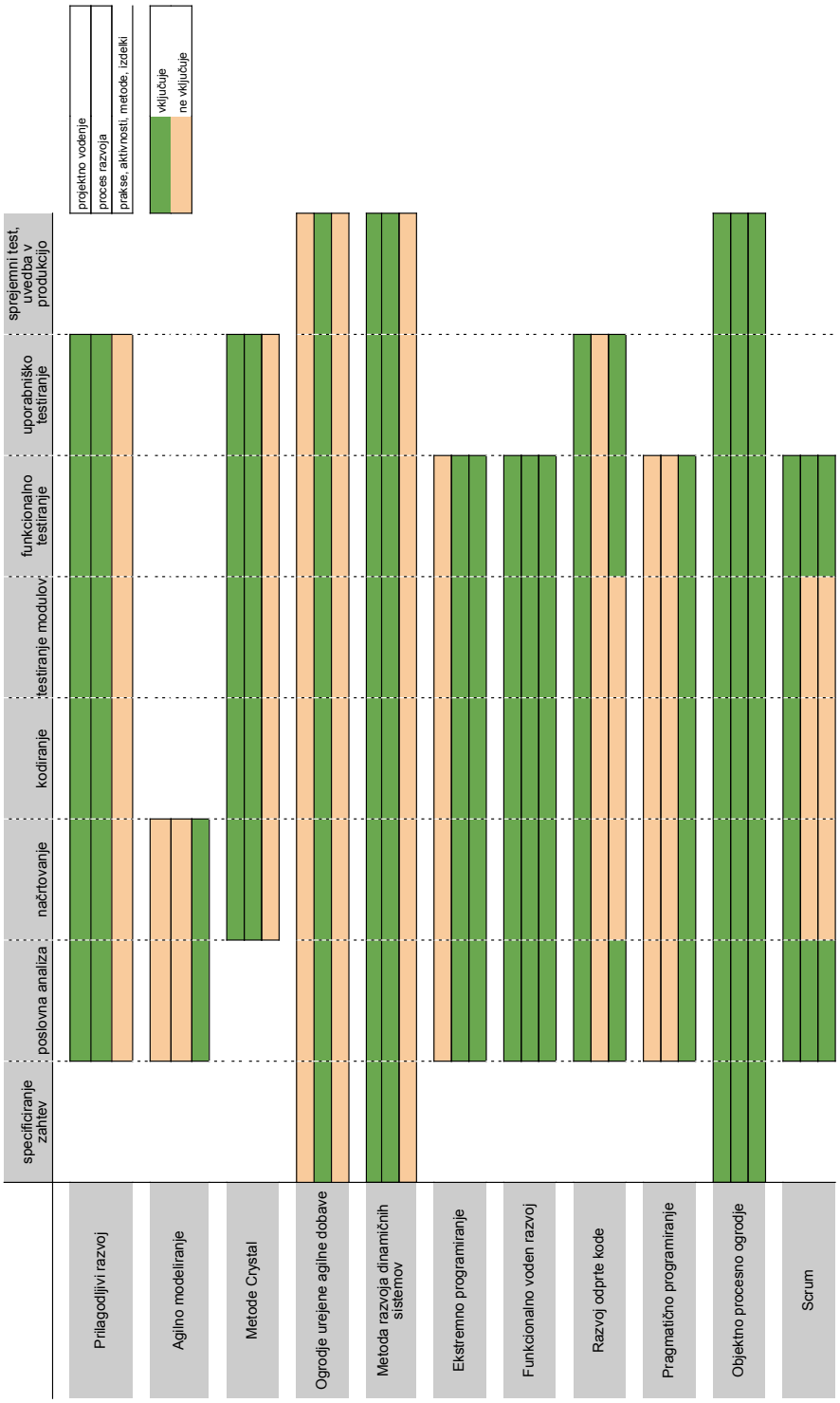
- deležu celotnega razvojnega procesa, ki ga metodologija zajema,
- stopnji podrobnosti definicije delovnih postopkov,
- vodenju, nadzoru izvajanja razvoja,
- vlogah posameznikov, ki jih uvajajo v razvoj ter
- terminologiji.

V [3, 21] so različne agilne metodologije zelo podrobno analizirane in primerjane. Rezultati primerjave so zelo nazorno prikazani na sliki 4.5, ki je prevedena iz [3, str.95]. Veliko raziskav in primerjav agilnih pristopov najdemo tudi v slovenski literaturi, med drugim v [65].

Na sliki 4.5 so navedene tudi tri metode, ki niso strogo agilne, vendar tudi temeljijo na iterativnem in inkrementalnem razvoju, zato so uvrščene v primerjavo:

- razvoj odprto kodne programske opreme (angl. Open Source Software Development),
- pragmatično programiranje (angl. Pragmatic Programming),
- objektno procesno ogrodje (angl. Rational Unified Process – RUP).

S tremi črtami na sliki 4.5 pri vsaki metodologiji so prikazana področja, ki so v posamezni metodologiji zajeta ali ne. Zelena obarvanost črte pomeni, da metodologija v navedeni fazi pokriva področje, oranžna pa pomeni da ne. Prva črta predstavlja podatek ali metodologija govori o načinih vodenja projekta (agilnem, prilagodljivem vodenju projektov seveda). Druga ponazarja



Slika 4.5: Primerjava agilnih metodologij

ali metodologija govori o razvojnem procesu in ga določa. Tretja pa ponazarja ali metodologija definira postopke, dobre prakse za delo po metodologiji ter katere aktivnosti in izdelke je potrebno po metodologiji izdelati.

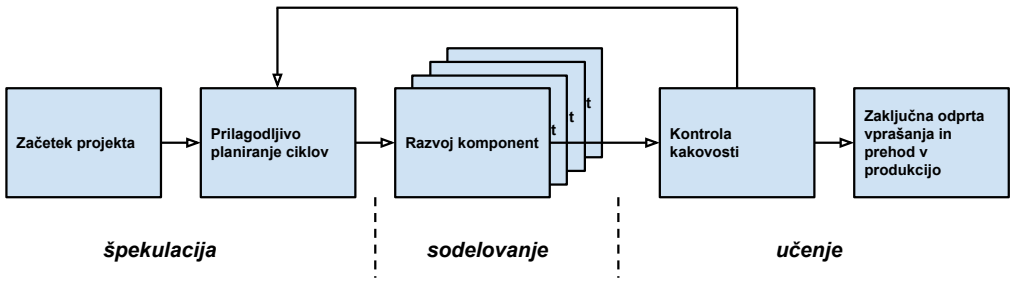
Od naštetih agilnih metodologij le dve zajemata celoten proces razvoja, ki je celovito predpisan v objektno procesnem ogrodju (angl. RUP). To sta metodi urejene agilne dobave (angl. DAD) in razvoja dinamičnih sistemov (angl. DSDM). V primerjavi z RUP se razlikujeta, da nobena izmed njih ne definira podrobnih postopkov in izdelkov, ki jih je potrebno pripraviti. Ogrodje agilne dobave (DAD) definira le proces razvoja po agilnem principu in ga integrira v okolje delovne organizacija (angl. enterprise). Metoda razvoja dinamičnih sistemov (DSDM) pa poleg procesa določa še nadzor, usmerjanje in spremljanje razvoja (vodenje projekta), ki zagotavljajo, da je končna rešitev za uporabnika najboljša.

Vse ostale metodologije pokrivajo manjši del celotnega razvojnega procesa. Najožja pri tem je metoda agilnega modeliranja (angl. AM), ki pa definira predvsem postopke dokumentiranja poslovne analize in načrtovanja. Ta metodologija je temelj metode agilno procesnega ogrodja (AUP) in ogrodja urejene agilne dobave (DAD).

4.9.1 Prilagodljivi razvoj programske opreme

Metoda prilagodljivega razvoja programske opreme (angl. Adaptive Software Development – ASD) [26, 25] se osredotoča na razvoj velikih, kompleksnih sistemov. Bistvo metode je, da spodbuja inovativnost in izdelavo prototipov za veliko različnih možnosti izvedbe, a vendar zagotavlja uspešnost izvedbe projekta. To zagotavlja s cikličnim razvojem in tremi fazami vsakega cikla, ki že s poimenovanjem določajo pomen sprememb in iskanja optimalnih rešitev za metodologijo. Faze vsakega cikla so:

- špekulacija (angl. speculate),
- sodelovanje (angl. collaborate),
- učenje (angl. learn).



Slika 4.6: Življenjski cikel metodologije ASD

Če jih iz terminologije metodologije prevedemo na tradicionalno poimenovanje, lahko fazi špekulacije rečemo tudi faza planiranja in usklajevanja zahtev. Faza sodelovanja se pokriva s fazo analize, načrtovanja in razvoja. Govori pa o ključnem pomenu timskem delu v vseh fazah, kar pomeni, da v vseh fazah razvoja dejavno sodelujejo vse vloge pri razvoju. Tako izvajalci kot tudi uporabniki, odločevalci in ostali deležniki morajo ves čas aktivno sodelovati. Metodologija trdi, da le s sodelovanjem vseh vlog lahko izziv razvoja raziščemo iz vseh zornih kotov in na ta način poiščemo najboljše rešitve. Fazi učenja pa je analogna faza testiranja (od funkcionalnega do uporabniškega), katere pomen je, da se iz nje naučimo spoznanj trenutne iteracije (pripravljenega prototipa) in jih uporabimo za izboljšanje v naslednji iteraciji. Testiranja po metodologiji niti več strogo ne ločimo na testiranje modulov, funkcionalno testiranje, uporabniško testiranje, ker so vsi člani del ekipe in stalno sodelujejo skupaj ter tako vse teste izvedejo hkrati.

Metodologija po shemi 4.6 na začetku definira ključne rezultate projekta in določi datume, do kdaj morajo biti rezultati projekta na voljo. Vsak cikel se potem planira sproti in predlagano naj traja od štiri in osem tednov.

Bistvo te metodologije je v končnem rezultatu in kakovosti delujoče programske opreme in ne na definiranih postopkih in vmesnih izdelkih, ki morajo nastati. V metodologiji se stalno izvaja testiranje, ki mu tu bolj po industrijsko avtor reče kar kontrola kakovosti. Rezultati stalne kontrole kakovosti so podlaga za naslednje cikle razvoja, kamor stalno vključujemo tudi spremembe

in nova spoznanja, ki nastanejo že v samem razvoju (fazi sodelovanja). Za kontrolo kakovosti in vključevanja sprememb in spoznanj v fazi sodelovanja avtor metode (Highsmith [26]) predlaga čim pogostejše skupne delavnice vseh udeležencev odgovornih za razvoj, ki jih po metodologiji ne poimenuje, ampak gre za vse vloge, ki sodelujejo pri razvoju (analitiki, načrtovalci, razvijalci, predstavniki uporabnikov, vodje projektov/skupin). Tem delavnicam (angl. Joint Application Development Sessions – JAD sessions) metodologija ne predpisuje časovnega okvira in pogostosti, ampak to prepušča odločitvi razvojne ekipe.

Zadnja faza po metodologiji je potem končni pregled z odprtimi vprašanji in prenos delujočega sistema v produkcijsko okolje. Ključno pri tej fazi, poleg prehoda v produkcijsko delovanje je, pregledati vse kar smo se tekom razvoja naučili (angl. lessons learned) in to vključiti v repozitorij dobrih praks in izkušenj za uporabo pri novem razvoju.

4.9.2 Agilno modeliranje

Metoda agilnega modeliranja (angl. Agile Modeling - AM) [6] je zelo specializirana za tradicionalni fazi poslovne analize in načrtovanja. Določa predvsem vrste in načine dokumentiranja, ki naj nastanejo v agilnih metodologijah poleg končnega izdelka (programske kode in delujoče programske opreme) ter načine komunikacije med člani razvojnih ekip. Na ta način metoda v uporabi agilnih metodologij ne nastopa samostojno, ampak kot sestavni del agilnih metodologij, ki bolj pokrivajo procesni del razvoja (kot so AUP, DAD, DSDM, Scrum, XP).

Analogno tradicionalnem pristopu in dokumentaciji po metodologiji RUP, ta metoda definira načine dokumentiranja za specificiranje zahtev, poslovno analizo in načrtovanje programske opreme. Metoda temelji na štirih vrednotah, ki so ključne za uspeh razvoja:

- komunikacija,

- preprostost,

- odzivnost,

- pogum.

Metoda spodbuja oblikovanje majhnih razvojnih ekip, stalno prisotnost uporabnika oziroma naročnika ter neposredno komunikacijo. Uvaja štirinajst najboljših praks za modeliranje (analizo in načrtovanje), ki jih razporedi v štiri kategorije sorodne zgoraj naštetim vrednotam (iterativno in inkrementalno modeliranje, timsko delo, preprostost, validacija). Omenjene najboljše prakse pa so naslednje [4]:

- ravno prav dobro modeliranje (angl. Just Barely Good Enough), ki govori da mora biti dokument ali model ravno prav dober in nič več,

- zelo priporočljivo je že takoj na začetku pripraviti model visoko-nivojske arhitekture (angl. Architecture Envisioning),

- modeliranje malo vnaprej (angl. Look Ahead Modeling) svetuje, da kljub temu, da ekipa z vsemi vlogami dela hkrati, je smiselno modelirati nekaj dni preden se vsebine lotijo razvijalci,

- posluževati se je potrebno čim večjega nabora tehnik modeliranja, da smo lahko najbolj učinkoviti z različnimi vlogami posameznikov v razvoju (angl. Multiple Models),

- aktivno sodelovanje uporabnikov (angl. Active Stakeholder Participation) je nujno za uspeh razvoja. Pod uporabniki si predstavljamo uporabnike, njihove predstavnike ali druge odločevalske vloge, ki imajo odgovore na vprašanja razvojne ekipe in moč, da odločajo o zahtevah in njihovih prioritetah,

- iz uporabniških zahtev je potrebno vnaprej izluščiti ključne poslovne cilje, skupno videnje novega sistema ter visoko-nivojske zahteve (angl. Requirements Envisioning),

- prioritizacija zahtev (angl. Prioritized Requirements) je osnova agilnega pristopa, saj zahteva, da se vse zahteve uredijo po prioriteti. V vsakem ciklu razvoja se potem razvijajo zahteve z najvišjo prioriteto,

- iterativno modeliranje (angl. Iteration Modeling) se izvede v začetku vsakega cikla kot analiza in načrtovanje zahtev, ki se bodo v ciklu razvijale. Pri tem je pomembno, da naj bi ta del (analiza in načrtovanje) v vsakem ciklu potekala le okrog 5% časa,

- testno voden razvoj (angl. Test Driven Development - TDD) [13, 10] je tehnika, kjer razvijalci najprej napišejo testni program preden začnejo z razvojem. Na ta način se osredotočijo na ključne vsebinske izzive, ki jih morajo implementirati in po razvoju preveriti, če delujejo ustrezno,

- hitro modeliranje (angl. Model Storming), ki podobno kot kresanje idej (angl. brainstorming) govori o spontanih, na hitro organiziranih delavnicah (angl. Just-In-Time (JIT) Sessions), kjer izmenjamo mnenja in pripravimo model za izbrano problematiko,

- vzporedno dokumentiranje (angl. Document Continuously) govori, da je vzporedno z razvojem potrebno tudi dokumentirati, namesto da se dokumentira vnaprej ali po koncu razvoja,

- pozno dokumentiranje (angl. Document Late) je druga tehnika dokumentiranja, ki pravi, da je potrebno dokumentacijo (sploh tehnično) napisati takrat, ko so razrešena že vsa odprta vprašanja in se razvoj bliža koncu,

- vsaka informacija mora biti shranjena samo na enem mestu (angl. Single Source Information), da se ne sprašujemo o pravih verzijah, lokacijah, itd,

- izvajalne specifikacije (angl. Executable Specifications) so tehnika, ki v povezavi s TDD govori o tem, da so najboljše specifikacije takšne, ki

jih lahko izvedemo. Takšen primer so testni programi (testni scenariji) za izvajanje testov enot, ki dajo jasne rezultate o uspešnosti.

4.9.3 Agilno procesno ogrodje

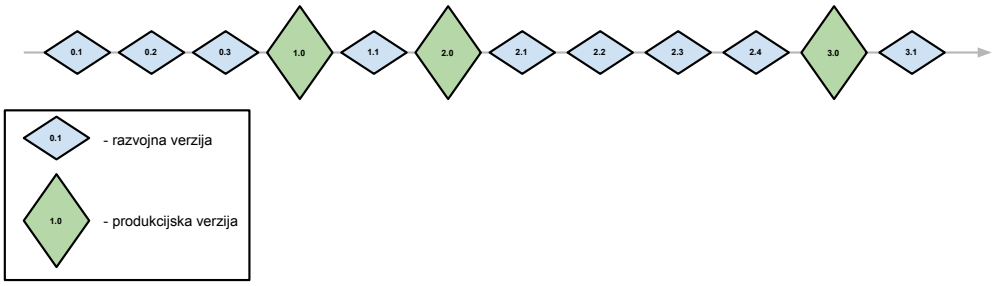
Metoda agilnega procesnega ogrodja (angl. Agile Unified Process - AUP) je okrnjena izvedba metode RUP (angl. Rational Unified Process), ki jo je razvil Scott Ambler [5]. Govori o preprostem procesu razvoja poslovnih programskih rešitev na agilen način. V tej metodi so uporabljene druge agilne metode kot so testno voden razvoj (TDD), agilno modeliranje, agilna obravnava sprememb. Metoda temelji na naslednjih principih:

- člani ekip vedo kaj delajo,
- preprostost,
- agilnost,
- osredotočanje na aktivnosti z visoko dodano vrednostjo,
- neodvisnost od orodij,
- vsak naj si agilno procesno ogrodje zgradi po svojih potrebah.

Ključno za metodo AUP je, da uvaja dvojno iteracijo v razvojnih ciklih, katerih rezultata sta:

- razvojna verzija programske opreme, ki je primerna za funkcionalno testiranje (angl. Development Release),
- produkcijska verzija programske opreme, ki gre v produkcijsko delovanje (angl. Production Release).

Metoda ne uvaja točno določenega zaporedja obeh verzij. Razvojna verzija naj bi bila pogosta in rezultat vsakega razvojnega cikla. Na vsake nekaj razvojnih ciklov pa naj bi se dosedanje razvojne verzije združile v produkcijsko verzijo, kot je prikazano na sliki 4.7.



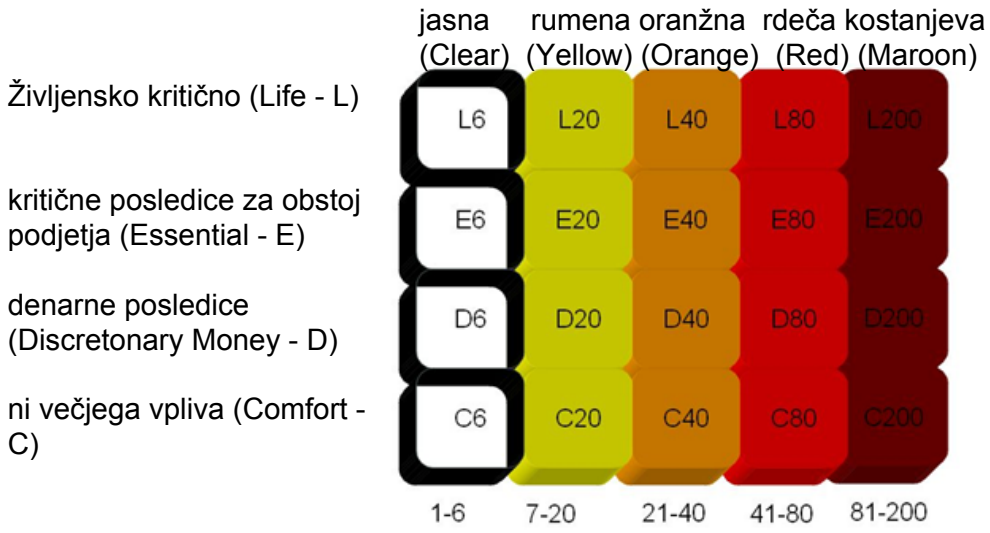
Slika 4.7: Shema priprave verzij po metodi AUP

4.9.4 Metode Crystal

Metode Crystal (angl. Crystal Methods) so skupina metodologij, ki jih je razvil Alistair Cockburn[18] na podlagi dolgoletnih izkušenj pri razvoju različnih projektov. Izkušnje kažejo, da se uspešni projekti v praksi ne držijo točno določenih posameznih metodologij, ampak med različnimi metodami in dobrimi praksami kombinirajo in si te tudi prilagodijo svojim potrebam. Da pa lahko to raznolikost zaobjamemo v neko grobo formalizirano metodologijo, je projekte ločil glede na velikost projektne ekipe in kritičnost uspešnosti projekta, ki se deli na štiri stopnje:

- zelo nizka (angl. Comfort),
- nizki stroški neuspeha (angl. Discretionary Money),
- visoki stroški neuspeha (angl. Essential Money),
- življenjsko kritični (angl. Life).

Vsakem nivoju se potem določi odtenek barve (od svetle do temnejše), ki ji pripada posamezna metodologija, kot je prikazano na sliki 4.8. Priporočilo avtorja je, da se glede na oba kriterija potem izbere ustrezna metodologija in uporabi pri razvoju.



Slika 4.8: Matrika različnih metod Crystal

Fokus metod Crystal ni na procesu dela, ampak je predvsem na timskem delu in vsem kar naredi timsko delo uspešno:

- ljudje,
- interakcija,
- pripadnost,
- znanje,
- talent,
- komunikacija.

Skupna vsem metodam pa so področja, ki po avtorjevem mnenju zagotavljajo uspešnost razvoja:

- pogoste dobave (angl. Frequent Delivery). Tako kot vse agilne metode priporoča hitre razvojne cikle. Glede na izbrano metodo (velikost in kritičnost projekta) priporoča cikle od enega tedna do enega kvartala,

- vidno izboljšanje (angl. Reflective Improvement) govori o tem, da se morajo razvojne ekipe vsake toliko časa ločiti od rednega dela in izmenjati mnenja o najboljših praksah dela z drugimi razvojnimi ekipami. Na ta način se postopki dela v organizaciji izboljšujejo,

- neposredna komunikacija (angl. Close Communication) je ključna za uspeh projekta. Najbolje je, da je ekipa v isti sobi, kar omogoča stalno izmenjavo mnenj. Za večje projekte pa morajo ekipe biti blizu skupaj (v istem nadstropju, v isti stavbi). Količina komunikacije "z zamudo" kot so telefonski klici, e-pošta, dokumentacija se tako zelo zmanjša,

- osebna varnost (angl. Personal Safety). Vsak član ekipe se mora počutiti kot del te ekipe. Tako lahko odprto komunicira, si izmenjuje mnenja in dosega najboljše rezultate,

- osredotočenost (angl. Focus) je pomembna z vidika operativnega dela vsakega posameznika ter vidika projektnih ciljev. Prvi govori o tem, da posameznik v skupini ne sme imeti stalnih prekinitev in menjavanj področij dela, saj bo le tako delo najbolj učinkovito opravljeno. Drugo pa je jasno določanje in zavedanje o končnih projektnih ciljih, kar je naloga koordinatorja ekipe ali vodje projekta,

- dostopnost uporabnikov (angl. Easy access to expert users). Uporabniki oziroma njihovi predstavniki morajo biti dosegljivi ves čas ali kolikor se le da biti prisotni v ekipi. Pomembno je tudi, da je uporabnik pravi, ki izvaja operativno delo in tako res ekspert na svojem področju,

- tehnično okolje, ki omogoča avtomatizirano testiranje, nastavljanje konfiguracij in enostavne integracije. To je osnova za kratke razvojne cikle in hitro dobavo nadgrajene programske opreme v uporabniško testiranje in produkcijsko delovanje.

4.9.5 Ogrodje urejene agilne dobave

Metodologija urejene agilne dobave (angl. Disciplined Agile Delivery - DAD) je osredotočena na proces agilnega razvoja v večjih organizacijah [8]. Ker metodologija govori o postopnem uvajanju agilnih metodologij v organizacijo, bom tej posvetil več pozornosti. Pri predlogu novega procesa dela za razvoj sistema za bančno poslovanje in predlogu uvedbe agilnega pristopa v celotni banki, bom namreč uporabil več zgledov in priporočil te metodologije.

Ta metodologija določa ogrodje celotnega življenjskega cikla razvoja z namenom umestitve agilnih pristopov dela v večje organizacije. Ostale metode (Scrum, XP, AM, ASD) se bolj osredotočajo samo na posamezen projekt in predvsem na razvojno ekipo, ta pa vpliva na procese dela celotne organizacije. Metodologija definira fazni pristop implementacije agilnih pristopov v organizacijo po naslednjem vrstnem redu [7]:

1. osredotočiti se je potrebno na dodano vrednostjo rešitev, ne samo na dobavo uporabnikom,
2. razvojni cikel metode Scrum naj se razširi na celoten proces razvoja v organizaciji,
3. znane agilne metode naj se prilagodijo potrebam organizacije in še nadgradijo,
4. privzeti je potrebno strategije upravljanja,
5. uporabiti je potrebno ciljno usmerjen pristop za prilagajanje in spremljanje uspešnosti.

Metodologija ne predpisuje natančno definiranih postopkov, dokumentov, vlog sodelujočih, temveč vsebuje priporočila in dobre prakse kako agilne pristope prilagoditi potrebam lastne organizacije.

V razvojni cikel se v tej metodologiji vpletejo vidiki celotne organizacije in vpliv razvoja na njo. Celoten cikel podobno kot po metodi ASD razdelimo v tri faze z drugim poimenovanjem, ki so tudi vidne na sliki 4.9:

- začetna faza (angl. inception),
- faza gradnje (angl. construction),
- faza prenosa (angl. transition).

V začetni fazi se definira natančna vsebina razvoja in identificirane zahteve uredi po prioritetnem vrstnem redu. Pri pripravi seznama zahtev in določanja priortitet se upošteva vse omejitve (sredstva, število zaposlenih) in priporočila (strategija organizacije, smernice, tehnološka strategija).

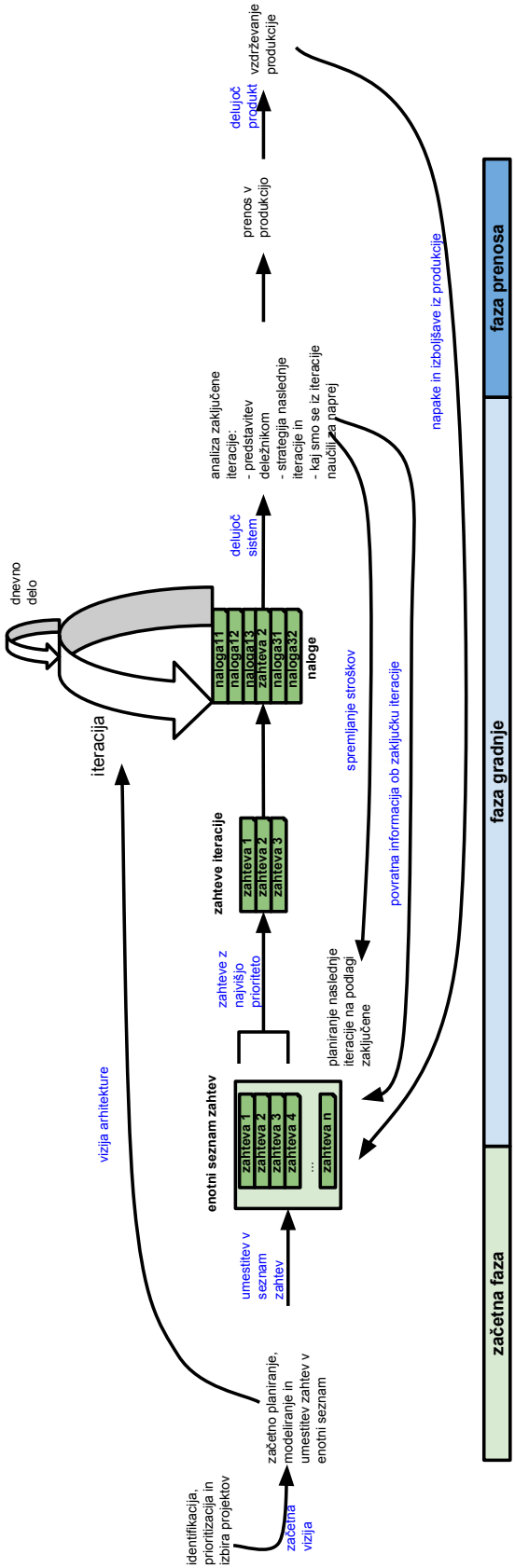
V fazi gradnje oziroma konstrukcijski fazi se izvaja razvoj, kjer za vsak cikel:

- iz seznama zahtev vzamemo zahteve z najvišjo prioriteto,
- jih podrobneje razdelimo na opravila,
- opravila razvijemo po agilnem pristopu (z dnevnim usklajevanjem, npr. Scrum) in sprotnim testiranjem ter
- predstavimo vsem odgovornim za uspešnost rešitve (uporabnikom, vodjem).

Glede na rezultat predstavitve odgovornim za uspešnost rešitve, se potem odloči ali je do sedaj razvita rešitev primerna za prehod v produkcijsko delovanje. Če je primerna, se začne faza prenosa, kjer rešitev prenesemo v produkcijsko delovanje in spremljamo zadovoljstvo uporabnikov in ostalih odgovornih.

Vzporedno s prenosom neke pripravljene verzije rešitve v produkcijsko delovanje pa se nadaljuje nov cikel gradnje, kjer implementiramo naslednje zahteve iz seznama in pri tem upoštevamo rezultate in spoznanja zadnjega cikla. V primeru, ko smo vse zahteve že implementirali in so z delujočo rešitvijo vsi udeleženci zadovoljni, se razvoj konča.

Metodologija predlaga, da za vsako aktivnost v predlaganem razvojnem procesu pregledamo vse vidike, ki vplivajo na izvedbo aktivnosti. Prav tako



Slika 4.9: Življenjski cikel po metodologiji DAD. Vir: [7, str.11]

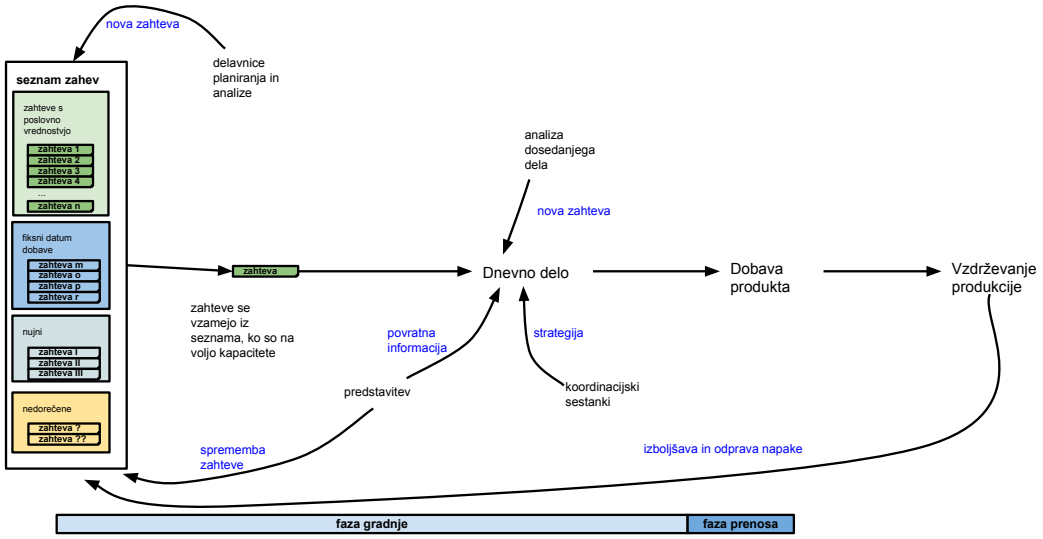
predlaga, naj se za izvedbo aktivnosti poslužujemo različnih tehnik, ki izhajajo iz drugih metodologij.

Z vidika celotne organizacije in uvedbe agilnih pristopov je zelo pomembna koordinacija in izmenjava informacij med vsemi organizacijskimi enotami. Metodologija zato predlaga naslednji proces koordinacije, ki vključuje vse vidike na katere je potrebno biti pozoren:

- izmenjava informacij,
- jasna lastništva in odgovornosti,
- koordinacija ekipe,
- koordinacija področja,
- koordinacija v okviru IT,
- koordinacija urnikov dobave,
- koordinacija med lokacijami.

Ključni vidiki, ki so upoštevani pri metodologiji DAD so naslednji:

- ljudje so ključni za uspeh razvoja,
- usmerjenost k učenju in nadgrajevanju znanj. Ljudje si morajo izmenjavati znanja ter se učiti na lastnih spoznanjih (npr. prejšnjih razvojnih ciklov),
- hibridni pristop - ne implementirajmo samo ene metodologije, ampak vzemimo stvari iz različnih metodologij, ki so za našo organizacijo najbolj primerne,
- pokriva celoten razvojni cikel,
- ciljno usmerjeni procesi,
- usmerjenost k uporabni končni rešitvi,



Slika 4.10: Princip stalnih dobav po metodologiji DAD. Vir: [7, str.13]

- usmerjenost k dodani vrednosti in upoštevanju tveganj,

- zavedati se pomena metodologije za celotno organizacijo.

Skrajni cilj razvojnega procesa po tej metodologiji je zagotoviti stalne dobave (angl. continous delivery). To v praksi pomeni, da nimamo več ločene začetne faze v kateri pripravljamo zahteve. Začetna faza postane samostojen proces ažuriranja seznama vseh odprtih zahtev, ki upošteva vse potrebne vidike celotne organizacije 4.10:

- poslovna strategija organizacije,

- tehnološka strategija,

- nove usklajene uporabniške zahteve,

- dopolnjene (spremenjene) zahteve iz razvojnih ciklov,

- napake in želje po izboljšavah delovanja.

Tako v samem razvojnem ciklu poznamo samo še fazo gradnje, v sklopu katere razvijemo izbrane zahteve iz skupnega seznama. Na koncu te faze pa je še zelo kratka faza prenosa, kjer se samo preveri ali je funkcionalnost primerna glede na zahtevo. Ko se to potrdi, se prenese v produkcijsko delovanje in izmeri zadovoljstvo vseh udeležencev.

4.9.6 Metoda razvoja dinamičnih sistemov

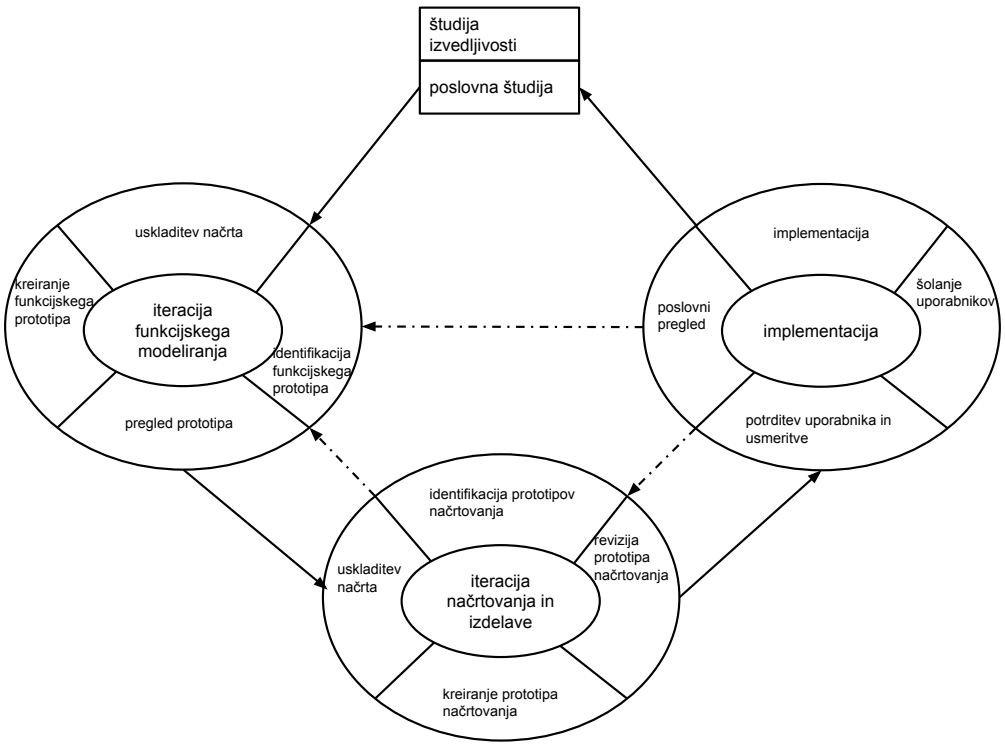
Metoda razvoja dinamičnih sistemov (angl. Dynamic Systems Development Method - DSDM) [61] je najbolj podrobna od vseh, ki jih obravnavamo. Vključuje vse faze procesa razvoja in natančno določa postopke, vloge in dokumentacijo, ki mora v posamezni fazi nastati. Metoda DSDM z vsemi predpisanimi postopki služi kot ogrodje za hiter razvoj programske opreme (angl. Rapid Application Deveopment – RAD).

Temeljna ideja metode DSDM sloni na drugačnem pogledu na razvoj. Namesto, da definiramo funkcionalnosti, ki jih bomo razvijali in na te potem ocenjujemo potreben čas ter ostale vire za razvoj, najprej določimo čas in vire, ki ga imamo na voljo. V definiran okvir časa in ostalih virov (predvsem ljudi) nato umestimo obseg razvoja funkcionalnosti programske opreme.

Proces razvoja po metodi DSDM je sestavljen iz petih faz, ki so prikazane na sliki 4.11. Od teh faz sta dve enkratni, ostale tri pa so iterativne ter inkrementalne:

- študija izvedljivosti (angl. feasibility study),
- poslovna študija (angl. business study),
- iteracija funkcijskega modeliranja (angl. functional model iteration),
- iteracija načrtovanja in izdelave (angl. design and build iteration),
- implementacija (angl. implementation).

V fazi študije izvedljivosti se oceni primernost metode za projekt razvoja izbrane programske opreme. Predvsem se preveri zrelost organizacije in sodelujočih na projektu za metodo DSDM. Prav tako se v tej fazi preveri tehnične



Slika 4.11: Faze metode razvoja dinamičnih sistemov. Vir: [3, str.62]

možnosti izvedbe razvoja in identificira tveganja pri razvoju. Kot rezultat te faze nastane poročilo o izvedljivosti, okvirni plan razvoja ter v določenih primerih grobi prototip za lažjo predstavo zahtevanih vsebin.

V fazi poslovne študije se analizira uporabniške zahteve do te mere, da v naslednji fazi lahko začnemo detajlno poslovno analizo. Na delavnicah z uporabniki se preveri vsebina zahtev in določi prioriteto njihove implementacije. Rezultat te faze je tudi grobi poslovni proces vsake zahteve, visoko-nivojska arhitektura sistema ter načrt priprave prototipov, ki določa okvirje za naslednje tri faze.

Faza funkcijskega modeliranja se lahko imenuje tudi faza poslovne analize. Vsako iteracijo te faze lahko razdelimo na štiri pod-faze:

- identifikacija funkcijskega prototipa, kjer se pripravi seznam zahtev, ki bodo predmet prototipa. Ta se pripravi iz zahtev na podlagi izkušenj prejšnje iteracije,
- usklajevanje plana, v sklopu katerega se definira kako in kdaj se implementira seznam zahtev, ki so bile identificirane za ta prototip,
- izdelava prototipa,
- pregled in analiza izdelanega prototipa.

Ko je iteracija te faze zaključena, ji sledi faza načrtovanja in izdelave, katere rezultat je testirana verzija sistema. Tudi ta faza ima enake pod-faze kot funkcijsko modeliranje.

Ko je prototip sistema testiran, sledi še faza implementacije. V sklopu te faze se prototip prenese v produkcijsko delovanje. Poleg samega produkta, se opravi tudi uvajanje uporabnikov in pripravi uporabniška navodila. Glede na delovanje prototipa se potem odločimo za ponovitev iteracij in dopolnitve funkcionalnosti ali pa zaključimo razvoj, če ugotovimo zadostno število implementiranih zahtev in je delovanje ustrezno.

Celotna DSDM temelji na praksah, ki se pojavljajo tudi v ostalih agilnih metodah:

- aktivno sodelovanje uporabnika je nujno,
- vsaka ekipa, ki izvaja DSDM mora biti ustrezno velika, da lahko sprejema avtonomne rešitve,
- delujočo programsko opremo je potrebno hitro dobaviti v uporabo,
- ključen kriterij ustreznosti iteracij je, da vsebuje prave funkcionalnosti,
- iterativni in inkrementalni razvoj je ključen, da se lahko prilagajamo spremembam,
- zagotavljamo sledljivost vseh sprememb v iteracijah,
- samo grobe zahteve se ne smejo spreminjati. Podrobne zahteve so nespremenljive znotraj iteracije, a se na podlagi rezultatov iteracije in drugih kriterijev lahko dopolnijo za naslednjo iteracijo,
- testiranje in integracija se izvaja stalno v okviru celotnega cikla,
- ključen je celovit pregled in transparenten prikaz informacij med vsemi udeleženci v razvojnem procesu.

4.9.7 Ekstremno programiranje

Ekstremno programiranje (angl. Extreme Programming – XP) je ena izmed bolj razširjenih agilnih metodologij za razvoj programske opreme. Predvsem je poznana po principih za hitro prilagajanje spremembam uporabniških zahtev ter tehnikah, ki zagotavljajo visoko kakovost končne programske rešitve.

Celoten razvojni proces, ki ga zajema metodologija ekstremnega programiranja je razdeljen v naslednje faze, ki so prikazane tudi na 4.12:

- faza raziskovanja (angl. Exploration phase),
- faza planiranja (angl. Planning phase),
- faza razvoja iteracij do dobave programske rešitve (angl. Iterations to release phase),

- faza priprave za prenos v produkcijo (angl. Productionizing phase),

- faza vzdrževanja (angl. Maintainance phase),

- faza dokončanja (angl. Death phase).

V fazi raziskovanja uporabniki pripravijo natančne opise zahtev (angl. stories), ki morajo biti vključene v prvi dobavi delujoče programske rešitve. Vzporedno s tem se izvedbeni del ekipe seznanja s tehnologijo, v kateri bo potekal razvoj, arhitekturo ciljnega sistema in vsebinami, ki so predmet razvoja.

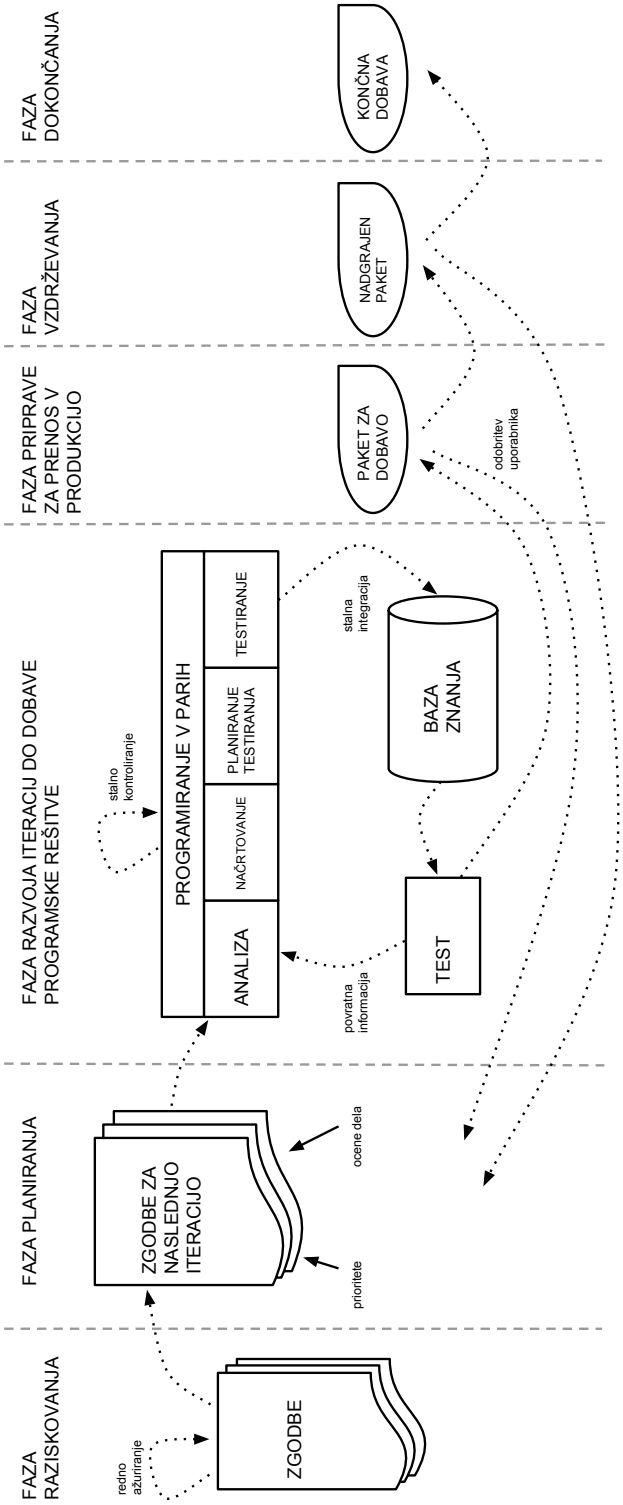
Tej fazi sledi planiranje, v sklopu katerega se določi prioritete uporabniških zahtev in potrdi vsebine prve dobave. V tej fazi se vse zahteve prav tako ocenijo s časom potrebnim za razvoj. Na podlagi teh ocen in prioritet uporabniških zahtev se zgradi časovnica celotnega razvoja.

V naslednjih fazah razvoja posameznih iteracij se uporabniške zahteve razvijajo. Na koncu vsake iteracije se izvede funkcionalno testiranje, ki potrdi, da je iteracija kandidat za produkcijo. V fazi priprave za produkcijo se izvaja še uporabniško ter performančno testiranje, ki potrdita ustreznost rešitve.

Ko je produkt delujoč v produkcijskem okolju, se začne faza vzdrževanja, ki je še posebej zahtevna vse dokler se vzporedno razvijajo nove uporabniške zahteve. Pri vseh spremembah in nadgradnjah je potrebno biti posebej pozoren, saj nadgradnja ne sme porušiti delovanja prejšnje verzije delujočega sistema. V pripravo novih verzij (iteracij) pa je vzporedno potrebno vključiti tudi vse odkrite napake v produkcijskem delovanju.

Na koncu sledi še faza zaključka razvoja, ki se prične, ko so vse uporabniške zahteve podprte in ni več novih zahtev uporabnikov. Razlog za fazo zaključka je lahko tudi drugačen v primeru, ko delujoča rešitev na prinaša zahtevanih rezultatov ali pa je predraga za nadaljnje vzdrževanje.

Principi ekstremnega programiranja, kot jih navaja Beck v [12] so naslednji:



Slika 4.12: Proces razvoja po ekstremnem programiranju. Vir: [3, str.19]

- v planiranje je potrebno aktivno vključiti razvijalce in uporabnike. Razvijalci pripravijo oceno za zahtevano funkcionalnost, potem pa se skupaj z uporabniki dogovorijo o rokih za izvedbo,
- nadgradnja sistema mora biti v produkcijskem okolju na mesečni ravni, v najslabšem primeru vsake dva do tri mesece,
- oblikujejo se modeli, ki omogočajo enoten pogled na zahtevo, tako da imajo vse vloge enak pogled na delujoč sistem,
- načrt rešitve mora biti enostaven, da omogoča hiter razvoj,
- razvoj mora biti voden na podlagi rezultatov testiranja,
- sistem je potrebno optimizirati in prilagajati,
- programiranje v parih [20],
- vsa dokumentacija in izvorna koda mora biti vedno na voljo za spremembo, tako da lahko vsakdo iz razvojne ekipe izvede nadgradnjo,
- stalno izboljševanje delujoče programske rešitve,
- delovnik mora biti 8-urni, ker je ta najbolj učinkovit,
- uporabnik mora biti stalno prisoten v razvojnih timih,
- razvijalci se držijo standardov kodiranja,
- delovno okolje mora biti takšno, da lahko komunikacija med sodelujočimi poteka nemoteno,
- pravila razvojne ekipe so definirana, ampak se lahko tudi spreminjajo in prilagajo novim potrebam.

Prednost metodologije ekstremnega programiranja je v tem, da le definira principe dela in najboljše prakse. Te principe se potem poveže in prilagodi specifičnim potrebam različnih razvojnih projektov.

V praksi je metodologija znana predvsem po nekaj principih oziroma tehnikah, kot so programiranje v parih (angl. pair programming) in visoki kakovosti programske kode kot rezultat takšnega način dela.

Za ekstremno programiranje je skoraj nujno, da je ekipa skupaj v isti sobi kar omogoča neposredno komunikacijo. Zaradi stalnega skupnega dela je metodologija primerna za manjše projekte, kjer sodeluje od 4 do 20 ljudi.

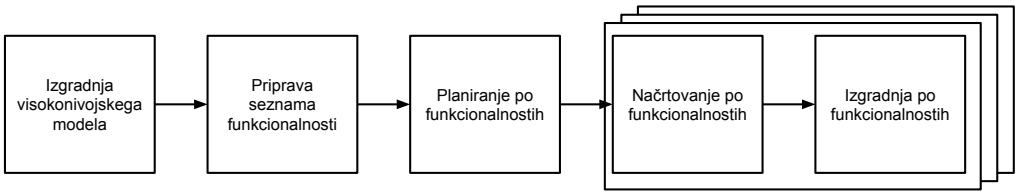
4.9.8 Funkcionalno voden razvoj

Metoda funkcionalno vodenega razvoja (angl. Feature Driven Development – FDD) [47] definira agilen in prilagodljiv princip razvoja programske opreme. Podobno kot ekstremno programiranje ne pokriva vseh faz celotnega razvojnega procesa, ampak se osredotoča predvsem na načrtovanje in razvoj (programiranje, kodiranje) delujoče programske rešitve.

Celoten razvoj se po metodologiji FDD izvaja na podlagi zahtevanih funkcionalnosti programske rešitve. Proces dela je razdeljen na pet faz kot je tudi nazorno prikazano na 4.13:

- izgradnja visoko-nivojskega modela,
- priprava seznama funkcionalnosti,
- planiranje po funkcionalnostih,
- načrtovanje po funkcionalnostih,
- izgradnja (razvoj) po funkcionalnostih.

V fazi izgradnje visoko-nivojskega domenskega modela se analizira uporabniške zahteve, ki morajo že biti analizirane in pripravljene v obliki primerov uporabe ali funkcionalnih specifikacij. Na podlagi tehničnih omejitev in uporabniških zahtev se pripravi visoko-nivojska arhitektura sistema in vanjo umesti področja iz uporabniških zahtev. Vsako področje se potem podrobneje analizira do izgradnje objektnega modela.



Slika 4.13: Faze funkcionalno vodenega razvoja. Vir: [3, str.48]

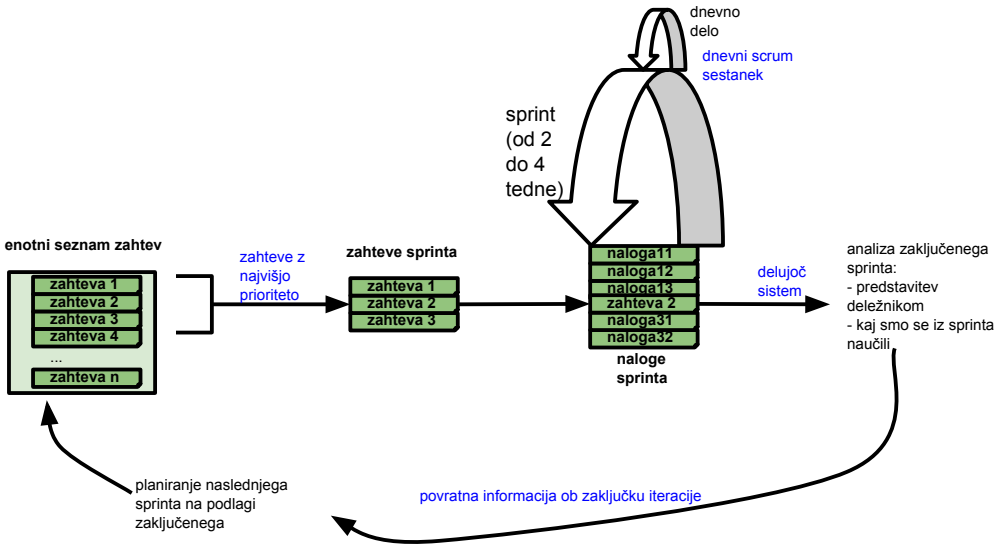
Nato se začne faza priprave seznama funkcionalnosti novega sistema. Iz vsakega področja se identificira seznam funkcionalnosti, ki jih mora končni sistem podpirati. Funkcionalnosti se lahko združuje več-nivojsko v funkcionalne sklope. In ko je seznam vseh funkcionalnosti tako podroben ter razumljiv in usklajen med uporabnikom, sponzorjem in člani razvojne ekipe, se lahko razvoj nadaljuje v naslednji fazi planiranja.

Planira se funkcionalne sklope in posamezne funkcionalnosti znotraj njih na način, ki ga uskladijo uporabniki in glavni razvijalec. Vrstni red je usklajen na podlagi tehničnih zahtev, da se pripravi ogrodje za končno rešitev in uporabniških prioritet po zaporedju funkcionalnosti, ki jih želijo na voljo.

Od tu naprej sledita fazi razvoja in sicer načrtovanja in kodiranja. Za vsak cikel teh dveh faz se iz planiranega seznama vzame določeno število funkcionalnosti in se jih načrtuje in takoj tudi razvije (kodira). Po priporočilih naj ta cikel traja od nekaj dni do največ dveh tednov. Rezultat tega cikla mora biti delujoč sistem, ki je testiran, integriran z ostalimi sistemi, ustrezno kvalitetno preverjen in podpira nove funkcionalnosti.

Ta metoda je v praksi zelo podobna klasičnim metodam kot je slapovni model, saj so prve tri faze povsem zaporedne in se fazi hitrega cikličnega (iterativnega in inkrementalnega) načrtovanja in razvoja ne moreta začeti dokler te niso narejene. Tudi vse potrebne vloge, ki nastopajo pri takem načinu razvoja (projektni vodja, vodja razvoja, glavni arhitekt, tester, ...) so zelo podobne tradicionalnim vlogam.

Metoda je zato zelo primerna za tranzicijo razvojne organizacije iz tra-



Slika 4.14: Proces metodologije scrum. Vir: [7, str.2]

dicionalnega načina dela v moderen agilni pristop. Princip je namreč ta, da moramo na tradicionalen način analizirati zahteve in iz njih izluščiti funkcionalnosti, ki jih potem s hitrimi cikli razvijemo in tako vključimo uporabnika in ponudimo delujočo rešitev. Zaradi faz podrobnega analiziranja uporabniških zahtev in planiranja preden se lotimo razvoja, je primerna za velike projekte, ker zagotavlja ustrezen nadzor izvajanja in jasno določene časovne mejnike.

4.9.9 Scrum

Scrum [56] je agilna metodologija, ki je danes sigurno najbolj poznana in razširjena. Ker je o tej metodologiji veliko napisanega tudi v slovenski literaturi [43, 41, 65], je ne bom podrobno opisoval. Metodologija ne pokriva celotnega razvojnega procesa temveč predvsem izvedbene faze od pripravljenega seznama zahtev do načrtovanja in razvoja teh zahtev ter priprave delujoče programske rešitve.

Metodologija se je najbolj prijela predvsem zaradi natančne definicije

postopka razvoja in terminologije, ki jo je uvedla za korake v postopku kot je prikazano tudi na 4.14.

Ko se začne razvoj nove programske rešitve, se najprej pripravi seznam zahtev tega produkta urejen po prioriteti (angl. Product Backlog). Vsako zahtevo iz seznama se razbije na manjše zahteve, ki gredo v razvoj – tako se pripravi seznam zahtev za cikel (angl. Sprint Backlog). En cikel po Scrum metodologiji traja od enega tedna do enega meseca. Najpogosteje je dolžina cikla dva tedna. Na začetku cikla se pripravi delavnica za planiranje cikla, znotraj cikla pa potekajo dnevni kratki sestanki (15 minut) na katerih vsak član razvojne ekipe pove posebnosti prejšnjega dne, svoj dnevni plan in posebnosti, ki bi jih bilo smiselno izpostaviti vsem članom ekipe. Na koncu vsakega cikla sledi še zaključni sestanek, na katerem se pogleda uspešnost planiranega cikla in eventualne neizvedene zahteve prenese v naslednji cikel.

4.9.10 Povzetek primerjave agilnih metodologij

Vse naštete in opisane agilne metodologije lahko med seboj primerjamo iz različnih vidikov, eden izmed teh je prikazan v tabeli 4.1. Predvsem ugotavljamo, da so si v principu zelo podobne, saj vse izhajajo iz izhodišč agilnega manifesta in temeljijo na prilagodljivi uporabi posameznih postopkov in najboljših praks za različne namene oziroma vrste razvoja. Če pogledamo na agilni pristop s praktičnega vidika, lahko rečemo, da gre za upoštevanje zdrave pameti in postopnosti pri razvoju. Tako namreč pravi Phillips [50] na strani 174 za avtorje agilnega manifesta, "ki zardijo ob besedah, da so naredili nekaj povsem novega in neverjetnega". Pogosto namreč odgovorijo, da gre le za "principe zdrave pameti."

Faze od načrtovanja do funkcionalnega testiranja so pokrite z metodologijami Scrum, ekstremnim programiranjem in funkcionalno vodenim razvojem. Enake faze pokriva tudi metodologija prilagodljivega razvoja, s tem da vključuje še uporabniško testiranje.

Če naštete metodologije pogledamo z vidika vodenja projektov, je to vključeno v metodologija ASD, Crystal, DSDM, FDD, Scrum. Te namreč

zagotavljajo vodenje in koordinacijo razvoja in usmerjanje k pravi končni rešitvi.

Zelo podrobno postopke definirajo v DSDM in Scrum. Pri obeh je tudi zelo pomembna uporaba prave terminologije in zastopanosti vlog, ki jih metodologiji definirata. Predvsem Scrum, ki je danes v zelo široki uporabi, uvaja povsem svoje poimenovanje. Namen tega je doseči prepoznavnost metode s specifičnim izrazoslovjem, ki ga ostale metode posplošijo. Kot primer navedimo cikel ali iteracijo, ki ji po metodologiji Scrum rečemo sprint. Scrum metodologija je zelo stroga glede postopkov dela, kar v nekaterih ostalih metodologijah in tudi za našo uporabo pri predlogu izboljšave razvojnega procesa ni nujno. Pri predlogu izboljšave želimo predvsem uvesti princip manj obsežnih in kratkih razvojnih ciklov in se ne toliko vezati na eno metodologijo. Želimo se predvsem vezati na idejo agilnega pristopa in iz znanih metod izluščiti tisto, kar je za nas najbolj uporabno.

Metodologija	Št. faz razv. procesa	Podrob. postopka [1-5]	Vodenje	Terminologija	Področje uporabe	Razširjenost metode
ASD	6	2	da	špekulacija, sodelovanje, učenje,	potrditev kakšnega koncepta	nizka
AM	2	3	ne	/		srednja
AUP	4	3	na	/	manjši projekti	nizka
Metode Crystal	5	2	da	/	vse vrste projektov	nizka
DAD	8	3	da	/	večje organizacije	srednja
DSDM	8	4	da	študija izvedljivosti, poslovna študija, iteracija funkcionalnega modela	za manjše projekte	srednja
XP	5	3	ne	metafore, programiranje v parih	majhni, raziskovalni projekti	visoka
FDD	5	3	da	/	veliki projekti, večje organizacije	srednja
Scrum	5	5	da	sprint, seznam zahtev	/	visoka

Tabela 4.1: Primerjava agilnih metodologij

Poglavje 5

Aplikacija agilnih in vitkih konceptov za rešitev identificiranih slabosti obstoječega načina dela

Kaj je cilj izboljšanega procesa razvoja:

- kako čim hitreje ponuditi delujočo rešitev (informacijsko podporo) od uporabnikove zamisli (angl. time to market),
- kako z manjšimi zmogljivostmi zagotoviti kar se da podoben obseg razvoja in ohraniti visoko kakovost sistema za bančno poslovanje.

Za izpolnitev obeh ciljev se bomo zatekli k agilnim tehnikam razvoja in vitkim principom [34].

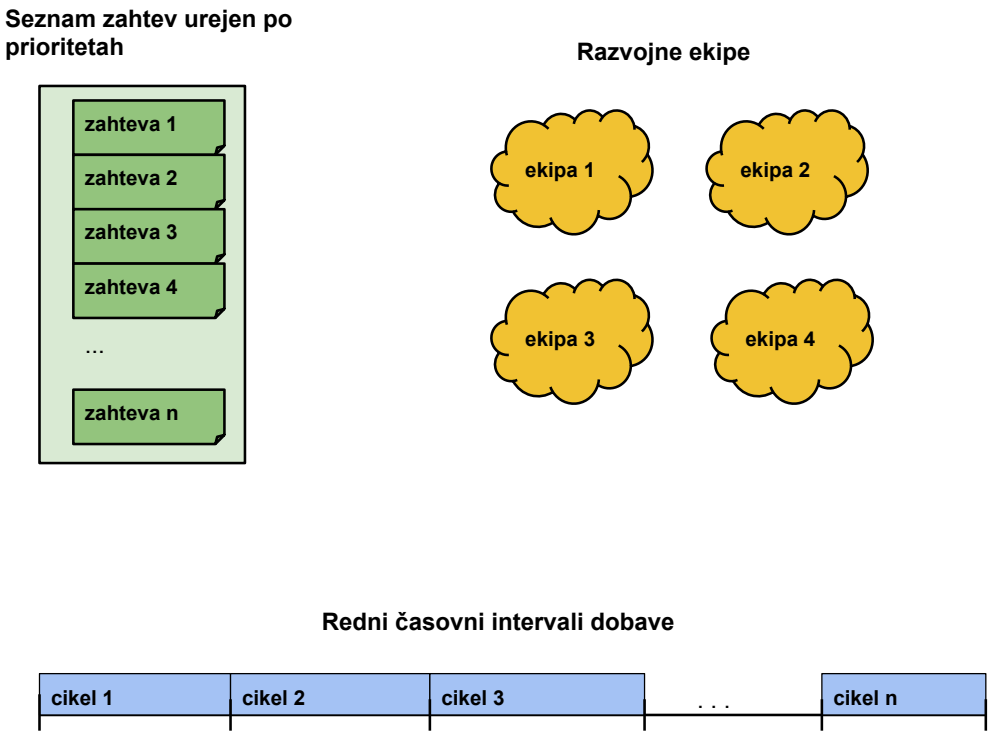
5.1 Kako hitreje ponuditi delujočo rešitev od zamisli?

Kako uporabnikom čim hitreje ponuditi delujočo rešitev [49] od same zamisli je ključen izziv, ki ga moramo rešiti pri razvoju sistema za bančno poslovanje.

Danes uporabniki namreč najmanj pol leta (1 produkcijski cikel – verzijo) čakajo na razvoj podpore njihove nove zahteve. V praksi je povprečen čas celo bližje enemu letu, saj se zamisel sama najprej oblikuje, potem analizira in umesti v seznam za planiranje verzij naslednjega leta in šele nato razporedi v ustrezen produkcijski cikel (verzijo).

Ključ za hitrejši čas do razvoja podpore zamisli uporabnika je skrajšanje razvojnega cikla. Ta v osnovi sploh omogoča, da imamo možnost pripraviti informacijsko podporo neke zamisli (uporabniške zahteve) v manj kot dosedanjih pol leta. Za skrajšanje razvojnega cikla se lahko poslužujemo različnih agilnih pristopov, ki smo jih opisali v poglavju 4. Kratki razvojni cikli ter iterativen in inkrementalni razvojni proces so namreč temelj vseh agilnih metodologij. Skrajšanje razvojnega cikla pa zato zelo vpliva na obstoječi tradicionalen proces razvoja. Potrebno je spremeniti organizacijo dela v procesu razvoja ter povečati vključenost uporabnikov. Do zdaj ločene faze razvoja, kjer je vsaka vloga izvedla svoj del nalog za celoten planiran razvojni cikel, je potrebno združiti in proces razvoja spremljati ter izvajati drugače – bolj odzivno oziroma agilno. Po obstoječi metodologiji dela izvajamo in spremljamo razvoj po fazah (analiza, načrtovanje, razvoj, testiranje), po novem agilnem pristopu pa moramo spremljati razvoj po vsebinah. Vsebine (uporabniške zahteve), ki jih razvijamo so del enotnega seznama vseh zahtev urejenih po prioritetah. Enoten seznam zahtev je poleg ostalih prikazanih na sliki 5.1 eden izmed ključnih gradnikov novega procesa razvoja. Za pripravo zahtev in umestitev zahtev na enotni seznam je potrebno v fazo priprave za razvoj vključiti uporabnike, poslovne analitike in del razvojne ekipe. Ta skupina mora analizirati zahtevo in jo glede na prioritete, strateške smernice ter izračunano dodano vrednost umestiti na ustrezno mesto na seznamu. Glede na prioritete in ocenjen obseg razvoja vsake zahteve, se le-te vzame iz seznama in razporedi v ekipo, ki bo izvajala razvoj. Velika sprememba v primerjavi s tradicionalnim načinom dela je tudi stalna vključenost uporabnika, da je lahko kratek razvojni cikel čimbolj učinkovito izveden.

Ko v prehodu iz tradicionalnega načina dela na agilne načine delo dosežemo



Slika 5.1: Ključni sestavni deli novega razvojnega procesa

ustrezno zrelostno fazo, je cilj, da si ekipe same po vrsti vzamejo zahteve iz skupnega seznama. Tako se zagotovi kontinuiran način izvajanja razvojnih ciklov, kot smo ga prikazali pri metodologiji DAD na sliki 4.10.

Ekipe se morajo oblikovati heterogeno. To spet pomeni spremembo v oblikovanju ekip, ki so danes oblikovane homogeno. Slednje so sestavljene iz posameznikov z enako vlogo v procesu razvoja. Tako imamo ekipo poslovnih analitikov, načrtovalcev, razvijalcev poslovne logike ter razvijalcev uporabniškega vmesnika. Prenos znanja tako poteka prek definirane dokumentacije in premalo neposredno – z osebnimi stiki. Nov proces dela zahteva heterogene ekipe, katerih sestava je lahko stalna ali dinamična. Heterogenost v pojmovanju ekip pomeni, da je vsaka ekipa sestavljena iz posameznikov z vsemi vlogami, ki so potrebne za izvedbo razvoja informacijskega sistema. V

ekipi je nujno tudi uporabnik oziroma predstavnik uporabnika ter vse ostale vloge kot so poslovni analitik, načrtovalec, razvijalec poslovne logike ter razvijalec uporabniškega vmesnika. Pri uporabniku je potrebno poudariti, da izvaja še uporabniško testiranje, poslovni analitik pa izvaja tudi funkcionalno testiranje. Velikost stalnih heterogenih ekip je tako od 5 do 8 posameznikov, saj izkušnje pokažejo, da moramo vloge razvijalcev v ciklu pogosto podvojiti, da dosežemo cilj cikla. Včasih pa se tudi zgodi, da moramo vključiti dodatno vlogo testerja, da bolj neodvisno preverimo delovanje. Ključno pri oblikovanju ekipe je to, da je ekipa stalno skupaj, najbolje v istem prostoru. Na ta način je prenos informacij zelo neposreden in hiter. Uporabnik je prisoten ves čas ali vsaj večino časa, kar zanj pomeni večjo vključenost. To mu omogoča, da takoj poda informacije o podrobnostih zahteve in tudi spremlja razvoj podpore zahteve. Na drugi strani pa razvojni ekipi nudi vse potrebne informacije za ustrezno pripravo rešitve uporabniške zahteve. Na ta način potem lažje sledimo racionalizaciji razvoja podpore novi zahtevi po Paretovem načelu [54]. V zagotavljanju informacijske podpore namreč velja, da je priporočljivo določen del procesa pustiti ročnega (20%), saj bi bil razvoj tega dela procesa preveč kompleksen (80%) in bi zahteval preveč časa za premajhno dodano vrednost [64, str.51-53].

Kakšna pa je lahko nova dolžina razvojnega cikla? V opisanih agilnih pristopih v poglavju 4 so v glavnem v uporabi od dva do štiri tedenski razvojni cikli, nekatere metode, ki vključujejo tudi aspekt celotne organizacije pa govorijo tudi o daljših ciklih (mesečnih, kvartalnih). Različne metodologije govorijo tudi o več nivojih teh ciklov. Eno so sami kratki razvojni cikli, ki trajajo največ mesec dni in katerih rezultat je produkt primeren za produkcijo. Drugo so potem produkcijski cikli, ki združujejo več razvojnih ciklov in katerih rezultat je na koncu res prehod v produkcijo. Za prehod iz obstoječega tradicionalnega razvoja, kjer je cikel pol leta, na agilni razvoj, kjer je cikel npr. en mesec, je potrebno spremeniti več stvari. Ker je proces prehoda za cel razvoj sistema za bančno poslovanje lahko dolgotrajen, je potrebno tega najprej aplicirati na delu razvoja. Nato je potrebno preve-

riti ustreznost sprememb (angl. proof of concept - POC) pred nadaljnjimi transformacijami in širjenjem novega načina dela na celoten razvoj sistema za bančno poslovanje. Ko pa se celoten razvoj tega sistema transformira na nov način dela, se šele odločimo o širitvi na celoten razvoj v banki [7].

Sam za spremembo na razvoju sistema za bančno poslovanje predlagam tritedenske razvojne cikle. V nadaljevanju bom povedal več zakaj takšen cikel. Ključna razlika glede na obstoječi način dela je, da morajo biti v novem razvojnem ciklu (treh tednih) za funkcionalnosti, ki se v tem ciklu razvijajo, izvedene vse "faze" (tradicionalno) razvoja: od analize zahteve, načrta izvedbe, do programiranja in testiranja. Na koncu cikla mora namreč podpora funkcionalnosti biti 100% (oziroma tako kot v starem ciklu, saj se vedno dopušča tudi možnost programskih hroščev) delujoča in pripravljena za prenos v produkcijsko okolje. Zakaj pripravljena (zrela)? Ker je po agilnih metodologijah praksa, da je rezultat razvojnega cikla kandidat za produkcijo (angl. release candidate). To pomeni, da gre lahko v produkcijsko okolje, lahko pa počaka kandidata naslednjega cikla ali več ciklov in gredo v produkcijsko delovanje skupaj.

Da lahko v novem, kratkem razvojnem ciklu, res zagotovimo ustrezen razvoj je nujno, da tradicionalne faze razvoja potekajo vzporedno. Na začetku razvojnega cikla celotna ekipa (vsi posamezniki, vse vloge) analizira uporabniško/e zahtevo/e, ki jo bodo v tem ciklu razvijali. Ko zahtevo celotna ekipa razume enako, se vsak loti svojega dela:

- poslovni analitik izvede podrobno poslovno analizo in pripravi primer uporabe,
- načrtovalec pripravi načrt rešitve z vsemi specifikacijami,
- razvijalci začnejo programirati.

Ker je ekipa stalno skupaj, se vprašanja in eventualne nejasnosti ali neznanke, rešujejo sproti. Nek del dokumentacije vsaj v prvih fazah prehoda mora ostati, je pa potrebno obseg dokumentacije zmanjšati, združiti v skupne dokumente in si po možnosti pomagati s CASE orodji, ki omogočajo

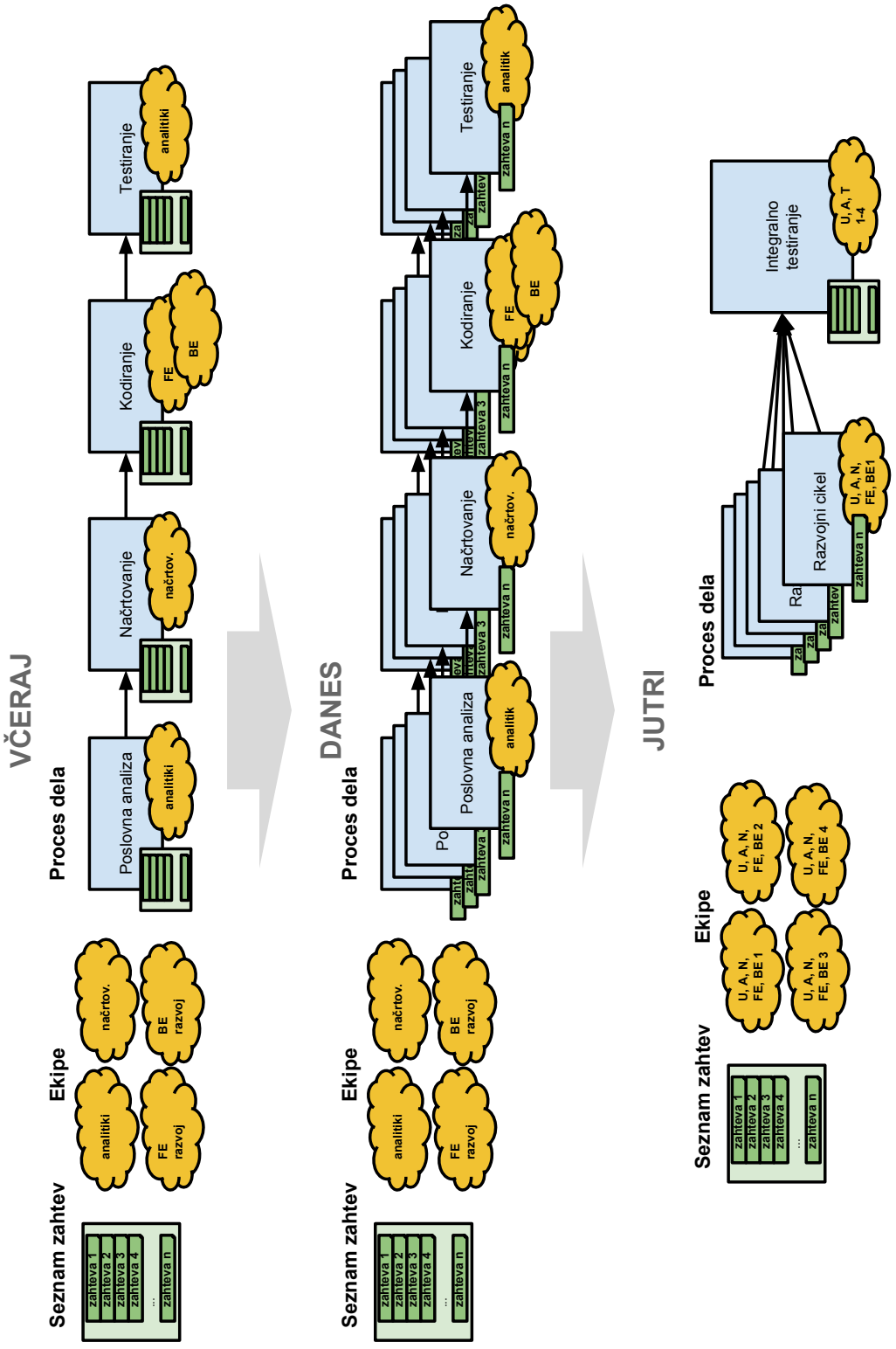
kontinuiran prehod od dokumentacije do programske kode. Namen primerov uporabe in tehničnih specifikacij je predvsem dokumentiranost za testiranje ustreznosti delovanja podpore in kasnejše vzdrževanje sistema.

Za prehod iz tradicionalnega načina na nov način zato predlagam postopen prehod kot je prikazan na sliki 5.2. Ideja je, da polletni obstoječi produkcijski cikel razdelimo najprej na nove tritedenske cikle, s čimer dobimo pet do sedem kandidatov za produkcijo, ki jih potem združimo v hkratni prehod v produkcijsko delovanje. Na ta način se postopoma pripravljamo na nov način dela in sproti odpravljamo potencialne nedorečenosti. Tako v prvi iteraciji novega razvojnega procesa še vedno ohranimo eno verzijo na pol leta. V naslednji ponovitvi bi lahko produkcijski cikel bil na tri mesece (kvartal). Za tem pa bi sledila iteracija, ki pomeni ciljno stanje, da je vsak kandidat za produkcijo (na koncu tritedenskega razvojnega cikla) tudi dejansko prenesen v produkcijsko delovanje.

5.2 Kako zagotoviti obseg razvoja in ohraniti kakovost sistema?

Za samo omogočanje krajšega razvojnega cikla so potrebne velike spremembe v razvoju. Enako pa zahteva tudi zmanjšanje kapacitet razvoja, kjer imamo v mislih predvsem zmanjšanje kadrov.

Ključno za to je, da je enotni seznam zahtev za razvoj res urejen po pravilnem vrstnem redu in skladen s strategijo in ostalimi poslovnimi smernicami banke. Na enem mestu morajo biti namreč prioritetno urejene zahteve uporabnikov iz vseh področij. Edino na ta način lahko zagotovimo, da bo razvoj informacijske podpore res usmerjen v pravo smer in bomo razvijali podporo za zahteve z najvišjo dodano vrednostjo. Tako je za vsako uporabniško zahtevo ali strateško smernico potrebno za umestitev v enotni seznam zahtev narediti analizo upravičenosti (angl. cost-benefit analysis). Rezultat te analize potem določa, ali zahteva sploh prinaša dodano vrednost in se uvrsti na enotni seznam. V primeru, da je analiza upravičenosti ustrezna, ta določa



Slika 5.2: Shema predlaganega prehoda k agilnem procesu razvoja

tudi prioriteto in ustrezno mesto na enotnem seznamu zahtev. Na ta način bomo na krovnem nivoju sledili principu izločanja nepotrebnega (angl. eliminate waste) in ne bomo razvijali nečesa na zalogo ali nečesa s prenizko dodano vrednostjo.

V samih tehnikah razvoja in testiranja so tudi možnosti izboljšav in avtomatizacije. Že za skrajšanje razvojnega cikla je ena od osnov tudi avtomatizacija testiranja. Tu govorimo predvsem o regresijskem testiranju, ki zagotavlja, da nov razvoj ni imel negativnega vpliva na obstoječe delovanje. Z avtomatskim testiranjem na avtomatiziran način preverimo delovanje obstoječih funkcionalnosti. Za ta namen imamo za vse funkcionalnosti posnete testne scenarije, ki jih samo izvedemo pred pripravo nadgradnje sistema. Rezultate testa se potem pregleda in v primeru uspešnosti, se pripravi nadgradnja sistema. Ta avtomatizacija prihrani veliko časa:

- na eni strani uporabnikom ali analitikom pri testiranju ni potrebno ročno preverjati še delovanja vseh starih funkcionalnosti - v odstotkih to pomeni kar 15-30% prihranka na času testiranja,

- na drugi strani je razvojna ekipa ob nadgradnji sistema lahko zelo mirna, ker ve, da so bili uspešno izvedeni avtomatski testi in je verjetnost napak na obstoječih funkcionalnostih zelo majhna,

- v celotnem razvojnem ciklu se tako več ne ukvarjamo z odpravljanjem napak, ki so posledica prepletenosti programske kode (ko sprememba ne enem delu kode lahko pokvari obstoječe delovanje), temveč vse moči usmerjamo v razvoj in testiranje novih funkcionalnosti. Glede na izkušnje dela na projektu za razvoj sistema za bančno poslovanje, se na ta način prihrani vsaj 10% časa pri odpravljanju napak, ki so nastale zaradi soodvisnosti (prepletenosti) programske logike.

Glede na težnje po manjšanju razvojne ekipe je nujno avtomatizirati tudi postopke priprave programskih paketov nadgradenj in distribucije teh paketov med okolji. Za ta del, ki je delno povezan tudi z avtomatiziranim testiranjem obstajajo moderni pristopi, kot je npr. DevOps [62], ki med drugim

uvajajo orodja za avtomatizacijo postopkov priprave programskih paketov in distribucijo paketov med okolji. Teh ponovitev je v novo predlaganih hitrih in kratkih razvojnih ciklih še več kot v starem procesu razvoja, zato prinašajo še večji prihranek časa kot ročna priprava paketov programske opreme.

Za ustrezno spremljanje novega načina razvoja je zelo priporočljiva uporaba orodja za spremljanje razvoja po zahtevah. Ker splošno orodje za spremljanje zahtev v banki že uporabljamo, lahko s prilagoditvijo načina dela omogočimo tudi spremljanje novega procesa razvoja. Ključna sprememba - izvajanje ter spremljanje razvoja po vsebinah ter vzporedno delo vseh vlog znotraj posamezne ekipe - prinese v razvoj več neposredne komunikacije in zmanjšuje potrebo po formalni dokumentaciji, ki je temelj tradicionalnih pristopov razvoja. Zato je potrebno narediti pregled in optimizacijo zahtevane dokumentacije. Pregledati je potrebno katere dokumente zares potrebujemo in izločiti nepotrebne ter mogoče določene vrste informacij oziroma obstoječih več dokumentov združiti v enega. Kot primer optimizacije oziroma zmanjšanje števila dokumentov navajam ukinitev sekvenčnih diagramov. Ključno pri sekvenčnih diagramih je zaporedje klicev zalednih sistemov iz uporabniškega vmesnika. Samo klicanje iz nivoja integracijskega vmesnika do podsistemov pa je že del specifikacije vmesnika, zato bi lahko dokument sekvenčnega diagrama opustili. Morali pa bi klice zalednega sistema prikazati na ekranski sliki, ki je priloga primera uporabe (PU).

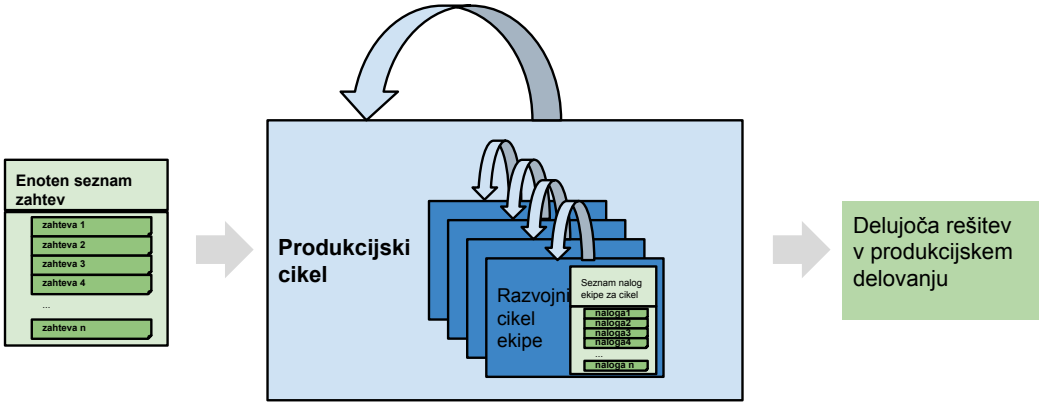
Še en korak naprej v avtomatizaciji dela lahko prinese uporaba ogrodja (angl. framework), kjer vsi sodelujoči v razvoju delajo v orodjih tega ogrodja. Takšen primer je danes zelo popularno ogrodje BPM (angl. Business Process Management) [31, 48]. To ogrodje prinaša novo paradigmo razvoja in omogoča sledljivost od poslovnega procesa do informacijske podpore. Poslovni analitik v orodje najprej nariše poslovni proces nove uporabniške zahteve na nekaj najvišjih nivojih. Tega potem načrtovalec razdeli še nekaj nivojev v globino, tako da identificira vse potrebne storitve za izvedbo aktivnosti procesa. Identificirane storitve pa potem razvijalci razvijejo in dajo na voljo v ogrodje za nadaljnjo uporabo. Ko je proces na najnižjem nivoju im-

plementiran, ga lahko preverimo na vseh višjih nivojih in tudi že preskusimo delovanje v uporabniškem vmesniku, ki je del ogrodja. Kar je ključno pri vseh modernih ogrodjih je to, da omogočajo enostavno postavljanje metrik (angl. KPI – Key Performance Indicator) na katerem koli nivoju procesa. Kar je še dodatno, že orodje samo analizira delovanje procesa in identificira ozka grla ter predlaga izboljšave procesa. Z dinamičnim postavljanjem metrik v proces lahko zelo enostavno spremljamo delovanje na tehničnem nivoju, uporabniškem nivoju kot tudi na nivoju vodstva (managementa) ter ga prav tako na vseh naštetih nivojih enostavno izboljšujemo.

Pri uvajanju vseh novih modernih metodologij in orodij v razvoj je potrebno biti previden. Da organizacija oziroma proces razvoja preide do zrelostne faze, mora izvajanje določene nove metodologije teči nekaj časa. Uvajanje vsake spremembe v proces namreč vzame svoj čas. Ponavadi se dodana vrednost vpeljave novega procesa pokaže šele po nekaj ponovitvah ter prilagoditvah lastnim potrebam. Ne smemo se niti vseh sprememb lotiti hkrati, ampak mogoče predlagane spremembe izvajati v različnih ekipah in potem združevati izboljšave. Prav za proces uvajanja modernih konceptov dela v razvoj je zato smiselno uporabiti agilne in vitke pristope. Na ta način se za uvedbo vsake spremembe najprej oceni dodano vrednost spremembe (naredimo analizo upravičenosti). Na koncu implementacije pa učinke sprememb zmerimo ter primerjamo z začetno oceno upravičenosti. Če so učinki pozitivni, se lotimo naslednjih korakov do ciljnega stanja, v nasprotnem primeru pa preverimo zakaj učinki niso pravi. Ali smo pri uvedbi kaj zgrešili, ali nismo pripravljeni na spremembo, ali proces za naše delo ni primeren, ali pa smo se spremembe lotili na napačen način?

5.3 Predlagani novi razvojni proces

V prvi fazi uvedbe novega načina dela za razvoj sistema za bančno poslovanje smo se odločili, da produkcijska cikla – nova verzija sistema v produkcijsko okolje – ostaneta enaka in sicer dve verziji na leto. Tako bomo v prehodni fazi



Slika 5.3: Produkcijski in razvojni cikel

še vedno razlikovali med produkcijskim in razvojnim ciklom kot je prikazano na 5.3. Velika sprememba pa je, da se ne izvaja več faza planiranja kot jo poznamo danes. Vzporedno z razvojnim procesom namreč poteka proces priprave in analize uporabniških zahtev ter umeščanje zahtev na prioritetno urejen seznam zahtev. Tako se za razvojni proces zahteve enostavno vzamejo po vrsti iz tega seznama.

Da pa bi v okviru planiranega polletnega produkcijskega cikla kar najbolj optimizirali delo in naredili čim več novih funkcionalnosti in prinesli največjo dodano vrednost končnim uporabnikom, smo se odločili za spremembo dela znotraj cikla z agilnimi in vitkimi principi. V naslednjih uvedbah agilnih pristopov v razvojni proces pa predlagamo, da se tudi prehod v produkcijo lahko izvede večkrat na leto – mogoče kvartalno ali najbolje mesečno.

Produkcijski cikel vsake verzije sistema za bančno poslovanje traja od štiri do pet mesecev. Odločili smo se za naslednje spremembe, ki tudi nakazujejo postopen prehod iz klasičnih razvojnih tehnik v moderni agilni razvoj.

ORGANIZACIJA RAZVOJNE EKIPE

1. Vse razvojne kadre, ki delajo na sistemu za bančno poslovanje smo razdelili v štiri heterogene skupine - ekipe. Stalni člani vsake skupine

so:

- poslovni analitik (1),
- načrtovalec (1),
- BE razvijalec (1),
- FE razvijalec (1).

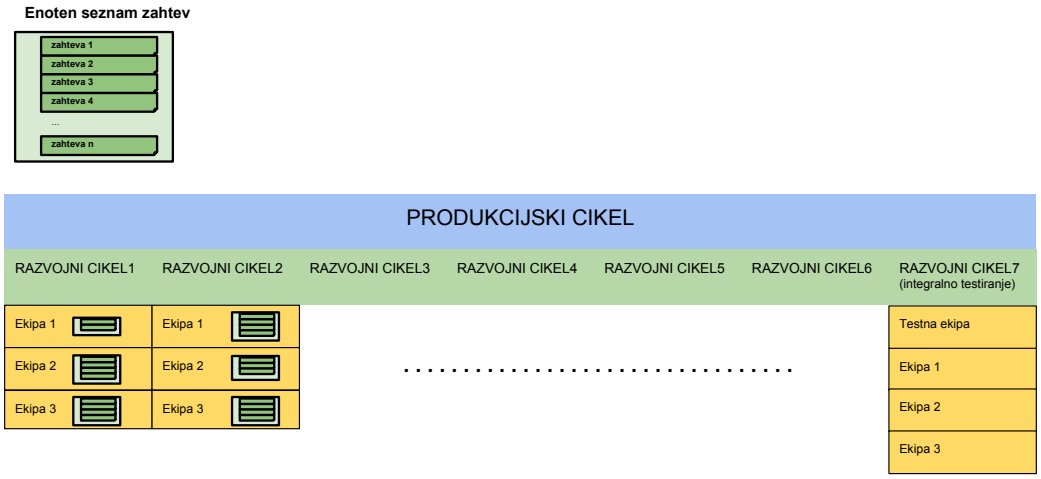
Poleg teh so del ekipe tudi vsi uporabniki oziroma njeni predstavniki. Prisotnost je odvisna od vsebin, ki se izvajajo v vsakem razvojnem ciklu.

2. Razvojni cikel traja tri tedne. Na ta način v okviru produkcijskega cikla dobimo pet do sedem razvojnih ciklov.

3. Definirali smo posebno ekipo za vzdrževanje sistema. Ti rešujejo reklamacije uporabnikov in pripravljajo razna ročno pripravljena poročila, enkratne poizvedbe ali posebne nepodprte enkratne dogodke.

4. Vodja projekta oziroma koordinator razvoja sistema za bančno poslovanje skrbi za usklajevanje dela med ekipami tako kot v starem načinu dela, skrbi za ustrezno izvajanje razvojnih ciklov ter izpolnjevanje časovnih okvirov definiranih ciklov.

5. Tako kot v starem načinu dela, tudi v predlaganem novem načinu, koordinator razvoja stalno sodeluje z vodji posameznih ekip. Ideja je, da se vsaka ekipa samoorganizira, kar pomeni, da se vodja ekipe avtomatsko določi izmed članov ekipe na podlagi obnašanja (angl. “drive”) v ekipi. Ekipa sama vidi, kdo ima najboljše koordinacijske, komunikacijske ter vodstvene sposobnosti in ta postane vodja. Ključno pri tem je, da vodja ni določen vnaprej ali od zunaj, ampak da ga ekipa izbere sama med svojimi člani. S tem si pridobi veliko zaupanje ekipe.



Slika 5.4: Predlagani nov proces dela in razporeditev dela po ekipah

RAZPOREDITEV DELA MED EKIPAMI

1. Zahteve iz skupnega seznama zahtev, ki pridejo na vrsto za produkcijski cikel, se ne razdelijo vnaprej, ampak jih ekipe po vrsti jemljejo iz seznama kot je prikazano na sliki 5.4. Prioritete oziroma vrstni red seznama zahtev določa tudi zaporedje dela posamezne ekipe. Tak postopek predlagam za začetne faze prehoda iz klasičnega v agilnejši način dela. V naslednjih fazah prehoda v agilni razvoj pa se na produkcijski cikel sploh ne oziramo več, ampak ekipe stalno izvajajo samo razvojne cikle in zahteve pobirajo iz skupnega seznama.

2. Uporabniške zahteve iz skupnega seznama so lahko preobsežne za en sam razvojni cikel. Zato jih je potrebno razbiti na posamezne dele, rečemo jim naloge (angl. tasks) na nivoju razvojnega cikla. Naloge morajo biti tako velike, da so lahko kot samostojna zaključena celota in se tudi samostojno vključijo v dopolnitev funkcionalnosti sistema.

3. Vsaka ekipa tako zahteve znotraj produkcijskega cikla razbije po razvojnih ciklih na obvladljive naloge, ki jih postavi v ustrezni vrstni red razvojnih ciklov.

PROCES

1. V začetnih fazah prehoda na agilni in vitki način dela predlagam, da je prvi teden vsakega razvojnega cikla bolj analitično, načrtovalsko in vsebinsko naravnan. Namen tega je, da se poslovne zahteve natančno analizira in razbije na naloge, ki jih je potrebno realizirati.

2. Ko se vsi člani ekipe podrobno seznanijo z vsebino in enako razumejo kaj je za narediti, se delo razdeli na posamezne vloge v ekipi:

   - poslovni analitik napiše poslovne spremembe oziroma primere uporabe (v nadaljevanju PU),
   - načrtovalec pripravi tehnični koncept rešitve, spremembe na podatkovni bazi in specifikacije programskih vmesnikov,
   - razvijalci (BE in FE) razvijajo programsko kodo.

3. Pri vzporednem delu več različnih vlog znotraj ekipe je pomembno, da med seboj stalno komunicirajo in se usklajujejo o morebitnih težavah. Zaradi tega je pomembno, da so locirani blizu skupaj, po možnosti v isti sobi, da lahko neovirano neposredno komunicirajo.

4. Kaj bo posamezna ekipa naredila v sklopu enega razvojnega cikla se določi na podlagi izkušenj preteklega razvoja in na osnovi ocen potrebnega dela, ki v prvem tednu cikla nastanejo. Zato je obvezno, da do konca drugega tedna razvojnega cikla zaključimo s pripravo PU in načrtovalskih dokumentov. V tretjem tednu tako poteka samo še kodiranje in vzporedno funkcionalno testiranje.

5. Odkrite napake testiranja se uvrščajo na seznam nalog v razvojnem ciklu glede na prioriteto in se sproti odpravljajo.

6. Na koncu vsakega razvojnega cikla preverimo, ali smo vse kar smo si zadali tudi realizirali in če je zahtevana funkcionalnost v delu definiranem za razvojni cikel, ustrezno implementirana in delujoča (potrjen test).

7. Glede na rezultate izvedbe na koncu razvojnega cikla, se potem določi oziroma korigira vsebina naslednjega razvojnega cikla:

   - če prejšnji razvojni cikel ni realiziral vsega, se odprte zadeve uvrstijo na seznam novega razvojnega cikla in se z ostalimi zahtevami za ta cikel iz skupnega seznama razporedijo glede na prioritete,
   - če smo prejšnji razvojni cikel v popolnosti izvedli, potem se v nov razvojni cikel umesti naslednje planirane zahteve iz skupnega seznama.

8. Zadnji razvojni cikel pred prehodom v produkcijo je v glavnini namenjen uporabniškem in integralnem testiranju. Na začetku tega cikla morajo vse zahteve planirane za produkcijski cikel že biti funkcionalno delujoče in testirane, saj le na ta način lahko zagotovimo, da bomo do zaključka zadnjega razvojnega cikla (planiranega datuma prehoda v produkcijo) zagotovili ustrezno delovanje nadgrajenega sistema za bančno poslovanje.

Pri novem načinu dela pa se vseeno pojavijo težave in možnosti izboljšav. Predvsem jih je potrebno odpravljati postopoma in na podlagi izkušenj, saj bomo le na ta način prišli do optimalnega procesa. Glavne zaznane težave oziroma ključne dileme so naslednje:

1. Ali lahko definiramo stalne ekipe, glede na to da posamezniki trenutno pokrivajo več vsebin in so za določeno vsebino v različnih vlogah to različne osebe (npr. FE razvijalec 1 je specialist za kredite in CRM, načrtovalec 1 je specialist za kredite in finančne transakcije, BE razvijalec 1 pa je specialist za finančne transakcije in osebne račune. Zahteva v produkcijskem ciklu se nanaša na kredite in za to se oblikuje ekipa kjer so FE razvijalec 1, načrtovalec 1 in BE razvijalec 2). Zaradi takšnih težav predlagam, da do oblikovanja stalnih ekip pridemo postopoma:

   - tako, da bomo do stalnih ekip prišli v več iteracijah novega načina

dela in uskladili znanja ljudi znotraj ekipe na ista vsebinska področja,

- v prvih fazah se bodo ekipe oblikovale bolj "plavajoče". Na ta način bodo ekipe v enem razvojnem ciklu v eni sestavi, v drugem razvojnem ciklu pa v drugi sestavi.

2. Ali res lahko zagotovimo, da bomo pred zadnjim razvojnim ciklom imeli vse razvito in funkcionalno testirano?

   - Zagotoviti tega ne moremo, ampak za razliko od prejšnjega slapovnega načina lahko to trdimo in zagotovimo z veliko večjo verjetnostjo. Zakaj? Ker takoj ko je funkcionalnost razvita, že začnemo testirati in se ne lotimo nove vsebine, dokler ta ni ustrezno delujoča. S pravim testom ne čakamo do zadnje faze v slapovnem načinu (testiranje), ko se nam je pogosto dogajalo, da smo potem še dodajali marginalne funkcionalnosti, ki so podaljšale razvoj in jih na koncu nismo bili zmožni testirati.

5.4 Odprava identificiranih glavnih težav v novem procesu

Z opisanim novim procesom razvoja lahko hitro odpravimo glavne pomanjkljivosti oziroma možnosti izboljšav v obstoječem načinu dela, ki smo jih identificirali v poglavju 3.4.

Prvo težavo pregrobo definiranih uporabniških zahtev enostavno rešujemo že z uvedbo skupnega seznama zahtev. Za umestitev na ta seznam mora zahteva namreč že biti ustrezno analizirana in ocenjena z analizo upravičenosti, ki zajema prihranke uvedbe in strošek implementacije. Poleg tega bomo uporabili še ostale principe vitke (angl. lean) metodologije. Ti nam med drugim narekujejo, da ne delamo stvari na zalogo (angl. eliminate waste), ker bo ta razvoj, v času ko bo postal aktualen, že v neskladju z novimi zahtevami. Drug, zelo pomemben vitki princip pa je, da ne poskušamo najprej vse v

podrobnosti definirati, potem pa to šele razvijati (angl. decide as late as possible), ampak poskušamo zelo hitro uporabniku (ki je postavil zahtevo) zagotoviti nek prototip (angl. deliver as fast as possible). S tem bo dobil občutek kako bo podpora narejena, kar mu bo omogočalo lažje razmišljanje o naslednjih nivojih podrobnosti zahteve.

Drugo težavo zahtev, ki se širijo in spreminjajo, **tretjo težavo** prenosa informacij in **četrto težavo** neizpolnjevanja mejnikov, nam rešujejo kratki razvojni cikli ter oblikovane heterogenih ekip. Dejstvo, da z oblikovanjem heterogenih ekip vse vloge v procesu razvoja sedijo skupaj pomeni, da je prenos informacij neposreden ter da se mnenja vseh vlog že na začetku uskladijo in upoštevajo. Tako ne more priti do konceptualne spremembe, ko že programiramo rešitev, saj so že v analizo in načrtovanje vključeni vsi tehnični vidiki razvoja.

Peto težavo skrajševanja časa za programiranje in testiranje prav tako rešujemo z novim načinom dela. Vsako uporabniško zahtevo namreč razbijemo na obvladljive naloge, ki jih je tudi lažje razviti in preveriti ustreznost delovanja. Ker se funkcionalno testiranje izvaja že sproti, znotraj posameznega razvojnega cikla, to pomeni, da povratno informacijo o uspešnosti testa dobimo takoj in ne šele v posebni fazi testiranja kot v starem načinu dela.

Šesta težava z ustreznim prioritiziranjem zahtev je prav tako rešena že z vodenjem skupnega seznama zahtev ter umeščanjem novih zahtev na ta seznam. Tako že v času analize zahteve in umeščanja na seznam uskladimo vse interese različnih uporabnikov, skladnost s strateškimi cilji banke in tako zagotovimo, da so na seznamu res “prave” zahteve za razvoj.

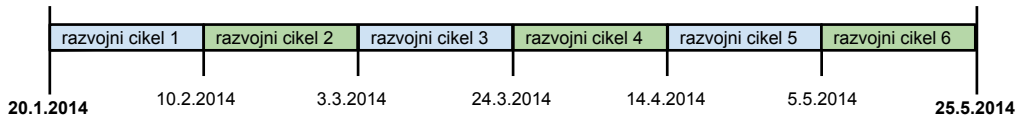
Poglavje 6

Izvedba novega procesa razvoja na delu sistema

Predlagani nov proces razvoja, opisan v poglavju 5, smo skušali že delno implementirati na delu razvoja sistema za bančno poslovanje. Proti koncu leta 2013, ko so bile vsebine za obe verziji leta 2014 že definirane, sem vizijo o spremenjenem procesu predstavil vodstvu razvoja. Moj predlog je bil, da bi eno izmed večjih vsebin pomladanske verzije, na kateri bi tudi sam sodeloval kot načrtovalec in koordinator, izvajali po novo predlaganem procesu dela. Zaradi prevelike vpetosti v sam razvoj in vzdrževanje ter časovne stiske ter zamika v razvoju jesenske verzije 2013, se uradno nismo uspeli dogovoriti o novem načinu dela. Zato sem se tega na delu razvoja lotil samostojno in tako želel preizkusiti primernost novo predlaganega procesa dela.

Takoj na začetku lahko že identificiram prvi izziv pri uvajanju novih metodologij v takšno organizacijo kot je banka. Da se v obstoječih postopkih dela res nekaj lahko začne spreminjati se lahko zelo hitro zgodi, da preteče leto dni ali več:

- najprej je potrebno idejo predstaviti na vseh nivojih vodstva in zaposlenih v razvoju, s čimer se pri vseh vzbudi interes,
- potem je potrebno pri vseh vzpostaviti razumevanje in pridobiti vsaj delno podporo,



Slika 6.1: Produkcijski cikel pomlad 2014

- nato se na delu razvojne ekipe pripravi in uskladi nov proces dela ter na realnem primeru izvede potrditev novega koncepta (angl. proof of concept – POC),

- po pričakovano uspešni potrditvi koncepta sledi postopno širjenje, ki ponavadi traja dlje časa, odvisno od podpore, vključenosti in interesa vodstva.

Del predlaganega procesa dela smo tako preizkusili pri razvoju podpore za nove vrste gotovinskih blagajn – avtomatskih blagajnah. Za dodatno informacijo nja dodam, da smo po starem načinu dela implementacijo te podpore ocenili na obseg razvoja štirih posameznikov (poslovni analitik, načrtovalec, FE razvijalec, BE razvijalec) za celotno pomladansko verzijo. Tudi razvoja za to funkcionalnost po novem procesu smo se zato lotili s takšno ekipo, ki smo ji dodali še predstavnika uporabnikov. Tako smo 20.1.2014 začeli z delom vsaj delno na nov način. Sam sem obdobje od 20.1.2014 in do planiranega prehoda v produkcijsko delovanje razdelil na tritedenske cikle na sliki 6.1.

Tako smo 20.1.2014 začeli s prvim ciklom razvoja podpore delu z avtomatskimi blagajnami. Iz ene poslovne zahteve smo v uvodnih delavnicah članov ekipe, ki smo jo razširili na dva posameznika v vsaki vlogi zaradi prenosa informacij, identificirali enajst nalog. Teh enajst nalog smo potem postavili v urejen seznam in jih okvirno razdelili v tri razvojne cikle.

Za prvi razvojni cikel smo se odločili za implementacijo treh nalog, ki so morale biti narejene za postavitev temeljev za nadaljnji razvoj podpore. Ker je šlo za povsem novo področje dela in prilagoditev delovanja obstoječih

temeljnih konceptov seje za stranko in gotovinskega poslovanja, smo uspeli na koncu v prvem ciklu implementirati le dve nalogi. Vzporedno je potekal še neodvisen razvoj komunikacijskega modula (tretja naloga), ki se je delno končal s postavitvijo osnovnih funkcionalnosti na koncu drugega cikla. Do zamika oziroma zamude v prvem ciklu je prišlo zaradi več usklajevanja in dogovarjanja o konceptih implementacije podpore. Ke se ekipa dejansko ni 100% ukvarjala le s tem razvojem (kar bi po vseh agilnih metodologijah in tudi predlaganem novem procesu bilo pravilno), je prišlo do zamude. Ekipa tudi ni bila skupaj v istem prostoru in se je sestajala na delavnicah, kar spet ni povsem v skladu s predlaganim novim procesom.

V drugem razvojnem ciklu smo se tako lotili četrte naloge in poleg te implementirali še planirani nalogi pet in šest. Potem pa smo vse ostale naloge implementirali v tretjem razvojnem ciklu, v katerem smo izvedli več kot smo prvotno ocenili.

Tako smo na koncu tretjega razvojnega cikla že lahko funkcionalno testirali delovanje nove podpore. Potem smo potrebovali še cel cikel za testiranje in odpravo pomanjkljivosti in bili pripravljeni na uporabniško testiranje. Tega smo izvedli v maju in potem naredili prehod v produkcijsko delovanje podpore v juniju 2014. Do zamika v mesec junij je prišlo zaradi neke druge nepopolne funkcionalnosti, kar je zahtevalo prestavitev prenosa v produkcijsko delovanje celotne pomladanske verzije sistema za bančno poslovanje.

Iz zgoraj opisanega se lahko jasno razbere, da je bil prvi poizkus izvajanja novo predlaganega izboljšanega procesa razvoja zelo improviziran. Zato je tudi zelo težko iz njega izvesti kakršnekoli analize in sprejeti usmeritve za naprej. Kar lahko rečem je, da smo se trudili in na koncu implementacije tudi ugotovili, da smo porabili znatno manj časa od planiranega. Namesto planirane izvedbe po starem procesu v 400 človek/dneh, smo to naredili v 260 človek/dneh, kar je 35% prihranek. Ali je to posledica drugačnega načina dela, ali samo slaba začetna ocena, je zaradi improviziranega dela zelo težko oceniti.

Prav zato predlagam za vnaprej, da se novi proces dela res najprej uskladi

z vsemi potrebnimi vpletenimi in se šele ob podpori in konsenzu vseh, ta res začne izvajati.

Sem se pa že iz samih pogovorov, predstavitev in razmišljanj z drugimi člani razvojne ekipe sistema za bančno poslovanje, dokopal do nekaj ugotovitev, ki jih je nujno potrebno upoštevati v nadaljnjih implementacijah novega procesa. Tekom improviziranega prvega poskusa pa se je prav tako pokazalo še nekaj pomanjkljivosti, pomislekov in idej.

6.1 Zaznane pomanjkljivosti pri prehodu na nov proces dela

Najprej naj povem, da so me odzivi ob pogovorih o določenih spremembah načina dela zelo presenetili. Predvsem so me presenetili obrambni mehanizmi v smislu, zakaj bi nekaj v svojem delu spreminjali, če pa svoje delo opravljamo kar dobro. Ugotovil sem, da je ključno pri uvajanju sprememb to, da:

- se spremembe uvajajo postopoma ena za drugo,
- imamo pri tem podporo vodstva na vseh ravneh,
- dosežemo najprej široko poznavanje in razumevanje vsebine spremembe,
- jih uvajamo s podporniki, ki jih bodo dejansko izvajali, ker jo bodo na ta način njihovi sodelavci lažje sprejeli.

Napisane trditve bom pojasnil na primerih, ki sem jih doživel pri pogovorih in promociji spremenjenega razvojnega procesa na razvoju sistema za bančno poslovanje.

Za uvedbo agilnih pristopov v proces razvoja je potrebno spremeniti več stvari. Ker gre za drugačen proces dela, predvsem pri spremljanju izvajanja, je potrebno spremeniti tudi ustaljeno miselnost. To pomeni, da se z roki zaključka posameznih faz po tradicionalnem slapovnem pristopu ne smemo več obremenjevati. Ko sem dobil podporo vodstva razvoja sistema za bančno

poslovanje za nov razvojni proces in smo ga že improvizirano izvajali, so se še vedno pojavljala vprašanja in trditve v smislu:

- "Ali je poslovna analiza že zaključila PU-je?",
- "ok za zaključek poslovne analize je že mimo, a še nimamo PU-ja!".

Ta pogled je potrebno spremeniti in se osredotočiti le na to:

- katere zahteve iz skupnega seznama zahtev so v delu in
- kje (časovno) v trenutnem razvojnem ciklu se nahajamo in kakšen je status zahtev v delu.

Preverja in spremlja se lahko potem še :

- v koliko razvojnih ciklih bo posamezna uporabniška zahteva zaključena in
- se sproti preverja zadovoljstvo uporabnika s pripravljeno podporo znotraj in na koncu vsakega razvojnega cikla.

Zato je v novem pristopu ključno, da se najprej pripravi prioritetno urejen seznam zahtev za sistem za bančno poslovanje. Ko se razvoj začne, se jemljejo zahteve po vrsti iz seznama in se do konca razvojnega cikla razvojne ekipe ne prekinja. Če vzporedno pride do spremembe vrstnega reda na seznamu zahtev ali se pojavijo nove nujne zahteve, se te najprej uvrstijo na enotni seznam. Če so prioritete ustrezne, se pojavijo na vrhu seznama. Po koncu trenutnega razvojnega cikla se potem razvojna ekipa v novem razvojnem ciklu spet loti po vrsti zadev iz vrha seznama. Ko sta proces vzdrževanja enotnega seznama zahtev (z analizo in umeščanjem novih zahtev na pravo mesto) in proces razvoja pravilno usklajena, se lahko lotimo naslednjih predlaganih sprememb, kot so ustanovitev stalnih heterogenih ekip in optimizacije dokumentacije.

Na drug primer negativnega odziva sem naletel pri predlogih in razmišljanju o agilnem razvoju in avtomatiziranem testiranju znotraj razvojnih ciklov. Agilni razvoj v osnovi pravi, da uporabniku najprej predstavimo ogrodje

(prvi prototip), da dobi občutek kako bo podpora narejena. Potem to prvotno ogrodje izboljšujemo in usklajujemo z uporabnikom do končne rešitve, ki je usklajena s strani razvijalcev in uporabnikov in tako predstavlja optimalno rešitev za vse vpletene. Ko sem predlagal ta način dela, sem dobil odgovor, da to ni smiselno in optimalno glede na današnji način dela. Po njem poslovna analiza in načrtovalci pripravijo vso dokumentacijo, potem razvijalci to razvijejo in ponudijo uporabniku v testiranje. Če se v testiranju pojavi konceptualni problem, je razlog v poslovni analizi in načrtu rešitve, kar po mojem mnenju ni pravilno. Če želimo doseči optimalen proces razvoja in pri tem skrbeti tudi za zdravo delovno klimo, je pomembno, da se vseh težav lotimo skupaj vsi člani razvojne ekipe. Prav tako se razvijalcem ne zdi sporno, da vsako verzijo nadgrajenega sistema uporabniki testirajo najmanj mesec dni. Ko sem iz obeh naslovov (konceptualnih napak in dolgega testiranja) omenil pisanje testnih programov vzporedno s kodiranjem funkcionalnosti za avtomatsko testiranje, sem zopet naletel na negativen odziv. Dejstvo namreč je, da v prihodnje pri razvoju sistema za bančno poslovanje ali kateregakoli drugega razvoja, ne bo mogoče pridobiti velike testne ekipe, ki bo lahko dlje časa izvajala testiranje. Prav zato se bo s kvaliteto pripravljene programske rešitve v nadaljevanju potrebno še veliko ukvarjati. Potrebno se bo poslužiti npr. principov programiranja v parih iz ekstremnega programiranja, ki je na prvi pogled počasen. Ampak, če se pogleda celoten razvojni cikel na obstoječi način, temu prišteje čas testiranja in odpravljanja napak ter to primerja z načinom programiranja v parih, se ugotovi, da je slednji krajši in zato hitrejši [20]. To pomeni, da se bo tudi sprememb v v aktivnostih programiranja potrebno lotiti.

V luči optimiziranja načina dela so iterativni in inkrementalni razvoj ter avtomatsko testiranje ključni. Res da pisanje testnih programov vzame nekaj dodatnega časa, vendar na koncu prihrani veliko količino napornega testiranja in ravno razvijalcem zagotavlja “miren spanec”. Ko pripravijo nadgradnjo, namreč izvedejo avtomatske teste in če ti ne javijo nobene napake, je nadgradnja pripravljena za uporabnike in jih tako ne skrbi kakšne napake se

lahko pojavijo ob začetku testiranja.

Že pri pripravi predloga izboljšanega razvojnega procesa sem zapisal, da je oblikovanje stalnih heterogenih ekip lahko težavno in se ne bo moralo vzpostaviti takoj. Predvsem je to povezano z naslova preteklega dela, ker imajo posamezniki v različnih vlogah različna področja dela, ki so ga do zdaj opravljali. Sam sem pri poskusu novega načina deloval v navidezni ekipi, v katero smo posameznike z znanji, ki smo jih za razvoj potrebovali, vključevali sproti po potrebi. Tisti glavni del ekipe, ki je izvajal razvoj pa je bil stalni. Pri oblikovanju ekip v prihodnje bo prav zato potrebnih več iteracij, da do ekip, ki bodo izvajale res 100% svojega časa samo en razvojni cikel za en produkt, na koncu res pridemo. Prav tako je pri oblikovanju ekip pomembna kompatibilnost članov, njihov skupni interes in potem samo-organiziranje oziroma izbira vodje.

Za uvedbo takšnih novosti je zato potreben čas in sprememba načina razmišljanja pri vseh udeleženih v razvoju od uporabnikov pa do razvijalcev. Obstoječi način dela je v razvoju sistema za bančno poslovanje utečen. V osnovi se je definiral pred devetimi leti in se bistveno ni spreminjal v celotnem obdobju vse do danes. Sproti so se uvajale izboljšave, katere uvedba vsake je prav tako trajala kakšno leto, vendar so bile v primerjavi s predlaganimi spremembami procesa razvoja vse to zelo majhne zahteve.

Zato je potrebno biti potrpežljiv in se uvedbe predlaganih sprememb lotiti po vrsti. Vsaka sprememba v utečenem delu je namreč lahko težavna. Ključno je, da se vsebino spremembe dobro komunicira z vsemi vpletenimi in pridobi kar se da široko podporo. Tisti, ki se jih sprememba neposredno dotika, pa so ključni za uspešnost uvedbe in širjenja pozitivne podpore pri svojih sodelavcih in vodstvu. Zato je nujno potrebna dobra komunikacija pred vsakršnimi nadaljnjimi spremembami v procesu dela na razvoju sistema za bančno poslovanje. Še toliko bolj pa je komunikacija ključna za celoten razvoj informacijskega sistema banke, saj se le z zgledi dobre prakse uspešno širijo.

Poglavje 7

Sklepne ugotovitve

Na koncu naj še enkrat povzamem glavne vsebine magistrske naloge ter njene cilje. Glavni cilji so bili:

- predstaviti proces razvoja programske opreme (angl. as-is) na primeru sistema za bančno poslovanje,
- opisati moderne trende pri razvoju programske opreme,
- predlagati izboljšave procesa razvoja (angl. to-be) sistema za bančno poslovanje z uporabo opisanih trendov.

Vsi cilji naloge so bili doseženi, za naprej pa ostaja odprta implementacija predlaganega izboljšanega procesa razvoja. Ta je bil namreč pripravljen na teoretični osnovi in izkušnjah drugih avtorjev pri prehodu na agilne in vitke metodologije razvoja. Velik del predlaganega izboljšanega procesa je tudi plod razmišljanj s sodelavci razvoja sistema za bančno poslovanje, lastnih razmišljanj ter izmenjav mnenj in izkušenj z drugimi strokovnjaki znotraj banke ter tudi izven nje. Obstoječi proces razvoja projekta ponuja namreč veliko dobrih praks, ki sem jih tudi opisal v poglavju 3, in jih je smiselno uporabiti tudi v novem procesu. Nekatere lahko ostanejo povsem nespremenjene, druge pa je smiselno prilagoditi agilnim principom.

Kar je bilo narejeno z preizkusom uvedbe novega načina dela, je bilo dokaj improvizirano in zato na tem področju ostaja največ dela za naprej.

Ugotovil sem, da predlagane spremembe niso tako majhne, kot se mogoče zdijo meni. Z opisanimi dilemami in odporom na katerega sem naletel lahko ugotovim, da so spremembe večje kot se zdijo. Predvsem postavljajo nove temelje načina razmišljanja o razvoju informacijskega sistema, katerih uvedba v organizacijo kot je banka lahko traja več let. Če se ozrem nazaj na svojih pet let sodelovanja pri razvoju sistema za bančno poslovanje, lahko prav tako ugotovim, da smo v tem času marsikateri postopek spremenili in ga optimizirali, izboljšali kakovost razvoja, vendar so bile te spremembe male v primerjavi s predlagano konceptualno novim procesom razvoja. Zato je nujno, da se uvedbe novega procesa ne improvizira in ga ne uvaja od spodaj navzgor (angl. bottom – up), ampak se najprej pridobi podporo vodstva na vseh potrebnih ravneh za uvedbo sprememb. Vodstvo se namreč mora zavedati pomena novega načina dela in nuditi podporo:

- najprej razvojni ekipi, ki bo izvajala projekt za potrditev novega načina dela (angl. proof of concept) in ugotovila pravi način postopnega prehoda na nov proces dela,

- kasneje pa vsem, ki so kakorkoli vključeni v razvoj informacijskega sistema, da bo ta potekal hitreje in bolj kvalitetno.

Glede na trende v svetu IT [21, 63, 34] pa lahko vsekakor potrdim, da so agilni, predvsem pa vitki principi razvoja prava smer za prihodnost banke in upam, da bomo tej smeri sledili tudi v prihodnje.

Literatura

[1] *Vodnik po znanju projektnega vodenja (PMBOK vodnik), tretja izdaja.* Project Management Institute, 2008.

[2] *Reputational risk and IT in the banking industry, Findings from the 2012 IBM Global Reputational Risk and IT Study.* IBM Press, 2012.

[3] P. Abrahamsson, O. Salo, J. Ronkainen, in J. Warsta, "Agile software development methods: Review and analysis," 2002.

[4] S. Ambler, "Agile modeling (am) home page;effective practices for modeling and documentation; poskus dostopa 25.8.2014." [Online]. Na naslovu: http://www.agilemodeling.com

[5] ——, *Agile modeling: effective practices for extreme programming and the unified process.* John Wiley & Sons, 2002.

[6] S. W. Ambler, "Agile modeling: A brief overview." *pUML*, št. 7, str. 7–11, 2001.

[7] ——, "Going beyond scrum: Disciplined agile delivery," *White Paper Series*, 2013.

[8] S. W. Ambler in M. Lines, *Disciplined agile delivery: A practitioner's guide to agile software delivery in the enterprise.* IBM Press, 2012.

[9] J. Arlow in I. Neustadt, "Uml and the unified process," *Practical Object-Oriented Analysis & Design, London, GB: Pearson Education Limited*, 2002.

[10] M. Balantič, "Testno voden razvoj programske opreme v javi ee," diplomsko delo, Univerza v Ljubljani, 2013.

[11] K. Balasubramanian, A. Gokhale, G. Karsai, J. Sztipanovits, in S. Neema, "Developing applications using model-driven design environments," *Computer*, št. 39, zv. 2, str. 33–40, 2006.

[12] K. Beck, *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.

[13] ——, *Test-driven development: by example*. Addison-Wesley Professional, 2003.

[14] B. Boehm in C. Abts, "Cots integration: Plug and pray?" *Computer*, št. 32, zv. 1, str. 135–138, 1999.

[15] G. Booch, J. Rumbaugh, in I. Jacobson, "The unified modeling language user guide," *Reading, PA: Addison-Wesley*, 1999.

[16] G. Candea, S. Bucur, in C. Zamfir, "Automated software testing as a service," in *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 2010, str. 155–160.

[17] A. Cockburn, *Writing effective use cases*. Addison-Wesley Longman, 1999.

[18] ——, *Crystal clear: a human-powered methodology for small teams*. Pearson Education, 2004.

[19] ——, "Using both incremental and iterative development," *STSC CrosssTalk (USAF Software Technology Support Center)*, št. 21, zv. 5, str. 27–30, 2008.

[20] A. Cockburn in L. Williams, "The costs and benefits of pair programming," *Extreme programming examined*, str. 223–247, 2000.

[21] T. Dybå in T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information and software technology*, št. 50, zv. 9, str. 833–859, 2008.

[22] M. Fowler, K. Beck, J. Brant, W. Opdyke, in D. Roberts, "Refactoring: Improving the design of existing code addison-wesley," *Reading, MA*, 1999.

[23] M. Fowler in J. Highsmith, "The agile manifesto," *Software Development*, št. 9, zv. 8, str. 28–35, 2001.

[24] C. Hibbs, S. Jewett, in M. Sullivan, *The art of lean software development: a practical and incremental approach*. O'Reilly Media, 2009.

[25] J. Highsmith, *Agile Project Management: Creating Innovative Products*. Addison Wesley Longman Publishing Co., Inc., 2004.

[26] ——, *Adaptive software development: a collaborative approach to managing complex systems*. Addison-Wesley, 2013.

[27] P. Hines in N. Rich, "The seven value stream mapping tools," *International journal of operations & production management*, št. 17, zv. 1, str. 46–64, 1997.

[28] W. J. Hopp in M. L. Spearman, "To pull or not to pull: what is the question?" *Manufacturing & Service Operations Management*, št. 6, zv. 2, str. 133–148, 2004.

[29] J. Horswill, *Designing and Programming CICS Applications*. "O'Reilly Media, Inc.", 2000.

[30] K. Jelnikar, "Avtomatsko funkcionalno testiranje programske opreme," diplomsko delo, Univerza v Ljubljani, 2009.

[31] J. Jeston in J. Nelis, *Business process management*. Routledge, 2014.

[32] M. S. John Parkinson, Mike Yorwerth, *Masters of Lean IT, How 3 Visionary IT Executives Maximize Value and Minimize Waste*. CA, 2009.

[33] M. B. Jurič, *SOA Approach to Integration: XML, Web services, ESB, and BPEL in real-world SOA projects*. Packt Publishing Ltd, 2007.

[34] N. B. Kindler, V. Krishnakanthan, in R. Tinaikar, "Applying lean to application development and maintenance," *The McKinsey Quarterly*, zv. 3, str. 99–101, 2007.

[35] D. Krafzig, K. Banke, in D. Slama, *Enterprise SOA: service-oriented architecture best practices*. Prentice Hall Professional, 2005.

[36] V. Kumar, *Customer relationship management*. Wiley Online Library, 2010.

[37] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and the Unified Process*. Prentice Hall PTR, Upper Saddle River, NJ, 2001.

[38] C. Larman in V. R. Basili, "Iterative and incremental development: A brief history," *Computer*, št. 36, zv. 6, str. 47–56, 2003.

[39] H. L. Lee, "Ultimate enterprise value creation using demand-based management." Stanford global supply chain management forum, 2001.

[40] J. N. Luftman, P. R. Lewis, in S. H. Oldach, "Transforming the enterprise: The alignment of business and information technology strategies," *IBM systems journal*, št. 32, zv. 1, str. 198–221, 1993.

[41] V. Mahnič, "A capstone course on agile software development using scrum," *Education, IEEE Transactions on*, št. 55, zv. 1, str. 99–106, 2012.

[42] ——, "Applying kanban principles to software development," *IT for practice 2013*, str. 89–96, 2013.

[43] V. Mahnič in I. Rožanc, "Students' perceptions of scrum practices," in *MIPRO, 2012 Proceedings of the 35th International Convention*. IEEE, 2012, str. 1178–1183.

[44] E. J. Mishan in E. Quah, *Cost-benefit analysis*. Routledge, 2007.

[45] T. Ōno, *Toyota production system: beyond large-scale production*. Productivity press, 1988.

[46] N. Oza, C. Ebert, in P. Abrahamsson, "Lean software development," *IEEE Software*, št. 29, zv. 5, str. 22–25, 2012.

[47] S. R. Palmer in M. Felsing, *A practical guide to feature-driven development*. Pearson Education, 2001.

[48] K. Pant in M. B. Jurič, *Business process driven SOA using BPMN and BPEL: From business process modeling to orchestration and service oriented architecture*. Packt Publishing Ltd, 2008.

[49] K. Petersen in C. Wohlin, "Measuring the flow in lean software development," *Software: Practice and experience*, št. 41, zv. 9, str. 975–996, 2011.

[50] D. Phillips, *The software project manager's handbook: principles that work at work*. John Wiley & Sons, 2004, št. 3.

[51] U. Poljšak, "Prenova poslovnih procesov in integracija informacijskih sistemov na primeru projekta novo bančno okence," magistrsko delo, Univerza v Ljubljani, 2008.

[52] M. Poppendieck in M. A. Cusumano, "Lean software development: A tutorial," *Software, IEEE*, št. 29, zv. 5, str. 26–32, 2012.

[53] M. Poppendieck in T. Poppendieck, *Lean software development: an agile toolkit*. Addison-Wesley Professional, 2003.

[54] R. Sanders, "The pareto principle: Its use and abuse," *Journal of Product & Brand Management*, št. 1, zv. 2, str. 37–40, 1992.

[55] A.-W. Scheer in M. Nüttgens, *ARIS architecture and reference models for business process management*. Springer, 2000.

[56] K. Schwaber in M. Beedle, *Agile Software Development with Scrum*. Prentice Hall PTR, 2001.

[57] S. Sedigh-Ali, A. Ghafoor, in R. A. Paul, "Software engineering metrics for cots-based systems," *Computer*, št. 34, zv. 5, str. 44–50, 2001.

[58] M. Silič, M. Colnar, M. Krisper, R. Rupnik, M. Bajec, I. Rozman, M. Heričko, T. Domajnko, M. Jurič, A. Živkovič *et al.*, "Emris–enotna metodologija razvoja informacijskih sistemov," *Ljubljana: Vlada Republike Slovenije, Center Vlade RS za informatiko*, št. 4, 2000.

[59] D. K. Sobek, A. C. Ward, in J. K. Liker, "Toyota's principles of set-based concurrent engineering," *Sloan management review*, št. 40, zv. 2, str. 67–84, 1999.

[60] F. Solina, *Projektno vodenje razvoja programske opreme*. Fakulteta za računalništvo in informatiko, 1997.

[61] J. Stapleton, *DSDM, dynamic systems development method: the method in practice*. Cambridge University Press, 1997.

[62] M. Walls, *Building a DevOps Culture*. "O'Reilly Media, Inc.", 2013.

[63] X. Wang, K. Conboy, in O. Cawley, "Leagile software development: An experience report analysis of the application of lean approaches in agile software development," *Journal of Systems and Software*, št. 85, zv. 6, str. 1287–1299, 2012.

[64] K. Waters, *All About Agile: Agile Management Made Easy!* CreateSpace, 2012.

[65] A. Weitzer, "Primerjava in vrednotenje procesov razvoja programske opreme," magistrsko delo, Univerza v Ljubljani, 2009.

[66] R. K. Wysocki, *Effective Project Management: Traditional, Adaptive, Extreme, Sixth edition.* John Wiley & Sons, 2012.

[67] S. Yoo in M. Harman, "Regression testing minimization, selection and prioritization: a survey," *Software Testing, Verification and Reliability*, št. 22, zv. 2, str. 67–120, 2012.