UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

MAGISTRSKO DELO

RAZVOJ INFORMACIJSKEGA SISTEMA ZA ANALIZO BANČNIH PODATKOVNIH VIROV

Ljubljana, september 2021 ŽAN KOREČIČ

IZJAVA O AVTORSTVU

Podpisani Žan Korečič, študent Ekonomske fakultete Univerze v Ljubljani, avtor predloženega dela znaslovom Razvoj informacijskega sistema za analizo bančnih podatkovnih virov, pripravljenega v sodelovanju s svetovalcem red. prof. dr. Mirom Gradišarjem.

IZJAVLJAM

1. da sem predloženo delo pripravil samostojno;
2. da je tiskana oblika predloženega dela istovetna njegovi elektronski obliki;
3. da je besedilo predloženega dela jezikovno korektno in tehnično pripravljeno v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani, kar pomeni, da sem poskrbel/-a, da so dela in mnenja drugih avtorjev oziroma avtoric, ki jih uporabljam oziroma navajam v besedilu, citirana oziroma povzeta v skladu z Navodili za izdelavo zaključnih nalog Ekonomske fakultete Univerze v Ljubljani;
4. da se zavedam, da je plagiatorstvo – predstavljanje tujih del (v pisni ali grafični obliki) kot mojih lastnih – kaznivo po Kazenskem zakoniku Republike Slovenije;
5. da se zavedam posledic, ki bi jih na osnovi predloženega dela dokazano plagiatorstvo lahko predstavljalo za moj status na Ekonomski fakulteti Univerze v Ljubljani v skladu z relevantnim pravilnikom;
6. da sem pridobil/-a vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v predloženem delu in jih v njem jasno označil/-a;
7. da sem pri pripravi predloženega dela ravnal/-a v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobil/-a soglasje etične komisije;
8. da soglašam, da se elektronska oblika predloženega dela uporabi za preverjanje podobnosti vsebine z drugimi deli s programsko opremo za preverjanje podobnosti vsebine, ki je povezana s študijskim informacijskim sistemom članice;
9. da na Univerzo v Ljubljani neodplačno, neizključno, prostorsko in časovno neomejeno prenašam pravico shranitve predloženega dela v elektronski obliki, pravico reproduciranja ter pravico dajanja predloženega dela na voljo javnosti na svetovnem spletu preko Repozitorija Univerze v Ljubljani;
10. da hkrati z objavo predloženega dela dovoljujem objavo svojih osebnih podatkov, ki so navedeni v njem in v tej izjavi.

V Ljubljani, dne _____ Podpis študenta: _____

KAZALO

KAZALO TABEL

KAZALO SLIK

SEZNAM KRATIC

angl. – angleško

API – (angl. Application Programming Interface); Vmesnik uporabniškega programa

CSV – (angl. Comma separated value); Vrednosti ločene z vejico

DOM –(angl. Document Object Model); Objektni model dokumenta

ECB SDW – (angl. European Central Bank Statistical Data Warehouse); Statistično podatkovno skladišče Evropske centralne banke

HTML – (angl. Hyper Text Transfer Protocol); Jezik za označevanje nadbesedila

HTQL – (angl. Hyper Text Query Language); Jezik za poizvedovanje v nadbesedilu

HTTP – (angl. Hyper Text Transfer Protocol); Protokol za prenos hiperteksta

IS – (angl. Information System); Informacijski sistem

IT – (angl. Information Technologies); Informacijska tehnologija

SURS – Statistični Urad Republike Slovenije

WPF– (angl. Windows Presentation Platform); Windows prezentacijska platofrma

XAML – (angl. Extensible Application Markup Language); Razširljiv aplikacijski označevalni jezik

UVOD

Turbulentni in vse bolj konkurenčni trgi, na katerih morajo sodobne organizacije delovati in poslovati, pričakujejo od svojih informacijskih sistemov, da se hitro razvijajo, hitro konfigurirajo in da jih je enostavno vzdrževati. Te lastnosti so izrednega pomena, saj imajo informacijsko tehnološke (v nadaljevanju IT) rešitve odločilno vlogo pri konkurenčnosti organizacij. Na znanju temelječi družbi, v kateri živimo, organizacije vse bolj tekmujejo v spretnosti pri pravočasnem in ustreznem odzivanju na tržne pritiske, priložnosti in spremembe (Rosemann & vom Brocke, 2015, str. 105).

Potencial informacijskih sistemov za izboljševanje procesa odločanja in izboljšanje uspešnosti organizacije je že nekaj časa poudarjen v literaturi o poslovni vrednosti IT (Davern & Kauffmanm, 2000, str. 123). V študijah o uspešnosti je bilo ugotovljeno, da informacijski sistemi podpirajo pravočasne odločitve, ki povečujejo prednost, spodbujajo inovacije in zmanjšujejo negotovost, ki je značilna za današnje poslovno okolje (Melville, Kraemer & Gurbaxani, 2004, str. 287). Visokokakovostne informacije so pomembne, zanesljive, točne in pravočasne (Wixom & Todd, 2005, str. 86) ter omogočajo izboljšanje kakovosti odločitev in lahko posledično tudi izboljšajo uspešnost podjetja. Da bi izkoristili prednosti visokokakovostnih informacij, podjetja vse več vlagajo v IT (Verhoef, 2005, str. 316).

Da organizacije lahko ostanejo konkurenčne, je potrebno, da izkoristijo tehnološki napredek pri razvoju informacijskih sistemov. Zaradi tega so tehnološka kompleksnost in hitro spreminjajoče se zahteve temeljne ključne besede v današnjem času pri razvoju informacijskih sistemov. Organizacije morajo poleg uporabe ustreznih metodologij razvijanja ustvariti tudi tehnološko infrastrukturo, ki jim bo omogočila pravočasno in učinkovito razvijanje ter vzdrževanje njihovih informacijskih sistemov.

Namen magistrskega dela je izboljšati informacijski sistem banke. Analiziral bom trenuten sistem za zajem podatkov v izbranem oddelku banke in dodatne funkcionalnosti, ki jih želijo uporabniki.

Cilj magistrskega dela je nov sistem, katerega izid bo:

- večja kakovost podatkov,
- večja hitrost prenosa podatkov od vira do podatkovne baze,
- hitrejši dostop do podatkov,
- nižji stroški vzdrževanja,
- enotna centralna podatkovna baza,
- beleženje delovanja sistema.

V magistrskem delu bom uporabil pridobljeno znanje in kompetence iz dodiplomskega študija na Fakulteti za računalništvo in informatiko ter magistrskega študija na Ekonomski fakulteti. Poleg znanja in kompetenc, pridobljenih s študijem, bo uporabljeno tudi večletno praktično znanje, pridobljeno iz izkušenj v postavljanju in načrtovanju novih informacijskih sistemov. Tovrstno znanje bom združil s teoretičnim proučevanjem literature. Literatura obsega poslovno informatiko tujih in domačih avtorjev ter spletnih virov. Uporabil bom metodo sistemske analize in oblikovanja informacijskih sistemov ter metodo sinteze nove rešitve. Potrebne informacije za načrtovanje in postavitev novega sistema bom pridobil z zbiranjem zahtev zaposlenih na oddelku banke.

1 RAZVOJ INFORMACIJSKIH SISTEMOV

V prvem poglavju bomo opisali različne načine oziroma metodologije načrtovanja ingradnje raznih informacijskih rešitev. Opisali bomo življenjski cikel razvoja; kakšne faze so vključene; različne pristope h gradnji IS; kaj je značilno za tradicionalne pristope in kaj je specifično pri agilnih pristopih. Izpostavili bomo tudi objektno usmerjen pristop in značilnosti takega pristopa. Poleg tega bomo opisali različne možnosti pridobivanja programske opreme od lastnega razvijanja do nakupa obstoječe opreme in najema zunanjih izvajalcev. Na koncu si bomo še pogledali, kaj je analiza stroškov in koristi, in zakaj je to uporabno pri razvijanju novih sistemov.

1.1 Življenjski cikel razvoja informacijskih sistemov

Življenjski cikel razvoja informacijskih sistemov je temeljna metoda razvijanja informacijskih rešitev, kot tudi za druge projekte, ker ta pristop temelji na izvajanju več faz vsekvenčnem zaporedju (Gradišar, Jaklič & Turk, 2007, str. 166). Omenjeni pristop k razvoju informacijskih rešitev je zelo strukturiran in je zasnovan za upravljanje velikih projektov, kateri vključujejo več programerjev in sistemov, ki imajo velik vpliv na organizacijo. Zahteva jasno in predhodno razumevanje, kaj naj bi rešitev počela, in ni podložen do sprememb pri oblikovanju sistema (Bourgeois, 2014, str. 106). Obstajajo različne opredelitve faz te metodologije, večina avtorjev pa navaja naslednje faze:

Prva faza življenjskega cikla razvoja sistemov vključuje preliminarno analizo, s katero lahko ugotovimo, ali je koncept izvedljiv; predlagamo alternativne rešitve v primeru, da prvotni koncept ni izvedljiv; izvedemo analizo stroškov in koristi. Pred koncem preliminarne analize razvijalci izvedejo še študije izvedljivosti, da ugotovijo, ali naj popravijo obstoječi sistem ali naj ustvarijo popolnoma nov sistem, ki bo nadomestil obstoječega (O'Brien & Marakas, 2011, str. 486). Študija izvedljivosti vsebuje naslednje komponente (Stefanou, 2003, str. 332; O'Brien, & Marakas, 2011, str. 487):

- Operativna izvedljivost - ocenjuje, kako dobro predlagani sistem rešuje probleme, kako izkorišča definirane priložnosti pri opredelitvi sistema in kako sistem izpolnjuje zahteve, opredeljene v preliminarni fazi analize zahtev;
- Ekonomska izvedljivost - določa pozitivne gospodarske koristi, ki jih bo sistem zagotavljal (analiza stroškov in koristi);
- Tehnična izvedljivost – preučuje, ali bo sistem deloval pravilno, in preveri potencialne ovire;
- Pravna izvedljivost – preverja, ali je predlagan sistem v skladu z zakonskimi zahtevami;
- Načrtna izvedljivost – preverja, ali je sistem mogoče razviti pravočasno in prepoznati alternativne možnosti;
- Organizacijska in družbena izvedljivost – preučuje, ali je sistem sprejemljiv na vseh ravneh;
- Strateška izvedljivost – preverja, ali sistem ustreza strateškim planom organizacije.

Faza sistemske analize vključuje definiranje ciljev projekta v opredeljene funkcije in operacije novega sistema. Ta postopek vključuje zbiranje in interpretacijo dejstev, diagnosticiranje problemov in zbiranje priporočil za izboljšanje sistema. K doseganju ciljev projekta bodo še dodatno pomagale analize informacij končnih uporabnikov o potrebah in odpravljanje morebitnih nedoslednosti in nepopolnih zahtev (Sharma, 2017, str. 520).

Tretja faza je načrtovanje sistema. V tej fazi življenjskega cikla opisujemo želene lastnosti in operacije sistema. Cilj faze načrtovanja je pretvoriti vse zahteve v podrobne specifikacije, ki zajemajo vse vidike sistema, in tudi oceniti varnostna tveganja. Nazadnje je potrebno dobiti odobritev, da se ta faza zaključi in se premakne v fazo razvijanja.

V četrti fazi sledi razvijanje. Programska koda se začne pisati v tej fazi, kjer programerji uporabijo dokument iz prejšnje faze načrtovanja kot vodilo za razvijanje sistema (Stefanou, 2003, str. 334). Rezultat te faze je začetni delujoč program, kateri izpolnjuje zahteve, ki so bile določene v fazi analize sistema in fazi načrtovanja sistema.

V peti fazi je vključeno testiranje sistema, razvitega v prejšnji fazi; sistem gre skozi več strukturiranih testov. Prvi test oceni posamezne dele kode za napake in neujemanja. Sledi sistemski test, v katerem se preizkusijo različne komponente sistema, da se zagotovi njihovo pravilno delovanje. Končni test vključuje končne uporabnike, ki bodo uporabljali ta sistem, da testirajo in se prepričajo, ali jim novi sistem ustreza glede na njihove standarde. Vse nepravilnosti, napake ali problemi, ki se izkažejo v tej fazi, morajo biti odpravljeni in nato se sistem še enkrat ponovno testira (Bourgeois, 2014, str. 105).

Faza implementacije se začne, ko je novi sistem razvit in preizkušen ter se ga začne vgrajevati v organizacijo. Ta faza vključuje usposabljanje uporabnikov za nov sistem, zagotavljanje dokumentacije za uporabo in pretvorbo podatkov iz prejšnjega sistema v novega. Izvedba ima lahko tudi različne oblike, odvisno od vrste sistema, števila uporabnikov in od nujnosti, da sistem začne delovati (Tilley & Rosenblatt, 2016, str. 21).

Sedma faza je vzdrževanje novo postavljenega sistema v organizaciji, ki se začne po zaključku implementacije. V fazi vzdrževanja ima sistem vzpostavljen strukturiran podporni proces za prijavljanje napak, zahteve za nove funkcije, ki gredo skozi ocenjevanje in nato v izvedbo (Hoffer, George & Valacich, 2020, str. 11). Za vsako novo različico sistema so narejene sistemske posodobitve in varnostne kopije. V tej fazi se lahko šele pojavijo zahteve po spremembah prvotne programske opreme.

Zadnja faza v življenjskem ciklu je faza upokojitve. V tej fazi se oblikujejo načrti za prenehanje uporabe informacij, strojne opreme, informacijskega sistema in načrt za prehod na novi sistem. Namen tega je pravilno premikanje, arhiviranje, zavržba ali uničenje informacij, strojne in programske opreme, ki se jo nadomešča na takšen način, da preprečuje kakršno koli nepooblaščeno razkritje občutljivih podatkov. Upokojevanje sistema zagotavlja ustrezen prehod na nov sistem, poseben poudarek pa je namenjen pravilnemu ohranjanju in arhiviranju podatkov, ki so bili uporabljeni v prejšnjem sistemu.

Celoten postopek je potrebno opraviti v skladu z varnostnimi zahtevami organizacije (Broad, 2013, str. 40; Radack, 2009, str. 2). Metodologija življenjskega cikla razvoja sistemov je včasih poimenovana tudi kot slapovna (angl. waterfall) metodologija, in sicer v primerih, ko se koraki izvajanja izvedejo samo enkrat v zaporedju. Naslednji korak se izvede šele, ko je bilprejšnji korak dokončan. Po vsakemu koraku se mora organizacija odločiti, kdaj naj preide na naslednji korak. Projekti, ki uporabljajo to metodologijo, lahko včasih trajajo mesece ali leta. Zaradi svoje nefleksibilnosti in razpoložljivosti novih programerskih tehnik in orodijso bile razvite številne druge metodologije za razvoj programske opreme (Bourgeois, 2014, str. 106).

1.2 Tradicionalni/klasični pristop

Metodologije, kot so slapovni ali V model, se imenujejo tradicionalne metodologije razvoja programske opreme in so kategorizirane pod težke metodologije (Nikiforova, Nikulsins & Sukovskis, 2009, str. 230). Te metodologije temeljijo na izvajanju zaporedij sekvenčnih korakov, kot so opredelitev zahtev, izdelovanje rešitve, testiranje in uvajanje. Ta pristop zahteva točno opredelitev in dokumentiranje zahtev že pred začetkom projekta.
Za tradicionalne metodologije razvoja programske opreme so značilne štiri faze. Prva faza je določitev zahtev za projekt in določitev potrebnega časa za izvedbo različnih faz razvoja ter hkrati poskus napovedi morebitnih težav, ki se lahko pojavijo tekom izvedbe projekta. Ko so zahteve določene, je naslednji korak, ki sledi, prehod v fazo načrtovanja in arhitekturnega planiranja, kjer se izdela tehnična infrastruktura v obliki diagramov ali modelov. Diagrami in modeli potencialno razkrijejo težave, s katerimi se lahko projekt sooča med napredovanjem, in razvijalcem zagotavljajo izvedljiv načrt za implementiranje. V primeru, da so vsi vpleteni v projekt zadovoljni z arhitekturnim in oblikovnim načrtom, se projekt premakne v fazo razvijanja, kjer se programska koda začne izdelovati, dokler niso doseženi zaželeni cilji. Razvijanje se pogosto razdeli na manjše obvladljive naloge, ki se razdelijo med različne ekipe na podlagi kompetentnosti. Faza testiranja se pogosto prekriva s fazo razvijanja z namenom, da se že zgodaj zagotovi obravnavo problemov; ko se projekt

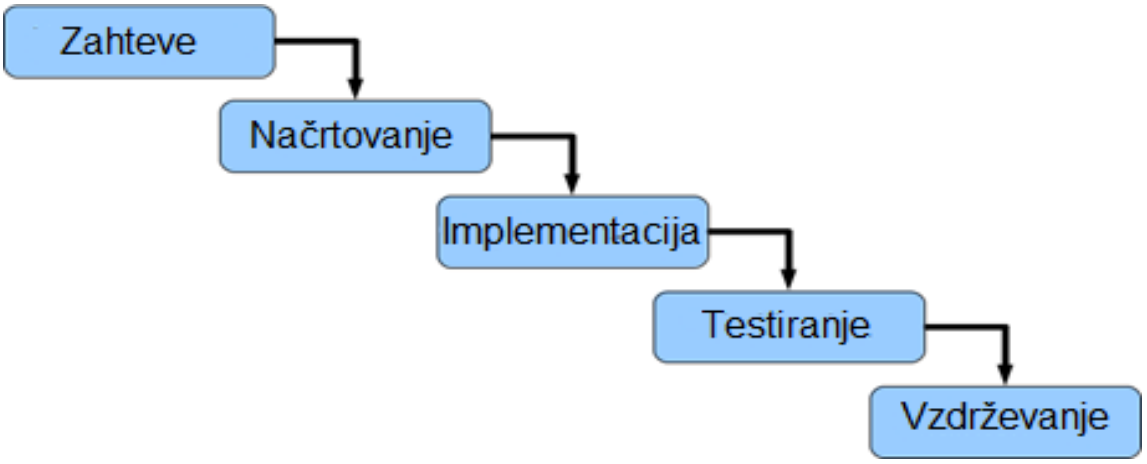
približuje zaključku in so razvijalci tudi blizu izpolnjevanju projektnih zahtev, se nato v cikel testiranja vključi še stranka in projekt se zaključi šele, ko je stranka zadovoljna z rezultatom. Tradicionalne metode razvoja programske opreme so odvisne od skupka vnaprej določenih procesov in dokumentacije, ki je napisana med samim delom in služi kot vodič za nadaljnje razvijanje (Nikiforova, Nikulsins & Sukovskis, 2009, str. 231).

Uspeh projekta, ki deluje po tradicionalnem principu, temelji na poznavanju vseh zahtev, določenih pred začetkom razvoja, kar pomeni, da je uvajanje sprememb v razvojnem življenjskem ciklu nekoliko težavno. Ta pristop pa tudi olajša določanje stroškov projekta, postavitev časovnih rokov in razporeditev virov (Leau, Loo, Tham & Tan, 2012, str. 163).

1.2.1 Slapovna metoda

Slapovna metoda življenjskega cikla razvijanja programske opreme je zaporedni postopek razvijanja programske opreme, pri katerem se napredek skozi faze izvajanja premika navzdol podobno kot slapovi. Slapovni model opredeljuje več zaporednih faz, katere je potrebno zaključiti eno za drugo, in premik v naslednjo fazo se opravi šele, ko je bila predhodna faza popolnoma končana (Munassar & Govardhan, 2010, str. 96). Zaradi tega je ta metoda rekurzivne narave, saj lahko vsako fazo neskončno ponavljamo, dokler ne dobimo izpolnjenih rezultatov (Chowdhury, Bhowmik, Hasam & Rahim, 2017, str. 2). Na sliki 1 prikazujemo različne faze slapovnega modela.

Slika: 1: Potek faz slapovne metode



Prirejeno po Chowdhury, Bhowmik, Hasam & Rahim (2017, str. 2).

Metoda je sestavljena iz petih faz (Bassil, 2012, str. 2):

1. Faza analize - Pogosto znana kot specifikacija zahtev za programsko opremo, je popoln in celovit opis delovanja sistema, katerega se razvija. V tej fazi morajo analitiki opredeliti funkcionalne in tudi nefunkcionalne zahteve. Funkcionalne zahteve so običajno opredeljene s primeri uporabe, kateri opisujejo interakcijo med uporabniki in sistemom. Vključujejo zahteve kot so namen, področje uporabe, perspektivo,

funkcije, atribute sistem, značilnosti uporabnikov, specifikacije funkcionalnosti, zahteve za vmesnike in zahteve za podatkovne baze. V nasprotju s tem se nefunkcionalne zahteve nanašajo na različna merila, omejitve in zahteve, ki veljajo pri načrtovanju in delovanju sistema. Vključujejo lastnosti, kot so zanesljivost, razširljivost, razpoložljivost, zmogljivost in standarde kakovosti.

2. Faza načrtovanja - Gre za postopek načrtovanja in reševanja problemov programske rešitve, ki jo izdelujemo. Ta faza vključuje razvijalce in oblikovalce sistemov, da opredelijo načrt rešitve, ki vključuje zasnovo algoritmov, arhitekture sistema, konceptualno shemo podatkovne baze in zasnovo logičnega diagrama, grafičnega vmesnika in definicijo podatkovnih struktur.
3. Faza implementacije - Ta faza se nanaša na uresničitev poslovnih zahtev in specifikacij v izvršljiv program, podatkovno bazo, spletno stran ali programsko komponento s programiranjem in uvajanjem. V tej fazi se izvaja proces pretvorbe vseh zahtev in načrtov v proizvodno okolje, kjer se ustvari vsa programska koda, podatkovne baze in vse datoteke, potrebne za delovanje novega sistema.
4. Faza testiranja - Znana tudi kot faza preverjanja in potrjevanja. To je postopek, pri katerem preverjamo, ali programska rešitev ustreza prvotnim zahtevam in specifikacijam ter ali izpolnjuje svoj predvideni namen. Preverjanje je postopek ocenjevanja programske opreme z namenom, da se ugotovi, ali izdelek v dani razvojni fazi izpolnjuje pogoje, določene na začetku te faze; potrjevanje pa je postopek vrednotenja izdelka med ali na koncu razvojnega procesa, da se ugotovi, ali izpolnjuje določene zahteve.
5. Faza vzdrževanja - Gre za postopek, v katerem spreminjamo programsko rešitev po zaključeni fazi implementacije z namenom, da se izboljša delovanje, popravi napake ter izboljša zmogljivost in kakovost. V tej fazi se lahko izvajajo dodatne vzdrževalne dejavnosti, vključno s prilagajanjem programske rešitve novim uporabniškim zahtevam in povečevanju zanesljivosti sistema.

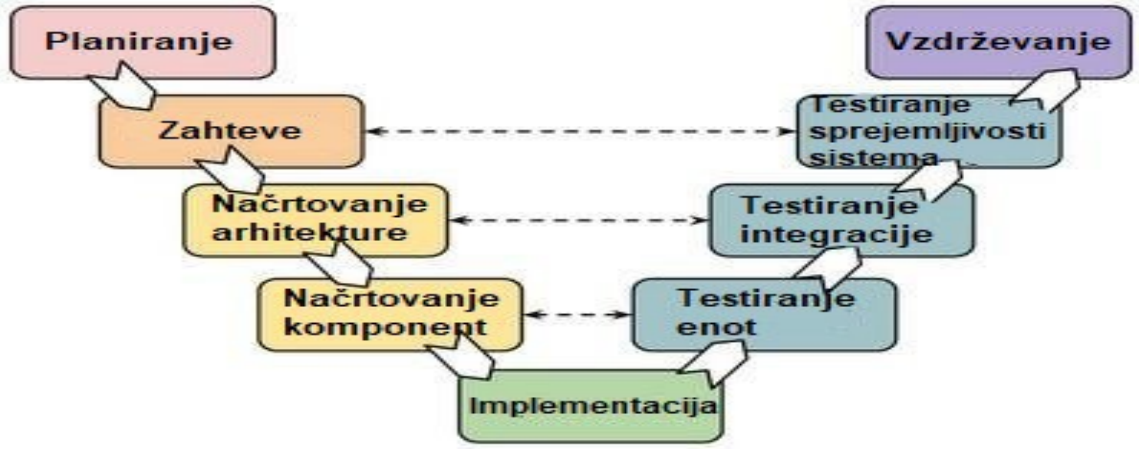
1.2.2 V model

V osnovi je V model podoben slapovnemu modelu, pri katerem se faze življenjskega cikla izvajajo zaporedno in kjer mora biti vsaka faza zaključena pred začetkom naslednje. Poudarek pri V modelu je na testiranju in procedure testiranja se razvijejo že zgodaj v življenjskem ciklu pri vsaki stopnji pred fazo razvijanja. Model se začne s fazo zbiranja zahtev in še pred začetkom razvoja se izdela načrt za testiranje sistema. Ta načrt za testiranje se osredotočana izpolnjevanje funkcionalnosti, zbranih v prvi fazi (Ragunath, Velmourougan, Davachelvan, Kayalvizhi & Ravimohan, 2010, str. 114).

Faza načrtovanja arhitekture se osredotoča na arhitekturo in obliko sistema. Načrt testiranja integracije se ustvari v tej fazi z namenom, da se preizkusi zmožnost medsebojnega delovanja različnih programskih delov sistema. V fazi načrtovanja komponent se začne načrtovanje dejanskih programskih komponent in hkrati se ustvarijo tudi načrti za testiranje teh enot. Sledi faza implementacije, pri kateri se začne razvijanje sistema, in ko se ta faza konča, se pot izvajanja modela nadaljuje po desni strani, kot je prikazano na sliki 2,kjer se tudi

uporabljajo načrti za testiranje, ki so bili razviti tekom izvajanja.

Slika: 2: Potek faz V modela



Prirejeno po Chowdhury, Bhowmik, Hasam & Rahim (2017, str. 3).

1.2.3 RAD

Metodologija hitrega razvijanja aplikacij (angl. rapid application development, v nadaljevanju RAD) omogoča hiter razvoj informacijskih sistemov od faze načrtovanja do zaključka z razmeroma nizkimi stroški (Geambaşu, Jianu, Jianu & Gavrilă, 2011, str. 483). Programska oprema je razdeljena na manjše komponente, kar olajša spreminjanje skozi celoten razvojni proces. Za projektne komponente so določeni časovni okviri, ki se jih ne sme prekoračiti. Funkcionalnosti so prioritizirane in se po potrebi zmanjšujejo, da ustrezajo postavljenemu časovnemu okviru, če je to potrebno. Ta metodologija uporablja druge metode, kot so JAD, spiralni model ali hitro prototipiranje. V RAD projektih izgled projekta, ki je prikazan med prototipiranjem, potem postane dejanski produkt. Tako kot pri drugih metodologijah, je tudi tu komponente mogoče ponovno uporabiti (Daud, Bakar & Rusli, 2010, str. 1665). Za razliko od strukturiranih metodologij, ki vključujejo večje število faz za dokončanje programske opreme, metodologija RAD predlaga par korakov, ki aktivno vključujejo razvojno skupino in uporabnike, kar posledično vodi do hitrejšega razvijanja projekta.

Življenjski cikel razvoja po metodologiji RAD je razdeljen na pet stopenj (Geambaşu, Jianu, Jianu & Gavrilă, 2011, str. 484):

- Inicializacija;
- Zahteve;
- Načrtovanje;
- Implementacija;
- Uvajanje.

Vsaka stopnja vključuje izvedbo dodatnih faz. Vsaka faza je razdeljena na dodatne tri korake (Geambaşu, Jianu, Jianu & Gavrilă, 2011, str. 484):

- Priprava - Organizirana je vrsta virov, ki bodo predstavljeni, obravnavani in spremenjeni med pripravo,
- Seja - Različni udeleženci v procesu razvoja programske opreme se sestanejo z namenom, da sprejemajo odločitve o načinu vodenja prihodnjih dejavnosti,
- Zaključek - Rezultat seje je oblikovan v obliki sklepov, ki bodo upoštevani v procesu razvijanja programske opreme.

Pri projektih, razvitih s to metodologijo, so odgovornosti jasno razdeljene med različne udeležence. Ta metodologija se osredotoča na uporabnike, ki so aktivno vključeni v razvojni proces, zato je zadovoljstvo uporabnika na višji ravni (Amlani, 2012, str. 8). RAD prinaša številne prednosti v primerjavi z metodologijami, ki so se uporabljale pred njegovim pojavom, kar vodi k zmanjšanju časa, potrebnega za pridobitev končnega sistema, zmanjšanju stroškov in ublažitvi tveganja neuspeha z vključevanjem uporabnikov v razvojno skupino. Zaradi uporabe prototipov RAD omogoča uporabnikom interakcijo z različnimi verzijami sistema že v zgodnjih fazah razvojnega procesa. Spreminjajoče se zahteve je mogoče hitro vključiti v sistem, vendar pa obstajajo tveganja pri tej metodologiji; ker se zahteve običajno ne upoštevajo sistematično in ker ekipa hitro deluje skozi iteracije projekta, je možno, da spregledajo pomembne zahteve.

1.3 Agilni pristopi

Agilni pristop temelji na ideji o postopnem in iterativnem razvijanju, pri katerem se faze znotraj življenjskega cikla razvijanja vedno znova pregledujejo. Iterativno se izboljšuje programsko opremo z uporabo povratnih informacij strank za približevanje rešitvi (Szalvay, 2004, str. 8; Martin, Newkirk & Koss, 2003, str. 7; Chan & Thong, 2009, str. 811). Pri agilnem pristopu je življenjski cikel razvoja namesto izvajanja zaporedja sekvenčnih korakov, ki se običajno izvaja pri tradicionalni metodologiji, razdeljen na manjše dele imenovane inkrementi ali iteracije, v katerih se vsak od teh inkrementov dotika tradicionalnih faz razvoja. Najpomembnejši dejavniki agilnega pristopa so sledeči (Fowler & Highsmith, 2001, str. 29; Williams, 2010, str. 5):

- Zgodnje vključevanje strank;
- Iterativno razvijanje;
- Samoorganizajoče se ekipe;
- Prilagoditve spremembam.

Agilni pristop vodenja projektov je namenjen predvsem kreativnim, inovativnim projektom, kot so raziskovalni projekti ali novi inovativni načini za razvoj izdelkov. Za vse takšne projekte je značilna visoka stopnja negotovosti, nejasni cilji projekta ali nepopolne in nepredvidljive zahteve, za katere bi lahko domnevali, da se bodo med projektom bistveno spremenile (Špundak, 2014, str. 942).

1.3.1 Ekstremno programiranje

Ekstremno programiranje lahko opredelimo kot lahko metodologijo, ki olajša majhnim skupinam razvijalcev načrtovanje in iterativno razvijanje programske opreme tako, da dosežejo višje kakovosti programske opreme in večjo produktivnost na odziv hitro spreminjajočih zahtev. Za ekstremno programiranje je značilna intenzivna stopnja interakcije s strankami med postopkom razvoja programske opreme (Kent, 2000, str. 2).

Značilnosti ekstremnega programiranja so (Shore, 2007, str. 40):

- Zahteve kot zgodbe - Uporabniki zahteve predstavijo kot scenarije, ki so nato oblikovani kot zgodbe. Razvijalci vsako zgodbo razdelijo na manjše naloge, ki jih kupec še naprej prioritizira;
- Preprostost - Ekstremno programiranje je naklonjeno razvijanju programske opreme z najpreprostejšimi oblikami, dodatne funkcije pa se lahko dodajajo po potrebi, kadar jih stranka zahteva;
- Neprekinjena interakcija - vključuje visoke ravni interakcije s strankami preko povratnih informacij. Sodelovanje s stranko poteka v pogostih majhnih iteracijah, s čimer se zagotovi, da stranka ostane seznanjena z napredkom razvoja programske opreme. To omogoča tudi hitro prilagajanje sprememb programske opreme glede na povratne informacije strank;
- Testno orientiran razvoj - testni primeri za naloge se napišejo že pred njenim grajenjem. Testiranje ostaja integralni del celotne metodologije;
- Refaktoriranje - spodbuja grajenje visoko kakovostnih rešitev z refaktoriziranjem obstoječih rešitev, s čimer dosežejo večjo zanesljivost kode in zmanjšuje zapletenost;
- Programerske dvojice - edinstven koncept te metodologije je vključevanje programerjev v dinamične dvojice, kar povzroča izboljšano komunikacijo, zmanjšanje količine delovnega časa in delovnih obremenitev.

1.3.2 Scrum metodologija

Scrum ponuja prilagojen način dela na različnih projektih, kateri imajo različne zahteve, in ima prednosti, kot je prilagodljivo izbiranje zahtev za sprinte, ter ne vsebuje posebnih postopkov, ki jih je potrebno upoštevati.

Potek dela pri scrum metodi je sestavljen iz tesnega sodelovanja med scrum ekipo, vodjo projekta in lastnikom produkta v neprekinjenih iteracijah razvijajoče se programske opreme. Glavna vloga vodje projekta je odpravljati ovire, medtem ko je scrum ekipa več funkcionalna skupina, sestavljena iz razvijalcev, testerjev in strokovnjakov z različnih področij, ki so ključni za razvoj, kar tudi vodi do vsestranskega in inovativnega končnega izdelka (Srivastava, Bhardwaj & Saraswat, 2017, str. 865).

Sprint je sestavljen iz majhne ekipe, ki dela na dodeljeni nalogi od enega do treh tednov. Sprint backlog je dokumentacija o vseh zahtevah za trenutni sprint in ta backlog tudi določa,

katere naloge se bo izvedlo v sprintu. Product backlog je seznam zahtev, ki jih določi lastnik izdelka in se imenujejo uporabniške zgodbe. Razdeljen je na sprint backloge, po tem sledi načrtovanje sprinta, ki vključuje metode za izvedbo sprinta. Na koncu vsakega dne se izvede dnevni scrum z namenom, da se pregleda napredek dela za tisti dan.

Cilj vsakega sprinta je prikaz potencialnega končnega izdelka in na koncu sledi pregled sprinta z lastnikom produkta, da se prikaže novi inkrement razvijanja. Med sprintom ni mogoče spreminjati ciljev, lahko pa pri vsakem inkrementu lastnik produkta doda nove funkcionalnosti, ki prej niso bile določene. Z uporabo retrospekcije se naslednji sprint načrtuje na podlagi znanja in ugotovitev, ki so bila pridobljena v prejšnjem sprintu.

1.4 Objektno usmerjen pristop

Objektno usmerjeni pristop poudarja sistemsko analizo in načrtovanje z vidika končnih uporabnikov sistema, kar pomeni, da je objektno usmerjen pristop usmerjen na primere uporabe. Primer uporabe zajema scenarije uporabe sistema. Uporablja se za določanje funkcionalnih in operativnih zahtev sistema, ki ga je potrebno zgraditi, z opisom načina interakcije uporabnikov s sistemom. Ta pristop sprejema razvojno paradigmo na osnovi komponent z uporabo zapisa diagramskega jezika za modeliranje komponent sistema (Barros, 1994, str. 103). Vse metode razvijanja programske opreme lahko izvedemo na strukturiran način ali na objektno usmerjen način oziroma pristop. V objektno usmerjenim pristopom so sistemske komponente sestavljeni objekti, ki so koncepti ali entitete. Za objekte so značilni atributi in metode, ki vsebujejo seznam procedur. Objekti so avtonomni, vendar lahko komunicirajo z uporabo s tako imenovanimi komunikacijskimi sporočili. Enkapsulacija je lastnost, ki omogoča skrivanje podatkov in metod pred zunanjim svetom, kar omogoča, da se predhodno zasnovani objekti ponovno uporabijo v novih sistemih (Zhou, Greenwell & Tannock, 1994, str. 115). Dostop do podatkov in metod objekta se doseže preko predpisanih vmesnikov ter z uporabo sporočil. Enkapsulacija podatkov in metod znotraj objektov omogoča sestavljanje razredov, kar so abstraktni koncepti za logično združevanje povezanih objektov. To vodi do ustvarjanja knjižnic objektov in razredov za večkratno uporabo. Osnovna lastnost dedovanja pomeni, da vsak objekt podeduje atribute in metode razreda, ki je nad trenutnim. Dedovanje omogoča razširitev obstoječih vrst objektov, pri čemer je pomembno, da je programska koda modulov narejena za ponovno uporabo bodisi za razširitev obstoječih aplikacij ali za hiter razvoj novih (Abreu & Carapuça, 1994, str. 4).

Popularnost objektno usmerjenih pristopov v devetdesetih temelji na konceptu, da je programsko opremo, ki temelji na modularnosti, ponovni uporabi in samostojnosti objektov, lažje vzdrževati, razširiti in prilagoditi kot običajno programsko opremo. Sistemi, razviti s preizkušenimi komponentami za ponovno uporabo, so sistemi z manjšimi tveganji in se razvijajo veliko hitreje kot tisti, ki so razviti na običajen način. Objektno usmerjeni pristopi se konceptualno razlikujejo od običajnih metodologij za razvijanje sistemov. Objektno usmerjeni pristopi poudarjajo osnovne lastnosti objektov, enkapsulacijo, klasifikacijo in dedovanje. Konvencionalne metodologije ločeno obravnavajo podatke in

postopke, ki se uporabljajo pri preučevanju določenega problema. Osnovne faze tradicionalnega življenjskega cikla razvoja sistemov so še vedno prisotne v objektno usmerjenih pristopih, kot so analiza, načrtovanje in razvijanje (Zhou, Greenwell & Tannock, 1994, str. 116).

Objektno usmerjena analiza se osredotoča na preiskovanje problemov. Začne se z oceno specifikacij kupcev in ustvarjanjem primerov uporabe. Primer uporabe opisuje alternativne scenarije, kako lahko uporabnik in drugi sistemi komunicirajo s sistemom, katerega je potrebno zgraditi, preden nadaljujejo z izdelavo modela objektno usmerjene analize. Glavna naloga sistemskih analitikov v objektno usmerjeni analizi je prepoznavanje objektov, ki se nanašajo na določeno težavo, prepoznavanje instanc, ki predstavljajo posamezen pojav objekta v razredu in združevanje objektov v razrede, s katerim zagotavljamo opise in navedbo njihovih namenov. Na vsakem koraku ustvarjanja objektno usmerjene analize sistemski analitiki ob upoštevanju uporabniških zahtev razvijejo, testirajo in dokumentirajo vlogo vsakega razreda, njihove interakcije in izvedene operacije objekta v razredu (Coad & Yourdon, 1991, str. 19).

Objektno usmerjeno načrtovanje se ukvarja z logičnim reševanjem problemov. Za načrtovanje sistemov in posameznih objektov so bile predlagane različne tehnike modeliranja. Objektno usmerjeno načrtovanje identificira in določi podsisteme, potrebne za izvajanje nekaterih funkcij, definira večplastno sistemsko arhitekturo ter opisuje objekte, razrede in komunikacijske mehanizme. Cilj tega načrtovanja je razviti podroben model, kije sestavljen iz programskih modulov, kako upravljati podatke in naloge, ter uporabniškega povezovanja, ki ga je mogoče uresničiti z uporabo objektno usmerjenega jezika (Wirfs-Brock & Johnson, 1990, str. 113).

Objektno usmerjeno razvijanje je dejavnost, s katero se zasnova objektno usmerjenega načrta spremeni v programsko kodo z uporabo programiranja. Testiranje sistema oziroma aplikacije je dodatna dejavnost, ki se izvaja v tem koraku. Objektno usmerjena tehnologija je še posebej pomembna pri gradnji sodobnih informacijskih sistemov na arhitekturi odjemalec- strežnik. Objekti so lahko v istih ali ločenih programskih modulih, podatkovna baza je lažje uporabljena v porazdeljenih okoljih. Lahko se tudi vzpostavi boljša revizijska sled dejavnosti, ker operacije določajo, kako naj se uporabniške zahteve izvajajo.

1.5 Pridobivanje programskih rešitev

Pridobivanje novih programskih rešitev lahko opravimo na več različnih načinov. V tem podpoglavju bomo izpostavili več različnih načinov, kako lahko to opravimo in prikazali, kaj imajo različni ponudniki na voljo in kakšne so prednosti oziroma omejitve pri posameznem načinu.

1.5.1 Lasten razvoj

Običajno se podjetja odločijo za lastno razvijanje, da izpolnijo svoje edinstvene poslovne zahteve z namenom, da zmanjšajo spremembe v poslovnih procedurah in politikah, ter da upoštevajo omejitve obstoječih sistemov in tehnologije. Podjetja se pogosto odločajo, da bodo razvila lastno programsko opremo, ker noben komercialno dostopen programski paket ne ustreza njihovim edinstvenim poslovnim zahtevam.

Lahko se tudi odločijo za razvoj lastne programske opreme, če razpoložljivi komercialni paketi zahtevajo spremembe trenutnega načina poslovanja ali spreminjanje obstoječih poslovnih procesov. Namestitev novih programskih paketov skoraj vedno zahteva določeno stopnjo sprememb v načinu poslovanja podjetja in če bo namestitev komercialnih paketov preveč disruptivna, se bo organizacija odločila, da namesto tega razvije lastno programsko rešitev (Tilley & Rosenblatt, 2016, str. 210).

Vsa na novo nameščena programska oprema mora delovati z obstoječimi sistemi. Kot primer lahko vzamemo podjetje, ki že ima obstoječi sistem, s katerim bo novi sistem moral komunicirati in bo na trgu težko najti komercialni programski paket, ki bo pravilno deloval z obstoječim sistemom. V takšnih primerih lahko podjetje razvije lastno programsko rešitev, s katero bo zagotovilo, da bo novo razviti sistem znal komunicirati z že postavljenim obstoječim sistemom (Setende, 2012, str. 1).

Drugi možni razlog za interni razvoj programske opreme je, da mora nov sistem delovati tudi z obstoječo strojno opremo in s starejšimi sistemi. To zna zahtevati posebno zasnovo, nadgradnjo okolja ali programsko opremo, ki bo lahko delovala v okviru omejitev.Sistemski analitik mora obravnavati problem med analizo tehnične izvedljivosti pri preliminarni analizi in v fazi sistemske analize mora še ugotoviti, ali je interni razvoj programske opreme boljša opcija kot nakup že obstoječe programske opreme (Hoffer, George & Valacich, 2020, str. 30).

Z razvijanjem lastnega sistema lahko podjetja usposobijo IT osebje, ki bo razumelo organizacijske poslovne funkcije in potrebe. Podjetja z lastnimi informacijskimi viri in zmogljivostmi zagotavljajo konkurenčno prednost, saj imajo lastno ekipo, s katero se lahko hitro odzovejo na pojave poslovnih težav in priložnosti. Če podjetju primanjkujejo notranji viri, je poslovno podpiranje odvisno od zunanjih izvajalcev.

1.5.2 Zunanje izvajanje

Zunanje izvajanje je prenos razvijanja, delovanja ali vzdrževanja informacijskih sistemov zunanjemu podjetju, ki zagotavlja navedene storitve v zameno za denarno plačilo. Zunanje izvajanje se lahko nanaša na sorazmerno majhne programske naloge, najem programske opreme pri ponudniku storitev, izvajanje osnovnega poslovnega procesa ali upravljanje celotne informacijske funkcije podjetja (Alpar & Saharia, 1995, str. 199).

Tradicionalno so podjetja IT naloge prenašala na zunanje izvajalce kot način nadzora nad stroški in obvladovanja hitrih tehnoloških sprememb. Trend zunanjega izvajanja je vplival tudi na prodajalce programskih paketov, ki so marketing tudi ustrezno prilagodili. Podjetje, ki ponuja zunanje izvajanje, se imenuje ponudnik storitev in nekateri ponudniki se osredotočajo na specifične programske rešitve, nekateri drugi ponujajo poslovne storitve, kot sta obdelovanje naročil in zaračunavanje strankam (Feeny, Lacity & Willcocks, 2005, str. 41). Nekateri drugi ponujajo programske rešitve za celotna podjetja, ki vključujejo in uporabljajo funkcije, kot so računovodstvo, izdelovanje in nadzorovanje zalog.

Ponudniki aplikacijskih storitev so podjetja, ki zagotavljajo programsko aplikacijo ali dostop do te aplikacije z zaračunavanjem uporabe ali naročnine. Običajno tudi ponujajo komercialno dostopno programsko opremo, kot so podatkovne baze. Če se podjetja odločijo za ponudnike aplikacijskih storitev, podjetjem ni treba načrtovati, razvijati, implementirati ali vzdrževati, ker bo za to poskrbel ponudnik (Karimi-Alaghehband & Rivard, 2020, str. 2).

Nekatera podjetja ponujajo internetne poslovne storitve, ki nudijo močne spletno podprte funkcionalnosti, kot so obdelovanje naročil, obračunavanje in uporabljanje odnosov s strankami. Drug izraz za tovrstne ponudnike je upravljano gostovanje, saj zunanje podjetje upravlja sistemsko delovanje. Podjetja, ki ponujajo programsko opremo kot storitev, so razvila plačilne strukture, ki temeljijo na tem, kako stranke v določenem časovnem obdobju uporabljajo to aplikacijo. Obstaja več modelov, vključno s fiksnim plačevanjem, naročnino ali glede na uporabo storitve. Model s fiksnim plačevanjem uporablja določeno provizijo, ki temelji na določeni ravni storitve in uporabniške podpore. Model plačevanja glede na uporabo storitve zaračunava različno provizijo glede na obseg uporabe ali operacij, ki jih izvaja uporabnik preko storitve (Tilley & Rosenblatt, 2016, str. 208).

Ko se podjetje odloči za zunanje izvajanje računalniških storitev, naredi pomemben korak, ki lahko vpliva na vire, poslovanje in donosnost podjetja. Ključne naloge informacijskih sistemov je potrebno oddati zunanjim izvajalcem le, če je rešitev stroškovno privlačna in zanesljiva ter ustreza dolgoročni strategiji podjetja. Zunanje izvajanje lahko poleg dolgoročnih strateških posledic sproži še dodatne dileme. Podjetje mora na primer prenesti občutljive podatke zunanjemu ponudniku storitev in mu zaupati, da bodo ohranili varnost, zaupnost in kakovost. Prav tako mora podjetje pred začetkom uporabe zunanjega izvajalca skrbno pregledati vprašanja v zvezi z zavarovanjem, potencialno odgovornostjo, licenciranjem in lastništvom informacij, garancijami ter reševanjem problemov.

1.5.3 Proizvajalci paketne programske opreme

Podjetja se lahko tudi odločijo, da ne bodo najela zunanjih izvajalcev za pridobitev nove programske opreme in imajo potem na voljo kupovanje obstoječe komercialne programske opreme, ki je hkrati alternativa lastnemu internemu razvoju. Prednosti nakupa komercialne programske opreme pred lastnim razvijanjem vključuje nižje stroške, manj porabljenega časa za uvajanje sistema, dokazano zanesljivost in učinkovitost, manj potrebe po osebju za

tehnično vzdrževanje in ponujanje novejših nadgradenj sistema, ki jih ponuja prodajalec (Tilley & Rosenblatt, 2016, str. 212).

Ker številna podjetja uporabljajo programske pakete, prodajalci opreme stroške razvijanja porazdeljujejo na stranke, ki so pakete kupile. V primerjavi z internim razvojem so komercialni programski paketi vedno cenejši, zlasti pri začetnih naložbah. Kljub temu, da so začetni stroški manjši, lahko potem kupljena oprema vključuje dodatne stroške, ki so lahko posledica prekinitve poslovanja in spreminjanja poslovnih procesov. Ob nakupu je oprema že oblikovana, sprogramirana, testirana in dokumentirana. Programska oprema mora biti nameščena in integrirana v sistemsko okolje, kar lahko traja precej časa. Čeprav je izvajanje hitrejše, lahko pomeni še dodatne stroške, kot so usposabljanje osebja in modifikacija programske opreme (Tilley & Rosenblatt, 2016, str. 213).

Podjetja, ki uporabljajo komercialne programske pakete, lahko zmanjšajo število razvijalcev in sistemskih analitikov v njihovi organizaciji. Hkrati se lahko tudi informacijsko osebje osredotoča na dele sistema, ki jih programski paket ne more izpolniti.

Ponudniki programske opreme redno nadgrajujejo svoje rešitve z dodajanjem novih izboljšav in inovativnosti pri vsaki novi različici opreme. Pri nadgrajevanju prodajalci upoštevajo predloge trenutnih uporabnikov, ko načrtujejo prihodnje nadgradnje.

1.5.4 ERP

Celovita programska rešitev ERP (angl. Enterprise Resource Planning) je programska oprema, ki jo podjetja uporabljajo za upravljanje in integracijo pomembnih delov svojega podjetja. Številne programske aplikacije ERP so za podjetja pomembne, ker jim pomagajo pri načrtovanju virov, tako da v en sam sistem vključijo vse procese, ki so potrebni za vodenje njihovih podjetij. ERP programski sistem lahko vključuje tudi načrtovanje, nakup zalog, prodajo, trženje, finance, človeške vire in še več (Ruivo, Johansson, Sarker & Oliveira, 2020, str. 2).

Nekatere prednosti ERP vključujejo prost pretok komunikacije med poslovnimi področji, en sam vir informacij in natančno poročanje podatkov v realnem času. Z ERP programsko opremo ima vsak oddelek še vedno svoj sistem, vendar je do vseh sistemov mogoče dostopati prek ene aplikacije z enim vmesnikom (Hoffer, George & Valacich, 2020, str. 31). Programska oprema tudi omogoča različnim oddelkom lažjo komunikacijo in izmenjavo informacij s preostalimi deli podjetja. Sistem zbira informacije o dejavnosti in stanju različnih oddelkov ter te informacije daje na voljo drugim, kjer jih je mogoče produktivno izrabiti.

Ponudbe ERP so se skozi leta razvile od tradicionalnih modelov programske opreme, ki uporabljajo fizične odjemalske strežnike do programske opreme v računalniškem oblaku, ki ponuja oddaljeni spletni dostop.

Podjetja uvajajo ERP iz različnih razlogov, kot so širjenje poslovanja, zmanjšanje stroškov in izboljšanje poslovanja. Vključevanje in avtomatizacija poslovnih procesov odpravlja odvečnosti, izboljšuje natančnost in povečuje produktivnost. Oddelki z medsebojno povezanimi procesi lahko zdaj sinhronizirajo delo za doseganje hitrejših in boljših rezultatov. Nekatera podjetja imajo koristi od izboljšanega poročanja podatkov v realnem času iz enega samega vira. Natančno in popolno poročanje pomaga podjetjem, da ustrezno načrtujejo, napovedujejo in sporočajo stanje poslovanja organizaciji zainteresiranim stranem, kot so delničarji (Rainer & Prince, 2021, str. 324). ERP sistemi podjetjem omogočajo hiter dostop do potrebnih informacij za stranke, prodajalce in poslovne partnerje, kar prispeva k večjemu zadovoljstvu strank in zaposlenih, hitrejšemu odzivu ter večji stopnji natančnosti. S tem se tudi stroški pogosto zmanjšujejo, saj podjetje deluje boljučinkovito.

ERP sistemi ne odpravijo vedno neučinkovitosti znotraj podjetja, običajno ne dosežejo ciljev, ki so vplivali na namestitev te rešitve, ker podjetje ne želi opustiti starih delovnih procesov, ki niso združljivi s to rešitvijo. Nekatera podjetja tudi nerada opuščajo staro programsko opremo, ki je v preteklosti dobro delovala. Ključno je preprečiti, da se ERP projekti ne razdelijo na številne manjše projekte, ker to lahko povzroči večje stroške (Setende, 2012, str. 7).

1.5.5 Računalništvo v oblaku

Računalništvo v oblaku nam ponuja razpoložljivost računalniških virov, zlasti shranjevanja podatkov in računalniške moči brez neposrednega aktivnega upravljanja s strani uporabnika (Mell & Grance, 2011, str.2). Večji računalniški oblaki, ki danes prevladujejo, imajo pogosto funkcije razporejene na več lokacijah; če je povezava z uporabnikom razmeroma blizu, ji je dodeljena bližja lokacija strežnika namesto glavnega. Najpogostejši modeli storitev, ki nam računalništvo v oblaku ponuja, so (De la Prieta, Rodríguez-González, Chamoso, Corchado & Bajo, 2019, str. 226):

- Programska oprema kot storitev – Aplikacije, ki gostujejo v računalniškem oblaku in so dostopne preko spletnega brskalnika, namenskega namiznega odjemalca ali API-ja, ki se integrira z namiznim ali mobilnim operacijskim sistemom. V večini primerov se uporabo te storitve plačuje preko naročnin ali glede na uporabo. Nove posodobitve se lahko izkoristijo takoj, ko jih ponudnik doda, ne da bi uporabniki organizirali nadgradnjo. Podatki aplikacije so shranjeni v oblaku in v primeru okvare ali izpada se podatki ne izgubijo,
- Platforma kot storitev - razvijalcem ponuja strojno opremo, celoten nabor programske opreme, infrastrukturo in razvojna orodja. Ponudniki te storitve v svojem podatkovnem centru gostijo vse; strežnike, omrežja, programsko opremo operacijskih sistemov in podatkovno bazo. Te storitve so pogosto zgrajene okoli virtualiziranega računalniškega okolja (Song, 2020, str. 84),

– Infrastruktura kot storitev - Omogoča dostop do osnovnih računalniških virov, kot so fizični in virtualni strežniki, omrežje, shranjevanje podatkov prek interneta. Končnim uporabnikom omogoča, da lahko po potrebi prilagajajo in zmanjšujejo vire; imajo pa v primerjavi s predhodno opisanima modeloma najmanjši nadzor nad računalniškimi viri v oblaku.

Obstaja tudi več tipov računalniških oblakov (Jula, Sundararajan & Othman, 2014, str. 3812; Mell & Grance, 2011, str. 3), kot so javni oblaki, zasebni oblaki, oblaki za skupnosti in hibridni oblaki. Pri javnih oblakih ponudniki storitev ponujajo uporabnikom računalniške vire, kot so programske storitve preko oblaka, virtualne stroje, strojne opreme ter popolne infrastrukture in razvojne platforme za podjetja preko interneta. Našteti viri so lahko brezplačno dostopni ali pa se prodajajo na podlagi naročnin ali plačila glede na uporabo. Ponudnik javnega oblaka ima v lasti in upravlja ter prevzema vso odgovornost za podatkovne centre, strojno opremo in infrastrukturo, na katerih se izvajajo obremenitve njihovih strank. Javni oblak je večnajemniško okolje, kar pomeni, da si infrastrukturo podatkovnega centra ponudnika oblaka delijo vse stranke v tem oblaku.

Zasebni oblak je okolje, v katerem so celotna infrastruktura in računalniški viri namenjeni samo eni stranki in dostopni samo njej. Običajno tovrstne računalniške oblake gostijo na lokaciji podjetja, lahko pa gostujejo tudi na infrastrukturi neodvisnega ponudnika oblaka ali pa ga zgradijo na najeti infrastrukturi, nameščeni v zunanjem podatkovnem centru. Mnoga podjetja se odločijo za zasebni oblak pred javnim, ker je zasebni oblak boljši način za izpolnjevanje njihovih zahtev ali pa ker se njihovi delovni procesi ukvarjajo z zaupnimi dokumenti, intelektualno lastnino, osebnimi podatki, zdravstvenimi kartotekami, finančnimi podatki ali drugimi občutljivimi podatki (Abdurachman, Gaol & Soewito, 2019,str. 1327). V oblakih za skupnosti je infrastruktura namenjena uporabi izključno določeni skupnosti potrošnikov iz organizacij. Lahko je v lasti ene ali več organizacij, ki so v skupnosti in upravljajo računalniški oblak.

Hibridni oblak je kombinacija javnega in zasebnega okolja v oblaku. Povezuje zasebne storitve v oblaku in javne oblake v enotno, prilagodljivo infrastrukturo za izvajanje aplikacij in delovnih obremenitev organizacije. Cilj hibridnega oblaka je vzpostaviti mešanico javnih in zasebnih virov v oblaku, kar daje organizaciji prožnost, da izbere optimalni oblak za vsako aplikacijo ali delovno obremenitev. To organizaciji omogoča, da svoje tehnične in poslovne cilje doseže bolj učinkovito in tudi bolj stroškovno učinkovito, kot bi lahko samo z javnim ali zasebnim oblakom.

1.5.6 Odprtokodna programska oprema

Odprtokodna programska oprema je brezplačno dostopna, ne samo kot končni izdelek, temveč tudi njihova izvorna koda. Tako programsko opremo razvija skupnost zainteresiranih namesto zaposlenih v določenih podjetjih. Odprtokodna programska oprema opravlja enake funkcije kot komercialna programska oprema, kot so operacijski sistemi, elektronska pošta, podatkovne baze, spletni brskalniki in podobno. Ker so odprtokodni projekti

brezplačni, lahko razvijalci in vzdrževalci služijo tako, da zagotavljajo vzdrževanje in druge funkcionalnosti ali ponudijo osnovno različico projekta brezplačno in prodajajo bolj razširjeno različico (Hoffer, George & Valacich, 2020, str. 34).

1.6 Analiza stroškov in koristi

Analiza stroškov in koristi je sistematičen postopek, s katerim podjetja analizirajo, katere odločitve naj sprejmejo in katere zavržejo. Ni metoda razvoja, ampak je ločena analiza, ki se uporablja pri vseh metodah razvijanja. Analiza stroškov in koristi sešteje potencialne koristi, pričakovane od situacije ali dejanja in nato odšteje skupne stroške, ki so povezani s to odločitvijo (King & Schrems, 1978, str. 31). Analiza stroškov in koristi vključuje merljive finančne metrike, kot so zasluženi prihodki ali prihranjeni stroški zaradi odločitve o nadaljevanju projekta. Lahko vključuje tudi nematerialne koristi in stroške ali učinke odločitve, kot sta morala zaposlenih in zadovoljstvo strank.

Pred gradnjo oziroma začetkom novega projekta podjetja najprej opravijo analizo stroškov in koristi, da ocenijo vse potencialne stroške in prihodke, ki jih podjetje lahko ustvari s projektom. Izid analize bo določil, ali je projekt finančno izvedljiv ali bi moralo podjetje nadaljevati drug projekt. V mnogih modelih bo analiza stroškov in koristi upoštevala tudi oportunitetne stroške v postopku odločanja. Priložnostni stroški so alternativne koristi, ki bi jih lahko izkoristili pri izbiri ene alternative nad drugo. Z drugimi besedami, oportunitetni strošek je opuščena ali zamujena priložnost zaradi izbire ali odločitve (Mishan & Quah, 2020, str. 65). Upoštevanje oportunitetnih stroškov omogoča vodjem projektov, da pretehtajo koristi, ki jih prinašajo alternativni načini delovanja, in ne zgolj trenutna pot ali izbira, ki se upošteva v analizi.

Analitik ali vodja projekta mora uporabiti monetarno meritev za vse postavke na seznamu stroškov in koristi, pri čemer mora biti posebej pozoren, da ne podcenjuje stroškov ali precenjuje koristi. Rezultate skupnih stroškov in koristi je potrebno primerjati kvantitativno, da se ugotovi, ali koristi odtehtajo stroške. Če je smiselno, potem nadaljuje s projektom, v nasprotnem primeru pa mora podjetje pregledati projekt, da ugotovi ali lahko prilagodi povečanje koristi in zmanjšanje stroškov, da bo projekt sposoben za preživetje (Layard, 1994, str. 266).

Pri analizi stroškov in koristi je v postopek vgrajenih več napovedi in če je katera od napovedi netočna, so lahko rezultati vprašljivi. Za projekte, ki vključujejo majhne do srednje velike kapitalske izdatke in niso omejeni na čas dokončanja, lahko poglobljena analiza stroškov in koristi zadostuje za dobro informirano in racionalno odločitev. Pri zelo velikih projektih z dolgoročnim časovnim obdobjem morda analiza stroškov in koristi ne bo upoštevala pomembnih finančnih težav, kot so inflacija, obrestne mere, različni denarni tokovi in vrednosti denarne valute (Drèze & Stern, 1987, str. 911). Za takšne situacije bi lahko bila primernejša metoda analize proračuna kapitala vključno z neto sedanjo vrednostjo. Ena od prednosti uporabe neto sedanje vrednosti za odločanje o projektu je, da uporablja alternativno stopnjo donosa, ki bi jo lahko zaslužili, če projekt nikoli ne bi bil

izveden. Kljub temu se rezultat odločitve, sprejete na podlagi neto sedanje vrednosti, lahko razlikuje glede na vrednosti obrestne mere in je lahko rezultat tudi zavajajoč, ker po eni strani neto sedanja vrednost ne more biti neodvisna od vrednosti kapitalske naložbe, hkrati pa presežka dobička nad stopnjo donosa ni mogoče razložiti brez upoštevanja življenjske dobe koristnosti (Juhász, 2011, str. 52).

2 STRGANJE PODATKOV

V tem poglavju bomo opisali pojem spletnega strganja podatkov, kaj to sploh je in na kakšne načine lahko ta postopek opravljamo. Pojem se vrti okoli programskega pridobivanja različnih podatkov iz spletnih strani, pri čemer se pojavi tudi dilema o pravicah in vprašanje, če lahko sploh to izvajamo na določenih straneh ali ne. Zaradi omenjene dileme bomo tudi prikazali, kakšna je pravna podlaga pri izvajanju spletnega strganja.

2.1 Splošna definicija

Spletno strganje podatkov je tehnika oziroma način pridobivanja podatkov iz raznih spletnih virov in shranjevanje teh podatkov v datoteke ali podatkovne baze za kasnejšo uporabo in analizo. Spletne podatke pridobijo z uporabo protokola za prenos hiperteksta (angl. Hyper-Text Transfer Protocol, v nadaljevanju HTTP) ali pa tudi preko uporabe spletnih brskalnikov bodisi ročno preko uporabnikov ali samodejno z roboti in spletnimi pajki. Ker se na spletnih mestih nenehno in vsakodnevno ustvarja ogromna količina raznovrstnih podatkov, je spletno strganje postalo splošno znano kot učinkovita in močna tehnika zbiranja podatkov (Zhao, 2017, str. 2).

Spletno strganje je oblika podatkovnega rudarjenja. Splošni cilj te tehnike je pridobivanje podatkov iz spletnih virov ter njihova transformacija v razumljive oblike kot so Excel, podatkovne baze ali tekstovne datoteke (Saurkar, Pathare & Gode, 2018, str. 364).

Definiramo ga lahko tudi kot postopek sistematičnega pridobivanja in kombiniranja vsebin, ki nas zanimajo. V takem procesu bo programski agent, sicer znan tudi kot spletni robot, posnemal interakcijo med človekom in brskanjem po spletu. Korak za korakom bo robot dostopal do spletnih mest, kolikor jih je navedenih, razčlenil bo vsebino teh spletnih strani in izvlekel podatke, ki nas zanimajo ter pretvoril podatke v najprej določeno strukturo (Junjoewong, Sangnapachai & Sunetnanta, 2018, str. 2; Dewi & Chandra, 2019, str. 445).

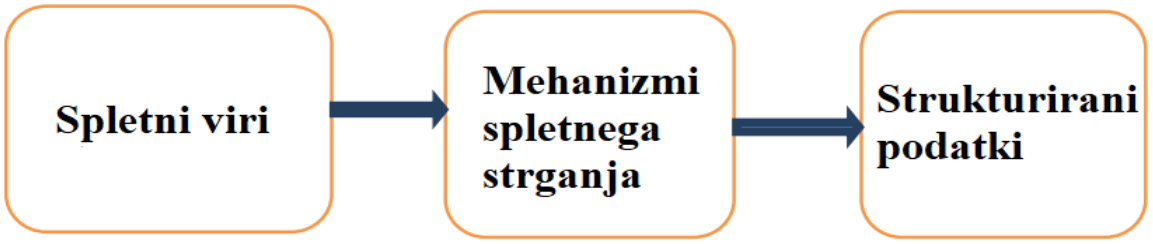
Na sliki 3 je prikazan potek spletnega strganja podatkov iz spletnih virov in te korake definiramo po sledečem (Glez-Peña, Lourenço, López-Fernández, Reboiro-Jato & Fdez-Riverola, 2014, str. 790):

1. Dostop do spletnega mesta - Robot za spletno strganje mora najprej vzpostaviti komunikacijo s ciljnim spletnim mestom preko protokola HTTP. HTTP je tekstovni internetni protokol, ki koordinira transakcije med odjemalcem, običajno spletnim

brskalnikom in strežnikom. Pri uporabljanju HTTP zahtevkov sta najpogostejši metodi GET, ki se uporablja pri zahtevah za vire, in POST, ki se uporablja pri oddaji spletnih obrazcev in nalaganju datotek. V HTTP zahtevku je tudi pomembno polje User-Agent, katero strežnik pregleda, da ugotovi, kdo dostopa do njihove vsebine. Poleg tega morajo spletni strgalniki podatkov tudi upoštevati pogoje uporabe, ki so določeni na spletni strani v datoteki robots.txt. Datoteka navaja, do katerih virov roboti ne smejo dostopati in kako morajo tudi upoštevati primernost časa za strganje podatkov, da slučajno ne pride do preobremenitve strežnika.

2. Razčlenjevanje HTML kode in pridobivanje vsebine - Ko robot pridobi HTML dokument od spletne strani, lahko spletni strgalnik izvleče željeno vsebino. Za ta namen lahko uporabimo ujemanje regularnih izrazov (angl. Regex) za pridobivanje željene vsebine ali pa lahko poleg tega vzpostavimo še dodatno programsko logiko za pridobitev podatkov. Alternativno lahko uporabimo že obstoječe programske knjižnice za razčlenjevanje HTML kode. Logika za pridobivanje podatkov mora biti tako narejena, da je odporna na morebitne spremembe v HTML kodi spletne strani in da lahko, če pride do sprememb, še vedno pridobi podatke.
3. Ustvarjanje končnega rezultata - Glavni cilj je preoblikovati pridobljeno vsebino v strukturirano obliko, ki je primerna za nadaljnje analize in shranjevanje. Nekatera orodja vnaprej upoštevajo naknadno obdelovanje rezultatov in zagotavljajo tudi podatkovne strukture, ki se lahko shranijo v pomnilnik računalnika ali pa v tekstovne rešitve, kot so nizi ali datoteke.

Slika: 3: Potek strganja podatkov



Prirejeno po Saurkar, Pathare & Gode (2018, str. 364)

2.2 Načini strganja podatkov

Načini strganja podatkov iz spletnih virov so sledeči (Sirisuriya, 2015, str. 136):

Kopiraj in prilepi - Ročno kopiranje podatkov iz vira v druge oblike sodi pod pojem spletnega strganja podatkov. Ta metoda ima zelo šibko točko v potencialni človeški napaki in lahko hitro postane dolgočasna pri večjih količinah podatkov.

Tekstovno iskanje in regularni izrazi - To je preprost in zmogljiv način za pridobivanje informacij iz spletnih strani, tehnika temelji na tekstovnem ujemanju po vnaprej določenih pogojih, ki jih definiramo v kodi.

HTTP programiranje - Tehnika se uporablja za pridobivanje podatkov iz statičnih in dinamičnih spletnih strani. Podatke je mogoče pridobiti s pošiljanjem HTTP zahtevkov na oddaljene spletne strežnike.

Razčlenjevanje HTML dokumentov - Strukturirani jeziki za poizvedovanje podatkov, kot sta XQuery in HTQL, se lahko uporabljajo za razčlenjevanje HTML dokumentov in za pridobivanje ter preoblikovanje njihove vsebine.

Razčlenjevanje objektnega modela dokumenta (angl. Document Object Model, v nadaljevanju DOM) - Programi lahko pridobijo dinamične vsebine z uporabno skript na odjemalčevi strani preko spletnih brskalnikov. Kontrole znotraj brskalnikov lahko tudi razčlenijo vsebine spletnih strani.

Programska oprema za strganje podatkov - Na voljo je veliko programskih orodij, s katerimi lahko prilagodimo rešitve za strganje podatkov na spletu. Programska oprema lahkoposkuša samodejno prepoznati podatkovno strukturo spletne strani ali pa z uporabo vmesnika za posnemanje interakcij med uporabnikom in spletno stranjo, s katerim lahko odpravi potrebo po ročnem pisanju kode za strganje podatkov.

Vertikalne zbirne platforme - ustvarjajo in nadzirajo množico robotov brez nujne neposredne prisotnosti uporabnikov. Robustnost platforme se meri s kakovostjo pridobljenih informacij in njihovo razširljivostjo (kako hitro lahko doseže do stotine ali tisoče spletnih mest).

Prepoznavanje semantičnih pripisov - Spletne strani lahko vsebujejo metapodatke ali semantične oznake in pripise, ki jih je mogoče uporabiti za iskanje določenih specifičnih podatkov. Če so pripisi vdelani v stran, lahko to tehniko obravnavamo kot poseben primer razčlenjevanja DOM. V drugem primeru se pripisi, ki so organizirani v semantični plasti, shranjujejo in upravljajo ločeno od spletne strani, tako da lahko strgalniki podatkov pridobijo podatkovno shemo ter navodila s tega sloja, preden izvedejo strganje spletne strani.

Računalniški vid - Z uporabo strojnega učenja in računalniškega vida lahko poskušajo prepoznati in pridobiti informacije iz spletnih strani z vizualno interpretacijo strani.

2.3 Pravna podlaga

Čeprav so raziskovalcem na voljo številna orodja in tehnologije za pomoč pri izvajanju spletnega strganja podatkov (Munzert, Rubba, Meißner & Nyhuis, 2014), je zakonitost le - tega na pravnem področju še vedno sivo območje (Snell & Menaldo, 2016, str. 2). Tukaj zakonitost opredeljujemo kot skladnost med veljavno zakonodajo in pravnimi teorijami. Zaenkrat še ne obstaja zakon, ki bi bil neposredno povezan s spletnim strganjem. Upoštevajo se predvsem temeljne pravne teorije in zakoni, ki obravnavajo kršitev avtorskih pravic, kršitev pogodb, zakon o računalniških prevarah in zlorabah (angl. Computer Fraud and Abuse Act)(Dreyer & Stockton, 2013, str. 3).

2.3.1 Pogoji uporabe

Lastniki spletnih mest lahko učinkovito preprečijo programski dostop do njihovih spletnih mest tako, da to izrecno prepovedo v pravnem dokumentu o pogojih uporabe, ki so objavljeni na spletnem mestu. Neupoštevanje teh pogojev lahko privede do kršitve pogodbe s strani uporabnika spletnega mesta (Dryer & Stockton, 2013, str. 2). Uporabnike spletnega mesta lahko lastniki začnejo preganjati zaradi kršitve pogojev, če uporabniki z lastnikom spletnega mesta sklenejo izrecno pogodbo o spoštovanju pravilnika o pogojih uporabe.

2.3.2 Avtorsko zaščitena gradiva

Strganje in ponovno objavljanje podatkov ali informacij, ki so zaščitene pri lastniku spletnega mesta, lahko privede do kršitve avtorskih pravic (Dryer, & Stockton, 2013, str. 2), tudi če podatki, ki jih uporabniki ustvarijo, niso v lasti spletne strani. Za primer lahko vzamemo spletno trgovino, na kateri uporabniki pišejo mnenja o izdelkih, ta mnenja niso nujno v lasti spletne trgovine. Poleg tega idej tudi ni mogoče zaščititi z avtorskimi pravicami, lahko zaščitimo samo posebno obliko ali predstavitev teh idej. Tako lahko z avtorskimi podatki ustvarjamo povzetke avtorsko zaščitenih podatkov. Lahko še vedno uporabljamo avtorsko zaščiteno gradivo, ampak le v omejenem obsegu po načelu poštene uporabe.

2.3.3 Namen strganja po spletu

Kakršna koli nezakonita ali goljufiva uporaba podatkov, pridobljenih s spletnim strganjem, je prepovedana z več zakoni. Na primer, osebo, ki dostopa do podatkov s spleta, za katere je znano, da so zaupni in zaščiteni, se lahko preganja po zakonu o računalniških prevarah in zlorabah (Dryer & Stockton, 2013, str. 2). V spletu se pogosto zgodi, da nekdo zavestno dostopa do bolj ”kakovostnih” vsebin in jih nato preprodaja ali nadaljuje z dostopom do vsebine po nepooblaščenem kanalu, potem ko je od lastnika spletnega mesta prejel poziv ”prenehaj in opusti” (Snell & Menaldo, 2016, str. 2).

2.3.4 Poškodbe spletnega mesta

Če spletno strganje preobremenjuje ali celo poškoduje spletno mesto ali strežnik, se lahko odgovorno osebo za nastalo škodo kazensko preganja (Dryer, & Stockton, 2013). Škoda mora biti materialna in enostavno dokazljiva na sodišču, da je lastnik spletnega strežnika oziroma mesta upravičen do finančnega nadomestila.

2.3.5 Zasebnost posameznikov

Zbiranje podatkov iz spletnih strani lahko nenamerno ogrozi zasebnost posameznikov, ki uporabljajo oziroma obiskujejo te spletne strani (Mason, 1986, str. 6). S primerjanjem podatkov, zbranih s spletnih mest, s podatki iz drugih spletnih strani in nespletnimi viri lahko raziskovalci nenamerno razkrijejo identiteto tistih uporabnikov, ki so ustvarili

podatke (Ives & Krotov, 2006, str. 598). Tudi če zasebnost posameznika ni kršena, je težava v tem, da morda uporabniki spletnih mest niso dovolili uporabe svojih podatkov tretjim osebam. Uporaba teh podatkov brez privolitve je kršitev pravic raziskovalcev (Buchanan, 2017).

2.3.6 Zasebnost organizacij in poslovne skrivnosti

Tako kot imajo posamezniki pravico do zasebnosti, imajo tudi organizacije pravico, da nekatere vidike svojega poslovanja ohranijo zaupno (Mason, 1986, str. 7). Samodejno strganje podatkov spletnih mest lahko nenamerno razkrije poslovne skrivnosti ali zaupne podatke o organizaciji. Prav tako lahko razkrije nekatere podrobnosti in morebitne napake v načinu shranjevanja podatkov na spletnem mestu (Ives & Krotov, 2006, str. 603). Vse to lahko škoduje ugledu podjetja, ki stoji za spletnim mestom in povzroči materialne in finančne izgube.

2.3.7 Zmanjševanje vrednosti za organizacije

Če nekdo dostopa do spletnega mesta brez spletnega vmesnika, namenjenega brskanju po strani, se posledično uporabnikom ne bodo prikazali izpostavljeni oglasi, ki jih spletno mesto uporablja za monetizacijo svoje vsebine. Poleg tega lahko projekt spletnega strganja privede do ustvarjanja podatkovnih izdelkov, ki brez kršitve avtorskih pravic manjša verjetnost, da stranka kupi podatkovni izdelek od prvotnega lastnika podatkov. Z drugimi besedami, podatkovni izdelek, ustvarjen s pomočjo spletnega strganja, lahko neposredno ali posredno konkurira poslovanju lastnika spletnega mesta (Hirschey, 2014, str. 915). Vse to lahko vodi do finančnih izgub lastnika spletnega mesta ali vsaj do nepravične razdelitve vrednosti iz lastništva podatkov (Mason, 1986, str. 7).

3 ANALIZA TRENUTNEGA SISTEMA

V tem poglavju bom analiziral, kakšno funkcijo ima trenutni sistem na oddelku, kaj sploh omogoča in zakaj je pomemben pri vsakodnevnih opravilih zaposlenih, pokazal bom tudi kako je postavljen trenutni sistem na izbrani banki ter še opisal na kakšen način deluje, s kakšnimi težavami se uporabniki spopadajo pri uporabi sistema, s kakšnimi tehnologijami je bil ustvarjen in kakšne so zahteve, ki jih bo potrebno upoštevati pri izgradnji novega sistema.

3.1 Vloga IS na oddelku

Natančnost in ažurnost podatkovnih baz je na naši izbrani banki ključnega pomena za delovanje modelske infrastrukture. Makroekonomski modeli so seveda ocenjeni na zgodovinskih podatkih, vendar pa lahko nova realizirana vrednost bistveno vpliva na prihodnje napovedane vrednosti, zato je izjemno pomembno, da so novi razpoložljivipodatki kar se da hitro na voljo za pripravo osveženih napovedi. Na izbranem oddelku

znotraj banke vsakodnevno obdelujejo veliko količino podatkov, saj pripravljajo izvedene časovne vrste, zato se veliko obdelav izvaja ponoči, da strežnika ne obremenjujejo čez dan, ko ga uporabljajo za druge naloge. Če bi prišlo do napak pri vnosu podatkov v bazo, kar je osnova za vse nadaljnje izračune, bi to lahko vodilo do velikih zamud, saj bi se izvedba programa zaustavila in postopek izračuna bi morali ponoviti tekom delovnega dne. Do podobnih težav bi prišlo, če bi prenos podatkov trajal predolgo, saj bi morali ostale obdelave prestaviti. Poleg avtomatiziranih procesov pa osvežene podatke občasno potrebujejo tudi takoj po objavi, zaradi priprave raznih poročil, zato mora obstajati tudi takojšnji zagon prenosa podatkov. Skrajšan čas prenosa podatkov v podatkovne baze jim tako omogoča več časa za analizo in sprotno obveščanje odgovornih o spremembah. Polegčasovnega poteka prenosa podatkov je pomembno tudi prekrivanje časovnih vrst, če pride do spremembe metodologije na viru. Tudi to težavo je bilo s posodobljenim prenosom podatkov mogoče omiliti, saj sedaj ob vsakem prenosu program sam pregleda ujemanje časovne vrste s trenutnimi podatki v bazi in preveri, če je struktura časovne vrste na viru enaka tej v podatkovni baze oddelka, sicer pa na neujemanje opozori.

Na oddelku uporabljajo predvsem javno dostopne podatke, ki so objavljeni na različnih spletnih straneh. Da zaposleni pri iskanju podatkov izgubijo čim manj časa in da pri prenašanju podatkov s spleta ne prihaja do napak in neskladnosti, podatke, ki jih uporabljajo za spremljanje makroekonomskih gibanj, zbirajo na sistematičen način, jih prenašajo na dnevni ravni in zagotavljamo čim bolj nemoteno delovanje. Analitiki tako lahko dostopajo do urejenih časovnih vrst s pripadajočimi opisi, s pomočjo analitičnega orodja EViews pa so že vnaprej pripravljeni tudi določeni programi za preračun in prikazovanje podatkov.

Pri delovanju oddelka je pomembno, da imajo čim več gradiva vnaprej pripravljenega, tako da se v primeru predvsem izrednih nalog lahko hitro odzovejo in v relativno kratkem času naredijo zanesljivo analizo. Pri tem je pomembno, da imajo avtonomijo pri informacijskem sistemu za zajem podatkov, saj jim to omogoča, da se lahko hitro odzivajo na spremembe na viru podatkov in da si lahko ob poljubnem času osvežimo podatke z zadnjimi razpoložljivimi. Podatki, ki jih večinoma uporabljajo, so specifični v tem, da so javno objavljeni in niso v neki vnaprej predpisani obliki oz. formatu, zato se mora informacijski sistem ves čas prilagajati spremembam na viru.

Če informacijskega sistema ne bi imeli organiziranega na ta način, bi analitiki porabili veliko več časa za iskanje in zbiranje podatkov, isto opravilo iskanja prenašanja podatkov bi se hkrati odvijalo pri več uporabnikih. Isti podatki bi se shranjevali na različne lokacije mrežnih ali lokalnih diskov, pri tem pa bi bili podatki zaradi različnega časa ažuriranja lahkorazlični. Poleg tega bi bila verjetnost napake pri ločenih ročnih prenosih s spleta veliko večja.

Če podatki niso pravočasno in zanesljivo posodobljeni, to predstavlja tveganje za neustrezno in zavajajoče informiranje ostalih v banki ter javnosti.

3.2 Opisniki

Opisniki so navadne tekstovne datoteke oziroma na starem sistemu Excel datoteke, ki hranijo vse informacije o časovnih serijah glede na vir in v kateri sklop podatkov sodijo te časovne serije. Hranijo se informacije kot so:

- Interno ime časovne serije - pod katerim imenom se shrani v lokalno podatkovno bazo),
- Identifikacijski ključ na viru – kombinacija dimenzij, ki sestavlja celotno časovno serijo na viru,
- Opis časovne serije – kaj ta serija predstavlja,
- Opis dimenzij - kaj predstavljajo različne vrednosti dimenzij.

3.3 Trenutna informacijska rešitev

Rešitev, ki je trenutno postavljena v naši izbrani banki, je delujoča že nekaj let, a je zelo časovno potratna, nepregledna, težko je razvozlati, kje je kaj šlo narobe, poleg avtomatiziranih prenosov podatkov iz spletnih virov je tudi veliko zajemov, ki jih je potrebno ročno izvajati. Rešitev je tudi v celoti decentralizirana in je težko spremljati, kdaj in od kod so nekateri podatki prišli, kdo jih je vnesel; nekateri podatki, ne pa vsi, se ne zapisujejo v centralno podatkovno bazo banke, ampak se shranjujejo v več lokalnih bazah na internem omrežju.

Vsi trenutno postavljeni zajemi so narejeni v različnih tehnologijah in so na različne načine postavljeni. Na banki trenutno obdelujejo podatke treh večjih ponudnikov in še dodatne podatke, ki jih vnašajo ročno preko različnih oblik virov. Trije glavni viri so sledeči:

- Statistični Urad Republike Slovenije (SURS),
- Eurostat,
- European Central Bank Statistical Data Warehouse (ECB SDW).

Največ podatkov se zbira iz zgoraj naštetih virov in še iz dodatnih virov, ki so na voljo na različnih spletnih straneh, nekatere pridobijo tudi od drugih institucij preko elektronske pošte. Pri glavnih virih je zajem podatkov v celoti avtomatiziran, vendar je za vsak vir drugačna postavljena rešitev, rešitve pa si med seboj niso podobne. Za dodatne vire izven glavnih (Ministrstvo za finance, The Economist, itd.) pa postopek ni avtomatiziran in morajo uporabniki ročno vnašati podatke, vendar so v primerjavi z glavnimi viri ti podatki bistveno manj obsežni. Prenos podatkov se izvaja na dnevni ravni ob določenih urah dneva, neavtomatizirane zajeme pa uporabniki izvajajo samo, ko so novi podatki na voljo pri viru. Na oddelku zaposleni shranjujejo tudi lastne interne izračune, ki so sestavljeni iz podatkov, pridobljenih od zgoraj navedenih virov.

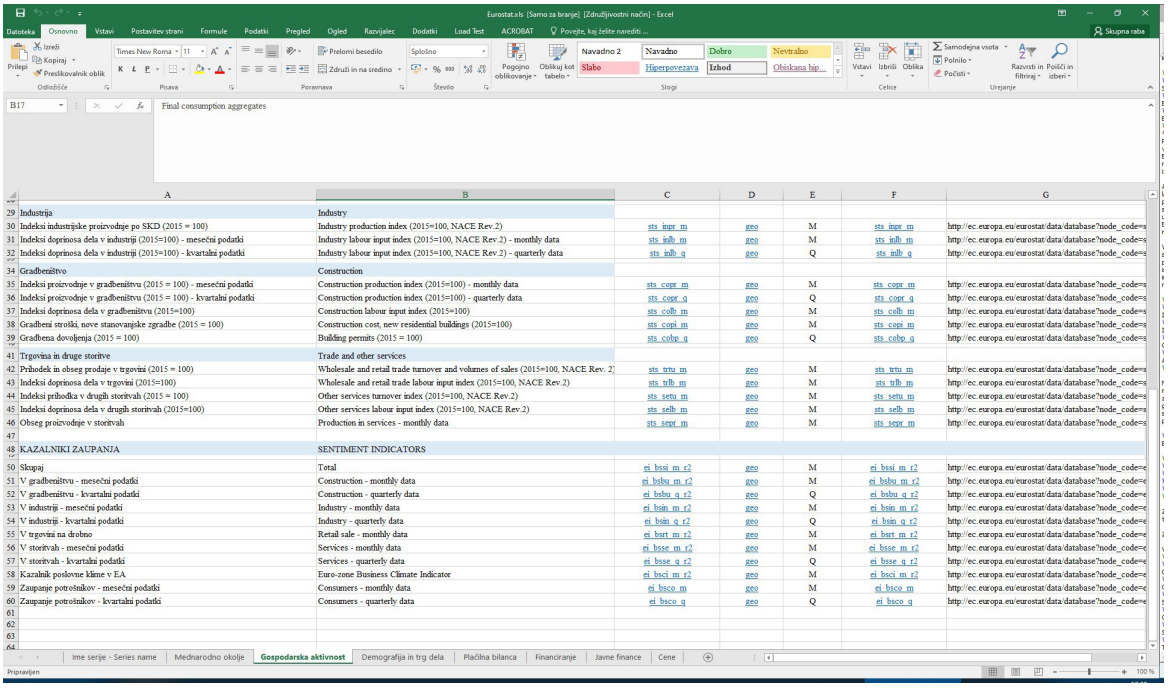
Avtomatizirani procesi za posodabljanje podatkovnih baz in grafov se izvajajo na ločenem strežniku, ker postopki pridobivanja podatkov trajajo zelo dolgo in s tem načinom se izognejo čakanju na lastnih službenih računalnikih. Zajemi se začnejo popoldan in zaključijo zgodaj zjutraj, na strežniku se zajemi ne smejo izvajati istočasno kot ostali, ker

če pride do prepletanja, se lahko začnejo pojavljati napake in se posledično podatki ne bi posodobili. Za avtomatizirane prenose podatkov (SURS, Eurostat in ECB SDW) se podatki prenesejo preko različnih aplikacij in nato se shranijo v fizični obliki (Excel) na skupnem omrežnem disku. Na disk se shranijo tudi zato, ker nekateri uporabniki nimajo dostopa do lokalnih podatkovnih baz oddelka in morajo potem uporabljati te Excel datoteke s podatki. Nato sledi postopek, ki je definiran v Visual Basic kodi znotraj različnih Excel datotek, ki prenešene podatke pretvorijo v ustrezen format, ki ga bo EViews prepoznal in nato tudi vpisal v lokalne podatkovne baze. Med trajanjem tega postopka se po potrebi tudi opisniki avtomatsko ustvarijo oziroma se spremenijo, če program zazna spremembe med potekom zajema. Na koncu se po potrebi tudi zažene EViews, kateremu kot parameter dodamo skripte, ki izvedejo dodatne izračune in se to tudi zapiše nazaj v podatkovne baze.

Vsaka procedura pridobivanja podatkov ima svoj način ustvarjanja in spreminjanja opisnikov za časovne vrste za vsak sklop podatkov posebej. Na sliki 4 je prikazan primer Excel datoteke, v kateri so shranjeni vsi sklopi podatkov za vsak vir posebej. Vsak sklop podatkov je vezan na drugo Excel datoteko, v kateri se nahajajo informacije za izbrani sklop podatkov. Viri, ki zajemajo več področij, pa imajo vmesnik, razdeljen po vsebinah, ki so vsaka posebej povezane s svojim opisnikom časovnih vrst.

Na sliki 4 je prikazana tudi trenutna oblika enega izmed opisnikov, kjer uporabniki nimajo nikjer iskalnika, ki bi poskrbel za iskanje željene serije, ampak morajo lastnoročno iskati po opisih, kje bi se lahko ta časovna serija nahajala. V primeru, da uporabnik že ima serijo in želi imeti še druge serije iz tega sklopa podatkov, lahko znotraj programa EViews prebere opis serije in nato poišče ta sklop v opisnikih na skupnem omrežnem disku.

Slika: 4: Primer opisnika za Eurostat



Vir: lastno delo.

Na oddelku imajo tudi datoteko v kateri je shranjen seznam avtomatiziranih grafov, ki jih osvežujejo po vsaki posodobitvi podatkov. Ta procedura posodabljanja grafov se zaključi le, če je možno graf shraniti. Narejeno je tako, da mora za posodobitev grafa procedura imeti na voljo dve datoteki, ki skrbita za uspešno posodobitev. Ena datoteka skrbi za obliko grafa, druga pa za prenos podatkov iz lokalnih baz in nato se iz obeh generira tretja datoteka, ki je skupek prejšnjih dveh. V primeru, da prvi dve datoteki nista na voljo (manjkajo ali ju ima nekdo drug odprti), se graf ne bo posodobil, ker bi prišlo do napake in posledično bi se celoten postopek ažuriranja ustavil. Zaposleni lahko tudi sami posodabljajo samo izbrane grafe in ni nujno, da ažurirajo celoten seznam naenkrat. V ozadju, ko se posodablja, se zažene Visual Basic skript, v katerem se napiše EViews koda za pridobivanje podatkov iz lokalne baze. Naslednji makro, ki se zažene po pridobivanju podatkov, pa podatke, ki jih je prenesel, nato še vstavi v graf.

EViews, prikazan na sliki 5, je sodoben ekonometrični, statistični in napovedni program, ki ponuja zmogljiva analitična orodja. Z uporabo te programske opreme lahko hitro in učinkovito upravljamo s podatki, izdelujemo ekonometrične in statistične analize, ustvarjamo napovedi ali simuliramo modele ter gradimo grafe in tabele za objave ali vključitve v druge aplikacije.

Znotraj programske opreme je dodana tudi možnost pisanja programske kode za avtomatiziranje postopkov, ki jih uporabniki izvajajo s programom.

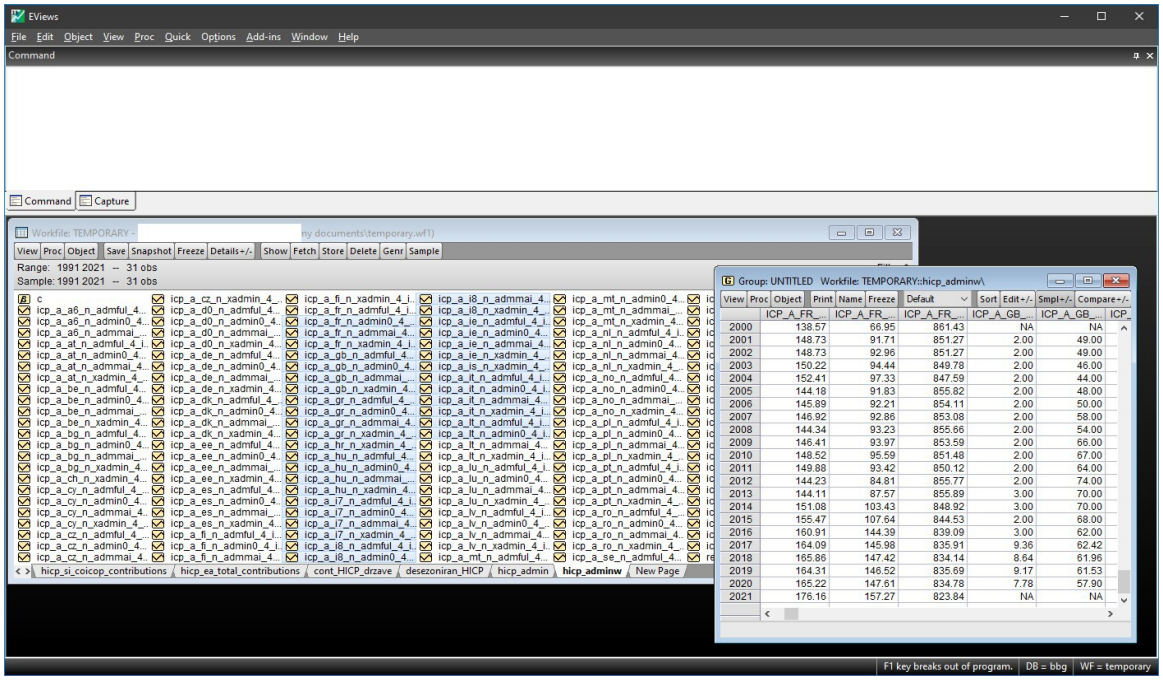
Vsebuje naslednje funkcionalnosti:

- Obdelovanje podatkov;
- Obdelava podatkov časovnih serij;
- Statistične analize;
- Testiranje in vrednotenje;
- Napovedovanje in simuliranje;
- Izdelovanje grafov;
- Programiranje skript za avtomatsko delovanje;
- Podpira zunanje vmesnike in dodatke.

3.4 Pomanjkljivosti obstoječe rešitve

Pri dnevni uporabi sistema na izbrani banki smo preučili vse podrobnosti delovanja obstoječega sistema. Pri sistemu za pridobivanje podatkov se je izkazalo, da je vzdrževanje tega sistema težavno, ker so datoteke, zadolžene za zajem podatkov, zelo razpršene po celotnem skupnem disku. Vsaka datoteka vsebuje delčke kode in to predstavlja problem pri ugotavljanju napak, ker moramo nato pregledovati več datotek naenkrat, da ugotovimo, kje se problem nahaja. Pomanjkljivo deluje tudi sistem sporočanja napak. Ko se zajemi izvajajo na strežniku in če pride do napake, se celoten proces zaustavi, ne prikaže nobene podrobnosti napake, temveč le generično sporočilo, da se je program ustavil. Take situacije predstavljajo velik problem pri vzdrževanju, ker ni jasno, kaj točno je bilo narobe in potem

Slika: 5: Primer uporabe programa EViews



Vir: lastno delo.

uporabniki porabijo veliko časa za ugotavljanje ter reševanje napak, še posebno, ko je koda za zajem razpršena čez več datotek. Vsak zajem za glavne vire (SURS, Eurostat, ECB SDW) je narejen na drugačen način. Za SURS je narejen večinoma v programski kodi Visual Basic in pri tem lahko sami uporabniki ugotavljajo, kje je potrebno izvesti popravila. Pri drugih dveh rešitvah so zajeme postavili zaposleni na oddelku informacijske tehnologije. Ko pride do težav oziroma zahtev po spremembi, morajo zaposleni oddati zahtevek preko internega portala banke in nato morajo počakati, da zadevo na oddelku informacijske tehnologije popravijo. Taka situacija lahko povzroči veliko nevšečnosti, če je zelo malo časa za pripravo analize na novih podatkih, ker je treba potem rezultate predstaviti drugim oziroma napisati povzetke. Tudi ko zaposleni na oddelku informacijske tehnologije popravijo zajeme,so potem še vedno prisotne dodatne procedure, ki se izvajajo na našem oddelku, in še ne pomeni, da je potem zajem v celoti popravljen.

Pri uporabi opisnikov se je izkazalo, da v takšni obliki, kot so trenutno postavljeni, vsebujejo zelo podrobne opise sklopov podatkov oziroma področij in tudi časovne serije vsebujejo mnogo informacij, ki so na voljo tudi na spletni strani ponudnika podatkov. Zaposleni so zadovoljni s takšno obliko, vendar se je izkazalo, da je iskanje v časovnih serijah zelo potratno, še posebno, če ni jasno, kaj točno iščejo ali pa pregledujejo, in če so iskani podatki sploh na voljo. Iskanje je potratno, tudi če poznamo serijo in njeno ime, še vedno je potrebno poiskati, v katerem sklopu se nahaja, in v primeru, da ne vemo natančno, katere podatke potrebujemo, moramo potem vse opise pregledovati. Glavni opisnik je sestavljen tako, da vsebuje samo opise lokalnih podatkovnih baz, ko bazo izberemo pa moramo še prebrskati, pod katero področje sodijo iskani podatki in nato

pregledati še opise sklopov podatkov znotraj področij, da lahko dobimo iskano serijo.

S procesom posodabljanja grafov smo ugotovili tudi eno ključno napako, ki vpliva na celotno izvajanje. V datoteki, kjer je definiran seznam grafov za posodabljanje, grafe ažurira sekvenčo in če naleti na kakšno težavo ne glede, na kateri poziciji je trenutno, se ustavi posodabljanje za vse grafe, ki sledijo tistemu, ki je zaznal težavo. Takšen način izvajanja predstavlja težavo, ko se grafi ažurirajo avtomatsko čez noč in ko zaposleni pridejo na delo, opazijo, da nekateri grafi niso posodobljeni, ker se je proces posodabljanja nekje ustavil. Sledi ugotavljanje, pri kateremu grafu je prišlo do napake in zakaj je prišlo do napake ter popravljanje le te.

Med vsemi naštetimi problemi najbolj izstopa in tudi zaposlene moti predolg čas, ki ga sistem porabi za zajem podatkov ter zapis le teh v podatkovno bazo. Celotni zajem za SURS v povprečju traja minimalno petdeset minut in zelo pogosto se zgodi, da vodstvo zahteva analize že isti dan, ko pridejo podatki in če zajem traja toliko časa, potem izgubijo dobro uro samo zato, da se podatki vpišejo v bazo. Eurostat zajem v celoti traja več kot dvanajst ur, seveda lahko zajamejo samo sklope potrebnih podatkov, a tudi ta proces je časovno potraten. Pri enem izmed zajemov pa se je zgodilo, da so spreminjali način pridobivanja podatkov in jim nato ta zajem sploh ni deloval več mesecev. Zajem je bil narejen na oddelku informacijske tehnologije in tudi tamkajšnji zaposleni napake niso znali popraviti. Odzivnost in hitrost prenosa podatkov je ključnega pomena na oddelku in ta hkrati predstavlja največji problem, s katerim se soočajo na dnevnem nivoju.

3.5 Zahteve za nov sistem

Uporabniki so se odločili, da je prišel čas za zamenjavo sistema in so izrazili željo po zamenjavi oziroma nadgraditvi obstoječega sistema. Izrazili so željo, da bi bil nov sistem popolnoma centraliziran, da bi obstajala sledljivost dela pri uporabljanju sistema, saj bi s tem rešili problem vprašanj, kdo kaj dela, kam se podatki zapisujejo in od kod prihajajo. Hitrost izvajanja je na oddelku v banki ključnega pomena, saj v večini primerov podatki pridejo pozno, kar pomeni, da imajo zaposleni majhen časovni okvir za izdelavo analiz, če zajem podatkov traja predolgo. Izpostavili so tudi, da hočejo imeti bolj robusten modul za poročanje o napakah pri izvajanju zajemov podatkov, ker v trenutnem stanju je zelo težko razločiti, kje in zakaj je prišlo do napake, ker se nikjer ne pojavi okno ali zapis, ki bi vseboval bolj podrobno razlago, zakaj sistem ne deluje več, in je prepuščeno zaposlenim, da sami raziskujejo, na katerem koraku je prišlo do težave, kar predstavlja dodatno izgubo časa.

Veliko je tudi zajemov, ki jih morajo ročno delati, ker trenutni sistem ni dovolj fleksibilen, da bi podpiral obravnavo teh datotek oziroma virov. Pri ročnem vnosu so zaposleni dodatno obremenjeni, izgubljajo čas in lahko se tudi pojavi človeška napaka pri vnašanju podatkov v podatkovno bazo. Sistem vsebuje modul za avtomatsko ustvarjanje in posodabljanje grafov

s časovnimi serijami, ampak za potrebe prve faze prenove se bomo osredotočili samo na zajeme podatkov.

Izrazili so tudi, da nočejo vsega spremeniti in bi radi, da se nekatere določene funkcije in module ohrani pri gradnji ter uporabi novega sistema. V trenutni fazi nočejo narediti preskoka na centralno bazo banke, ampak želijo, da se ohranijo lokalne baze znotraj oddelka. Poleg tega hočejo tudi obdržati obstoječe opisnike (datoteke z informacijami o časovnih serijah) v taki obliko kot so, ker že dolgo delajo s takim zapisom in ne potrebujejo nobene nadgradnje. Novo postavljen sistem bo moral primarno rešiti problem hitrosti in sledljivosti izvajanja sistema.

Prednost, ki jo bo sistem prinašal, je seveda boljša produktivnost in efektivnost zaposlenih. V našem primeru, kjer delamo analizo, velja, da je potrebno upoštevati tudi varnostne politike banke o varstvu podatkov, kar nam postavlja dodatne zahteve pri odločitvi o novem sistemu. Na oddelku je večina podatkov, ki jih uporabljajo, zaupne narave in morajo biti seveda zaščiteni pred zlorabami in neavtoriziranimi dostopi znotraj banke ter zunanjimi entitetami.

4 RAZVOJ NOVEGA SISTEMA

Razvili smo nov informacijski sitem za zajem podatkov na podlagi zahtev in želja zaposlenih na oddelku naše izbrane banke. Prikazali bomo na kratko, kako izgleda vizualno, katere funkcionalnosti obsega, v katerih programskih tehnologijah je bil spisan in kako se ga uporablja, da lahko potem še prikažemo primerjavo med starim sistemom ter novim.

Izpostavili bomo tudi omejitve novega sistema, s katerimi se bodo morali uporabniki spopadati pri vsakodnevnih opravilih in vzdrževanju, saj je novi sistem zgrajen na drugačni podlagi od prejšnjega. Poleg vsega tega bomo v zadnjem poglavju tudi omenili, kje se lahko funkcionalnosti izboljšajo in katere dodatne nove funkcionalnosti so v načrtih v bližnji prihodnosti.

4.1 Izbira načina pridobitve nove rešitve

Pred začetkom projekta smo prestavili načine, ki so na voljo pri izbiri novega sistema za izbrani oddelek na banki. Predloge smo posredovali naprej vodstvu oddelka, da bi pridobili njihovo mnenje, katera izbira bi bila najboljša oziroma za katero se lahko odločimo za nadaljevanje projekta. Dobili smo povratne informacije in ker je projekt na ravni oddelka in so zahteve specifične samo za ta oddelek, smo se odločili, da bi problem rešili z internim razvojem novega sistema. Ta izbira bi nam seveda omogočila, da imamo popoln nadzor nad potekom dela, najbolj bi se prilagajala poslovnim procesom banke, integracija je preprosta, saj lahko že sproti med delom integriramo nov sistem z obstoječimi poslovnimi procesi. Prednost te izbire je tudi v tem, da že poznamo varnostne politike banke in vnaprej vemo, na kaj moramo paziti pri razvoju sistema. Lažje je tudi spreminjati sistem, če bi slučajno prišlo do sprememb med razvojem ali pa če bi uporabniki imeli nove zahteve.

Odločili smo se, da za izgradnjo novega sistema uporabimo objektno usmerjen pristop. Ker smo se odločili za lasten razvoj novega sistema, je tak pristop še najbolj primeren, saj lahko modeliramo sistem glede na primere uporabe s končnimi uporabniki sistema. Iz primerov uporabe bomo sestavili razrede ter funkcije, ki nastopajo, in te dele programske kode lahko ponovno uporabimo pri drugih delih sistema, saj postavljamo sistem za spletno strganje podatkov in se deli kode lahko uporabijo pri več razredih oziroma primerih. Pri tem pristopu lahko tudi naknadno dodajamo nove funkcionalnosti glede na potrebe in morebitne nove zahteve. S tem načinom bomo lahko tudi med gradnjo sproti pokazali napredek projekta in že delujoče dele sistema, ki jih bodo končni uporabniki že lahko uporabljali.

Za izvedbo celotnega projekta od načrtovanja do gradnje sem bil zadolžen sam. Na izbranem oddelku sem zaposlen kot edini sistemski analitik in razvijalec programske opreme in zato je bila izvedba celotnega projekta odvisna od mene. Projekt je v celoti trajal približno sedem mesecev, pri kateremu sem delal poln delavnik vsak mesec s tem, da sem imel še druge delovne obveznosti poleg tega projekta.

Slabosti, ki so prisotne pri tej izbiri, so, da bo zelo časovno potratna za razliko od takojšnjega nakupa programske opreme, saj je potrebno sistem postaviti od začetka. Razvoj in vzpostavitev sistema predstavlja zelo veliko dela za enega samega človeka. Časovni okvir, ki je bil postavljen za dokončanje projekta, je bil začetek maja 2021, kar pomeni, da je do tega meseca moral biti sistem postavljen v celoti in delujoč. Glede na to, da je na projektu delal en sam zaposleni, smo lahko izločili faktor visokih stroškov, saj je imela banka pri projektu zelo nizke stroške; to je mesečno plačo zaposlenega na projektu. Stroškov postavljanja strežnikov ni bilo, ker so bili strežniki že vzpostavljeni in se uporabljajo že obstoječi brez nakupa novih ali nadgradenj, saj so dovolj zmogljivi za delovanje novega informacijskega sistema. Kupiti je bilo treba le eno novo licenco razvijalskega orodja, saj se je projekt delalo z orodjem Microsoft Visual Studio 2019, ker pokriva vse funkcionalnosti, ki jih potrebujejo za uresničevanje projekta, poleg tega pa je Visual Studio tudi del standardne razvojne opreme na banki.

Glede stroškov smo lahko sklepali, da med razvijanjem sistema ne bodo previsoki, je pa projekt trajal bistveno dlje, kot če bi imeli na voljo skupino programerjev. Preobremenjenost zaposlenega je bila zelo visoka, saj je moral delati sam vse od postavljanja načrtov sistema, programiranja, testiranja, pa do postavljanja celotnega projekta na produkcijsko okolje. Poleg preobremenjenosti je pomembno vlogo igrala tudi odgovornost zaposlenega, saj je bil sam odgovoren za uresničitev celotnega projekta.

4.2 Izgled in delovanje novega sistema za zajem podatkov

Nova informacijska rešitev za zajem podatkov, ki jo vpeljujemo v izbrano banko, je razdeljena na dva dela. Prvi del je grafični vmesnik, ki služi za lažje navigiranje in upravljanje podatkov znotraj aplikacije. Grafični vmesnik je bil izdelan za uporabnike tako, da lahko vsi uporabljajo to rešitev na čim preprostejši in intuitiven način, tudi če nimajo nobenega predhodnega tehnološkega ali računalniškega znanja. V sakodnevna

opravila s podatki lahko preverjajo na preprost način, da samo odprejo aplikacijo in izpišejo se vse informacije o zajemih, ki so se izvajali tekom dneva in lahko takoj opazijo, če je med izvajanjem prišlo do kakršnihkoli težav.

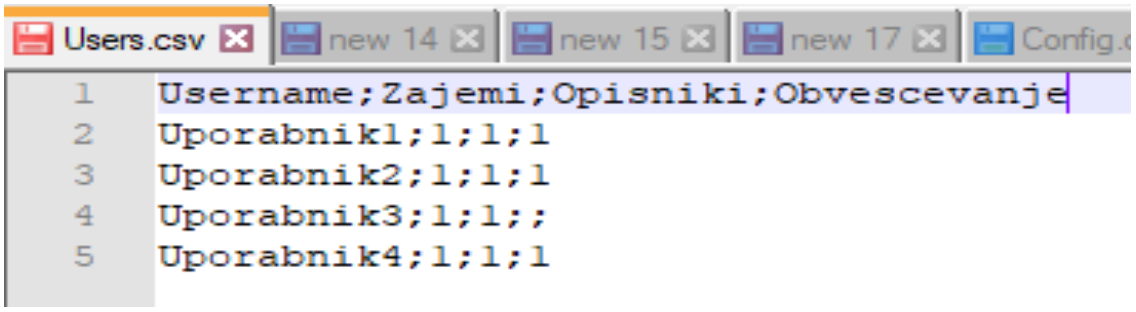
Drugi del tega sistema je program, ki zajema podatke iz spletnih virov. V osnovi je preprosta konzolna aplikacija, ki vsebuje kodo dostopa do podatkov spletnega vira, kako naj jih obdela oziroma pretvori v preprostejšo berljivo obliko in na koncu še zapis v lokalno podatkovno bazo. Pri spletnem zajemanju podatkov je programska koda za vsak spletni vir drugačna, saj ima vsak ponudnik podatkov strukturirano spletno stran na drugačen način in je skoraj nemogoče, da bi ena koda za zajem delovala pri več ponudnikih. V programu je definirana koda za vsak zajem posebej v ločenih datotekah, da se lahko takoj razloči, kje se nahaja koda za specifičen zajem. Največja razlika med zajemi je koda za pridobitev podatkov; lahko ima ponudnik že postavljen programski vmesnik za dostop do podatkov (angl. application programming interface, v nadaljevanju API) in se lahko nato samo sklicujemo na vmesnik in dobimo podatke takoj nazaj. Običajno imajo večji ponudniki podatkov to že postavljeno, a manjši ponudniki tega pogosto nimajo in je potrebno najti način za dostop do podatkov. Najbolj pogosta rešitev za take situacije je, da pridobimo kodo spletne strani in nato izčrpamo podatke direktno iz te kode spletne strani. Naslednja razlika med zajemi je obdelava podatkov, ki nam jih ponudnik posreduje, ko jih zaprosimo. Vsak ponudnik ima sestavljeno svojo obliko podatkov in moramo v kodi definirati, kako naj to obliko prebere ter jo pretvori v lažjo programsko berljivo kodo, ki nam tudi kasneje olajša zapis v podatkovno bazo. Ko imamo podatke shranjene v naši obliki, nam potem ostane le še zapis v podatkovno bazo in ta koda je pri vseh zajemih ista.

Glede na to, da sta to dve ločeni aplikaciji, lahko zajemamo podatke brez uporabe grafičnega vmesnika. Grafični vmesnik je narejen samo za lažjo uporabo in preglednost, zato lahko brez težav izvajamo zajeme podatkov brez uporabe le tega. Informacijska rešitev deluje tako, da grafični vmesnik in program za zajem podatkov komunicirata med seboj s pomočjo konfiguracijskih datotek, katere vsebujejo vrednosti ločene z vejico (angl. comma separated values, v nadaljevanju csv), ter parametrov, ki jih podamo programu. Grafični vmesnik generira te csv datoteke s podatki, ki jih uporabniki izberejo med uporabo, in nato se zažene zajem podatkov glede na informacije, ki so shranjene v csv-ju, in glede na podane parametre zagona. Uporabniki lahko tudi sami ročno ustvarijo csv-je, jih nato podajo kot parameter programu za zajem podatkov in program bo deloval enako kot preko grafičnega vmesnika. Takšna logika za komuniciranje je bila narejena z namenom, da lahko po potrebi tudi drugi programi izven tega projekta potencialno zaženejo zajem podatkov. Najpogostejši primer take uporabe je avtomatsko zaganjanje zajemov podatkov ob določeni uri na strežniku, kjer samo podamo parametre, kateri zajem naj zažene in ga nato program avtomatsko zažene v ozadju brez nujne prisotnosti uporabnika.

4.2.1 Konfiguracijske datoteke

Glavna komponenta za delovanje pri novemu sistemu so konfiguracijske datoteke. V teh datotekah se nahajajo vse informacije o uporabnikih, zajemih in tudi opisniki so zgrajeni po istem principu. Gre za običajni format, za besedilno datoteko, ki vsebuje z vejico ločene vrednosti. V prvi vrstici teh csv datotek je definirana struktura datoteke oziroma so našteti atributi, kaj vrednosti v naslednjih vrsticah prikazujejo po istem vrstnem redu kot atributi.

Slika: 6: Prikaz strukture datoteke o uporabnikih



Vir: lastno delo.

Prva datoteka, ki jo program prebere, je datoteka z informacijami o uporabnikovih pravicah, struktura te datoteke je prikazana na sliki 6. To obstaja samo z namenom, da to aplikacijo lahko uporabljajo samo uporabniki, ki so definirani znotraj datoteke. Poleg uporabniških imen je napisano tudi, katere komponente znotraj aplikacije lahko uporabniki uporabljajo, pravice se definirajo na naslednji način:

– 0 - Uporabnik nima dostopa do uporabe te komponente,
– 1 - Uporabnik ima dostop do uporabe naštete komponente.

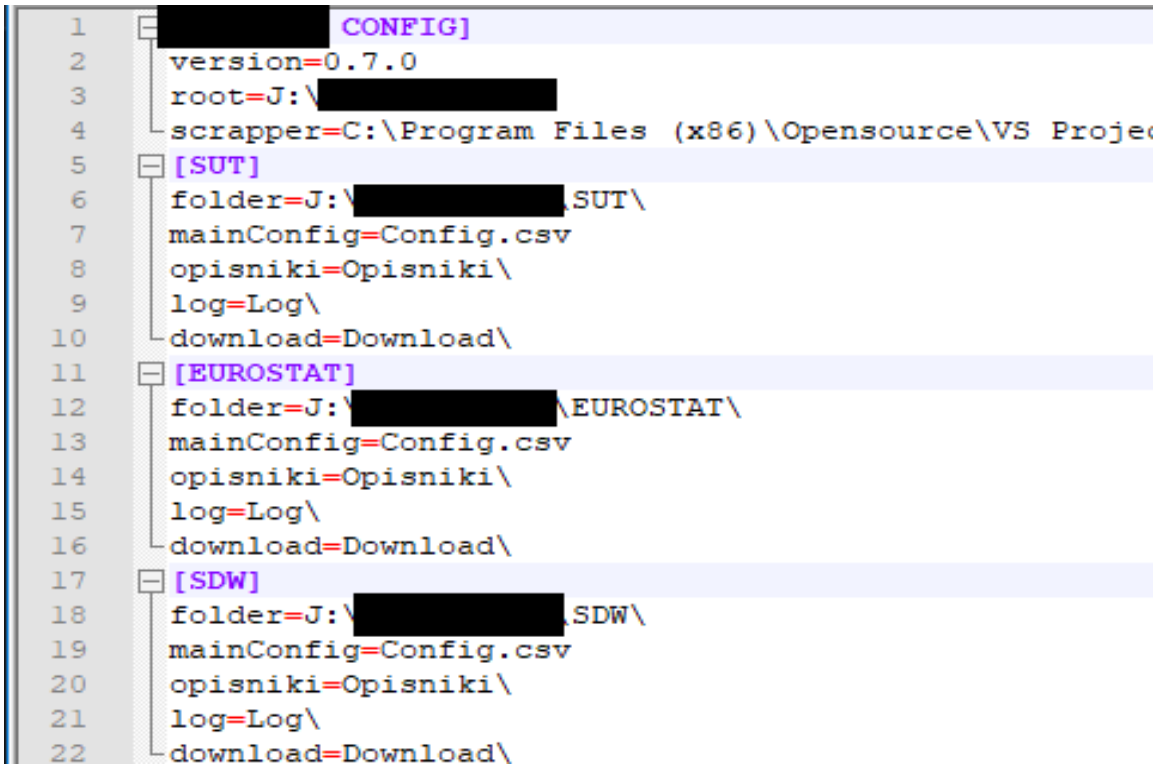
V primeru, da nekdo, ki ni definiran znotraj te datoteke, odpre aplikacijo, se bo aplikacija odprla, vendar bodo vse komponente znotraj onemogočene za uporabo. Lahko bo videl status zajemov, ki se izpišejo na uvodnem prikazu, o katerem pa bomo več povedali v naslednjem podpoglavju.

Naslednja konfiguracijska datoteka, ki jo bo program prebral, je datoteka z informacijami o direktorijih, kjer se nahajajo mape za zajeme na računalniku, ki so prikazane na sliki 7. Ta datoteka, za razliko od ostalih, v osnovi ni csv, ampak je inicializacijska datoteka (končnica .ini). To se uporablja v primerih, ko premikamo aplikacijo z enega mesta na računalniku na drugo, ne da bi morali popravljati poti znotraj kode, ampak lahko samo spremenimo pot v tej konfiguracijski datoteki in se bo potem uporabila skozi celotno aplikacijo. V omenjeni datoteki se na vrhu nahaja samo splošna definicija, katera je trenutna različica aplikacije, in pot do programa, ki zajema podatke. Pod splošno definicijo moramo še za vsak zajem, ki ga imamo, definirati informacije:

– Ime zajema – V oglate oklepaje napišemo poljubno kratico za ime zajema,

– folder - Pot do glavne mape zajema, kjer se nahajajo vse potrebne datoteke za pravilno delovanje zajema,
– mainConfig - Ime glavne konfiguracijske datoteke za zajem; v tej datoteki so definirani vsi sklopi podatkov, ki jih program zajema,
– opisniki - Pot do mape, v kateri so shranjeni vsi opisniki za ta zajem,
– log - Pot do mape, v kateri se bodo shranjevale informacije o poteku posamičnih zajemov, ki so definirani v mainConfig,
– download - Pot do mape, v kateri se bodo shranjevale JSON datoteke; to se uporablja v primerih, ko uporabnik hoče ročno preveriti podatke pri zajemu, če se pojavijo težave.

Slika: 7: Izgled inicializacijske datoteke



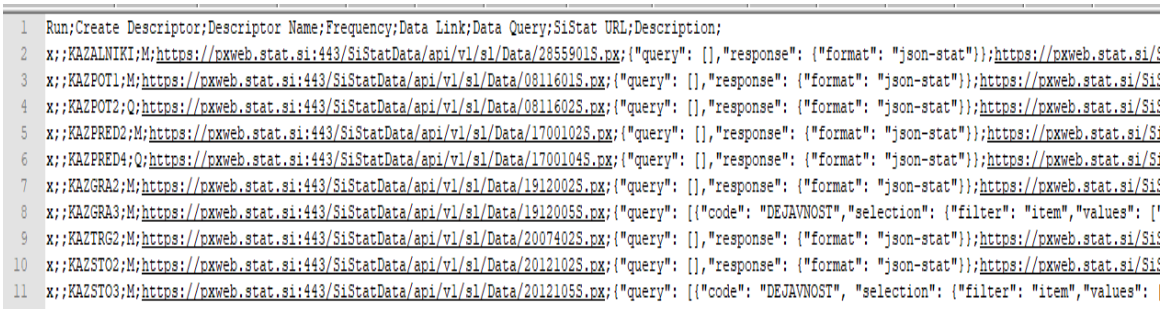
Vir: lastno delo.

Zadnji tip konfiguracijskih datotek, kot je prikazano na sliki 8, ki jih program uporablja, so za definiranje sklopov podatkov, ki jih program zajame iz podatkovnega vira. Ker se ponudniki podatkov razlikujejo med seboj glede strukture podatkov in načina pridobivanja podatkov, ima vsak zajem drugačno strukturirano konfiguracijsko datoteko. Seveda obstajajo atributi,ki so prisotni pri vseh datotekah, a največja razlika med njimi je v načinu pridobivanja podatkov iz vira. Kot primer lahko vzamemo, da en ponudnik potrebuje samoklic na eno spletno povezavo, nekateri drugi pa lahko potrebujejo še dodatne parametre pri pridobivanjupodatkov. Te datoteke so v grobem razdeljene na naslednje atribute:

– Run - S križcem ’x’ označimo, ali naj se zajem za ta sklop podatkov izvede ali ne;

– Create descriptor -S križcem ’x’ označimo, ali naj se pri zajemu še ustvari opisnik (generira se datoteka, kjer definira časovne serije in jim doda opis kaj predstavljajo);
– Descriptor name- Poljubno ime za sklop podatkov;
– Frequency - Kakšno časovno frekvenco imajo podatki (dnevni, mesečni, kvartalni ali letni);
– Spletna povezava - V ta atribut napišemo spletno povezavo do vira podatka. Lahko je samo ena ali pa po potrebi dodamo še dodatne atribute;
– Data Url -V to polje dodamo povezavo do podatkov na spletni strani za takojšni pregled, uporabno je za hiter pregled podatkov na viru, da preverimo, če je kaj novega ali pa če je šlo kaj narobe;
– Description - Dodamo opis sklopa podatkov, kaj podatki predstavljajo.

Slika: 8: Del konfiguracijske datoteke za celoten zajem



Vir: lastno delo.

4.2.2 Uvodni prikaz

Ob zagonu programa se nam pokaže uvodni zaslon, ki nam prikaže informacije o izvajanju vseh zajemov, ki so se izvedli na današnji dan. Na sliki 9 se prikaže status izvajanja zajemov po sklopih podatkov, prikažejo se informacije, katere časovne serije se niso napolnile, katere odvečne serije je tudi zajelo, katere niso definirane v opisnikih, in prikaže, za kateri datum je zadnji podatek za vsako serijo posebej. Na prikazane informacije lahko uporabnik hitro reagira, če je šlo kaj narobe.

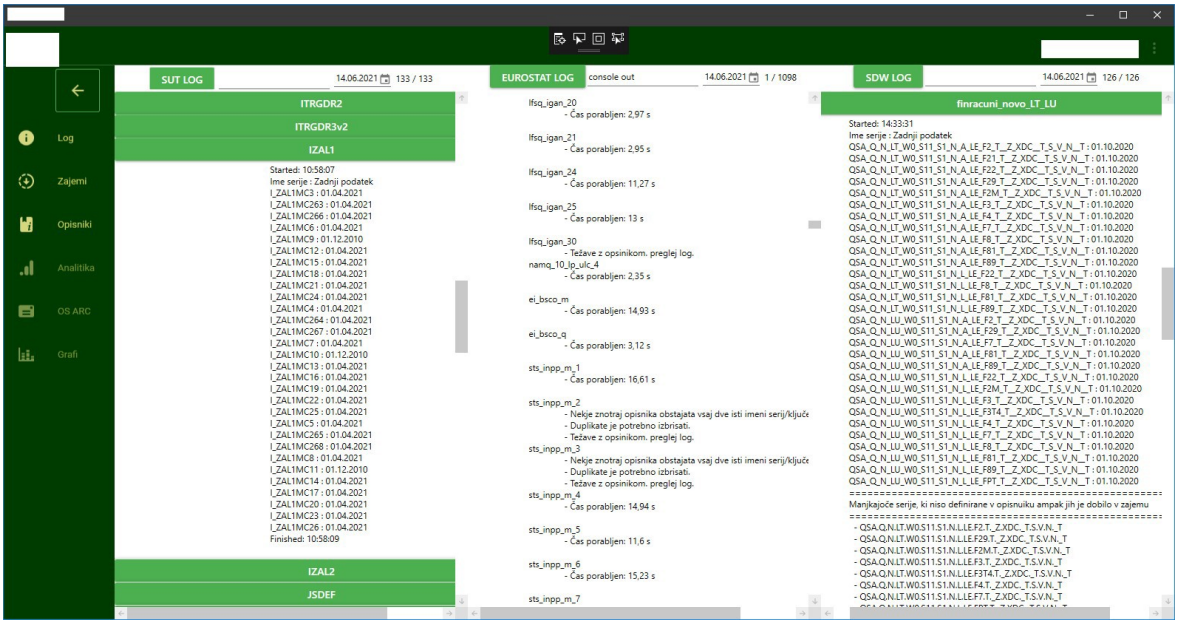
V zgornjem delu tega okna se samo prikaže uporabniško ime trenutnega uporabnika, ki je odprl aplikacijo. Program nima opcije za vpis z uporabniškem imenom in geslom, ampak avtomatično ob zagonu pobere uporabniško ime direktno iz računalnika, na katerem je uporabnik vpisan. Levi del okna vsebuje navigacijski meni, s katerim se lahko premikamo po aplikaciji. Ob zagonu je meni prikazan v strnjeni obliki, vsebuje samo ikone ostalih komponent, ima pa možnost, da se s klikom na prvo ikono meni razširi v bolj podroben pregled, kjer je poleg samih ikon dodan še opis, kaj te ikone predstavljajo za uporabnike začetnike, ki delovanja sistema še ne poznajo dobro.

V osrednjem delu so prikazani statusi izvajanja zajemov po sklopih podatkov. Ob kliku na ime zajema se prikaže status izvajanja tega zajema in potek dogajanja med samim izvajanjem. Informacije, ki se izpišejo, omogočajo uporabnikom hiter in enostaven pregled

nad samim izvajanjem ter na podlagi teh informacij se lahko potem ustrezno odločajo, kaj naj naredijo v primeru, da podatki manjkajo ali pa je prišlo do težav že pri pridobivanju podatkov. Informacije, ki se izpisujejo, so sledeče:

1. Napake pri pridobivanju podatkov- Med zajemanjem podatkov lahko pride do težav, kot so npr. strežnik ponudnika je nedosegljiv, spremenili so strukturo podatkov, interna napaka v kodi.
2. Čas izvajanja - Prikaže se, koliko časa je zajem za ta sklop podatkov potekal, ob kateri uri se je začel in kdaj končal.
3. Zadnji podatek za časovno serijo - Izpisuje se tudi za vsako časovno serijo posebej, kateri je zadnji datum za podatek; na ta način lahko hitro vidimo, ali so novi podatki že na voljo, pa tudi, če zajem deluje pravilno in prenese zadnji podatek.
4. Odvečne serije - Informacija predstavlja, katere dodatne časovne serije, ki niso definirane v opisnikih, smo prenesli, kar nam pomaga pri omejevanju prenosa podatkov, da prenesemo samo tiste serije, ki so definirane.
5. Spremembe v opisu serij - Obstaja možnost, da ponudnik podatkov spremeni definicije njihovih časovnih serij in zaradi tega se serije, ki so definirane v opisnikih, ne ujemajo s tistimi na viru in s to informacijo lahko prepoznamo, pri katerih časovnih serijah je prišlo do sprememb.

Slika: 9: Izgled uvodnega okna



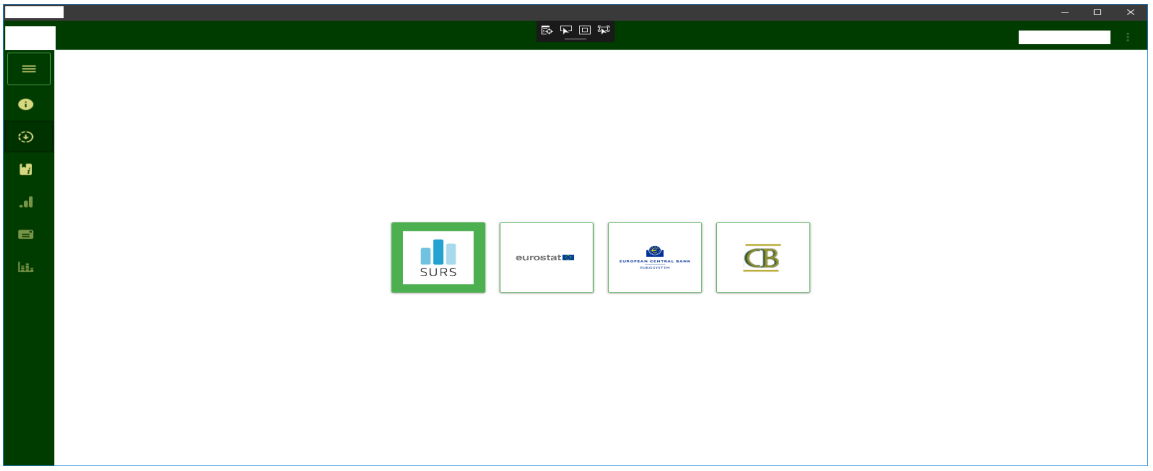
Vir: lastno delo.

Za primere, ko je definiranih veliko zajemov sklopov podatkov in postane zajemanje nepregledno ter je oteženo poiskati, kateri zajemi so naleteli na težave, je implementirana tudi opcija, da se na vrhu prikaže skrajšana oblika zapisa izvajanja vseh zajemov. V tej skrajšani obliki so zapisane informacije o časovnem izvajanju, kako dolgo je potekalo zajemanje za vsak sklop posebej, ter nastale napake med izvajanjem in zapis podatkov v podatkovno bazo, če je zajemanje slučajno naletelo na kakšne napake pri zapisu.

4.2.3 Zajemi

Na oknu prikazanem na sliki 10 se uporabniku pojavijo vse možne izbire zajemov, ki so trenutno definirani v kodi in so že pripravljeni za uporabo. S klikom na željeni zajem bo uporabnika preusmerilo na novo okno znotraj programa, ki je vezano na izbrani zajem.

Slika: 10: Izgled uvodnega okna za izbiranje zajemov



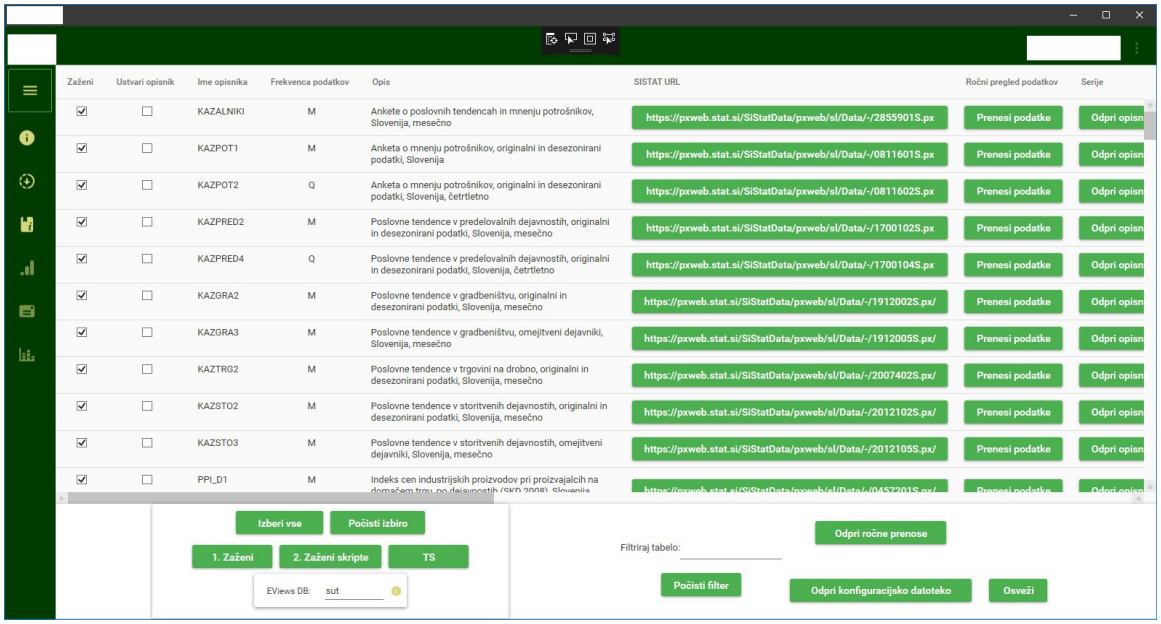
Vir: lastno delo.

V osnovi so okna za izvajanje strukturirana na enak način in se med seboj zelo malo razlikujejo. Vsi prikazujejo glavno konfiguracijsko datoteko, v kateri so definirani vsi sklopi podatkov za prenos znotraj tega zajema in je v grafičnem vmesniku prikazana kot tabela. Znotraj te tabele lahko uporabnik izbere, katere sklope podatkov naj zajame z namenom, da ni potrebno vedno izvesti celotnega zajema, ampak samo tiste, ki jih takrat potrebujejo.

Na sliki 11 nam večino zaslonskega prostora zasede tabela z zajemi. Znotraj te tabele imamo na voljo vse informacije, ki so zapisane tudi v sami konfiguracijski datoteki, vendar so v grafičnem vmesniku prikazane na bolj uporabniško prijazen način. Iz tabele lahko takoj razberemo, kaj vsak zajem posebej predstavlja in lahko na podlagi teh informacij izvajamo strganje podatkov po želji. V primeru, da uporabnik ne pozna tega sklopa, ima tudi možnost, da klikne na gumb, ki mu odpre opisnik, v katerem so zapisane vse časovne serije in njihovi opisi za lažje razumevanje sklopa. Dodana je tudi možnost, da se v tabeli prikaže gumb, ki uporabnika navigira na spletno stran ponudnika podatkov z namenom, da lahko takoj vidi, ali so na strani že novi podatki na voljo, brez da zgublja čas z izvajanjem zajemov. Uporabniku je tudi ponujena možnost, da prenese surovo obliko podatkov direktno iz vira z namenom, da pregleda strukturo podatkov znotraj odgovora spletnega strežnika in vidi, če je slučajno prišlo do kakšnih sprememb, da lahko nato ustrezno reagira na spremembe, ki jih je potrebno spremeniti tudi v sami kodi za zajem.

Spodnja vrstica je opremljena z gumbi za izvajanje in iskanje. Vsako okno za zajeme ima definiran gumb za zagon dejanskega zajema podatkov, ki generira začasno konfiguracijsko datoteko, ki vsebuje samo označene zajeme za zagon iz tabele. Ta začasna konfiguracijska

Slika: 11: Izgled vmesnika za upravljanje zajemov



Vir: lastno delo.

datoteka se nato posreduje programu za strganje podatkov, ki bo zajel samo tiste sklope podatkov, ki jih je uporabnik označil v tabeli. Poleg tega gumba se lahko po potrebi doda še dodaten gumb, ki bo po zaključku strganja podatkov izvedel še dodatne skripte kot npr. dodane kalkulacije, rebaziranje podatkov in podobno.

Dodana sta tudi dva dodatna gumba, ki služita samo za izbiranje celotne tabele za zagon in za to, da lahko na hiter način vse izberemo in zaženemo ali pa vse odkljukamo ter ročno izberemo samo določen izbor zajemov. Nahaja se tudi polje, v katero uporabnik napiše, v katero lokalno podatkovno bazo naj se tej podatki vpišejo. Večinoma tega polja ni potrebno spreminjati, ker je že vnaprej določeno, kam naj se vpišejo podatki. V primeru, da hočemo podatke prenesti v drugo podatkovno bazo, lahko preprosto tam spremenimo ime in ob naslednjemu zagonu programa se bodo podatki vpisali v drugo določeno podatkovno bazo. V primerih, ko imamo zelo obširno tabelo in je iskanje po njej zelo težavno, je dodana tudi opcija iskanja po njej. V tekstovno polje uporabnik vpiše poljuben niz, po katerem hoče iskati in se bo tabela filtrirala po vpisanem nizu in prikazala samo ujemanja, ki ustrezajo temu nizu. S tem olajšamo uporabniku iskanje in lahko takoj dobi zahtevane podatke. Da se trenutni filter počisti, lahko uporabnik to naredi na dva načina; prvi je s klikom na gumb Počisti filter, druga opcija pa je, da izprazni tekstovno polje sam in se bo tabela vrnila v prvotno stanje.

Ostali trije gumbi služijo samo za manjše specifične naloge in to so:

1. Odpri ročne prenose - odpre mapo, v kateri se nahajajo datoteke, ki vsebujejo surovo obliko podatkov iz vira; ta pot je tista, ki je definirana v inicializacijski datoteki za direktorije programa.

2. Odpri konfiguracijsko datoteko - odpre csv, v katerem so definirani vsi sklopi podatkov za trenutno izbrani zajem.
3. Osveži - Osveži tabelo, če je uporabnik spreminjal glavno konfiguracijsko datoteko.

4.2.4 Obveščanje

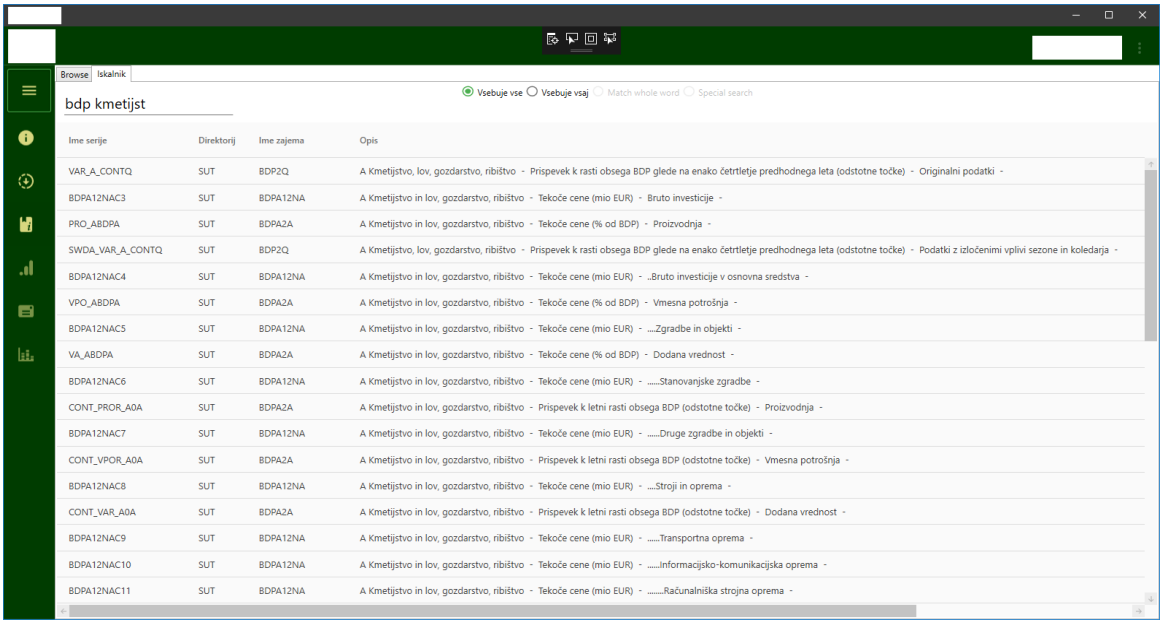
V ta program je bila vgrajena možnost pošiljanja elektronskih sporočil vsem uporabnikom, dodanim na poštno listo. Zaenkrat se pošilja samo status izvajanja zajemov, iste informacije, kot jih vidimo na uvodnem oknu, kjer se izpisujejo informacije o tem, kako so se zajemi izvedli, če je prišlo do napak in če se kateri podatki niso posodobili ali manjkajo. Ker je sam sistem nameščen na strežnik v banki, se morajo uporabniki vsako jutro povezati z njim in pregledati izpise.

S to komponento obveščanja se sedaj vsako jutro ob isti uri pošlje elektronsko sporočilo, v katerem so vsebovane tekstovne datoteke, ki hranijo skrajšan opis izvajanja vseh zajemov z namenom, da zaposleni lahko že takoj zjutraj vidijo, ali je šlo kaj narobe, ne da bi se priklopili na strežnik.

4.2.5 Opisniki

Ta komponenta služi samo za iskanje časovnih serij po vseh opisnikih naenkrat. Prejšnji postavljen sistem je imel pomanjkljivost, da ni imel opcije iskanja serij po vseh opisnikih naenkrat in so zato morali uporabniki ročno iskati po vseh opisnikih, kje bi se lahko iskana serija nahajala. Iskalnik prikazan na sliki 12 deluje tako, da ko uporabnik vnese iskani niz v polje, program sproži iskanje po vseh opisnikih in sproti prikazuje ujemanja v tabeli. Iskalnik išče ujemanja po opisu časovne serije in samem imenu serije.

Slika: 12: Izgled iskalnika po opisnikih



Vir: lastno delo.

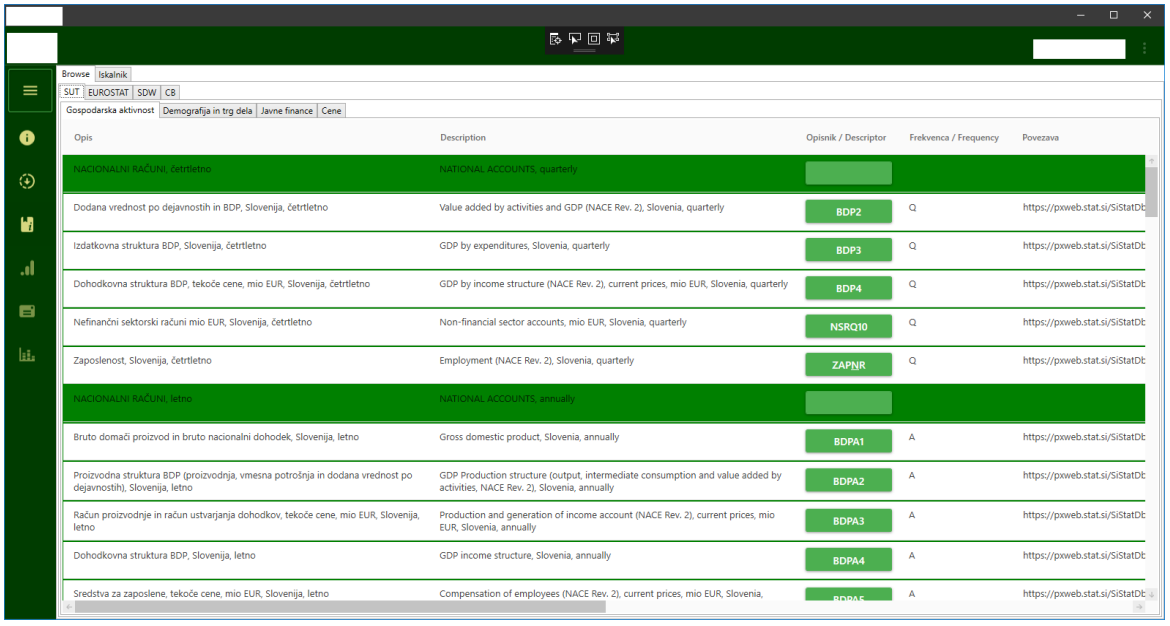
Vrhnji del okna vsebuje dva zavihka, ki uporabniku omogočata dva različna načina iskanja serij. Prvi, ki je privzeto izbran, je zavihek Iskalnik. Tukaj se nahaja tekstovno polje, v katerega napišemo poljuben niz iskanja, dodani pa sta še dve opciji, kako naj išče. Trenutno sta na voljo dve izbiri:

1. Vsebuje vse - če imamo to opcijo izbrano, bo iskalnik vrnil ujemanja samo v primeru, da vsebuje vse besede, ki jih je uporabnik napisal ne glede na to, kje se nahajajo v opisu serij.
2. Vsebuje vsaj - ta opcija bo iskalniku povedala, naj vrača ujemanja, če v opisu serije vsebuje vsaj eno besedo iz iskalnega niza.

Iskalnik je nastavljen tako, da začne iskati šele, ko uporabnik vnese vsaj tri črke; če bi iskali samo po eni črki, bi bilo ujemanj preveč in bi bilo iskanje nesmiselno. Ko je vpisana tretja črka v iskalnik, se sproži iskanje po vseh direktorijih, ki so definirani v inicializacijski datoteki pod rubriko opisniki.

Drugi zavihek, prikazan na sliki 13, ki je še prisoten poleg zavihka Iskalnik, je Brskaj. V tem zavihku se pojavi stari prikaz opisnikov, tako kot je bilo pred vpeljavo novega sistema, na željo nekaterih uporabnikov. Ima podoben izgled, kot je bil v preteklosti v Excel datotekah, in je razdeljen na zavihke, ki predstavljajo ponudnike podatkov, znotraj vsakega takega zavihka pa je tabela, ki vključuje vse sklope podatkov, njihove opise ter povezavo do opisnika na omrežnem disku. Na tem zavihku ni opcije iskanja in posledično morajo uporabniki ročno iskati na enak način kot pri prejšnjemu sistemu.

Slika: 13: Izgled starega brskalnika po opisnikih



Vir: lastno delo.

4.2.6 Zajem podatkov

Druga polovica projekta je dejanski program, ki zajema podatke iz spletnih virov. Ta del je popolnoma ločen in neodvisen od grafičnega vmesnika in ga lahko uporabniki uporabljajo brez poseganja v grafični vmesnik. Narejen je na način, da ga lahko zaganjajo tudi druge eksterne aplikacije brez potrebe po uporabi že narejenega vmesnika za upravljanje.

Za zagon mu je potrebno posredovati tri parametre:

1. Ime zajema- kot prvi parameter posreduje ime zajema, ki naj ga izvede, imena so definirana znotraj kode.
2. Avtomatski zajem:
   0 - Pomeni, da je ročni zajem in se bo na koncu odprl EViews program za pregled podatkov, preden se vpišejo v podatkovno bazo;
   1 - Pomeni, da je avtomatski zajem, torej se bodo po zaključitvi zajema podatki avtomatsko zapisali v bazo brez nujne uporabnikove prisotnosti.
3. Konfiguracijska datoteka - kot zadnji parameter podamo ime konfiguracijske datoteke, za katero želimo, da se izvede; v tej datoteki so definirani vsi sklopi podatkov za zajem.

Če uporabniki pred zagonom ne podajo vseh treh parametrov, se program ne bo zagnal in bo javil napako. S takim načinom zaganjanja si lahko tudi drugi zaposleni po želji sprogramirajo lasten grafični vmesnik v primeru, da želijo imeti lastno predstavitev tega vmesnika ali pa če imajo dodatne zahteve, ki bi služile samo njim.

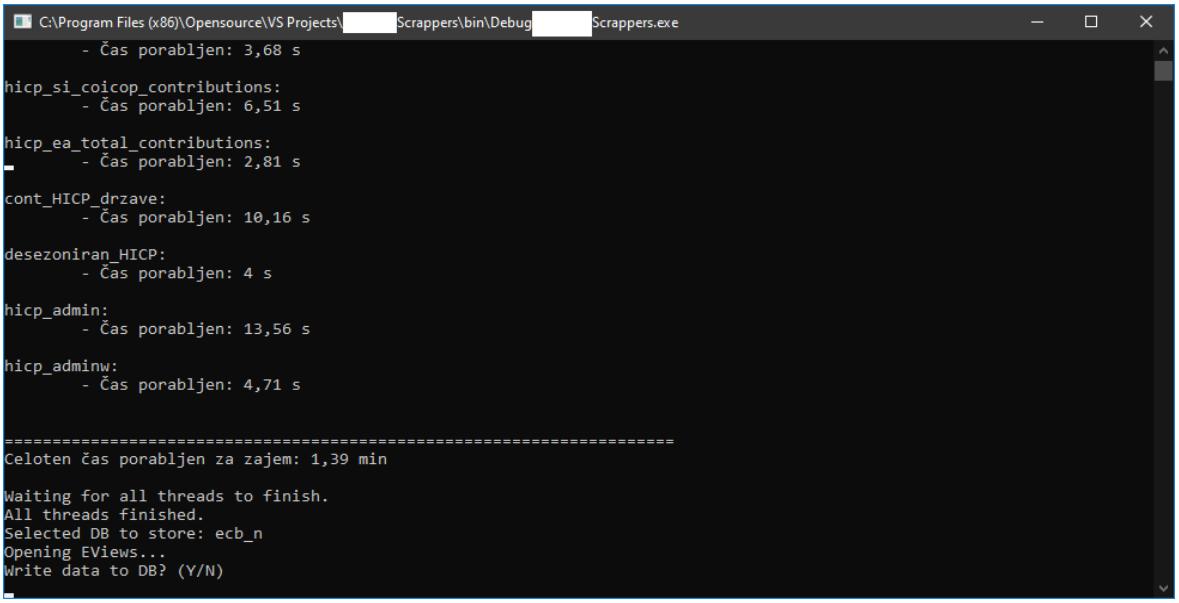
Ob pravilnem vnosu parametrov se bo začel izvajati program za strganje podatkov, prikazan na sliki 14. Najprej pregleda prvi podan parameter, ki je ime zajema. Na podlagi tega potem zažene ustrezno kodo, ki je namenjena temu zajemu. Struktura kode za pridobitev podatkov je vedno enaka, se pa za vsak vir posebej razlikuje v izvedbi. Najprej začne s tem, da inicializira vse potrebne spremenljivke znotraj kode za pravilno delovanje, kot so podatkovne strukture za shranjevanje podatkov, beleženje izvajanja in podobno. Poleg tega izvede branje konfiguracijske datoteke, ki je tudi tretji parameter pri zagonu, in vso njeno vsebino shrani v kodo. S tem shranimo vse spletne povezave, ki vodijo do podatkov na viru.

Sledi pridobitev surovih podatkov iz spletnega vira in to se izvede tako, da v kodi pošlje spletni zahtevek na povezavo, ki smo jo v prejšnjem koraku pridobili. Kot odgovor iz vira lahko dobimo napako, če smo v konfiguracijski datoteki napačno napisali spletno povezavo. Če je bilo vse pravilno napisano, bomo dobili podatke nazaj v takšni obliki, kakor smo zapisali v konfiguracijski datoteki (v kakšni obliki jih dobimo, je odvisno od ponudnika podatkov).

Po pridobitvi podatkov iz vira sledi transformacija le teh v vnaprej določeno obliko, ki smo jo definirali v programski kodi za lažjo berljivost in lažji zapis v podatkovno bazo. Istočasno, ko se to izvaja, se sproži tudi postopek generiranja opisnikov v primeru, da je uporabnik to označil, preden je zagnal program za strganje podatkov. V tej surovi obliki podatkov, ki jih dobimo iz vira, vsebujejo tudi informacije o časovnih serijah, katere preberemo in s tem

poleg zajema podatkov avtomatsko ustvarimo tudi opisnike, ki vsebujejo te serije ter njihove opise in seveda te časovne serije tudi avtomatsko poimenuje v interna imena, na katera se bodo zaposleni na banki sklicevali.

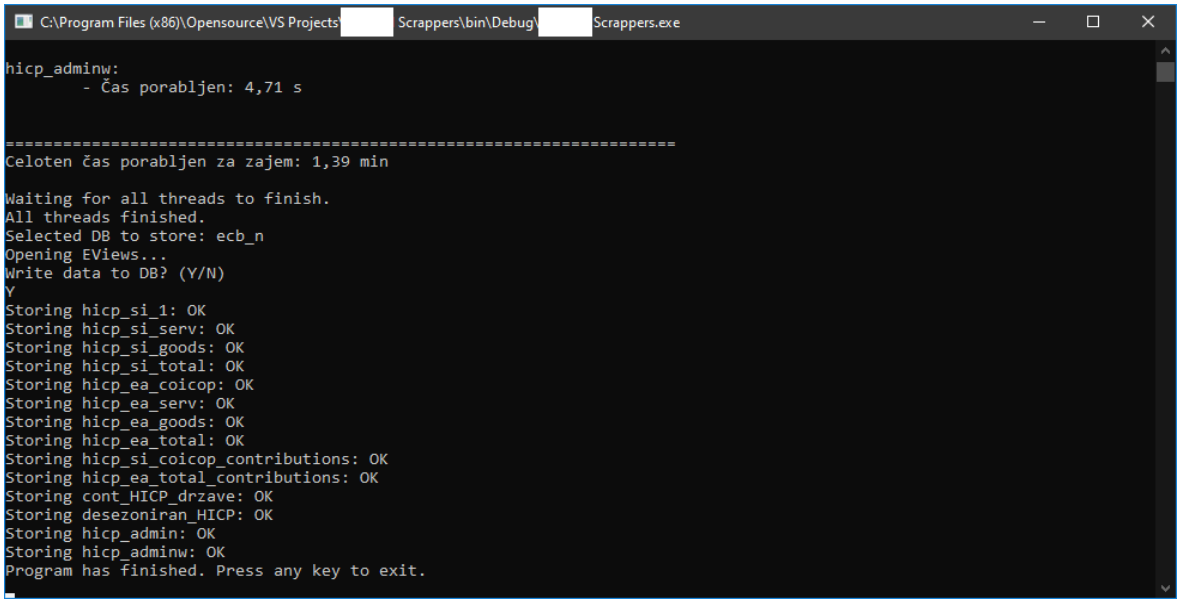
Slika: 14: Izgled programa/konzole za zajemanje podatkov



Vir: lastno delo.

Ko imamo vse podatke transformirane in opisnike narejene, sledi še zapis v podatkovno bazo, izgled zapisovanja v podatkovno bazo je prikazan na sliki 15. Na željo zaposlenih na oddelku se ti podatki ne zapisujejo v centralno bazo banke, pač pa v lokalne baze na skupnem omrežnem disku. Zapis podatkov se izvede s pomočjo uporabe programa EViews.

Slika: 15: Zapis podatkov preko programa za zajemanje podatkov



Vir: lastno delo.

V ta eksterni program s pomočjo kode v programu za zajeme prenesemo vse podatke v EViews in šele ko se prenos konča, se podatki zapišejo v bazo z uporabo programa EViews. Zapis se izvede glede na drugi parameter, ki ga podamo pred zagonom zajema. V primeru, da je bil izbran ročni vpis, se bo po končanem prenosu podatkov odprl program EViews, v katerem lahko uporabnik preveri veljavnost in točnost podatkov, ki smo jih zajeli, ter se na podlagi tega odloči, ali se bodo podatki zapisali ali ne. V primeru, da je bil izbran avtomatski vnos, se bodo po končanem prenosu podatkov avtomatsko tudi zapisali v lokalno podatkovno bazo brez potrebe po uporabnikovi potrditvi.

Po koncu zgoraj navedenih postopkov se zaključi proces strganja in zapisovanja podatkov. Med celotnim procesom izvajanja so dodane tudi kontrole, ki preverjajo pravilnost izvajanja in beleženja morebitnih napak med izvajanjem ter se kasneje prikažejo na uvodnem oknu v grafičnem vmesniku.

4.3 Dodajanje novih zajemov

Dodajanje novih zajemov podatkov iz drugih virov se ne izvede preko grafičnega vmesnika, ki bi imel to možnost s kakšnim preprostim načinom, temveč je potrebno spisati programsko kodo v orodju Visual Studio. Za programiranje novega zajema je potrebno imeti nekaj znanja iz spletnega strganja podatkov oziroma je treba vsaj poznati postopek, kako se pridobi vse potrebne informacije za izvedbo novega zajema. Potrebujemo naslednje informacije za dodajanje v novem sistemu:

1. Lokacija podatkov- Potrebno je dobiti informacijo, kje se na spletnem viru nahajajo podatki;
2. V kakšni obliki so - Ko dobimo lokacijo podatkov, moramo pridobiti še informacijo o tem, v kakšni obliki so, ali so v HTML kodi, ali lahko do njih dostopamo preko API-ja, ali so v datotekah, ki jih je potrebno prenesti, slike, itd.;
3. Kako najlažje priti do teh podatkov - Ob izpolnjenih zgornjih pogojih moramo še najti način, kako bomo programsko prišli do teh podatkov.

Pridobivanje lokacije podatkov je prvi korak in je najlažji. Ko najdemo nek nov ustrezen vir podatkov, ki bodo prišli prav za kasnejše analize, moramo najprej obiskati spletno stran ponudnika in poiskati točen spletni podnaslov, na kateremu se nahajajo iskani podatki. Lahko so vidni že na naslovni strani ali pa so skriti nekje drugje na kakšnih podstraneh.

Ko pridobimo točno lokacijo podatkov, v naslednjem koraku preverimo, če že obstaja kakšen direkten dostop do teh podatkov; kot na primer, da ponudnik že sam ponuja dostop preko API- ja. Možno je tudi, da nekje že obstaja možnost direktnega izvoza podatkov iz same spletne strani v takšne oblike datotek, ki jih ponudnik ponuja (Excel, tekstovne datoteke, itd.). V primeru, da zgornjih opcij ni na voljo na viru, je potrebno prenesti HTML kodo spletne strani in iz nje izluščiti podatke, ki jih želimo. Problem pri tem koraku lahko nastane, če na spletni strani zahtevajo, da se uporabniki registrirajo, kar pomeni, da ne moremo direktno

dostopati do želenih podatkov in je potrebno pri pisanju kode za zajem tega vira upoštevati tudi vpisovanje uporabniškega imena in gesla, preden lahko sploh pridemo do podatkov.

Ob pridobitvi vrhnjih dveh informacij sledi najtežji korak med temi tremi. Najti moramo način, kako bomo najlažje dostopali do želenih podatkov na spletni strani ponudnika. Če ima ponudnik postavljen vmesnik do podatkov (API), lahko direktno dostopamo do njih preko programskega klica, s tem da najprej preberemo dokumentacijo, ki je na voljo na njihovi spletni strani o tem, kako pravilno uporabljati vmesnik. V primeru, da dokumentacije ni, je potrebno preiskati spletno stran in najti navodila, kako to uporabljati. Če nimajo postavljenega vmesnika za dostop, ampak imajo na voljo direktni izvoz podatkov na njihovi strani, lahko shranimo spletno povezavo do izvoza, na katerega se bomo kasneje tudi sklicevali v programski kodi. Kadar nimamo na voljo direktnega dostopa ali izvoza, nam še preostane možnost, da si zabeležimo povezavo do spletne strani, kjer se nahajajo podatki, in bomo potem v kodi, ko bomo poslali programski klic na ta naslov, kot odgovor pridobili HTML kodo spletne strani, iz katere bo treba v programski kodi izluščiti podatke. Upoštevati moramo tudi primer, ko je potrebna registracija na viru za dostop do podatkov, kar se lahko zgodi samo pri izvozu podatkov s strani ali pa ko je potrebno te podatke izluščiti iz HTML kode. Preden bomo prišli do podatkov preko programskega klica znotraj kode, bo potrebno še pridobiti spletni naslov, na kateremu se zgodi prijava uporabnika v spletno stran in najprej bo potrebno poslati uporabniško ime ter geslo na spletni naslov in šele nato bomo izvedli programski klic do dejanskih podatkov.

Sledi korak, ko je potrebno spisati programsko kodo za strganje podatkov za nov vir podatkov v C#. Koda za to se razdeli na šest sekcij in pri vsaki sekciji je potrebno izpolniti njihovo funkcijo, da bo zajem deloval ter pravilno izvajal celoten proces zajemanja. Sekcije, katere je potrebno izpolniti so naslednje:

1. Inicializacija - V to sekcijo deklariramo vse spremenljivke, ki jih bo program potreboval za pravilno izvajanje; spremenljivke se dodajajo tudi tekom programiranja po potrebi, ni potrebno deklarirati vseh naenkrat, saj na začetku še ne poznamo celotnega obsega.
2. Obdelava konfiguracijske datoteke - V tem postopku se odločimo, kako naj bi konfiguracijska datoteka izgledala, vsebovati mora prvo vrstico, v kateri zapišemo, kakšna je struktura datoteke (katere podatke bomo shranjevali), ime zajema, frekvenco podatkov, opis tega sklopa podatkov, kaj predstavljajo, povezavo do spletnega vira, kjer se podatki nahajajo in po potrebi navedemo še dodatne atribute. V tej sekciji kode je potrebno nujno napisati, kako naj program prebere datoteko in izlušči te atribute iz nje ter jih tudi shrani v spremenljivke znotraj kode.
3. Obdelava podatkov iz vira - Glede na to, na kakšen način bomo pridobivali podatke (API, izvoz, HTML, ostalo), je v tej sekciji potrebno napisati, kako naj program pridobi podatke in kako naj jih prebere iz te oblike. V tem koraku je potrebno vse informacije, ki jih dobimo iz vira, shraniti v svoje spremenljivke; te informacije so podatki o časovnih serijah, opisi časovnih serij, velikosti posamičnih dimenzij, vse kar je

potrebno, da lahko sestavimo časovne serije. Če dobimo samo podatke časovnih serij v odgovoru vira, moramo potem vse ostalo sami ročno ustvariti.

4. Opisniki - Ta sekcija je odvisna od ponudnika vira, lahko se zgodi, da v odgovoru ponudnika dobimo samo vrednosti časovnih serij brez opisov, kaj te serije predstavljajo. V takem primeru moramo opisnik predhodno ročno narediti ali pa ga na kakšen drug način ustvariti. Kadar ponudnik v odgovoru posreduje tudi informacije o časovnih serijah, jih moramo v kodi izluščiti in na podlagi tega generirati opisnik za ta sklop podatkov. Opisnik mora vsebovati ime serije, za katero se uporabniki predhodno dogovorijo, podati je treba tudi identifikacijsko ime, ki je prisotno v odgovoru ponudnika za prepoznavanje časovnih serij, in opis serije, da uporabniki vedo, kaj te časovne serije predstavljajo.
5. Ustvarjanje serij – Vse, kar smo shranili pri sekciji obdelava podatkov iz vira, moramo prebrati in generirati časovne vrste v kodi po že vnaprej določeni obliki, ki nam kasneje olajša branje ter zapisovanje serij v podatkovno bazo. Vnaprej določena oblika je opisana v kodi; navedeno je, kako je strukturirana in katere informacije zahteva. Vse serije, ki jih generiramo, moramo nato še shraniti v seznam za vpis.
6. Zapis serij v bazo - Ta sekcija kode je pri vseh zajemih v tem projektu enaka in je ni potrebno vedno znova zapisovati. V tem postopku se mora seznam časovnih serij za vpis iz prejšnje sekcije prenesti v eksterno programsko opremo EViews, ki je zadolžena za zapis v podatkovno bazo. Šele ko se vsi podatki prenesejo v EViews, se lahko dokončno zapišejo v lokalne podatkovne baze, ki jih EViews upravlja.

Pri izdelavi novih zajemov je tudi zelo priporočljivo, da se dodajo kontrole, ki preverjajo veljavnost in pravilnost izvajanja. Kontrole se lahko dodajajo pri vsaki sekciji posebej z namenom, da imamo lažji nadzor nad izvajanjem in da lahko tudi prej opazimo, kje se morebitne težave pojavijo. Gre za preverjanje, ali so podatki v pravilni obliki, ali ima strežnik ponudnika težave, ali se opisniki pravilno generirajo in podobno. Vse take primere je potrebno zapisati v ločene tekstovne datoteke, da lahko uporabniki vidijo, kaj se dogaja med izvajanjem in če je prišlo do kakšnih napak. Te datoteke se kasneje tudi pojavijo na uvodnem oknu ob zagonu programa.

Po zaključku ustvarjanja novega zajema sledi še opcijska izgradnja vmesnika, kako zaganjati te zajeme preko grafičnega vmesnika. Na željo uporabnikov se lahko ta korak izpusti in se uporablja zajeme direktno ali pa ga zgradijo za lažje upravljanje. Pri tem postopku je potrebno narediti, da v grafičnem vmesniku prikaže konfiguracijsko tabelo zajema v obliki tabele in dodati še gumb za upravljanje zajemov. Za konec moramo ustvariti še direktorij na računalniku, v katerem bomo shranili konfiguracijsko datoteko, izpiske kontrol, ki se izvajajo v kodi med zajemanjem, in mapo z opisniki. V inicializacijski datoteki dodamo nov vnos z informacijami o direktoriju, ki bo hranil vse podatke za nov zajem.

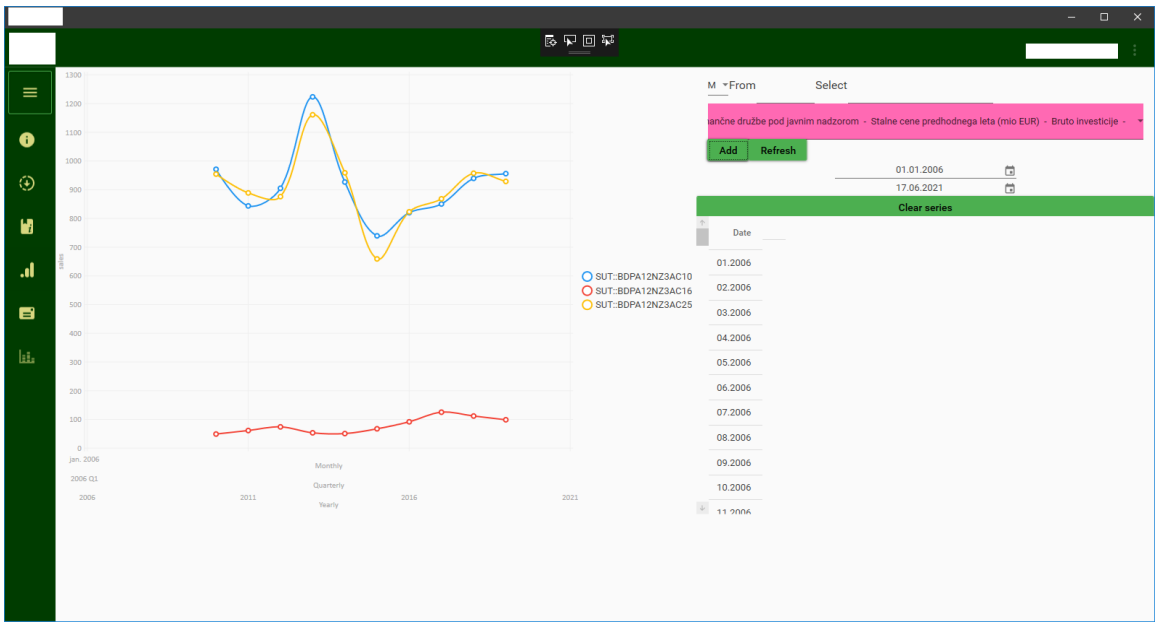
4.4 Nadaljnji razvoj

Aplikacija za strganje podatkov je v celoti postavljena, a so med gradnjo informacijskega sistema zaposleni izrazili dodatne želje. Želijo imeti še funkcijo avtomatskega izdelovanja grafov na podlagi podanih časovnih serij in še izvoz teh grafov v Excel datoteke ali slike. Ta funkcija bo morala vsebovati obstoječi seznam vseh grafov, ki jih posodabljajo, a bo oblika seveda morala biti pretvorjena tako, da bo delovala v novem sistemu, ker trenutno so vsi grafi zapisani v eni Excel datoteki. V tej datoteki so shranjena imena grafov, kratek opis, kaj predstavlja, frekvenca podatkov, in katere časovne serije so uporabljene za prikazovanje podatkov. Težava pri trenutni izvedbi te funkcije je, da obsega čez tisoč grafov in je generiranje zelo časovno potratno. Poleg časovnega problema je prisoten tudi problem beleženja napak pri izvajanju, kar pomeni, da uporabniki ne vedo, kateri grafi so naleteli na težave pri ustvarjanju, ker se nikjer ne pojavi kakršnokoli obvestilo, da je prišlo do napak. Posledično sledi, da morajo zaposleni ročno pregledovati, kateri grafi se niso posodobili ali manjkajo, in na podlagi tega potem ustrezno ukrepati.

V novem sistemu bo seveda funkcionalnost morala odpraviti prej naštete težave. V stranskem meniju sistema bo potrebno dodati novo ikono, ki bo uporabnikom omogočala ustvarjati in izvažati grafe. Potem bo potrebno narediti še grafične elemente, ki bodo po obliki morali biti podobni, kot je strukturirana obstoječa Excel datoteka z grafi, da bo uporabnikom čim bolj znana oblika. Ko bo grafična podoba dokončana, bo potrebno prenesti vse informacije o grafih iz Excel datoteke v nov sistem. Sprogramirati bo treba še možnost dodajanja grafov, brisanje grafov, urejanje grafov in ključno komponento beleženja izvajanja. Komponenta beleženja bo morala shranjevati informacije, kateri grafi so se uspešno izvedli, kateri so zadnji podatki, uporabljeni za prikaz podatkov, kateri so naleteli na težave in se posledično niso posodobili, poleg tega pa še te informacije posredovati preko modula obveščanja, ki bo poslal elektronsko sporočilo skrbnikom, da lahko vidijo to že takoj po prihodu v službo.

Uporabniki so izrazili še željo za funkcionalnost, ki bi omogočala, da bi se grafi izrisovali na zaslonu, takoj ko bi uporabnik izbral časovno serijo. Gre za hitro prikazovanje več časovnih serij v enem grafu z možnostjo, da upošteva različne frekvence podatkov. Na temu področju je že bilo nekaj vloženega dela in že obstaja prototipni model te funkcionalnosti. Na sliki 16 imamo na voljo iskalnik, ki bo na podlagi vnesenega niza poiskal vse serije, ki se ujemajo s tem nizom. Ko bo uporabnik izbral ustrezno serijo za dodajanje, se bo ta serija prikazala v glavnem grafu s toliko podatki, kolikor jih je uporabnik nastavil v tekstovnih poljih. Ta korak ponavlja, dokler nima vseh želenih serij izrisanih na grafu. Prikazane serije niso vezane samo na eno frekvenco podatkov, ampak so lahko mešane in se graf avtomatsko popravi glede na vnesene serije (dodajo se dodatne x in y osi). Ta funkcionalnost je še vedno testni prototip in ga je potrebno še dokončati ter optimizirati za hitrostno izvajanje, ker je v trenutnem stanju pridobivanje časovnih serij iz lokalnih baz še počasno in je na tem področju še veliko prostora za izboljšavo.

Slika: 16: Izgled prototipa hitrega izrisovanja časovnih serij na grafu



Vir: lastno delo.

Orodje za hiter pregled podatkov omogoča hitrejšo primarno analizo in pregled podatkov ter služi kot začetna točka bolj poglobljene analize. Za analitike je pogosto koristen grafični prikaz uporabljenih časovnih vrst, saj lahko tako hitro preverijo prisotnost trendov, strukturnih prelomov in povezav med časovnimi vrstami. Še posebej uporaben je prikaz časovnih vrst z različnimi frekvencami na istem grafikonu, saj to pomeni, da za sam prikaz in primerjavo podatkov ni potrebno ločeno izračunavati transformiranih časovnih vrst, kar znatno skrajša čas priprave grafikonov s koristnimi informacijami. Priprava podatkov je še vedno ključen del analitskega dela, ki pa je s primerno programsko opremo lahko znatno olajšan in pospešen.

4.5 Uporabljena tehnologija

Za realizacijo projekta smo uporabljali Microsoftovo programsko opremo Visual Studio 2019. Visual studio je integrirano razvojno okolje, s katerim lahko urejamo, razhroščamo in ustvarjamo programsko kodo. Integrirano razvojno okolje je vrsta programa, ki podpira s številnimi bogatimi funkcijami, ki se lahko uporabljajo za številne vidike razvoja programske opreme, in ima poleg standardnega urejevalnika kode in razhroščevalnika tudi prevajalnike, ki so zadolženi za prevajanje programske kode v strojno kodo, ki jo bo računalnik znal izvesti. Vsebuje tudi možnost avtomatskega dopolnjevanja kode z uporabo IntelliSense komponente, izdelavo grafičnih elementov in številne druge funkcije, ki olajšajo postopek razvoja programske opreme.

Vsa koda, ki je bila napisana za delovanje novega informacijskega sistema, je bila napisana v programskem jeziku C#. C# je sodoben, objektno orientiran programski jezik, ki razvijalcem omogoča izdelavo mnogih varnih in robustnih aplikacij, ki se izvajajo v

Microsoftovem .NET ekosistemu (Visual Studio). Od samega začetka se je v tem programskem jeziku dodajalo dodatne nove funkcije za podporo novim delovnim zahtevam in novim praksam oblikovanja programske opreme.

Grafična podoba sistema je bila napisana s pomočjo uporabe Windows Presentation Foundation (WPF), ki je tudi del .NET okolja. WPF je brezplačen in odprtokodni grafični podsistem, ki ga je razvil Microsoft za ustvarjanje grafičnih vmesnikov v aplikacijah, ki temeljijo na operacijskem sistemu Windows. WPF ločuje grafične elemente od same poslovne logike v programski kodi, kar pomeni, da se grafične elemente programira v XAML kodi, ki je del WPF podsistema, programsko logiko za te grafične elemente pa se piše v programski kodi C#.

4.6 Primerjava s prejšnjo rešitvijo

Glede na to, da je bil ključen dejavnik pri novemu projektu hitrost prenosa podatkov od spletnega vira do zapisa v podatkovno bazo, lahko sedaj primerjamo razlike v času. Za zajeme glavnih podatkovnih virov, kot so SURS, se je na starem sistemu porabilo petdeset minut za celoten zajem in vpis. Novo postavljeni sistem sedaj potrebuje petnajst minut za celoten postopek, kar je 70 % pohitritev; z uvedbo novega sistema zajemanja podatkov se ječas, potreben za pridobitev podatkov, bistveno skrajšal, kar omogoča zaposlenim na banki opazno izboljšavo pri pridobivanju podatkov; tudi če gre samo za zajem manjših sklopov podatkov, bo pohitritev tudi v teh primerih vidna. S pohitritvijo omogočamo, da lahko zaposleni prej začnejo izdelovati analize in povzetke teh podatkov v primerih, ko morajo naloge oddati že isti dan, ko pridejo novi podatki, ker se je postopek pridobivanja in zapisovanja podatkov bistveno zmanjšal. Pri zajemanju podatkov ima novi sistem tudi dodane funkcionalnosti, ki jih stari sistem ni vseboval. Ena izmed teh funkcionalnostije poročanje napak med procesom zajemanja podatkov. V novem sistemu se sedajbeležijo vse možne napake, ki se lahko zgodijo med izvajanjem, in se vse informacije o napakah zapišejo v tekstovne datoteke na omrežnem disku. Beleženje izvajanja lahko zaposlenim zelo pomaga pri ugotavljanju zakaj, na primer, se nekatere časovne vrste niso napolnile s podatki, tako da pregledajo te datoteke, v katere se zapisuje izvajanje za vsak zajem posebej, da je bolj pregledno. S to funkcionalnostjo zelo skrajšamo potreben čas za iskanje in ugotavljanje napak, ker imamo že vse to zapisano v tekstovnih datotekah brez potrebe bo razhroščevanju programske kode.

Ker je zajemov za vse sklope podatkov veliko, se datoteke z informacijami o napakah drastično povečajo v količini. Za odpravljanje te napake ima novi sistem ustvarjen pregledovalnik teh datotek na uvodnem prikaznem oknu, takoj ko uporabniki odprejo ta sistem. Na oknu imajo na voljo iskalnik za lažje iskanje specifičnih zajemov, poleg tega pa so vključene še posebne datoteke, ki hranijo strnjen opis izvajanja vseh zajemov. S to posebno datoteko lahko uporabniki naredijo hiter pregled celotnega izvajanja brez potrebe po pregledovanju vsake datoteke posebej. Za dodatek je bila tudi ustvarjena funkcija, ki se zažene vsak dan zgodaj zjutraj in pošlje strnjene opise izvajanja zaposlenim direktno v

elektronski poštni nabiralnik, da lahko že takoj, ko pridejo na delovno mesto, pregledajo, če je prišlo do kakšnih napak med izvajanjem zajemov podatkov.

Stari sistem je imel opisnike shranjene v več Excel datotekah, ki so poleg hranjenja informacij o časovnih vrstah služili tudi kot navigatorji med ponudniki virov in podatkovnih sklopov. Problem je nastal pri iskanju informacij o časovnih serijah, ker v Excel datoteke ni bilo vključenih nobenih iskalnikov, da bi lahko zaposleni prej dobili iskane podatke. Po starem postopku so morali te informacije ročno iskati tako, da so brali opise vseh podatkovnih sklopov, dokler niso dobili ustrezne informacije. Prenovljeni opisniki na novem sistemu delujejo na enak način kot prej, le da je sedaj vključen iskalnik, s katerim lahko uporabniki iščejo želene serije preko tega iskalnika, kar jim bistveno skrajša čas iskanja.

Novo postavljena rešitev na oddelku sedaj vsebuje vse naštete funkcionalnosti znotraj ene same aplikacije. Uporabniki lahko dostopajo do vsega na enemu mestu, medtem ko so bile vse te funkcionalnosti na stari način razmetane po več lokacijah na skupnem disku. Nov sistem je popolnoma centraliziran in s tem načinom tudi prihranimo nekaj časa. Aplikacija je tudi prenosljiva in namesto da se uporabniki povezujejo na spletni strežnik, si vsakokrat, ko iščejo podatke, lahko aplikacijo prenesejo na lastni službeni računalnik in jo uporabljajo brez uporabe strežnika. Poleg kopiranja aplikacije na računalnik bodo lahko prekopirali tudi konfiguracijske datoteke zajemov in opisnike ter še dodatno ustvarili konfiguracijsko datoteko, v kateri bodo nastavili poti do zajemov in opisnikov. Če bodo spreminjali opisnike ali zajeme, bodo spremembe vidne samo na njihovih računalnikih in ne na strežniku. Na projektu je delovala samo ena oseba in kot strošek razvijanja bomo upoštevali mesečno bruto plačo zaposlenega. Ob predpostavki, da je zaposleni na mesec opravil 160 ur dela, kar pomeni, da je bil mesečni strošek v višini 1.398,4 € bruto. Projekt je v celoti od začetka do konca trajal sedem mesecev, kar pomeni, da so stroški razvijanja programske opreme znašali skupaj 9.788,8 €. Še dodaten strošek pri izdelavi, poleg mesečne bruto plače zaposlenega, je bila uporaba programske opreme Visual Studio 2019 Enterprise različice, za katero nova licenca stane 5.000 € za eno leto. Celotni stroški projekta so bili skupaj z razvijanjem in licenco v višini 14.788,8 €. Za vzdrževanje novega sistema se trenutno opravi osem ur tedensko, kar na koncu meseca znaša 279,68 € bruto plače zaposlenega plus še letne obnovitve licence za Visual Studio katera znaša 2500€ in bi letno to naneslo 5.856,16 € bruto.

Za izračun koristi sem uporabil število porabljenih delovnih ur uporabnikov pri uporabljanju prejšnjega sistema in nato odštel število ur, ki jih porabijo pri uporabi novega sistema. Delovne aktivnosti, za katere sem izmeril porabljen čas za izvedbo, so naslednje:

- Prenašanje podatkov iz vira v podatkovno bazo;
- Popravljanje napak sistema;
- Pregledovanje podatkov;
- Dodajanje novih zajemov;
- Ustvarjanje analiz.

Pri uporabi starega sistema se je izkazalo, da so pri opravljanju delovnih opravil uporabniki letno porabili povprečno 2.976 delovnih ur, kar bi z njihovimi plačilnimi razredi znašalo v celoti 29.164,8 € bruto letno. Z uporabo novega sistema sem izmeril, da se je čas, porabljen za dokončanje enakih opravil, zmanjšal za 64,5 %, kar znaša 1.056 ur oziroma 10.348,8 € bruto letno. Na podlagi pridobljenih podatkov sem izračunal, da vrednost koristi znašajo v celoti 1.8816 €. Pri uporabi analize stroškov in koristi je potrebno določiti tudi diskontno stopnjo. Za vrednost denarja se v sedanjosti smatra, da je višja kot pričakovane vrednosti finančnih donosov v prihodnosti. Dlje ko je potencialna korist ali strošek v prihodnosti, manjša je njegova vrednost. Tukaj se za predvidene stroške in koristi projekta v času njegove življenjske dobe uporablja diskontna stopnja, da se vrednost prihodka v prihodnosti pretvoriv današnjo vrednost. Diskontna stopnja se razlikuje glede na vsako državo posebej, čas trajanja projekta in za kateri tip projekta gre. V Sloveniji je diskontna stopnja zelo nizka in sem za potrebe analize uporabil 1% (European Comission, 2021).

Tabela 1: Analiza stroškov in koristi

Analiza stroškov in koristi	Leto 0	Leto 1	Leto 2	Leto 3	Leto 4	Leto 5	Skupaj
Koristi v €		18.816	18.816	18.816	18.816	18.816	
Diskontna stopnja (1 %)		0,990	0,980	0,970	0,960	0,951	
Sedanja vrednost koristi v €		18.629,7	18.445,2	18.262,6	18.081,8	17.902,7	91.322
Stroški razvoja v €	14.788,8						14.788,8
Stroški vzdrževanja v €		5.856,16	5.856,16	5.856,16	5.856,16	5.856,16	
Diskontna stopnja (1 %)		0,990	0,980	0,970	0,960	0,951	
Sedanja vrednost stroškov v €		5.807,1	5.749,6	5.692,6	5.636,3	5.580,5	28.466,1
Obdobje vračila investicije	28.466,1 / 91.322 = 0,31 leta oziroma 3,7 mesecev						
5 letna stopnja donosa naložbe v %	(91.322 - (14.788,8 +28.466,1)) / (14.788,8 + 28.466,1) = 1,111 %						

Vir: lastno delo.

Iz rezultatov analize stroškov in koristi prikazanih v tabeli 1, ob predpostavki življenjske dobe rešitve 5 let, kažejo, da je investicija ekonomsko upravičena. Investicijase povrne nazaj v dobrih štirih mesecih. Koristi, ki so nastale zaradi te investicije, so, da imajo zaposleni na oddelku z novim sistemom bistveno hitrejši zajem in vpis podatkov v podatkovno bazo, kar jim omogoča več časa za ostala delovna opravila. Nov sistem omogočavse funkcije, ki so jih želeli znotraj ene aplikacije. Odpadla je potreba po iskanju različnih programov za vsako funkcijo posebej, ki se nahajajo na več lokacijah na omrežnem disku. Dodatno korist, ki so jo pridobili z novim sistemom, je tudi, da javlja vse možne napake pri izvajanju strganja podatkov in ostalih funkcij. Če pride slučajno do težav, jim bo sistem napake javil in zabeležil v tekstovne datoteke, ki jih tudi vsako jutro pošlje uporabnikom preko elektronskega sporočila.

SKLEP

Pri prenavljanju sistema za pridobivanje podatkov iz spletnih virov smo se odločali med več različnimi možnostmi pridobitve programske opreme. Na izbrani banki smo imeli možnost nakupa obstoječe opreme, ki bi opravljala te funkcionalnosti, najem zunanjih izvajalcev, ki bi ponudili že obstoječi produkt ali pa bi razvili celo rešitev za potrebe oddelka ali pa samostojni razvoj sistema znotraj banke. Obstoječa komercialna programska oprema bi rešila veliko problemov, ki smo jih zabeležili pri zbiranju zahtev. Za zahteve, ki so specifične samo za ta oddelek pa bi morali kljub temu dodatno razvijati funkcionalnosti, ki bi podpirale obstoječi sistem, ne samo na oddelku, pač pa tudi na ravni banke. Nakup programske opreme bi dodatno predstavljal problem v primerih, ko zajemi ne bi delovali več in bi bili odvisni od njih, kar bi terjalo veliko časa in stroškov za popravila.

Enake dileme bi bile prisotne tudi pri zunanjih izvajalcih, ker bi bili tudi s tem načinom odvisni od drugih, kar zadeva popravljanja. Ključni faktor na oddelku je hitrost in ko so zunanji izvajalci vpleteni v proces, lahko to predstavlja zakasnitev pri izvajanju vsakodnevnih opravil na oddelku. Spletno strganje podatkov je take narave, da podatki, ki jih pridobivamo, niso vedno dostopni na istem spletnem naslovu, ker se ponudniki virov lahko kadarkoli odločijo, da spremenijo postopek pridobivanja svojih podatkov, karpomeni, da je potrebno popravljati obstoječo kodo za pridobitev podatkov, da se boprilagajala novemu načinu.

Za uresničitev zahtev oddelka je bila zato najboljša odločitev interno razvijanje sistema. Tako lahko upoštevamo vse zahteve, ki so specifične za izbrano banko, in lahko nov sistem lažje prilagajamo in integriramo v obstoječe sisteme na ravni banke. Interni razvoj ima seveda tudi svoje zahteve, predvsem, da imajo zaposleni dovolj znanja za realizacijo projekta; čas vzdrževanja se bo bistveno povečal, stroški bodo bistveno manjši, kot če bi kupili obstoječo programsko opremo ali najeli zunanje izvajalce; je pa čas, porabljen za izgradnjo celotnega sistema, precejšen.

Prenovljeni sistem je popolnoma centraliziran in zaposleni imajo na voljo vse funkcionalnosti znotraj ene aplikacije. Zajem podatkov na novemu sistemu se je bistveno pohitril v primerjavi s prejšnjim in dodala se je tudi opcija beleženja statusa izvajanja zajemov ter napak, ki so se zgodile med pridobitvijo podatkov. Vse te informacije so na voljo uporabnikom na uvodnem zaslonu sistema, da lahko takoj vidijo, če je šlo kaj narobe in ustrezno reagirajo na podlagi teh informacij. Opisniki so bili razširjeni z možnostjo iskalnika, katerega lahko uporabljajo za iskanje časovnih serij na podlagi uporabnikovega vpisanega niza. Interni razvoj je od začetka do konca trajal okoli sedem mesecev, na projektu je delala samo ena zaposlena oseba. Nov sistem obsega vse zahteve, ki so bile definirane za začetno fazo in trenutno se zbirajo nove zahteve za nadaljnji razvoj oziroma vzdrževanje.

LITERATURA IN VIRI

1. Abdurachman, E., Gaol, F. L. & Soewito, B. (2019). Survey on threats and risks in the cloud computing environment. *Procedia Computer Science, 161*, 1325-1332.
2. Abreu, F. B. & Carapuça, R. (1994). Object-oriented software engineering: Measuring and controlling the development process. *Proceedings of the 4th international conference on software quality 3*(5), 1-8.
3. Alpar, P. & Saharia, A. N. (1995). Outsourcing information system functions: an organization economics perspective. *Journal of Organizational Computing and Electronic Commerce, 5*(3), 197-217.
4. Amlani, R. D. (2012). Advantages and limitations of different SDLC models. *International Journal of Computer Applications & Information Technology, 1*(3), 6-11.
5. Barros, O. (1994). Object-oriented case-supported development of information systems. *Journal of Systems and Software 24*(2), 95-113.
6. Bassil, Y. (2012). A simulation model for the waterfall software development life cycle. *International Journal of Engineering & Technology (iJET), 2*(5), 1-7.
7. Bourgeois, D. (2014). *Information systems for business and beyond*. Washington, DC: The Saylor Foundation.
8. Broad, J. (2013). *Risk Management Framework: A lab-based approach to securing Information Systems*. Waltham: Syngress.
9. Buchanan, E. (2017). Internet Research Ethics: Twenty Years Later. V M. Zimmer & K. Kinder-Kurlanda (ur.), *Internet Research Ethics for the Social Age: New Challenges, Cases, and Contexts* (str. xxix-xxxiii). Bern: Peter Lang International Academic Publishers.
10. Chan, K. Y. & Thong, Y. L. (2009). Acceptance of agile methodologies: A critical review and conceptual framework. *Decision Support Systems, 46*(4), 803-814.
11. Chowdhury, A. E., Bhowmik, A., Hasan, H. & Rahim, M. S. (2017). Analysis of the Veracities of Industry Used Software Development Life Cycle Methodologies. *AIUB Journal of Science and Engineering (AJSE), 16*(2), 75-82.
12. Coad, P. & Yourdon, E. (1991). *Object-oriented analysis*. Englewood Cliffs: Yourdon press.
13. Daud, N. M. N., Bakar, N. A. A. A. & Rusli, H. M. (2010). Implementing rapid application development (RAD) methodology in developing practical training application system. *2010 International Symposium on Information Technology, 3*, 1664-1667.
14. Davern, M. J. & Kauffmanm, R. J. (2000). Discovering potential and realizing value from information technology investments. *Journal of Management Information Systems, 16*(4), 121-143.
15. De la Prieta, F., Rodríguez-González, S., Chamoso, P., Corchado, M. & Bajo, J. (2019). Survey of agent-based cloud computing applications. *Future Generation Computer Systems, 100*, 223-236.
16. Dewi, L. C. & Chandra, A. (2019). Social media web scraping using social media developers api and regex. *Procedia Computer Science, 157*, 444-449.

17. Dreyer, A. J. & Stockton, J. (2013). Internet “data scraping”: A primer for counseling clients. *New York Law Journal, 7*, 1-3.
18. Drèze, J. & Stern, N. (1987), str. 909-989. The theory of cost-benefit analysis. *Handbook of public economics* (2. izd.). Amsterdam: Elsevier.
19. European Comission (2021). Reference/discount rates and recovery interest rates. Pridobljeno septembra 2021 iz https://ec.europa.eu/competition-policy/system/files/2021-09/reference_rates_base_rates2021_10%20.pdf
20. Feeny, D., Lacity, M., & Willcocks, L. P. (2005). Taking the measure of outsourcing providers. *MIT Sloan management review, 46*(3), 41.
21. Fowler, M. & Highsmith, J. (2001). The agile manifesto. *Software development, 9*(8), 28-35.
22. Geambaşu, C. V., Jianu, I., Jianu, I. & Gavrilă, A. (2011). Influence factors for the choice of a software development methodology. *Accounting and Management Information Systems, 10*(4), 479-494.
23. Glez-Peña, D., Lourenço, A., López-Fernández, H., Reboiro-Jato, M. & Fdez-Riverola, F. (2014). Web scraping technologies in an API world. *Briefings in bioinformatics, 15*(5), 788-797.
24. Gradišar, M., Jaklič, J. & Turk, T. (2007). *Osnove poslovne informatike*. Ljubljana: Ekonomska fakulteta.
25. Hirschey, J. K. (2014). Symbiotic relationships: Pragmatic acceptance of data scraping. *Berkeley Tech. LJ, 29*, 897.
26. Hoffer, J., George, J. & Valacich, J. (2020). *Modern System Analysis and Design*. Boston: Pearson.
27. Ives, B. & Krotov, V. (2006). Anything you search can be used against you in a court of law: Data mining in search archives. *Communications of the Association for Information Systems, 18*(1), 29.
28. Juhász, L. (2011). Net present value versus internal rate of return. *Economics & Sociology, 4*(1), 46-53.
29. Jula, A., Sundararajan, E. & Othman, Z. (2014). Cloud computing service composition: A systematic literature review. *Expert Systems with Applications 41*(8), 3809-3824.
30. Junjoewong, L., Sangnapachai, S. & Sunetnanta, T. (2018). ProCircle: A promotion platform using crowdsourcing and web data scraping technique. *2018 Seventh ICT International Student Project Conference (ICT-ISPC)*, 1-5.
31. Karimi-Alaghehband, F. & Rivard, S. (2020). IT outsourcing success: A dynamic capability-based model. *The Journal of Strategic Information Systems, 29*(1), 1-20.
32. Kent, B. (2000). *Extreme Programming Explained: Embrace Change*. Boston: Pearson.
33. King, J. L. & Schrems, E. L. (1978). Cost-benefit analysis in information systems development and operation. *ACM Computing Surveys (CSUR), 10*(1), 19-34.
34. Layard, P. R. G. (1994). *Cost-benefit analysis*. Cambridge: Cambridge University Press.
35. Leau, Y. B., Loo, W. K., Tham, W. Y. & Tan, S. F. (2012). Software development life cycle AGILE vs traditional approaches. *International Conference on Information and Network Technology, 37*(1), 162-167.

36. Martin, R. C., Newkirk, J. & Koss, R. S. (2003). *Agile software development: principles, patterns, and practices* (2. izd.). Upper Saddle River: Prentice Hall.
37. Mason, R. O. (1986). Four ethical issues of the information age. *MIS quarterly, 10*, 5-12.
38. Mell, P. & Grance, T. (2011). The NIST Definition of Cloud Computing. *NIST Special Publication 800-145*.
39. Melville, N., Kraemer, K. & Gurbaxani, V. (2004). Information technology and organizational performance: an integrative model of IT business value. *MIS Quarterly, 28*(2), 283-322.
40. Mishan, E. J. & Quah, E. (2020). *Cost-benefit analysis*. London: Routledge.
41. Munassar, N. M. A. & Govardhan, A. (2010). A comparison between five models of software engineering. *International Journal of Computer Science Issues (IJCSI), 7*(5), 94.
42. Munzert, S., Rubba, C., Meißner, P. & Nyhuis, D. (2014). *Automated data collection with R: A practical guide to web scraping and text mining*. Chichester: John Wiley & Sons.
43. Nikiforova, O., Nikulsins, V. & Sukovskis, U. (2009). Integration of MDA framework into the model of traditional software development. *Databases and Information Systems V*, 229-239.
44. O'Brien, J. A. & Marakas, G. M. (2011). *Management Information Systems* (10. izd.). New York: McGraw-Hill.
45. Radack, S. (2009). *The system development life cycle (sdlc)*. National Institute of Standards and Technology.
46. Ragunath, P. K., Velmourougan, S., Davachelvan, P., Kayalvizhi, S. & Ravimohan, R. (2010). Evolving a new model (SDLC Model-2010) for software development life cycle (SDLC). *International Journal of Computer Science and Network Security, 10*(1), 112-119.
47. Rainer, R. K. & Prince, B. (2021). *Introduction to information systems*. Chichester: John Wiley & Sons.
48. Rosemann, M. & vom Brocke, J. (2015). The six core elements of business process management. *Handbook on business process management 1*, 105-122.
49. Tilley, S. & Rosenblatt, H. J. (2016). *Systems analysis and design*. Boston: Cengage Learning.
50. Ruivo, P. Johansson, B., Sarker, S., & Oliveira, T. (2020). The relationship between ERP capabilities, use, and value. *Computers in Industry, 117*, 1-15.
51. Saurkar, A. V., Pathare, K. G. & Gode, S. A. (2018). An overview on web scraping techniques and tools. *International Journal on Future Revolution in Computer Science & Communication Engineering, 4*(4), 363-367.
52. Setende, H. (2012). *Differences, advantages and disadvantages between in-house development IT systems and industry standard ERP system.* Pridobljeno 3. avgusta

2021 iz https://www.academia.edu/4865003/Differences_advantages_and_disadvantages_between_in_house_dev
53. Sharma, M. K. (2017). A study of SDLC to develop well engineered software. *International Journal of Advanced Research in Computer Science, 8*(3), 520-523.
54. Shore, J. (2007). *The Art of Agile Development: Pragmatic guide to agile software development*. Cambridge: O'Reilly Media, Inc.
55. Sirisuriya, D. S. (2015). A comparative study on web scraping. *Proceedings of 8th International Research Conference*, 135-140.
56. Snell, J. & Menaldo, N. (2016). Web scraping in an era of big data 2.0. *Bloomberg Law News*.
57. Song, H. H. (2020). Testing and evaluation system for cloud computing information security products. *Procedia Computer Science, 166*, 84-87.
58. Srivastava, A., Bhardwaj, S. & Saraswat, S. (2017). SCRUM model for agile methodology. *2017 International Conference on Computing, Communication and Automation (ICCCA)*, 864-869.
59. Špundak, M. (2014). Mixed agile/traditional project management methodology–reality or illusion?. *Procedia-Social and Behavioral Sciences, 119*, 939-948.
60. Stefanou, C. J. (2003). System development life cycle. *Encyclopedia of Information Systems 4*, 329-344
61. Szalvay, V. (2004). An introduction to agile software development. *Danube technologies, 3*, 1-11.
62. Verhoef, C. (2005). Quantifying the value of IT-investments. *Science of computer programming, 56*(3), 315-342.
63. Williams, L. (2010). Agile Software Development Methodologies and Practices. *Advances in Computers, 80*, 1-44.
64. Wirfs-Brock, R. J. & Johnson, R. E. (1990). Surveying current research in object-oriented design. *Communications of the ACM, 33*(9), 104-124.
65. Wixom, B. H. & Todd, P. A. (2005). A theoretical integration of user satisfaction and technology acceptance. *Information systems research, 16*(1), 85-102.
66. Zhao, B. (2017). Web scraping. V L. A. Schintler & C. L. McNeely (ur.), *Encyclopedia of big data* (str. 1–3). Cham: Springer International.
67. Zhou, M., Greenwell, R. & Tannock, J. (1994). Object-oriented methods for manufacturing information systems. *Computer integrated manufacturing systems, 7*(2), 113-121.