UNIVERZA V LJUBLJANI
EKONOMSKA FAKULTETA

ZAKLJUČNA STROKOVNA NALOGA VISOKE POSLOVNE ŠOLE

IZZIVI TESTERJEV PROGRAMSKIH REŠITEV

Ljubljana, avgust 2022 MAJA PADARŠIČ

IZJAVA O AVTORSTVU

Podpisana Maja Padaršič, študentka Ekonomske fakultete Univerze v Ljubljani, avtorica predloženega dela z naslovom Izzivi testerjev programskih rešitev, pripravljenega v sodelovanju s svetovalcem doc. dr. Antonom Manfredo

I Z J A V L J A M

1. da sem predloženo delo pripravila 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 poskrbela, 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 pridobila vsa potrebna dovoljenja za uporabo podatkov in avtorskih del v predloženem delu in jih v njem jasno označila;

7. da sem pri pripravi predloženega dela ravnala v skladu z etičnimi načeli in, kjer je to potrebno, za raziskavo pridobila 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 študentke: _____

KAZALO

KAZALO TABEL

KAZALO SLIK

SEZNAM KRATIC

angl. – angleško

API – (angl. Application Programming Interface); Programski vmesnik

FSD – (angl. Functional Specification Document); Dokument funkcionalne specifikacije

IT – (angl. Information Technology); Informacijska tehnologija

UI – (angl. User Interface); Uporabniški vmesnik

XML – (angl. Extensible Markup Language); Razširljiv označevalni jezik

UVOD

Programska rešitev je v najsplošnejšem pomenu niz navodil, ki računalniku povedo kako naj opravi določeno nalogo. Z uporabo interneta so nam programske rešitve omogočile, da dobimo informacije v trenutku in lahko komuniciramo z ljudmi po vsem svetu. Programske rešitve nam ostajajo v pomoč pri reševanju različnih življenjskih izzivov – na primer pri potovanju iz enega kraja v drugega z uporabo zemljevida Google Maps, klepet s prijatelji s pomočjo WhatsAppa in deljenje fotografij na Instagramu. Poleg vsakodnevnih izzivov pa rešujejo tudi poslovne izzive in se v splošnem uporabljajo za povečanje produktivnosti in natančno izvajanje poslovnih funkcij (Muniraja, Jawahar & Ismail, 2015).

Da pa lahko programske rešitve delujejo tako, kot je uporabnik predvidel in učinkovito služijo svojemu namenu, je tester programskih rešitev nepogrešljiv. Tester programskih rešitev je del razvijalske ekipe, ki preverja kakovost in pravilnost delovanja razvite rešitve. Šele po uspešnem testiranju in dokončni potrditvi iz strani testerja, je rešitev pripravljena za uporabo pri končnem uporabniku. Na ta način se končnim uporabnikom prihranijo frustracije pri uporabi rešitve, ki ne bi delovala dobro, zmanjša se potreba po tehnični podpori in izboljša uporabniška izkušnja. Testerji hkrati poskrbijo, da razvojna ekipa in sam ponudnik programske rešitve v očeh kupcev izgledajo dobro, saj s kakovostnim testiranjem zagotavljajo, da programska rešitev deluje tako, kot je bila zasnovana in prvotno predstavljena kupcu.

Namen zaključnega dela je posameznikom, ki jih zanima delo na področju razvoja informacijskih rešitev, približati poklic testerja programskih rešitev, ter na primeru iz izbranega podjetja pokazati, s katerimi izzivi se soočajo testerji. Za ugotovljene izzive bodo podani predlogi izboljšav, kar je za podjetja, ki se ukvarjajo z razvojem programskih rešitev, lahko izhodišče za izboljšavo procesa testiranja programskih rešitev. Cilj je predstaviti poklic testerja programskih rešitev, ter veščine in znanja, ki so potrebna za opravljanje tega poklica. Želim jasno pokazati ugotovljene izzive na področju testiranja programskih rešitev, ter za njih podati konkretne rešitve in predloge za izboljšavo. Z zaključno nalogo želim pomagati posameznikom, ki jih zanima kako nastane programska rešitev in želijo začeti delati na tem področju, pa ne vedo kje začeti. Po prebranem delu bodo podrobno seznanjeni o odgovornostih in zadolžitvah testerjev, ter bodo lahko bolje pripravljeni na morebitno opravljanje takega dela. Sama sem opravljala študentsko delo kot tester programske rešitve. S pisanjem tega zaključnega dela želim podrobno raziskati in proučiti, kje je prostor za izboljšave, s katerimi se lahko pripomore k boljšemu in tekočemu poteku dela.

V uvodnem delu bom povedala kaj sploh je programska rešitev, ter predstavila metodologije razvoja programskih rešitev. Pri metodologijah se bom predvsem osredotočila na agilni pristop razvoja. Predstavljeni bodo tudi udeleženci, ki so navadno vključeni v razvoj programskih rešitev in njihova vloga. V naslednjem delu bo predstavljeno testiranje

programske rešitve, nivoji in metode testiranja. Podrobneje bo opisana vloga testerja programskih rešitev, ter s katerimi izzivi se lahko v vsakodnevnem delu le-ti soočajo.

Praktični del zaključne naloge bo predstavljal opis procesa testiranja programske rešitve v praksi in intervju s testerjem in razvijalcem v izbranem podjetju. Na podlagi intervjujev bom izvedela, s kakšnimi izzivi se soočajo testerji in s kakšnimi razvijalci pri delu s testerji. Na koncu bom predstavila temeljne ugotovitve in na podlagi ugotovitev predlagala izboljšave. V sklepu bom zapisala kako sem dosegla cilje zaključne strokovne naloge.

1 RAZVIJANJE PROGRAMSKIH REŠITEV

Razvoj programske rešitve se nanaša na sklop dejavnosti računalništva, ki so namenjene procesom ustvarjanja, oblikovanja, namestitve in podpore programski opremi (IBM, 2020). Pri razvoju programske rešitve vzamemo skupek zahtev uporabnika, jih analiziramo, oblikujemo rešitev problema in implementiramo rešitev na računalniku. Razvoj programske rešitve ni enak programiranju – programiranje je del implementacije v razvoju (Dooley, 2017, str. 1). Dandanes lahko skoraj povsod okoli nas opazimo stvari, ki uporabljajo programsko rešitev – ne samo v televizorjih, avtomobilih, ampak tudi v napravah kot so prodajni avtomati in kavni aparati - vsi ponujajo nekatere funkcije, ki jih poganja programska oprema in so narejeni tako, da se elektronski signali obnašajo po želenem vzorcu (Atlassian, 2022a).

Programska rešitev oz. oprema je splošni izraz za organizirano zbirko računalniških podatkov in navodil. Razdelimo jo na dve glavni kategoriji: sistemsko programsko opremo, ki zagotavlja osnovne funkcije, ki niso specifične za upravljanje nalog računalnika, in aplikacijsko programsko opremo, ki jo uporabniki uporabljajo za izvajanje določenih nalog oz. opravil. Sistemska programska oprema skrbi za kontrolo, integracijo in upravljanje posameznih komponent strojne opreme. Nekateri primeri le-te so operacijski sistem, gonilniki naprav, vdelana programska oprema, itd. Lahko bi tudi rekli, da je sistemska programska oprema 'posrednik' oz. vmesnik med uporabnikom (tj. aplikacijsko programsko opremo) in strojno opremo. Aplikacijska programska oprema deluje na zahtevo uporabnika na platformi, ki jo zagotavlja sistemska programska oprema – primeri aplikacijske programske opreme so urejevalnik besedila, podatkovne baze, multimedijska programska oprema, spletni brskalniki, itd. Aplikacijski programski opremi pravimo, da je uporabniški vmesnik, saj je vez med uporabnikom in aplikacijsko programsko opremo (Jordan University of Science & Technology, 2021).

Uporabniški vmesnik (angl. User Interface – UI) ima pomembno vlogo v programski rešitvi, saj je njegova sama zasnova primarnega pomena za prikazovanje tistih informacij, ki jih je uporabnik predvidel oz. zahteval. Vsaka manjša odločitev, sprejeta za oblikovanje uporabniškega vmesnika, lahko pozitivno ali negativno prispeva k programski rešitvi (Guntupalli, 2008). Pri uporabniškem vmesniku veliko vlogo igra tester programske rešitve,

saj skrbi za to, da je podoba usklajena skozi celotno aplikacijo, uporaba aplikacije čim bolj intuitivna in da so prikazane informacije, ki jih uporabnik pričakuje.

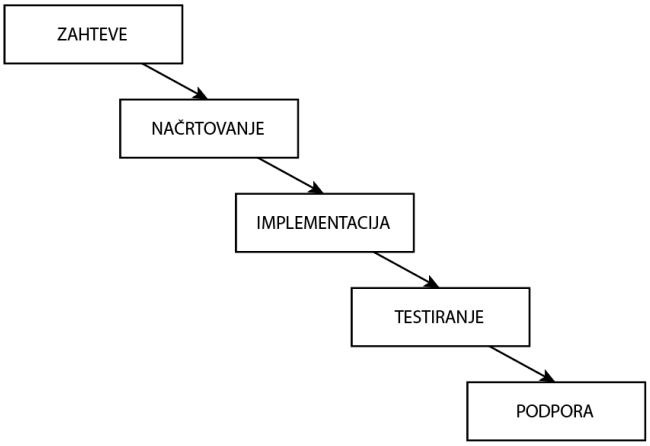
1.1 Metodologije razvoja programskih rešitev

Razvoj rešitve mora iti skozi določene faze življenjskega cikla, da je programska rešitev pripravljena za uporabo pri končnem uporabniku. Začne se z idejo, nato pa se zberejo vse zahteve za ustrezno rešitev. Temu sledi načrtovanje rešitve. Ko se ta faza zaključi, se prične kodiranje in hkratno odpravljanje napak (angl. Debugging), ki jih najde tester. Tester na koncu natančno pregleda in testira verzijo iz vidika končnega uporabnika. Ko zaključi s testiranjem in odobri verzijo, se izda nova verzija (angl. Release), ki je pripravljena za uporabo pri končnem uporabniku. Temu sledi vzdrževanje programske rešitve z odpravljanjem najdenih napak ali pa še dodaten razvoj, po zahtevah končnega uporabnika - stranke. Ko programska rešitev ne potrebuje več vzdrževanja ali pa se ne bo več uporabljala, sledi »upokojitev« rešitve (Dooley, 2017, str. 7).

Dooley (2017) razdeli modele razvoja programske rešitve na dva osnovna tipa – agilni in tradicionalni. Tradicionalni model se uporablja pri projektih, kjer so zahteve dobro znane in kjer je jasno, kakšen bo končni produkt. Temelji na načrtu, pri katerem gre projektna ekipa običajno skozi celoten življenjski cikel – od zbiranja zahtev do izdaje verzije, preden se začne delati na novi različici produkta. Kar se tiče razvojnih korakov in posameznih izdaj novih različic programske rešitve, je ta model bolj tradicionalen in strožji. Drugi tip je agilni razvojni model, ki danes prevladuje v hitro se spreminjajočih panogah; kot je razvoj programskih rešitev. Pri tem modelu gre projekta ekipa skozi delni življenjski cikel – običajno od faze načrtovanja do testiranja – te faze nekajkrat ponovi, preden se premakne do koraka izdaje produkta. Agilni model je postopen in deluje na osnovi tega, da majhne in bolj pogoste izdaje ustvarijo bolj stabilen produkt, v primerjavi z večjimi in manj pogostimi. V nadaljevanju bo opisana ena tradicionalna in ena agilna metodo razvoja programske rešitve. Na agilni metodi bo večji poudarek, saj se v informacijski tehnologiji (angl. Information Technology - IT) uporabljajo najpogosteje, če ne kar vedno.

Med tradicionalne metode štejemo model vodnega slapa (angl. Waterfall). Ta model se uporablja že zadnjih nekaj desetletij pri manjših in večjih projektih. Pri modelu vodnega slapa je vsaka posamezna faza popolnoma zaključena in preverjena, preden se nadaljuje z naslednjo. Kot vidimo na sliki 1, se najprej začne z zbiranjem, analizo in dokumentiranjem zahtev deležnikov. Sledi faza načrtovanja, implementacije in nato faza testiranja. Faza podpore se začne, ko je rešitev predana stranki. Ta model vključuje veliko dokumentacije, ki skrbi, da lahko projektna ekipa učinkovito sledi načrtu in razvije vse želene zahteve (Stober & Hansmann, 2010).

Slika 1: Model vodnega slapa



Prirejeno po Stober & Hansmann (2010, str. 16).

Kumar (2020) navaja, da je model vodnega slapa uporaben pri razvoju rešitev, ki imajo dobro definirane, jasne, fiksne in ne spreminjajoče se zahteve. Ko je delo določene faze zaključeno, je vrnitev k njej zelo zahtevna. Ta model tudi ne vsebuje smernic kako ravnati v primerih, ko pride do sprememb zahtev.

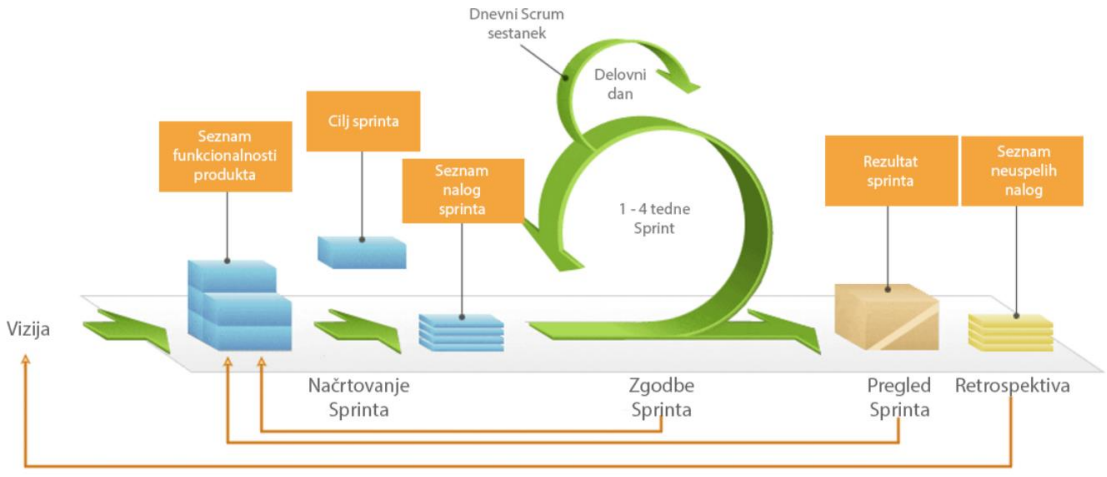
Obraten pristop tradicionalnemu razvoju programskih rešitev, je agilni pristop (angl. Agile). Vse se je začelo leta 2001, ko se je pojavil kratek dokument Manifest agilnosti (angl. Agile Manifesto), ki je bilo delo 17-ih podpisnikov. Ti so zaznali potrebo po alternativi takratnih metod razvoja programskih rešitev. Takratne metode so močno temeljile na načrtovanju, dokumentaciji in niso bile fleksibilne. V dokumentu je objavljenih 12 načel, ki naj bi jih upošteval vsak agilni izvajalec pri delu v projektnih ekipah razvoja programske rešitve. Načela govorijo o tem, da mora biti najvišja prioriteta vedno zadovoljitev stranke z zgodnjo in konsistentno dostavo programske rešitve. Dostava programske rešitve mora biti pogosta in izvedena v čim krajšem času – na nekaj tednov do največ nekaj mesecev. Spreminjanje zahtev je dobrodošlo, tudi pozno v razvoju. Komunikacija v ekipi je najbolj efektivna in učinkovita iz oči v oči. Pri projektu morajo osebe iz poslovnega in razvojnega vidika vsakodnevno tesno sodelovati med seboj. Posamezniki v projektni ekipi morajo biti motivirani. Omogočiti jim moramo le okolje in podporo, ter jim zaupati, da bodo svoje delo dobro opravili. Glavno merilo napredka je delujoča programska rešitev. Najboljša arhitektura, zahteve in načrtovanje izhajajo iz ekip, ki se same organizirajo. Ekipa mora v rednih intervalih komunicirati med seboj z namenom preverjanja napredka in odpravljanja možnih ovir (Stellman & Greene, 2014).

Agilni pristop daje poudarek na pomenu in vrednosti razvijalcev in članov ekipe, namesto da se osredotoča na dokončanje produkta. To omogoča razvijalcem in ostalim članom v ekipi bolj avtonomno delo. Lahko se osredotočijo na kakovost, namesto da bi samo izpolnili določeno funkcijo v ekipi. Je iterativen (ponavljajoči) pristop k razvoju, in uporablja

posamezne inkremente, ki zagotavljajo kakovostne rezultate. Primarni cilj je zadovoljiti stranko, ki je v agilnih projektih tako posel, kot končni uporabnik. Poudarka se ne da le na efektivnost in učinkovitost, ki ga programska rešitev za podjetje prinaša, temveč na pomembnost ljudi, ki bodo rešitev vsakodnevno uporabljali (Campbell, 2020). Med najpogostejše agilne metode štejemo Scrum in Kanban. Scrum metodologija bo v nadaljevanju tudi bolj podrobno opisana. Ta metodologija se uporablja tudi v izbranem IT-podjetju, ki bo predstavljeno v praktičnem delu tega dela.

Beseda skram (angl. Scrum) je têrmin, ki izhaja iz igre ragbija. Podobno kot igralci ragbi ekipe trenirajo za igro, Scrum spodbuja razvojno ekipo, da se uči skozi izkušnje, se samoorganizira med reševanjem težave in se nenehno izboljšuje (Drumond, 2020). Scrum projekti so organizirani v serijah iteracij oz. ponovitev, ki jih imenujemo sprint (angl. Sprint). Na sliki 2 vidimo prikaz agilne metodologije Scrum. Ponavadi posamezen sprint traja od dva do štiri tedne. Za vsak sprint se določi obseg dela, ki se ga opredeli glede na prioritetni seznam funkcionalnosti produkta (angl. Product backlog). Za oceno, koliko časa je potrebnega za dokončanje posameznega sprinta, se naredi seznam nalog (angl. Sprint backlog). Dnevno se organizirajo sestanki, na katerem se preverja napredek ekipe in odpravljajo morebitne ovire. Ekipa je samoorganizirana in odgovorna, da opravi vse naloge, ki so bile načrtovane v posameznem sprintu. V Scrum metodologiji imamo tri vloge – skrbnika metodologije (angl. Scrum master), ki skrbi da vsi procesi tečejo brez zapletov, lastnika produkta (angl. Product owner), ki skrbi da opravljeno delo prinaša čim večjo vrednost stranki, ter razvojno ekipo (angl. Team), ki je zadolžena za implementacijo načrtovanih nalog (Bauman, 2011).

Slika 2: Agilna metodologija Scrum



Prirejeno po Bauman (2011).

Posamezni sprint je sestavljen iz treh dogodkov. Prvi dogodek je sestanek za načrtovanje sprinta (angl. Sprint planning meeting), na katerem se dogovori katere naloge bodo implementirane. Vsakodnevno je organiziran dnevni sestanek (angl. Daily scrum meeting),

na katerem se preverja napredek ekipe pri implementaciji nalog, na koncu sprinta pa sledi sestanek namenjen pregledu sprinta (angl. Sprint review meeting), na katerem odgovorni v ekipi predstavijo inkrement produkta oz. rezultat sprinta (angl. Product increment) nadrejenim. Pred pričetkom novega sprinta se organizira retrospektiven sestanek (angl. Sprint retrospective meeting), kjer se pregleda kakovost razvojnega procesa (Bauman, 2011).

Agilni pristop pa nima samo prednosti, ampak tudi slabosti. Ena izmed najbolj očitnih je pomanjkanje dokumentacije, saj so količine podatkov strnjene v manjše uporabniške zgodbe (angl. User stories); to so razlage potreb določenih uporabnikov in zapisi, kako te potrebe izpolniti. Ti zapisi ne vsebujejo veliko podrobnosti in lahko pride do situacije, da razvijalci ne razumejo zahtev uporabnikov in nepopolno ali nepravilno izdelajo rešitev. Druga slabost agilnega pristopa, ki se velikokrat zgodi, je povečanje obsega projekta, saj se potrebe stranke nenehno spreminjajo, kar pa lahko vodi do preobremenjenosti ekipe. Prav tako morajo udeleženci ekipe v svojem urniku imeti na voljo čas za dnevni sestanek, kar lahko moti njihov potek dela in nenehno sodelovati s testerji, stranko in ostalimi deležniki projekta (Wrike, Inc., 2022).

1.2 Udeleženci v razvoju programske rešitve

Med udeleženci, ki so vključeni v razvoj programske rešitve, so navadno lastniki produkta, projektni vodja, arhitekt programskih rešitev, razvijalci programske rešitve, testerji programske rešitve in končni uporabniki.

Projektni vodja je predstavnik projektne ekipe, ki skrbi za organizacijo, administracijo in vodenje projekta. Arhitekt programskih rešitev iz tehničnega vidika oblikuje funkcionalnosti, odloča o tem, kako bodo funkcije razvite in katere tehnologije bodo uporabljene, ter izvaja ali nadzira pregled kode. Lastnik produkta zagotavlja vse potrebne informacije za razvoj produkta, podpira ekipo z vidika razvoja produkta in poslovne perspektive. Razvijalec oblikuje in implementira naloge v skladu s predloženimi specifikacijami, piše avtomatizirane teste na ravni enote in integracije, ter vzdržuje kodo in orodja v uporabi. Pomaga arhitektu pri pregledu kode, ter nudi podporo testerju pri testiranju in preverjanju. Tester je odgovoren za kakovost funkcionalnosti, ki jih razvije ekipa. Zagotavlja ustrezne postopke testiranja in usklajenost v skupini. Oblikuje in izvaja scenarije za testiranje programske rešitve, prijavlja napake, podpira razvijalce pri odpravljanju napak, piše poročila in po potrebi ponovno testira napake (Izbrano podjetje, 2021). Testiranje je ključna vloga za razvoj visoko kakovostne programske rešitve, saj odpravljanje napak pomeni višjo kakovost, zadovoljstvo stranke in končnih uporabnikov, ter stabilnost samega produkta.

2 TESTIRANJE PROGRAMSKE REŠITVE

Testiranje programske rešitve je postopek izvajanja programskega sistema z namenom ugotavljanja, ali ta ustreza definiranim specifikacijam in ali se pravilno izvrši v predvidenem okolju za razvoj programske rešitve. Testiranje ima pomembno vlogo pri doseganju kakovostne programske rešitve, saj s ponavljajočim ciklom testiranja – odkrivanja napak ter odpravljanja napak, izboljša kakovost produkta med samim razvojem (Naik & Tripathy, 2011). Lahko bi rekli da je testiranje tudi izvajanje procesa, z namenom iskanja napak v programu (Glenford, 2012). Ključni dejavnik za podporo pri testiranju je specifikacija, saj se z njo identificira pravilno in želeno delovanje programa. Testiranje se izvaja na novo razvitih programskih rešitvah, preden gredo v dejansko uporabo k končnemu uporabniku. Nepravilno vedenje je napaka programske rešitve. Napake so povzročene zaradi defekta v izvorni kodi, čemur pogosto rečemo hrošč v programski rešitvi (angl. Bug) oz. napaka (Whittaker, 2000). Tester najde in prijavi napako, vzrok napake pa običajno ugotovi razvijalec in napako odpravi.

Zelo priljubljeno programsko orodje, ki se uporablja pri agilnem razvoju programskih rešitev, je Jira. Uporablja se za upravljanje projektov, sledenje nalog ter napak, ki jih prijavi tester, komuniciranje s stranko, sledenje poteku dela in posameznim izdajam programske rešitve (Atlassian, 2022b). Jiro pogosto uporabljajo podjetja, ki razvijajo programske rešitve in je hkrati primerna za vse, ki potrebujejo orodje za upravljanje projektov in procesov. Je posebej prilagojena odzivnim, iterativnim in na stranke osredotočenim delovnim procesom agilne metodologije (Idalko, 2022). Z omenjenim orodjem lahko prioritiziramo, dodelimo, spremljamo in poročamo zahtevke; to so lahko napake v programski rešitvi, ki jih prijavi tester, izboljšave funkcionalnosti ali popis sprememb zahtev stranke.

Preden pa lahko tester prijavi napako v programski rešitvi, jo mora testirati. Vsaka programska rešitev gre skozi testiranja, katere lahko razdelimo na več nivojev in jih testirajo različni udeleženci z različnimi stopnjami tehničnega znanja, ali pa tehničnega znanja sploh nimajo. Testiranje na vseh teh nivojih je potrebno, da na koncu pridemo do tehnično izvedene rešitve, ki izpolnjuje želje stranke.

2.1 Nivoji testiranja

Testiranje programske rešitve gre skozi štiri nivoje, preden je rešitev pripravljena za končnega uporabnika. Nivoji testiranja si kronološko sledijo (Naik & Tripathy, 2011):

- Testiranje enot (angl. Unit testing)
- Integracijsko testiranje (angl. Integration testing)
- Sistemsko testiranje (angl. System testing)
- Testiranje sprejemljivosti (angl. Acceptance testing)

Prve tri nivoje testirajo različni udeleženci, ki so del razvijalske ekipe programske rešitve, testiranje sprejemljivosti pa izvede stranka oz. končni uporabnik, ki preveri, če so bile upoštevane vse njihove zahteve in ali je delovanje programske rešitve ustrezno (Naik & Tripathy, 2011).

Testiranje enot je testiranje na najmanjšem nivoju, kjer se testira posamezna enota (tj. posamezna funkcija, metoda, postopek, modul ali objekt) programske rešitve. Na ta način se izolira del kode in preveri njena pravilnost. Testiranje izvedejo razvijalci sami (Dooley, 2017).

Integracijsko testiranje je testiranje skupka modulov, ki so povezani drug z drugim. Programska rešitev je sestavljena iz različnih modulov, ki jih po navadi naredijo različni razvijalci. S tem se prepričamo, da se integrirajo skupaj in da ne prihaja do kakršnihkoli defektov v komunikaciji med njimi. Testiranje izvajajo razvijalci in testerji (Dooley, 2017).

Sistemsko testiranje je testiranje popolnoma integriranega sistema programske opreme. Je končna faza testiranja, kjer se testira celoten program iz perspektive uporabnika, izvaja pa ga tester. Pravimo mu tudi testiranje črne škatle (angl. Black-box testing), ker tester ne ve, kako je bila koda implementirana – pozna pa zahteve in na podlagi tega izvede teste (Dooley, 2017). Ko izvajalec, ki razvija programsko rešitev, konča s sistemskimi testi in popravi vse ali večino napak, je programska rešitev pripravljena za testiranje sprejemljivosti, ki ga izvede končni uporabnik (Graham, van Veenendaal, Evans & Black, 2008).

Testiranje sprejemljivosti se izvaja z namenom, da končni uporabnik preveri in potrdi ali programska rešitev ustreza definiranih zahtevam in ali je pripravljena na končno izdajo (Graham, van Veenendaal, Evans & Black, 2008). Rečemo mu tudi beta testiranje, saj testiranje izvedejo pravi uporabniki. Beta testiranje se izvaja po alfa testiranju in pred končno izdajo programske rešitve. Alfa testiranje izvajajo testerji na strani ekipe za razvoj programske rešitve. S končnim alfa testiranjem se zagotovi, da programska rešitev deluje brezhibno, preden jo damo na trg ali stranki v beta testiranje (JavaPoint, 2021).

2.2 Metode testiranja

Poznamo dve metodi testiranja – ročno in avtomatizirano. Kot že razberemo iz samega poimenovanja, se pri ročnem testiranju testni primeri izvajajo ročno. Nasprotno pa se pri avtomatiziranem testiranju testni primeri izvajajo s pomočjo orodij, skript in programskih rešitev.

2.2.1 Ročno testiranje

Pri ročnem testiranju tester ročno izvaja testne primere na uporabniškem vmesniku programske rešitve iz vidika končnega uporabnika. Pri tem ne uporablja nobenih orodij za avtomatizacijo testiranja. S svojim testiranjem preveri, ali programska rešitev deluje tako,

kot je bila zahtevana s strani stranke. V primeru da najde defekte oz. napake, jih prijavi razvijalcu. Razvijalec napako popravi in jo vrne testerju za testiranje (Singh, 2012).

Poznamo tri metode ročnega testiranja – po metodi bele škatle (angl. White-box testing), črne škatle (angl. Black-box testing) in sive škatle (angl. Gray-box testing).

Metoda bele škatle (poznana tudi kot logično usmerjeno testiranje) je metoda, pri kateri razvijalci testirajo notranjo strukturo programa – tj. preverjajo logike napisane kode. To velikokrat tudi pomeni zanemarjenje specifikacije zahtev. Zatem gre programska rešitev testerju, ki izvede testiranje po metodi črne škatle. Pri testiranju po metodi črne škatle (poznamo jo tudi kot podatkovno usmerjeno oz. vhodno-izhodno testiranje) gledamo program kot črno škatlo – ne poznamo notranje strukture programske rešitve oz. kode. Cilj je preko uporabniškega vmesnika preveriti ali program deluje v skladu z definirano specifikacijo stranke in iskanje okoliščin, v katerih ne deluje pravilno (Glenford, 2012). To metodo izvajajo testerji, s katero preverjajo funkcionalnost programske rešitve. Pogosta tehnika testiranja po metodi črne škatle je regresijsko testiranje. Z regresijskim testiranjem se preveri, da sprememba v programski kodi ne vpliva na obstoječo funkcionalnost produkta. To testiranje zagotavlja, da produkt še vedno deluje brezhibno z narejenimi novimi funkcionalnostmi, popravki napak ali kakršnimi koli spremembami obstoječih funkcij (JavaPoint, 2021). Testiranje po metodi sive škatle predstavlja kombinacijo metod bele in črne škatle. Izvedejo ga osebe z delnim znanjem notranje strukture programa. Namen te metode testiranja je iskanje napak zaradi nepravilne strukture kode ali nepravilne uporabe programske rešitve (Guru99, 2021).

2.2.2 Avtomatizirano testiranje

Postopek testiranja programske rešitve z uporabo avtomatiziranih orodij za iskanje defektov se imenuje avtomatizirano testiranje. Avtomatizirana orodja izvajajo vnaprej definirane akcije in generirajo rezultate. Tako lahko več časa namenimo drugim nalogam, hkrati pa avtomatiziramo teste, ki sicer vzamejo veliko časa – na primer regresijske teste. Kljub temu, da je potrebno tudi nekaj časa za vzdrževanje testnih skript, je še vedno najboljši način za povečanje učinkovitosti, pokritosti s testi in hitrosti testiranja (Guru99, 2021). Avtomatizacija testiranja je potrebna za sledenje tempa hitro naraščajočem obsegu in kompleksnosti programskih rešitev (O'Broin, 2015). Za primer prehoda iz ročnega na avtomatizirano testiranje je bila izvedena študija primera na industrijskem projektu, ki je razmeroma uspešno prešel iz ročnega testiranja na avtomatizirano z uporabo orodja za avtomatizirano testiranje. Prednosti avtomatiziranega testiranja so bile večja pokritost testov in krajši čas izvedbe testov. Teste, ki so prej trajali več ur, je bilo zdaj mogoče izvesti v nekaj minutah, ko so enkrat pravilno nastavljeni v avtomatiziranem orodju. Testi so nato ponovljivi in se zaženejo točno tako, ko so bili zapisani (Alegroth, Feldt & Olsson, 2013).

Dobri kandidati za avtomatizacijo testov so tisti, ki jih je težko izvesti ročno in tisti, ki so preveč zapleteni za ročno testiranje. Avtorja tudi navajata, da je vzdrževanje testnih skript

pogosta točka napak pri številnih projektih avtomatiziranja testov (Graham & Mark, 2012). Implementacija avtomatiziranja testiranja v podjetju je kompleksen podvig. Zgolj nakup orodja za avtomatizacijo redkokdaj izpolnjuje vse potrebe podjetja. Potreben je organiziran pristop in sodelovanje vodij projekta, ekipe za razvoj in ekipe za testiranje (O'Broin, 2015).

2.3 Vloga in izzivi poklica testerja programskih rešitev

Tester programskih rešitev igra kritično vlogo v razvoju programske rešitve. Odgovoren je za fazo testiranja v procesu razvoja programskih rešitev, z uspešnim testiranjem preprečiti, da bi do stranke prišla programska rešitev, ki bi imela napake v delovanju in poskrbeti, da rešitev deluje skladno z definirano specifikacijo. Tester ob testiranju programske rešitve razmišlja in jo uporablja z vidika končnega uporabnika. Ročno testiranje je ključni faktor za razvoj uporabniku prijazne programske rešitve. Njegova glavna odgovornost in želja je prispevanje k najvišji kakovosti končnega produkta (Onix-Systems, 2020).

Tester ima ustrezno znanje na področju testiranja, učinkovito komunicira z razvijalci in s stranko, izdeluje testne primere in dokumentacijo, analizira in poroča o rezultatih testiranja, ter nudi podporo stranki pri testiranju in uporabi programske rešitve. Zadolžen je za kakovost funkcionalnosti, ki jih razvije ekipa in izvaja testiranje na vseh ravneh (ročno in avtomatizirano testiranje). Odkriva, analizira in dokumentira najdene napake programske rešitve v izbrano programsko rešitev za upravljanje projekta. Oblikuje in izvaja scenarije za testiranje programske rešitve, podpira razvijalce pri odpravljanju napak in po potrebi ponovno testira napake. Razvita funkcionalnost je končana šele, ko jo potrdi tester. Tester sodeluje z razvijalci pri razvoju planov in možnih scenarijev za testiranje novih funkcionalnosti. Odgovoren je za pripravo in vzdrževanje testov za nove in obstoječe funkcionalnosti, ter za potrjevanje delovanja v produkcijskem okolju. Sodeluje pri načrtovanju in pisanju dokumentacije, ter spodbuja postopek za izboljšanje standardov, metod in orodij za zagotavljanje kakovosti. Tester mora biti radoveden in v programski rešitvi videti tisto, kar drugi spregledajo. Hkrati mora biti natančen, trmast, vztrajen in dovolj potrpežljiv, da najde čim več napak. Razumeti mora proces razvoja programskih rešitev in imeti dobre pisne sposobnosti (Izbrano podjetje, 2021).

Prva naloga na urniku testerja je testiranje funkcionalnosti ali popravka napak v programski rešitvi, ki ga dobi od razvijalca. Testiranje vključuje preverjanje same programske rešitve, da deluje kot je pričakovano in iskanje morebitnih napak. Ko tester prepozna napako, je njegova odgovornost, da skuša izslediti njen vzrok, to dokumentira in sporoči razvijalcu, ki bo kasneje to napako reševal. Ko je napaka rešena, sledi ponovno testiranje in končno poročilo o testiranju, ki da zeleno luč in potrdilo, da je test uspešen in napaka odpravljena. Tako je komunikacija in koordinacija z razvijalci zelo pomemben vidik vsakodnevnih aktivnosti testerja. Tester pa ne komunicira samo z razvijalci, ampak tudi z drugimi oddelki; npr. z poslovnimi analitiki, ki mu pomagajo razumeti, kako mora rešitev delovati (Kumari, 2018).

Testerji programskih rešitev se vsakodnevno soočajo s številnimi izzivi. Eno izmed najpogostejših je pomanjkanje dokumentacije, potrebne za testiranje. Dokumentacija je lahko slabe kakovosti, zapisana dvoumno, z manjkajočimi informacijami in zapisana za tistega, ki jo je napisal in ne tistega, ki jo bo bral. To otežuje delo pri ustreznem testiranju (Nayyar & Rizwan, 2009). Večina razvijalcev se zanaša na verbalno komunikacijo in testerjem ustno sporočajo o tem, kaj je potrebno potestirati. Ravno zaradi tega, lahko ostanejo nekatere komponente testa ne testirane, kar pomeni da lahko posledično v programski rešitvi ostanejo neodkrite napake. To vodi v nadaljnje testiranje in izgubo časa. Konkretna dokumentacija bi testerjem olajšala testiranje in zagotovila nemoteno delo (Kumari, 2018). Vseeno lahko kljub dobri dokumentaciji, neizkušeni testerji zaradi pomanjkanja znanja o programski rešitvi, odpirajo neveljavne napake ali pa ne razumejo zahtev (Zhang & Hunkapiller, 2017).

Testerji se soočajo tudi s časovnimi omejitvami pri testiranju. Potreben čas za izvedbo testiranja je pogosto ocenjen neustrezno, saj je namenjen čas za testiranje prekratek (Nayyar & Rizwan, 2009). Temu je tako, ker razvijalci v zadnjem trenutku zaključijo z razvojem, rešitev pa bi morala iti v kratkem času že naprej k stranki. To predstavlja velik pritisk na testerje, ki zaradi časovne stiske ne morejo opraviti tako dobrega testiranja, kot bi ga drugače. Rešitev izziva bi bil strog časovni razpored, ki bi se ga morali držati – tako bi vsak oddelek dobil pravičen delež svojega časa (Kumari, 2018). Za testerje, ki sodelujejo pri projektih večjega obsega, lahko postane regresijsko testiranje neobvladljivo, zaradi potrebnega obvladovanja testiranja trenutnih funkcionalnosti, preverjanja prejšnjih funkcionalnosti ter sledenja napakam. Delo je stresno zaradi časovnih omejitev, ker mora biti zaključeno v določenem času (Kumar, 2019).

Med testerji in razvijalci se lahko razvijejo tudi konflikti. Konflikti nastanejo zaradi tega, ker medsebojno ne komunicirajo primerno, učinkovito ali uspešno. Do tega recimo pride, ker razvijalec noče deliti znanja v podporo trditvi, da je napaka, ki jo je našel tester, veljavna. Ali pa tester slabo sporoči svoja pričakovanja glede delovanja rešitve in razvijalec posledično ne razvije ustrezne rešitve. Nastane lahko tudi pomanjkanje komunikacije med razvijalcem in testerjem, kar pomeni zamude in slabo kakovost dela. Prav tako lahko med njimi pride do negativnega vedenja in posledično slabega medsebojnega odnosa – na primer, če se tester baha in hvali s kritičnimi napakami, ki jih je našel v razvijalčevi kodi; ali pa se razvijalec čustveno odzove na identifikacijo napak v 'svoji kodi'. Do konfliktov pride lahko tudi, ker svojega dela ne dokumentirajo ustrezno in ne sledijo standardiziranim postopkom – to velja za razvijalce in testerje (Zhang & Hunkapiller, 2017).

3 PROCES TESTIRANJA PROGRAMSKIH REŠITEV IN IZZIVI TESTERJEV V IZBRANEM PODJETJU

3.1 Metoda raziskave

V empiričnem delu diplomskega dela bom predstavila proces testiranja programske rešitve v praksi, ter na podlagi intervjujev in opazovanja v izbranem podjetju raziskala, s katerimi izzivi se soočajo testerji programskih rešitev. Na drugi strani bom raziskala, v čem vidijo razvijalci, ki so v stalnem stiku s testerji, izzive testerjev. Do vprašanj, ki so bila zastavljena v intervjuju sem prišla na podlagi lastnih opažanj in teoretičnega dela.

Najprej bom na podlagi lastnih opažanj predstavila proces testiranja programske rešitve, saj sem v izbranem podjetju delala kot tester programskih rešitev in mi je proces zelo dobro poznan. Za raziskavo o izzivih testerjev programskih rešitev, odgovornostih, zadolžitvah, ter potrebnih veščinah in znanjih testerjev, sem izvedla pol strukturiran intervju s testerjem in razvijalcem v izbranem podjetju, z vsakim posebej. Intervju bo predstavljen po procesu testiranja programske rešitve. Intervju je potekal v živo in sem imela vnaprej definirana vprašanja v obliki zapisa. Vprašanja so bila zastavljena na način, da bi iz njih dobila odgovore, s katerimi bi čim bolj dosegla zastavljene cilje tega dela. Intervjuvancem sem zastavljala odprta vprašanja, ki so dopuščala proste odgovore.

3.2 Predstavitev izbranega podjetja in programske rešitve

Proces testiranja programskih rešitev bom opisala na podlagi lastnih izkušenj v izbranem podjetju. Izbrano podjetje deluje že več kot tri desetletja na trgu kot ponudnik programske rešitve za zavarovalništvo v Sloveniji in ima pisarne v šestih različnih državah po vsej Evropi. S svojo programsko rešitvijo pomagajo zavarovalnicam povečati digitalno zmogljivost in uspešnost na svojem področju. Programska rešitev temelji na najboljših praksah v industriji in povezuje celotno organizacijo s ponudniki storitev, strankami in partnerji. Sestavljena je iz različnih modulov – modul za administracijo polic, registracijo in pregled oseb, računovodstvo in fakturiranje, škode in ostalih modulov, ki se med seboj povezujejo in skupaj sestavljajo celovito zavarovalniško programsko rešitev.

Rešitev je sestavljena iz lastno razvite platforme in vsebinske konfiguracije iz področja zavarovalništva. Platforma ponuja konfiguracijske gradnike, ki zagotavljajo formaliziran pristop k razvoju, standardizirane programske vmesnike (angl. Application Programming Interface - API), uporabniške vmesnike, integracijo in potek dela. Platforma podpira celoten življenjski cikel razvoja in je dobro dokumentirana. Konfiguracija programske rešitve izkorišča gradnike, ki jih ponuja platforma in predstavlja poslovne značilnosti rešitve, ter standardno izvedbo zavarovalniških procesov, ki se prilagajajo s konfiguracijo in ne s kodiranjem (Izbrano podjetje, 2021).

3.3 Proces testiranja programske rešitve v izbranem podjetju

Delovni dan testerja v implementacijski ekipi se prične s pregledom e-mailov in statusov zahtevkov v Jiri. Zahtevki so lahko na primer naloge ali napake, ki jih prijavi stranka ali pa so odprti interno, znotraj razvojne ekipe. Ko so zahtevki pripravljeni na testiranje, si tester določi prioritete za testiranje – vse je odvisno od tega, kateri zahtevki morajo biti prioritetno testirani za namestitev v verzijo, ki se bo nameščala k stranki. Podjetje uporablja Scrum agilno metodologijo pri razvoju programskih rešitev in po projektu dnevno organizirajo t.i. Scrum dnevne sestanke preko spletne komunikacijske platforme. Na teh sestankih, ki niso daljši od 15 minut, se zbere celotna projekta ekipa. Vsak posameznik ekipe poroča o tem, kaj je delal prejšnji dan, kakšen je plan za delo v tistem dnevu in ali je naletel na kakšno težavo.

Tipične naloge, ki jih ima tester skozi dan, so ročno testiranje programske rešitve, ki jo je razvilo izbrano podjetje, ponovno testiranje napak in novih funkcionalnosti, ter kasneje tudi podpora stranki. Ročno testiranje pomeni, da se pri testiranju ne uporablja nobenega orodja, namreč se preko uporabniškega vmesnika preverja, da programska rešitev deluje po pričakovanjih. Takrat pravimo, da izvaja sistemsko testiranje (po metodi črne škatle). Da tester lahko pravilno testira, mora razumeti zahteve stranke. Zahteve so navadno popisane v samih zahtevkih na Jiri, ki so predmet testiranja. V primeru da zahteve niso popisane ali pa niso jasne, lahko tester pogleda v dokument funkcionalne specifikacije (angl. Functional Specification Document – v nadaljevanju FSD), ki so ga skupaj s stranko predhodno definirali interni poslovni analitiki. V FSD-ju so popisane vse zahteve stranke, ki so bile potrjene pred začetkom projekta. Če zadeve ne najde v tem dokumentu, lahko tester vpraša tudi stranko za pojasnila – tako si zagotovi, da je funkcionalnost narejena po zahtevah stranke.

Velikokrat se zgodi, da se pri testiranju določene funkcionalnosti ali ponovnem testiranju napake, odkrije nove napake. Tester napako prijavi v Jiro. Določi ime napake, tip zahtevka (na primer: napaka, naloga, izboljšava, nova funkcionalnost), prioriteto, komponente, oznake za boljšo organiziranost in razvrščanje zahtevkov, ter seveda opis napake. V podjetju je uveljavljena enotna oblika za opis napak, prikazana na sliki 3. V tem primeru je prijavljena napaka, ki uporabnikom rešitve onemogoča dodajanje plačilnih nalogov v nov paket, ki so bili prej v paketu, ki je bil izbrisan. Tester najprej zapiše korake za ponovitev oz. reproduciranje napake (angl. Repro steps), nato opiše dejansko obnašanje sistema (angl. Actual behaviour) in kakšno je pričakovano obnašanje (angl. Expected behaviour). Tester napako dodeli primernemu razvijalcu, odvisno od področja, na katero se nanaša napaka. Razvijalci znotraj ekipe so razporejeni v manjše ekipe po področjih, ki jih programska rešitev pokriva in vsaka ima svojega tehnično vodjo. Razvijalec, ki se loti reševanja napake, v primeru nejasnosti najprej kontaktira testerja, ki je napako prijavil. Ta mu pomaga pri reproduciranju napake, v primeru tehničnih vprašanj, pa se obrne na svojega vodjo.

Slika 3: Prijavljena napaka v Jiri

Description

Plačilni nalogi, ki so bili v paketu, ki je bil izbrisan, niso odstranjeni iz tega paketa.
Ti plačilni nalogi zaradi tega tudi ne morejo biti dodani v nov paket.

Repro

1. Računovodstvo - Plačilni nalogi - Paket plačilnih nalogov
2. Dodaj plačilne naloge v paket in shrani -> Paket je shranjen v Osnutku
3. Akcije - Izbriši

Dejansko

Plačilni nalogi, ki so bili vključeni v paket, niso odstranjeni iz paketa, po tem ko je bil izbrisan. Plačilni nalogi ne morejo biti dodani v nov paket.

Pričakovano

Ko je paket plačilnih nalogov izbrisan, so ti plačilni nalogi odstranjeni iz paketa in se jih lahko doda v nov paket.

Vir: lastno delo.

Interno oz. alfa testiranje poteka na primarnem razvojnem okolju (angl. Master staging) okolju in razvojnem okolju, ki je kandidat za namestitev na produkcijo (angl. Release staging). Ti dve okolji sta namenjeni za interno testiranje, za stranko pa je pri njih vzpostavljeno testno in produkcijsko okolje. Nova izdaja programske rešitve je najprej nameščena na testno okolje, kjer stranka izvede beta testiranje. Stranka ima na svoji strani ekipo uporabnikov, ki so prvi, ki uporabljajo aplikacijo in jo testirajo. Ko stranka potrdi verzijo na testnem okolju, se ta nato namesti na produkcijsko okolje in gre »v živo« (angl. Go-live).

Stranka ima prav tako dostop do projekta na Jiri, kamor lahko prijavlja napake oz. zahteve po novih funkcionalnostih. S tem se prične tudi potreba po podpori stranki, ki jo nudi tester. Programsko orodje Jira se v podjetju uporablja kot primarno sredstvo za komuniciranje s stranko. Tester je prva oseba, na katero se obrnejo uporabniki v primeru napak, vprašanj in glede določenih novih funkcionalnosti, ki jih želijo imeti. Enkrat na teden se tester in projektni vodja udeležujeta tudi sestanka s stranko, kjer je prisotna njihova ekipa uporabnikov, ki je zadolžena za testiranje in odgovorna oseba za koordinacijo testiranja. Na teh sestankih se pregleda statuse zahtevkov in obravnava morebitne želje oz. težave, na katere so uporabniki naleteli.

Programska rešitev temelji na lastni platformi, ki jo razvija interna ekipa za »razvoj in raziskave«, ter rešuje napake na platformi, samem jedru programske rešitve. Dodajajo ji nove funkcionalnosti in izboljšave. Ko izdajo novo verzijo platforme, jo arhitekt v implementacijski ekipi namesti na razvojno implementacijsko okolje. Takrat je potrebno

regresijsko testiranje celotne programske rešitve, ki poteka ročno. Tester z regresijskim testiranjem preveri, da sprememba verzije platforme ni vplivala na obstoječe funkcionalnosti programske rešitve. Za pomoč pri regresijskem testiranju so že vnaprej zapisani testni primeri v Jiri, ki jih je zapisal tester. Te testne primere se uporabi pri izvajanju »rutinskih« testov, kot so regresijski. Tester izvaja ročno testiranje, avtomatizirane teste pa napišejo razvijalci in jih poženejo vsakič, preden naredijo zahtevo za združitev spremembe kode na master ali na release okolje.

Na samem začetku implementacije projekta, tester sodeluje pri preverjanju pravilnosti migriranih podatkov, kot so npr. podatki o osebah na zavarovalnih policah, policah, škodah in računovodska analitika. Ti podatki so migrirani (preneseni) iz strankine prejšnje programske rešitve v novo programsko rešitev. V fazi razvoja programske rešitve, tester tudi preverja obstoječe delovanje »out of the box« funkcionalnosti na implementaciji (funkcionalnosti, ki so stranki na voljo za takojšnjo uporabo pri kupljenem produktu) in tudi išče možne anomalije od FSD specifikacije. Med drugim je naloga testerja tudi pisanje uporabniških navodil po posameznih modulih programske rešitve za stranko. V sklop testiranja prav tako spada pripravljanje različnih testnih primerov v razširljivem označevalnem jeziku (angl. Extensible Markup Language - XML) datotekah, poizvedbe po podatkovnih bazah in testiranje preko API programskega vmesnika.

3.4 Predstavitev intervjuvancev

V izbranem podjetju sem intervjuvala dve osebi, ki sta zaposleni na različnih delovnih mestih; eden kot tester in drugi kot razvijalec programskih rešitev. Zaradi lažjega sklicevanja na osebi v diplomskem delu, bo oseba, ki je zaposlena kot tester programskih rešitev, poimenovana kot oseba T, razvijalec programskih rešitev pa kot oseba R.

Oseba T je v podjetju že skoraj dve leti zaposlena kot tester programskih rešitev in strokovnjak za podporo. Je diplomiran ekonomist na študiju bančnega in finančnega managementa. Pred zaposlitvijo ni imel izkušenj z vrsto delo, ki ga zdaj opravlja v izbranem podjetju. Njegove delovne izkušnje so bile delo s CNC napravo v proizvodnji, opravljanje računovodskih del za manjša podjetja, organiziranje dogodkov in delo v gostinski industriji.

Oseba R je v podjetju zaposlena dve leti in pol. V izbranem podjetju je trenutno zaposlen na delovnem mestu razvijalec programskih rešitev in opravlja delo lastnika produkta. Lastnik produkta v Scrum metodologiji je tisti, ki zagotavlja, da gre razvoj produkta v zastavljeno smer. Njegova prva pozicija v podjetju je bila poslovni svetovalec, kjer je imel stik s stranko za analizo produkta, kmalu za tem pa je postal tudi konfigurator produktov. Pred zaposlitvijo v izbranem podjetju je imel deloma izkušnje s to vrsto dela, saj je delal na zavarovalnici, kjer je prodajal življenjska zavarovanja, kasneje pa kot finančni svetovalec. Iz tega vidika ima zelo dobro poznavanje zavarovalnih produktov in poslovne logike. Ko je prišel v izbrano podjetje, ni imel nič tehničnega znanja, vendar se je preko internih in spletnih delavnic naučil programiranja.

3.5 Rezultati raziskave

3.5.1 Intervju s testerjem programskih rešitev

Po predstavitvenem delu intervjuja sem nadaljevala z vprašanjem, kakšna znanja in veščine je oseba T pridobila v izbranem podjetju. Vprašanje je bilo zanimivo predvsem zato, ker je intervjuvanec povedal, da pred zaposlitvijo na tem delovnem mestu, ni imel nobenih izkušenj s to vrsto dela. Povedal je, da je v podjetju imel tečaj za tuji jezik, poleg tega pa ni imel nobenih uradnih izobraževanj, temveč se je vse naučil iz prakse – iz vsakodnevnega dela v podjetju. Testiranje preko uporabniškega vmesnika aplikacije je nadgradil z znanjem uporabe podatkovnih baz, pisanjem poizvedb in uporabo API programskega vmesnika za testiranje. Naučil se je razumeti implementirano kodo in sestavo konfiguracije programske rešitve v jezikovnem urejevalniku za kodiranje, pri čemer so mu tudi pomagali sodelavci razvijalci. Vse se je naučil znotraj podjetja.

Na vprašanje o tipičnih delovnih nalogah in odgovornostih, je intervjuvanec povedal, da je njegova osnovna naloga testiranje. Tester prejme zahtevek v Jiri, ki je pripravljen na testiranje in izvede testiranje. Če je test uspešen, se zahtevek zapre, sicer pa se ponovno odpre in se dodeli nazaj razvijalcu v implementacijo popravka. Naslednja izmed nalog je pomoč končnim uporabnikom (stranki) pri testiranju ali uporabi programske rešitve. Spremljati mora tudi, če uporabniki prijavijo napake ali nove zahtevke v Jiri, jih pregledati in dodeliti pravemu razvijalcu ali pa na projektno vodjo, če ne ve komu bi zahtevek dodelil. Na vrsto pridejo tudi analize zahtev stranke oz. končnih uporabnikov. Meni da je pri tem največja razlika med izkušenimi in neizkušenimi testerji, saj se naučiš kako se nekaj lahko naredi tehnično in kako se zahteva razloži razvijalcu. Ko je projekt tik pred produkcijo, kar pomeni da je produkt (programska rešitev) tik pred samo dostavo stranki, je zaradi tega veliko vsakodnevnih sestankov in koordinacije. Sicer pa, ko je programska rešitev v razvoju, tega ni tako veliko.

Pogovor sem nato usmerila na vprašanje, kakšne značilnosti mora po njegovem mnenju imeti tester programskih rešitev. Intervjuvanec je dejal, da ni nujno, da ima tester na začetku vse značilnosti, ki bi se pričakovale od testerja programskih rešitev. Vseeno pa mora imeti smisel za analitično razmišljanje, željo po učenju in proaktivnost. Na začetku tester ni vpleten v podporo stranki. Šele ko spozna osnove projekta in dobro pozna programsko rešitev, je lahko vključen v podporo stranki, sicer jim ne more veliko pomagati. Komunikacijske značilnosti pridobi z izkušnjami, ko je na primer soudeleženec pri sestankih s stranko in z opazovanjem, kako projektni vodja komunicira z njimi. Mora si znati tudi sam organizirati dan; npr. če pride pri testiranju do ponavljajočih se zadev, si skuša prilagoditi proces poteka dela. Primer je izvajanje regresijskih testov, ki se ponavljajo. Ko izvaja regresijske teste, si tako razdeli testiranje čez cel dan in vmes testira zahtevke, pripravljene za testiranje ali pa piše scenarije za regresijske teste.

Intervjuvanca sem povprašala, kaj je po njegovem mnenju potrebno izboljšave kar se zadeva njegovega vsakodnevnega opravljanja dela. Izpostavil je preobremenjenost testerjev. Trenutno dela kot tester na večjem projektu, ki je bil v času intervjuja na prehodu v produkcijo. To pomeni veliko sestankov, koordinacije in pritiska na testerja, zaradi istočasne podpore stranki in testiranja zahtevkov. V tej situaciji pri svojem delu ni vedel, česa naj se loti prvo - ali naj prijavi nov zahtevek, ima klic s stranko, ali ima sestanek s projektno vodjo, da bi se z njim organiziral. Za nastalo situacijo je predlagal dve rešitvi. Prva rešitev bi bila, da bi imeli na projektu zaposleno eno osebo, ki bi bila zadolžena samo za podporo stranki, koordinacijo in spremljanje zahtevkov na Jiri. Na ta način bi lahko tester svoj čas namenil dejanskemu testiranju in koordinaciji zahtevkov, ki so potrebni testiranja. Kot drugo rešitev za razbremenitev testerjev pa je predlagal, da bi imeli testiranje razdeljeno po modulih programske rešitve. Vsak modul (npr. računovodstvo, škode, osebe, …) bi tako imel specialista, ki bi bil na tekočem z vsemi zahtevki na posameznem modulu in bi bil odgovoren za podporo stranki. Po eni strani pa to ne bi bilo efektivno, če bi šel tester iz enega projekta na drug projekt, saj tam ne bi vedel nič o drugih modulih, zato bi morali o temu še razmisliti.

Nato sem intervju usmerila na izzive, s katerimi se srečujejo testerji programskih rešitev. Prvi izziv, katerega je omenil, je opisovanje prijavljenih napak oz. novih zahtev na Jiri. Težavo vidi v tem, da vsi deležniki ne upoštevajo dogovorjene strukture opisa, kako se mora napaka ali nova zahteva opisati. To se največkrat zgodi, ko zahtevek v Jiri prijavi razvijalec, saj ga odpre in opiše na način, da ga razume samo on in ne tisti, ki ga bo testiral. To pomeni, da se mora tester sam znajti in zapravljati čas z raziskovanjem kakšna sprememba je bila sploh narejena, kaj je potrebno potestirati in kakšno je pričakovano obnašanje. Naslednji izziv je pomanjkanje dokumentacije in izobraževanja o novih funkcionalnostih. Ko je v programsko rešitev implementirana nova funkcionalnost, ki je potrebna testiranja, se mora tester znajti sam. Dodobra mora preučiti kakšno je pričakovano delovanje nove funkcionalnosti. To naredi tako, da preveri, če je nova funkcionalnost popisana v FSD-ju, sicer pa mora govoriti z osebami, ki to vedo npr. interni poslovni analitiki. Testerji bi potrebovali kratko izobraževanje o novih funkcionalnostih, saj bi to močno pospešilo testiranje, ki bi bilo tudi bolj učinkovito in pravilno.

Intervjuvanec je kot izziv omenil tudi posodabljanje verzij platforme, ki se dogaja zelo pogosto. To za implementacijsko ekipo na nekem projektu pomeni, da je potrebno namestiti novo verzijo platforme, ki pa ni povsem konsistentna z implementacijo in tako lahko prihaja do napak v programski rešitvi zaradi same posodobitve. Za testerje le-to pomeni, da morajo večkrat izvesti popolno regresijsko testiranje programske rešitve, kar pa je zamudno. Testerjem bi bilo potrebno razložiti, kaj se pridobi z novo platformo, kje se lahko to vidi in kaj to pomeni za obstoječe funkcionalnosti. Za samega testerja je na samem začetku veliko nerazumevanja, na sestankih večinoma nič ne razume, že zaradi terminologije - to velja predvsem za testerje, ki nimajo prejšnjih izkušenj z delom v IT-podjetju. Udeležba in poslušanje na dnevnih sestankih je zato zelo pomembna, saj tako lahko zastavi kakšno

vprašanje in pridobi ogromno informacij, kasneje pa se ne izgublja časa z ugotavljanjem, ko pride do testiranja.

Prav tako pa obstajajo izzivi v povezavi s strankami. »*Največji izziv je, kako povedati stranki, da nekaj še ni pripravljeno za testiranje, ter na kakšen način jim to povedati, da ni toliko očitno. Prav tako je potrebno stranki razložiti kaj je v novi verziji, kako jo dobijo, kdaj lahko testirajo zahtevke, zakaj je zahtevek že rešen v Jiri, oni pa ga ne morejo potestirati*«. Zaradi tega se je tudi na projektu, kjer dela, naredila reorganizacija statusov zahtevkov v Jiri, da so uporabniki lahko lažje razumeli, kdaj je zahtevek dejansko pripravljen za njihovo testiranje. Izziv, ki ga je izpostavil, je tudi to, da je potrebno stranki razložiti, da se nekatere zadeve zaradi tehničnega razloga ne morejo implementirati na takšen način, kot so si zamislili. V tem primeru za pomoč prosi razvijalce, ki stranki objasnijo, zakaj to ni mogoče in kako se lahko funkcionalnost implementira na drugačen način. Če gre za poslovno logiko, se s stranko pogovori o zahtevah in utemelji zakaj nekaj ni smiselno narediti ter svetuje kakšna bi bila lahko optimalna rešitev. Prav tako prihaja do situacij, ko stranka odpre zahtevek, ki je sprememba že implementirane funkcionalnosti. V tem primeru ga mora odobriti strankin upravni odbor projekta (t.i. sponzor projekta). S tem se stranko omeji, da njihovi uporabniki ne odpirajo novih zahtev za spremembo delovanja funkcionalnosti in tudi mi, kot podjetje prihranimo čas za bolj pomembne stvari. »*Velikokrat se pa tudi zgodi, da kljub temu da stranka potestira neko funkcionalnost in jo potrdi, potem pa šele na realnih primeri vidi, kaj zares rabi in je takrat potrebno hiteti s popravki*«.

Izzivi z razvijalci prihajajo predvsem iz tega razloga, ker imajo oni perspektivo iz tehničnega vidika, tester pa iz poslovnega vidika. Tukaj lahko pride do konflikta razumevanja med obema stranema, ker drug drugemu ne znata razložiti, zakaj nekaj ne deluje oziroma na drugi strani, zakaj se le-to ne da implementirati in kako bi moralo delovati.

Na vprašanje, za koga je primerno delo testerja programskih rešitev, je intervjuvanec dejal, da je čisto odvisno od ambicije človeka. Sam je prišel iz gostinstva in dobro dela. Lahko pa pride nekdo iz podobne smeri, vendar pa mu bo delo dolgočasno in zato ne bo dober. Potrebno je imeti zanimanje, ni pa potrebno imeti nekih izkušenj. Če pa jih imaš, je to boljše zate in za podjetje. Posameznik, ki se prijavlja na pozicijo testerja programskih rešitev, mora biti pripravljen na vse zgoraj naštete izzive, hkrati pa se mora zavedati, da bo občasno delal izven svojih delovnih ur. Imeti mora željo po učenju in to početi samoiniciativno. Priporočljiva znanja in izkušnje so čisto odvisne od podjetja, za katerega se prijavlja. Dobro je imeti znanje iz področja, s katerim se podjetje ukvarja (npr. têrmini iz zavarovalništva, če se podjetje s tem ukvarja), hkrati pa moraš biti komunikativen, samostojen in iznajdljiv.

3.5.2 Intervju z razvijalcem programskih rešitev

Intervjuju s testerjem je sledil intervju z razvijalcem. Tipične vsakodnevne delovne in odgovornosti osebe R so koordinacija tima, komunikacija s stranko, reševanje odprtih vprašanj glede konfiguracije, koordinacija in spodbujanje razvijalcev, da naredijo svoje

zahtevke v sprintu, planiranje in delanje analiz za projekt ter ocenjevanje časa za izvedbo zahtevkov. S testerji komunicira vsak dan, saj mu na primer posredujejo povratne informacije za zahtevke, ki so jih testirali in potrdili, vendar pa pri stranki ne delajo, kot bi morali. Takrat se ugotavlja, ali je bila težava na internem površnem testiranju, ali je razvijalec že na začetku nekaj narobe naredil.

Oseba R je dejala, da po njegovem mnenju testerji testirajo preveč stvari, tudi tisto kar ni treba. Imajo tudi premalo poslovnega znanja in logike iz zavarovalništva. Tester mora razumeti kako stranka uporablja sistem pri vsakodnevnem delu. Vsak tester, ki pride v podjetje, bi morali imeti neko izobraževanje, s katerim bi to znanje lahko pridobil. *»Od testerja je odvisen razvoj programske rešitve za naprej, zagotavljanje kakovosti in ugled do stranke, zaradi slabega poznavanja poslovne logike pa so testerji pri testiranju lahko površni«.* Prav tako bi si testerji morali boljše organizirati čas pri testiranju, da v primeru vprašanj najprej komunicirajo z razvijalcem, ki je na zahtevku delal. Prav tako morajo biti testerji samostojni, da v Jiri sami poiščejo zahtevke, ki so pripravljeni na testiranje, ne da jih mora nekdo potiskati naprej, da vedo kaj testirati. Izziv so tudi regresijski testi, ker je potrebno testirati ogromno funkcionalnosti, kar pa testerjem vzame veliko časa, ki potem nimajo časa za redno testiranje zahtevkov, sploh če na projektu primanjkuje testerjev ali pa je en sam. Testerji morajo znati povezati vse module programske rešitve. Dobro bi bilo, da bi imeli testerje po modulih programske rešitve in da tudi ne bi bili omejeni na projekt, ampak bi se lahko med njimi premikali. Tester bi bil zadolžen za svoj modul in bi sodeloval z drugimi testerji, na podoben način kot testira stranka, kjer testirajo po področjih, npr. sektor premoženjskih zavarovanj testira svoj del, v sodelovanju z računovodstvom ali škodami. V tem primeru bi potrebovali tudi koordinatorja testerjev, ki bi zadeve koordiniral z vodjami modulov.

Testerji bi se morali tudi vključevati v vse sestanke statusov s stranko, saj testerji najbolj vedo, kakšen je status testiranja ali status posameznih zahtevkov. Tester bi bil zadolžen za koordinacijo, kdaj naj se določeni zahtevki dajo k stranki in bi delal zapiske na sestankih. Naslednji izziv, ki se pojavlja, so opisi prijavljenih zahtevkov v Jiri. Pri odpiranju zahtevkov se moramo držati zastavljene strukture pri opisovanju napake ali zahteve. Na ta način bi tudi spodbudili stranko, da se ob odpiranju novega zahtevka drži strukture opisa zahtevka, sicer bi jim tak zahtevek zavrnili, dokler opis nima dogovorjene strukture. S tem bi olajšali delo razvijalcem, ki bi točno vedeli kaj morajo narediti in kaj se od njih pričakuje. Olajšalo bi tudi delo lastnikom produktov in vodjem ekip modulov, ki bi lažje ocenili potreben čas za izvedbo zahtevka, tester pa bi na lažji način povedal stranki, kdaj bo zahtevek narejen. Zelo zaželeno je, da imajo testerji znanje SQL-a, pa to ne pomeni da mora sam znati pisati poizvedbe, ampak da zna v analitičnih tabelah podatkovne baze sam preveriti, da se na primer, vsi podatki izpišejo.

Intervjuvanec je na vprašanje, kako bi lahko skupaj izboljšali celoten potek razvoja programske rešitve, dejal, da je glavna točka komunikacija in znanje logike, za katero je programska rešitev namenjena. V našem primeru je to logika zavarovalništva. Tako se lahko

lažje določi smernice zavarovalnicam in jim tudi predlaga, da gredo v neko smer o kateri prej sploh niso razmišljali. Testerji morajo imeti občutek, do katere točke je ekipa tehnično sposobna neko zahtevo od stranke implementirati in stranki sporočiti, kdaj približno lahko neki zahtevek dobi. Ravno tako morajo vedeti, če je napaka, ki jo je prijavila stranka, res napaka v programski rešitvi. Če je, tester s stranko uskladi, koliko kritična je napaka, da se potem naprej sporoči razvijalcem in vodjem, kdaj mora biti napaka rešena. Ko ima tester enkrat poslovno znanje, je on tisti, ki definira testne scenarije, po primerih, ki jih priskrbi stranka. Ko se interno nadgradi platforma programske rešitve, mora tester znati razbrati spremembe, ki so bile narejene z novo verzijo platforme in napraviti regresijske teste. Problem je, ker imamo preveliko število novih verzij platforme. Nova verzija platforme je nameščena, največkrat preden testerji uspejo končati z regresijskimi testi prejšnje verzije. Prav tako posodabljanje platforme vzame veliko časa arhitektom in tudi vsem, ki delajo s konfiguracijo, saj je potrebna posodobitev lokalnega okolja.

Po mnenju intervjuvanca, je delo testerja programskih rešitev primerno za osebe, ki so po izobrazbi ekonomisti ali podobnih smeri, saj dobijo znanje komunikacije, osnove programiranja, znanje kako trg funkcionira, marketing, o zavarovalništvu, bančništvu, ... »*V podjetje je boljše vzeti nekoga, ki ima poslovno znanje, kot pa nekoga, ki ima tehnično znanje, saj se tehnično znanje lažje naučiš*«.

4 TEMELJNE UGOTOVITVE IN PREDLOGI IZBOLJŠAV

Po opravljenih intervjujih s testerjem in razvijalcem v izbranem podjetju, sem ugotovila, da sta njuni mnenji precej usklajeni glede izzivov, s katerimi se soočajo testerji. Oba sta izpostavila, da je izziv opisovanje zahtevkov v Jiri, saj se vsi deležniki ne držijo dogovorjene strukture opisa zahtevkov, kar oteži delo razvijalcem, ki zahtevek rešujejo in testerjem, ki ga kasneje testirajo. Ta izziv bi lahko rešili tako, da bi bila ena oseba zadolžena za to, da spremlja, če so opisi vseh odprtih zahtevkov v takšni strukturi, kot je bila dogovorjena. Če niso, bi se tak zahtevek zavrnilo, dokler ne bi bil opis v pravilni strukturi. Po drugi strani bi pa lahko zadevo rešili tako, da bi moral uporabnik že pri prijavljanju novega zahtevka obvezno vnesti opis po dogovorjeni strukturi, sicer zahtevka ne bi sploh mogel odpreti.

Razvijalec in tester sta se nekako strinjala tudi glede pomanjkanje poslovnega znanja pri testerjih. Tester je posebej izpostavil pomanjkanje dokumentacije, in da je potrebno samoiniciativno raziskati, kako mora neka nova funkcionalnost v programski rešitvi delovati. Podoben izziv je v raziskavi ugotovil tudi Kumari (2018), ki je dejal, da je pomanjkanje dokumentacije, potrebne za testiranje, eno izmed najpogostejših izzivov testerjev. Zhang & Hunkapiller (2017) pa sta dejala, da lahko testerji kljub dokumentaciji, zaradi pomanjkanja znanja o programski rešitvi, odpirajo neveljavne napake, ali pa ne razumejo zahtev. Potrebno bi bilo organizirati interna izobraževanja o novih funkcionalnostih in dati možnost spletnih izobraževanj preko zunanjih izvajalcev, kjer bi testerji pridobili ustrezno znanje, prav tako pa bi to znanje prišlo prav razvijalcem. Na

implementacijskih projektih bi zelo pomagalo vključevanje testerjev na sestanke s stranko od samega začetka implementacije. S tem bi tester dobil znanje, kako poslovni proces pri stranki poteka, kar bi mu kasneje pomagalo pri testiranju, saj bi razumel kako stranka razmišlja in uporablja programsko rešitev. Izbrano podjetje zaposlenim ponuja dostop do izobraževalnih vsebin, prav tako do posnetkov internih izobraževanj, ki pa jih je premalo in se ne nanašajo na določene funkcionalnosti programske rešitve.

Tako prvi kot drugi intervjuvanec, sta izpostavila izziv zaradi pogostega posodabljanja verzij platforme, ki testerjem vzame ogromno časa za izvedbo regresijskih testov programske rešitve, saj morajo preveriti, če zaradi nove verzije platforme prihaja do napak v implementaciji. Do napak zelo pogosto prihaja pri verzijah z več spremembami, kar je izpostavil tudi Kumar (2019). Posodabljanje platforme vzame veliko časa tudi vsem ostalim deležnikom v implementacijski ekipi. Potrebno bi bilo imeti manj verzij platforme z večjim številom sprememb, ki bi morale biti jasno opisane. Tako bi testerji na implementaciji vedeli, kje lahko potencialno prihaja do napak v implementaciji, hkrati pa bi manj časa namenili regresijskim testom, saj bi bile nove verzije platform manj pogoste. Razvijalec je izpostavil, da testerji testirajo preveč stvari, tudi tisto kar ni treba in je izven zahtevka, ki se ga testira. Biti morajo samostojni in sami poiskati zahtevke, ki jih je potrebno potestirati ter v primeru vprašanj glede določenega zahtevka, najprej govoriti z razvijalcem, ki je zahtevek reševal, in ne z vodjo ekipe posameznega modula. Če pride do takšne situacije, bi bilo potrebno testerje spomniti, na koga naj se najprej obrnejo pri testiranju in da naj se osredotočijo na testiranje zahtevka. V primeru da pri zahtevku najdejo dodatne napake, ki so izven obsega zahtevka, se za to odpre nov zahtevek v Jiri. Samostojnosti testerjev pri iskanju zahtevkov, ki jih je potrebno potestirati, bi se omogočilo s tem, da bi se določilo prioritete na posameznih zahtevkih. Tako bi tester takoj prepoznal, kateri zahtevek je potrebno potestirati najprej.

Preobremenjenost testerjev je velika še posebej v času, ko gre projekt »v živo« oz. v produkcijo. Na testerja pritiska stranka, ki potrebuje podporo pri testiranju, hkrati pa zahtevki stojijo, če niso testirani pravočasno. V tem času je pogosto, da morajo testerji delati tudi izven delovnih ur. Kot je predlagal že intervjuvanec T, bi lahko to rešili tako, da bi imeli na projektu dve osebi, ki bili zaposleni na področju zagotavljanja kakovosti. Ena bi bila zadolžena samo za podporo stranki, koordinacijo in spremljanje zahtevkov v Jiri, druga pa za testiranje zahtevkov. Lahko bi da bi imeli testerje razdeljene po modulih programske rešitve in ne bi bili omejeni na projekt, ampak bi se lahko med njimi premikali. Tester bi bil zadolžen za svoj modul in bi sodeloval z drugimi testerji, na podoben način kot testira stranka, kjer testirajo po področjih.

Intervjuvanec T je poudaril tudi izzive, ki nastanejo, ko je tester vključen v podporo stranki. Stranka navadno ni podkovana v tehničnem znanju, zato je izziv razložiti stranki, zakaj neke funkcionalnosti ne morejo imeti točno takšne, kot so si jo zamislili, kdaj lahko določeni zahtevek testirajo in kako sporočiti stranki, da nekaj še ni pripravljeno za testiranje. Navedene izzive se rešuje z izkušnjami, s katerimi tester pridobi t.i. mehke meščine oziroma znanje kako govoriti s stranko, da ji jasno in razumljivo razloži ter odgovori na vprašanja.

Pri razlagi, zakaj stranka nečesa ne more dobiti, kot si je zastavila, lahko pomaga razvijalec, ki pove iz tehnične plati, tester pa to prevaja v stranki bolj razumljiv jezik.

Za testerja, ki se na novo zaposli, je na začetku veliko nerazumevanja, sploh če oseba prej ni delala v IT-podjetju in ne pozna specifične terminologije. Izkušnje pridobi skozi čas in z aktivnim poslušanjem sodelavcev na dnevnih sestankih in seveda z izkazano radovednostjo po učenju. Kot smo izvedeli iz intervjuja, ni nujno, da ima tester na začetku vse značilnosti, ki bi se pričakovale od testerja programskih rešitev. Intervjuvanec, ki je sedaj zaposlen kot tester, pred zaposlitvijo na tem delovnem mestu ni imel nobenih izkušenj s to vrsto dela. Vse se je naučil iz vsakodnevnega dela v podjetju in z željo po učenju. Vsekakor je dobro, da pride tester z znanjem SQL in uporabe podatkovnih baz, da ima smisel za analitično razmišljanje, željo po učenju, samoiniciativnost in je timsko naravnan. Ne smemo izpustiti znanja angleščine, ki je na tem področju nepogrešljiva, sploh pri delu v mednarodnih podjetjih, kjer se dnevno komunicira s sodelavci in s strankami v tujem jeziku.

Iz intervjuja smo spoznali, da je velik izziv testerjev pomanjkanje poslovnega znanja iz področja, s katerim se v IT-podjetju ukvarjajo. S primernim znanjem, bi tester hitreje osvojil in razumel programsko rešitev, jo pravilno testiral in pripomogel k dodani vrednosti, ki jo programska rešitev stranki prinaša. Testerji so tisti, ki s kakovostnim testiranjem poskrbijo, da programska rešitev deluje tako, kot je bila zasnovana in prvotno predstavljena kupcu.

V tabeli 1 so predstavljeni zaznani ključni izzivi, s katerimi se soočata razvijalec in tester tekom opravljanja svojega vsakodnevnega dela v izbranem podjetju. Z znakom x je v tabeli označeno, kateri izmed intervjuvancev je problematiko izpostavil. V večina primerih sta jo izpostavila oba. V zadnjem stolpcu je po lestvici pomembnosti (1 – zelo nepomembno, 2 – nepomembno, 3 - niti pomembno, niti nepomembno, 4 – pomembno, 5 – zelo pomembno) ocenjena pomembnost navedenih izzivov, ocenjeno na podlagi zaznavanja, katere izzive sta intervjuvanca najbolj izpostavila tekom intervjuja in o katerih je bilo največ razglabljanja.

Tabela 1: Opredeljeni izzivi pri testiranju programskih rešitev

Opredeljen izziv	**Razvijalec**	**Tester**	**Pomembnost izziva**
Preobremenjenost. Ena oseba zadolžena za vse – testiranje, podporo stranki, koordinacijo in spremljanje zahtevkov na Jiri	x	x	5
Neustrezno dokumentiranje prijavljenih napak oz. novih zahtev na Jiri, ki ne sledi standardiziranim postopkom	x	x	4
Pogosto posodabljanje verzij platforme vzame veliko časa za testiranje, razvijalcem pa za posodobitev lokalnega okolja na zadnjo verzijo platforme	x	x	4

Pomanjkanje poslovnega znanja in prijava neveljavnih napak	x	x	5
Pomanjkanje dokumentacije potrebne za testiranje po specifikaciji		x	5
Ogromno časa porabljenega za izvedbo regresijskih testov	x	x	4
Testira se preveč, in tisto kar ni potrebno	x		2
Organizacija časa in dela pri testiranju	x		2
Neuspešna komunikacija med razvijalcem in testerjem		x	2
Komunikacija s stranko		x	3
Delo izven delovnih ur		x	3

Vir: lastno delo.

Po zaključku intervjuja lahko trdim, da se navedbe glede izzivov testerjev iz strani testerja in razvijalca ne razlikujejo v veliki meri. Oba se strinjata, da bi bila potrebna razbremenitev testerjev, na način, da bi imeli eno osebo, ki je zadolžena samo za testiranje, druga pa za podporo stranki, ali pa da bi imeli testerje, ki bi bili zadolženi za svoj modul znotraj programske rešitve in bi sodelovali med seboj – tako kot to počne stranka, ko testira in uporablja programsko rešitev v praksi. To bi tudi zmanjšalo potrebo po delu izven delovnih ur. Strinjata se tudi glede prijavljanja novih zahtevkov v Jiri, ki niso opisani v dogovorjeni strukturi, kar upočasnjuje proces obdelave zahtevka za razvijalce in testerje. Enako velja za pogosto posodabljanje verzij platforme programske rešitve in čas, ki se porabi za izvajanje pogostih regresijskih testov. Velik poudarek iz obeh strani je bilo tudi pomanjkanje poslovnega znanja pri testerjih, ki dodatno pride do izraza zaradi pomanjkanja dokumentacije in izobraževanj o funkcionalnostih, ki se na primer na novo razvijejo v programski rešitvi. Kdaj pride tudi do situacije, ko je komunikacija med razvijalcem in testerjem neuspešna, prav tako je na začetku potrebno razumeti, kako uspešno komunicirati s stranko. Med drugimi pa je pomembno, da testerji pri testiranju najprej komunicirajo s ključnimi osebami, so samostojni in si znajo sami organizirati čas ter delo.

SKLEP

Cilj zaključnega dela je bil predstaviti poklic testerja programskih rešitev, jasno pokazati ugotovljene izzive na področju testiranja programskih rešitev in za njih podati konkretne rešitve in predloge za izboljšavo. Cilje sem dosegla s pomočjo lastnih izkušenj s to vrsto dela, intervjujev z osebama, zaposlenima na poziciji tester programskih rešitev in razvijalec programskih rešitev, ter s pomočjo teoretičnih izhodišč. Za ugotovljene izzive sem podala predloge izboljšav, kar je lahko za podjetja, ki se ukvarjajo z razvojem programskih rešitev, izhodišče za izboljšavo njihovega procesa testiranja programskih rešitev.

V empiričnem delu diplomskega dela je bil iz lastnih izkušenj predstavljen proces testiranja programske rešitve v izbranem podjetju. Predstavljene so bile tipične naloge testerja, potek testiranja in na katerih okoljih se le-to izvaja, prijavljanje napak v Jiri, kako in kdaj se izdajajo nove verzije programske rešitve stranki, kakšno vlogo ima tester pri podpori stranki in drugo sodelovanje znotraj razvojne ekipe. Tako so bralci dobili razumevanje in vpogled v delo testerja programskih rešitev.

Na podlagi intervjujev, ki sta bila izvedena v izbranem podjetju, ki razvija lastno programsko rešitev za zavarovalnice, smo izvedeli, da mora imeti tester programskih rešitev predvsem komunikacijske sposobnosti, smisel za analitično razmišljanje, željo po učenju in biti proaktiven. Priporočljivo je, da ima tester poslovno znanje; kot je bilo povedano v intervjuju, je boljše vzeti nekoga, ki ima poslovno znanje, ko pa nekoga, ki ima tehnično znanje, saj se le-to lažje nauči. Iz naloge izvemo tudi, da za poklic testerja ni potrebno imeti posebnih izkušenj s to vrsto dela in saj se lahko vse znanje pridobi iz vsakodnevnega dela v podjetju – seveda pa je to odvisno od vsakega podjetja posamezno. Vsekakor pa je nujno, da ima oseba željo in zanimanje za delo v IT-ju.

Ugotovljeni izzivi testerjev programskih rešitev izhajajo predvsem iz organizacije časa in dela, procesov dokumentiranja ter tudi ljudi, ker imajo pomanjkanje potrebnega znanja. Raziskava obravnavane teme izzivov ima tudi omejitve, saj sta bila za potrebe diplomskega dela, intervjuvana posamezna predstavnika udeležencev v razvoju programskih rešitev – testerjev in razvijalcev, znotraj implementacijskega projekta v izbranem podjetju. Če bi v intervju vključila več udeležencev iz različnih projektov in iz različnih podjetij, bi se rezultati raziskave zagotovo razlikovali, hkrati pa bi pridobili vpogled v več različnih izzivov, s katerimi se testerji programskih rešitev na splošno soočajo.

S tem zaključnim delom sem pokazala, kako je v današnjem svetu v resnici pomembna vloga testerja programskih rešitev. Po prebrani nalogi so posamezniki, ki jih zanima delo v IT-ju, dobro seznanjeni s tem poklicem, podjetja, ki se ukvarjajo z razvojem programskih rešitev pa so dobila izhodišča za možno izboljšavo procesa testiranja njihovih programskih rešitev.

LITERATURA IN VIRI

1. Alegroth, E., Feldt, R. & H. Olsson, H. (2013). *Transitioning Manual System Test Suites to Automated Testing: An Industrial Case Study.* Pridobljeno 21. julija 2022 iz https://www.researchgate.net/publication/261487145_Transitioning_Manual_System_Test_Suites_to_Automated_Testing_An_Industrial_Case_Study
2. Atlassian. (2022b). *A brief overview of Jira*. Pridobljeno 4. februarja 2022 iz https://www.atlassian.com/software/jira/guides/getting-started/overview
3. Atlassian. (2022a). *Software development*. Pridobljeno 20. februarja 2022 iz https://www.atlassian.com/software-development

4. Bauman, M. (2011, 3. november). *Razvoj Intrixa po agilni metodologiji Scrum* [objava na blogu]. Pridobljeno 5. oktobra 2021 iz https://www.intrix.si/blog/projektno-vodenje/razvoj-intrix-a-po-agilni-metodoligiji-scrum
5. Campbell, A. (2020). *Agile: Essentials of Team and Project Management. Manifesto for Agile Software Development.*
6. Dooley, J. (2017). *Software Development, Design and Coding.* Galesburg, Illinois, USA: Apress.
7. Drumond, C. (2020). *What is Scrum?*. Pridobljeno 16. januarja 2021 iz https://www.atlassian.com/agile/scrum
8. Glenford, J. M. (2012). *The art of software testing.* Hoboken, New Jersey: JohnWiley & Sons, Inc.
9. Graham, D. & Mark, F. (2012). *Experiences of test automation: case studies of software test automation.* Crawfordsville: Addison-Wesley.
10. Graham, D., van Veenendaal, E., Evans, I. & Black, R. (2008). *Foundations of Software Testing.* Cengage Learning Emea.
11. Guntupalli, R. C. (2008). *User Interface Design – Methods and Qualities of a Good User Interface Design* (magistrsko delo). Pridobljeno 17. oktobra 2021 iz https://docplayer.net/21787339-User-interface-design-methods-and-qualities-of-a-good-user-interface-design.html
12. Guru99. (2021). *Software Testing Tutorial*. Pridobljeno 10. januarja 2021 iz https://www.guru99.com/software-testing.html
13. IBM. (2020). *What is software development?*. Pridobljeno 20. december 2021 iz https://www.ibm.com/topics/software-development
14. Idalko. (2022). *The Intuitive Jira Guide For Users*. Pridobljeno 2. avgusta 2022 iz https://www.idalko.com/jira-guide/
15. Izbrano podjetje. (2021). *Interni vir izbranega podjetja.* Ljubljana.
16. JavaPoint. (2021). *Manual Testing*. Pridobljeno 10. marca 2021 iz https://www.javatpoint.com/manual-testing
17. Jordan University of Science & Technology. (2021). *Introduction to Computers*. Pridobljeno 12. marca 2021 iz https://www.just.edu.jo/~mqais/CIS99/PDF/Ch.01_Introduction_%20to_computers.pdf
18. Kumar G., A. (2019). A review on challenges in Software testing. *Journal of Information and Computational Science, 9*(6), 166-174.
19. Kumar, V. (8. oktober 2020). *A complete guide on software development technologies* [objava na blogu]. Pridobljeno 26. marca 2021 iz https://blogs.sap.com/2020/10/08/a-complete-guide-on-software-development-methodologies-2020/#Rapid
20. Kumari, J. (2018, 18. Julij). *A day in the life of a software tester* [objava na blogu]. Pridobljeno 29. julija 2022 iz https://medium.com/@jaiyantikumari.13/a-day-in-the-life-of-a-software-tester-7bb947894b08
21. Muniraja, S. B., Jawahar, B. K. & Ismail, B. S. (2015). IT Application in Business. *International Journal of Engineering Research & Technology, 3*(18), 1-2.

22. Naik, K. & Tripathy, P. (2011). *Software Testing and Quality Assurance: Theory and Practice.* Hoboken, New Jersey: John Wiley & Sons, Inc.
23. Nayyar, I. & Rizwan, J. Q. (2009). Improvement of Key Problems of Software Testing in Quality Assurance. *Science International (Lahore), 21*(1), 25-28.
24. O'Broin, C. (2015). *The Impact of Test Automation on Software Testers.* Dublin.
25. Onix-Systems. (2020, 2. marec). *What Does a Software Test Engineer Do During Software Development?*. Pridobljeno 19. marca 2021 iz https://onix-systems.medium.com/what-does-a-software-test-engineer-do-during-software-development-a870e6adb495
26. Singh, Y. (2012). *Software testing.* New York: Cambridge University Press.
27. Stellman, A. & Greene, J. (2014). *Learning Agile: Understanding Scrum, XP, Lean, and Kanban* (1. izd.). USA: O'Reilly Media.
28. Stober, T. & Hansmann, U. (2010). *Agile Software Development, Best Practices for Large Software Development Projects.* Böblingen: Springer Heidelberg.
29. Whittaker, J. A. (2000). What Is Software Testing? And Why Is It So Hard? *IEEE SOFTWARE,* 70-79.
30. Wrike, Inc. (2022). *What Are the Disadvantages of Agile?* Pridobljeno 24. julija 2022 iz https://www.wrike.com/agile-guide/faq/disadvantages-of-agile/
31. Zhang, X. & Hunkapiller, J. T. (2017). Sources of Conflict between Software Developers and Testers. *Issues in Information Systems, 18*(3), 53-61.