VILNIAUS UNIVERSITETAS

EKONOMIKOS IR VERSLO ADMINISTRAVIMO FAKULTETAS

Strateginio informacinių sistemų valdymo

magistro programa

2 kurso studentė

Rugilė Žiukienė

MAGISTRO BAIGIAMASIS DARBAS

TESTAVIMO PROCESAS ŠIANDIENINĖJE PROGRAMŲ KŪRIMO APLINKOJE	**TESTING PROCESS IN MODERN SOFTWARE DEVELOPMENT**

Darbo vadovas: lekt. A. Gavrilov

Vilnius, 2022

TURINYS

LENTELIŲ SĄRAŠAS

PAVEIKSLŲ SĄRAŠAS

SANTRUMPŲ SĄRAŠAS

Terminas	**Paaiškinimas**
Ad-hoc	Veiklos atlikimo būdas, kai viskas daroma be struktūros, planavimo, nesilaikant gerųjų praktikų
Agile	Metodologija, grindžiama iteraciniu modeliu, didelį dėmesį teikianti komunikacijai, turinti savo principus bei metodus
CD	Nuolatinis diegimas (angl. *continues development*)
CI	Nuolatinis integravimas (angl. *continues integration*)
CI/CD	Bendras išsireiškimas, apimantis nuolatines veiklas (integravimą bei diegimą)
DevOps	Mąstymo būdas, kuris skatina CI/CD veiklas (angl. *Development & Operations*)
IT	Informacinės technologijos
PĮ	Programinė įranga
TMA	Testavimo brandos įsivertinimas (angl. *test maturity assesment*)
TPI	Testavimo proceso pagerinimas (angl. *test process improvement*)
TMA/TPI	Bendras išsireiškimas, apimantis testavimo įsivertinimo bei pagerinimo veiklas

ĮVADAS

2013m. „Amazon“ prarado 4.8 milijonų dolerių dėl atsiradusio gedimo, kuris buvo sutaisytas per 40min. Tai yra apytiksliai 120.000 dolerių per minutę. Įmonės vadovai pasimokė iš savo klaidos ir testavimo procesą padarė prioritetiniu. Šiandien, atsitikus panašiai problemai, suma būtų keliolika kartų didesnė, nes informacinių technologijų vaidmuo žmonių gyvenime nuolat auga. (Krasner, 2018)

Programinės įrangos testavimas – vienas iš patikimiausių procesų, norint užtikrinti gerą produkto kokybę. Tačiau jis nėra lengvai vykdomas – reikalauja didelių laiko bei pinigų sąnaudų, daug žinių bei patirties. Tai apima planavimą, pasiruošimą, atlikimą, analizę ir daug kitų veiklų. Šias veiklas suvaldyti padeda *agile* metodologija, kuri teigia, jog testavimas turi būti pradėtas ne tada, kai produktas suprogramuotas, o kartu su viso projekto pradžia. (Anand & Dinakaran, 2016) Pagal ISTQB 2018m. atliktą apklausą, 79% apklaustųjų teigia, kad testuojant naudojasi *agile* metodologija. Lyginant su 2016m., matomas 10% augimas. Tai rodo vis didėjantį *agile* populiarumą. (ISTQB, 2018)

Augant IT produktų paklausai, atsiranda poreikis kurti itin greitai, bet su aukšta kokybe. Dėl to vien tik rankinio testavimo nebeužtenka – atsiranda poreikis pradėti automatinio testavimo procesus, kas su savimi atneša poreikį testuotojui turėti ne tik analitinių žinių, bet ir techninių – išmanyti vieną ar kelias programavimo kalbas, testavimo karkasus, technikas, įrankius.

2019m. atliktame tyrime, buvo tirta virš 500 darbo skelbimų testuotojams, siekiant išsiaiškinti, ko reikalauja darbdaviai. Buvo atskleista, kad dabartinėje rinkoje neužtenka turėti vien tik patirties testavimo procese. Testuotojas turi būti itin aukštos kvalifikacijos su stipriomis techninėmis žiniomis. Daugelis darbdavių jau turi infrastruktūrą, skirtą testavimo procesui, bet automatinio testavimo srityje jaučiamas didelis kvalifikuotų darbuotojų trūkumas. Dėl šios priežasties, testuotojai, turintys geras technines žinias, turi itin didelę paklausą. (Florea & Stray, 2019)

Minėtame ISTQB tyrime buvo užduotas klausimas, ką, jų įmonės testavimo procese, reikia gerinti. 64% atsakiusiųjų pasirinko testavimo automatizavimą, 47% - žinias apie testavimo procesą, 45% - bendradarbiavimą tarp programuotojų ir testuotojų. Šie rezultatai mažai tesiskyrė nuo 2016m. atliktos tokios pat apklausos. Koreliuojančiai, klausime apie didžiausią testavimo iššūkį, daugiausiai balsų sulaukė, vėlgi, automatinis testavimas. Deja, net 21% teigė, jog jų testavimo procese automatinio testavimo nėra, o beveik pusė respondentų atsakė, jog padengimas automatiniais testais yra iki 20%. (ISTQB, 2018)

Įdomus tyrimas buvo atliktas 2019m. kuriame bandyta rasti koreliaciją tarp automatinių testų kiekio, defektų radimo greičio bei klientų pasitenkinimo. Rezultatai parodė, kad komandos, kurios neturi testavimo automatizavimo proceso, turi 13% didesnę tikimybę rasti defektus prieš pat produkto išleidimą į produkciją. Įmonės, kurios automatizuoja didžiąją dalį savo testų, turi 18% didesnę tikimybę turėti gerus klientų atsiliepimus. (Florea & Stray, 2019)

Vertinant testavimo procesą, taip pat svarbu paminėti DevOps (angl. *Development & Operations*) mąstymo būdą, kuris siekia suderinti stabilų sistemos veikimą su nuolatiniu jos keitimu, atnaujinimu. Jis apjungia programų kūrėjų (angl. *Development*) ir sistemas prižiūrinčių IT specialistų (angl. *Operations*) tikslus bei padeda abejoms pusėms efektyviau dirbti savo darbą.

DevOps tikslus padeda pasiekti CI/CD (angl. *Continuous integration and continuous delivery*) praktikos – nuolatinis integravimas bei nuolatinis diegimas, kas reiškia, jog padarytas atnaujinimas iškart pasiekia galutinį vartotoją. (Rajkumar et al., 2016) Tai testavimo procesą padaro dar sudėtingesniu – privalu testuoti itin dažnai bei greitai, tuo pačiu metu užtikrinant gerą produkto kokybę. CI/CD praktikos dar labiau iškelia testavimo kainą, tačiau tik laikinai, jei užtikrinamas tinkamas procesas, aukšta jo branda. Įtvirtinus šias praktikas, testavimo kaštai stipriai sumažėja. 2019m. buvo atlikti apklausa apie CI/CD praktikas. Rezultatai rodo, jog 50% naudoja nuolatinį integravimą, 38% nuolatinį tiekimą ir 29% nuolatinį diegimą, o 25% visų apklausų planuoja greitu metu pradėti naudoti šias praktikas. Tai patvirtina, jog DevOps praktikų populiarumas auga. (Landscape, 2019)

Nors sukurta daugybė praktikų, metodų bei įrankių, kurie turėtų palengvinti testavimą, tačiau, šis procesas daugelyje kompanijų yra toli nuo aukšto brandos lygio ir dažniausiai būna atliekamas *ad-hoc* būdu – be struktūros, planavimo, nesilaikant geriausių praktikų. Tai veda prie neefektyvaus darbo, atsiranda didelė rizika išleisti produktą su kritiniais defektais. Neinvestavus į testavimo proceso gerinimą dabar, nebrandaus darbo kaina projekto pabaigoje gali išaugti net keliasdešimt kartų. Tačiau testavimo proceso gerinimas irgi turi savų iššūkių – neaišku nuo ko pradėti, kokį gerinimo modelį pasirinkti, atsiranda darbuotojų pasipriešinimas pokyčiams.

Šio magistro darbo **tikslas** – įvertinti testavimo proceso problematiką dabartinėje programinės įrangos kūrimo aplinkoje.

Magistro darbo **uždaviniai**:

1. Išnagrinėti mokslines publikacijas, siekiant išsiaiškinti, kas iki dabar žinoma apie testavimo proceso brandą
2. Išskirti problemų aibę, su kuriomis praktiškai susiduriama valdant testavimo procesą bei testuojant PĮ

3. Suformuluoti sąrašą pageidautinų dabartinio testuotojo kompetencijų, kurios palengvintų testavimo proceso gerinimą
4. Sudaryti išplėstinę apibendrintą testavimo proceso įsivertinimo bei pagerinimo schemą
5. Išvadose apibendrinti rezultatus ir pateikti rekomendacijas

Pirmajame darbo skyriuje gilinamasi į testavimo proceso tyrimo erdvę, atliekama susijusių literatūros šaltinių analizė (1 uždavinys). Antrajame skyriuje apibrėžta bei pagrįsta pasirinkto kokybinio interviu tyrimo metodika, kuri buvo pagrindas siekiant išskirti esmines testavimo proceso problemas bei sudarant vertingiausių testuotojo kompetencijų sąrašą. Trečiajame skyriuje, interviu tyrimo rezultatų apibendrinime, susisteminta gauta informacija (2, 3 uždaviniai) bei, apjungus literatūros analizės bei atlikto interviu tyrimo metu gautus rezultatus, sudaryta išplėstinė apibendrinta testavimo proceso įsivertinimo bei pagerinimo schema (4 uždavinys). Galiausiai, išvadose apibendrinta viso darbo rezultatai bei pateiktos rekomendacijos skaitytojui (5 uždavinys).

1. TESTAVIMO PROCESO PROBLEMATIKA TYRIMO ERDVĖJE

Apsibrėžti tikslus, kurių reikia, norint pagerinti ir kontroliuoti testavimo proceso fazes, nėra lengva užduotis. (Camargo et al., 2015) Tam palengvinti, kompanijos neretai į pagalbą pasitelkia testavimo proceso brandos įsivertinimą (angl. *TMA– test maturity assesment*) ir po to atlieka testavimo proceso pagerinimą (angl. *TPI – test process improvment*).

Atliekant literatūros apžvalgą buvo siekiama išsiaiškinti, kas iki dabar žinoma apie TMA/TPI veiklas bei kaip į tai įsipina *agile* metodologija ir CI/CD praktikos.

Visų pirma, literatūros apžvalgoje aptarta, kas apskritai yra testavimo proceso brandos įsivertinimas ir pagerinimas, sutrumpintai išreiškiamas kaip TMA/TPI. Kokie siūlomi metodai jiems atlikti, kokie požymiai parodo, kad laikas gerinti testavimo procesą. Taip pat siekiama suprasti, su kokias iššūkiais susiduriama, bandant vykdyti TMA/TPI bei kokia nauda gaunama, pagerinus procesą.

Vėliau atskleista *agile* metodologijos esmė bei nagrinėta, kaip ji atsispindi įvairiuose TMA/TPI modeliuose.

Trečia, bandyta išsiaiškinti kas yra DevOps bei kokiomis nuostatomis jis grindžiamas. Atskleista CI/CD praktikų svarba bei kokia naudą jos atneša, siekiant pagerinti testavimo procesą. Nagrinėta, kaip visa tai atsispindi TMA/TPI modeliuose.

Galiausiai, siekiant išskirti aktualiausią nagrinėtą informaciją, buvo sudarytos dvi lentelės – susisteminti TMA/TPI modeliai ir išskirti svarbiausi naudoti šaltiniai su juose esančia esmine informacija. Lenteles galima rasti atitinkamai 1 bei 2 darbo prieduose.

Ši literatūros apžvalga gali padėti visiems, susijusiems su programinės įrangos kūrimu, pasižiūrėti į testavimą iš platesnės perspektyvos, kuri apima visą testavimo procesą. Tai gali paskatinti identifikuoti esamas spragas, jas įsivertinti ir, esant poreikiui, atlikti testavimo proceso pagerinimą.

Literatūroje galima rasti nemažai atliktų analizių, nukreiptų į TMA/TPI modelius. Tačiau šis darbas skiriasi tuo, jog nei vienoje iš jų nebuvo žiūrėta iš *agile* metodologijos bei DevOps praktikų perspektyvų, kurios yra itin reikalingos šiandieninėse programų kūrimo veiklose.

1.1. Testavimo įsivertinimas/pagerinimas

Norint kokybiškai atlikti TMA/TPI, tyrinėtojai bei praktikai tam siūlo įvairius būdus ir metodus. (Garousi, Felderer, & Hacaloglu, 2017) Tačiau prieš pradedant gilintis į siūlomus metodus, visų pirma, svarbu suprasti bendrą TMA/TPI procesą. Jis buvo aiškiai apibrėžtas dar

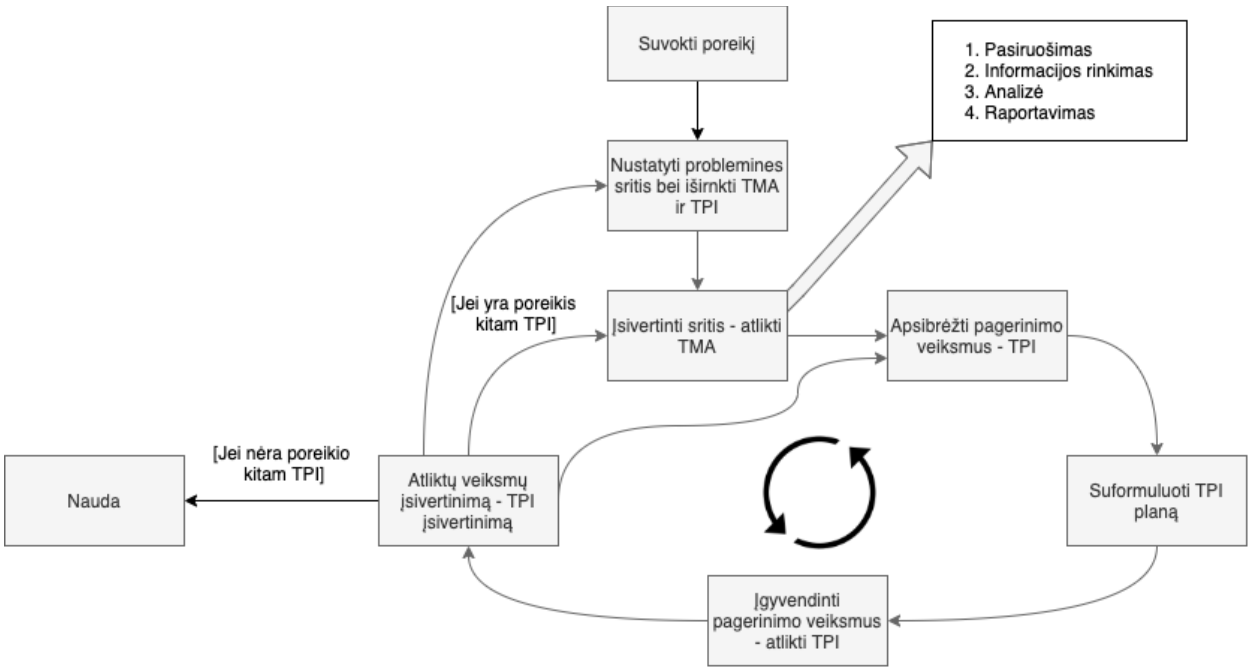
1999 m. profesorių Koomen ir Pol išleistoje knygoje. (Tim Koomen, 1999) Apibendrintas procesas, pavaizduotas 1 paveikslo diagramoje, prasideda suvokus poreikį, jog reikia gerinti produkto kokybę, kas reiškia, kelti testavimo proceso brandą.

Sekantys žingsniai yra identifikuoti testavimo proceso problemines sritis ir pagal tai pasirinkti tinkamus TMA/TPI modelius. Svarbu pabrėžti, kad po šio žingsnio prasideda ciklišas procesas, kartojamas tol, kol yra poreikis – įsivertinti problemines sritis, nustatyti, kaip bus bandoma spręsti iškilusius sunkumus, suformuluoti aiškų planą. Galiausiai, testavimo inžinieriai ar visa komanda turi įgyvendinti nustatytą planą bei įsivertinti gautus rezultatus bei naudą. Jei įsivertinus rezultatus matoma, jog visi tikslai pasiekti ir didesnių spragų testavimo procese nebeliko, galima užbaigti veiklą. Tačiau vėl pajutus poreikį, naujai atsiradusias problemas, patartina kartoti šį procesą.

Nors šis procesas yra pagrindas daugeliui kuriamų TMA/TPI modelių, tačiau sulaukia kritikos dėl savo neišbaigtumo. 2017 m. straipsnyje tyrėjai teigė, jog suvokus poreikį atlikti TMA/TPI, svarbus momentas yra iškomunikuoti jį visoms susijusioms šalims, pabrėžti svarbą bei naudą ir gauti jų palaikymą. To nepadarius, dažnai TMA/TPI daromas atmestinai, neskiriant tam nei laiko, nei resursų, neišgaunant naudos. Kas lemia tik išeikvotus resursus, o testavimo procesas nėra pagerinamas. (Garousi, Felderer, & Hacaloglu, 2017)

1 paveikslas

TMA/TPI apibendrintas procesas



Šaltinis: Tim Koomen, 1999.

1.1.1. Faktoriai, liudijantys TMA/TPI taikymo poreikį organizacijoje

Pradėti taikyti TMA/TPI veiklas komandoje, departamente ar visoje organizacijoje, turi būti aiškūs poreikiai, kurie skatintų skirti laiką/energiją bei pinigus proceso gerinimui.

Paprastai poreikis atsiranda tada, kai keletas paskutinių atliktų projektų nepasiekia nusistatytų lūkesčių bei tikslų. Taip pat jis neretai kyla ir iš verslo/departamento pusės, kai tenka mažinti išlaidas arba atsiranda tikslas siekti aukštesnio klientų pasitenkinimo produkto kokybe. Abejais atvejais pradedama analizė, siekiant išsiaiškinti, kas galėjo sąlygoti prastą darbo rezultatą. (Kulkarni, 2006)

Kadangi bet kokios programinės įrangos kūrimo esminis tikslas yra patenkintas galutinis vartotojas, todėl kokybė yra pagrindinis prioritetas. Už kokybę atsakinga visa komanda, tačiau didžiausia našta krenta ant testuotojų pečių. Būtent dėl to spragos testavime yra vienos svarbiausių, kurias reikia spręsti pirmas. (Krasner, 2018)

2017 m. buvo atlikta plataus spektro literatūros analizė (angl. *Multivocal Literature Review*), kurioje poreikius atlikti TMA/TPI suskirstė į keturias kategorijas (Garousi, Felderer, & Hacaloğlu, 2017):

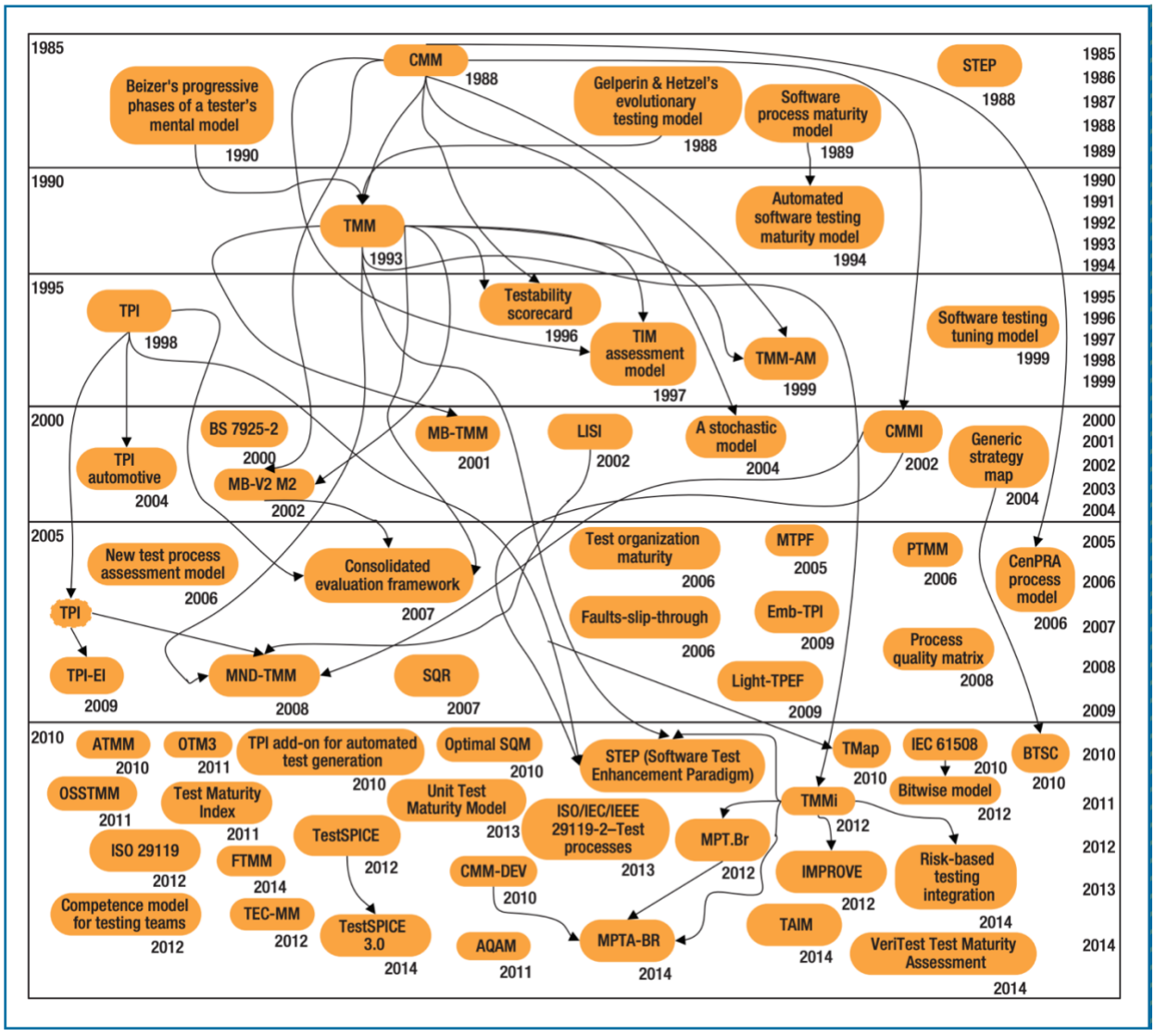
1. Proceso ir veiklų - konkretūs pavyzdžiai poreikių, kuriuos lemia spragos procese, yra dėmesio trūkumas testavimo veiklai, mažas pačių testuotojų darbo efektyvumas bei produktyvumas, nesugebėjimas pasiekti nusistatytų tikslų, vidinių susijusių asmenų nepasitenkinimas, testavimo pabaigos (angl. *exit*) kriterijų trūkumas, nenustatyta testavimo strategija/planas.
2. Programinės įrangos kokybės – didelis kiekis defektų dėl žemos testavimo kokybės, tiesioginis santykis tarp testavimo proceso kokybės bei galutinio produkto kokybės ir išteklių trūkumas, skiriamas testavimui.
3. Piniginis – paskaičiuota, kad testavimo fazė kainuoja 30-50% bendro projekto kūrimo vertės. Tai grindžiama tuo, kad procesas dažniausiai orientuotas į techninius dalykus, o ne siekiama išgauti maksimalią naudą verslui. (Zanoni et al., 2014) Kitas svarbus aspektas, jog susiję asmenys leis atlikti TMA/TPI tik tada, jei išlaidų ir naudos analizė parodys, kad nauda didesnė už sąnaudas.
4. Laiko ir planavimo – vėlavimas išleisti programinę įrangą į produkcinę aplinką dėl per vėlai rastų defektų, ką dažniausiai įtakoja didelės apimties rankinis testavimas, automatinio testavimo nebuvimas.

1.1.2. Testavimo proceso gerinimo metodai atliekant TMA/TPI

Praktikai siūlo itin platų testavimo proceso gerinimo metodų pasirinkimą. 2 paveiksle pateiktas chronologinis evoliucinis grafas, vaizduojantis TMA/TPI modelius bei jų ryšius – dažniausiai modeliai kuriami ankstesniųjų pagrindu. Taip atsitinka, kai senasis modelis nebeatlaiko nuolat kintančių bei augančių testavimo proceso poreikių arba kai norima akcentuotis/spręsti konkretesnę problemą, o ne gerinti visą procesą bendrai. Diagramoje matosi, kaip šių modelių kiekis su kiekvienais metais stabiliai auga, kas reiškia vis didėjantį jų aktualumą bei poreikį.

2 paveikslas

TMA ir TPI modelių evoliucija ir jų ryšiai



Šaltinis: Garousi, Felderer, & Hacaloğlu, 2017.

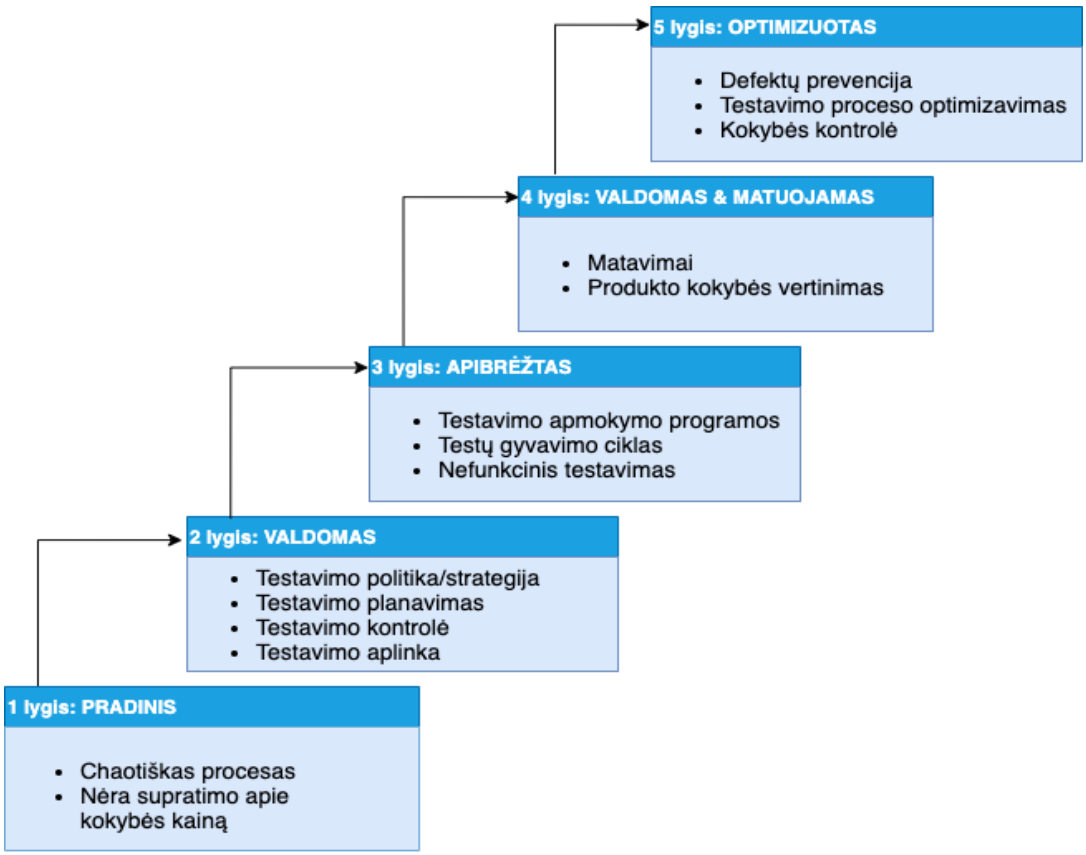
Pirmas modelis, CMM (angl. *Capability Maturity Model*), buvo pasiūlytas dar 1987 m. ir turėjo didžiulę įtaką programinės įrangos kūrėjams. Jis buvo kuriamas su mintimi, kad galutinio produkto kokybė stipriai priklauso nuo programinės įrangos kūrimo procesų kokybės. Modelio esmė yra padėti išsirinkti procesų gerinimo strategijas apsibrėžiant esamą situaciją bei identifikuoti dabartines problemas. (Paulk et al., 2011) Procesas gali būti apibrėžtas kaip veiklos, metodai, praktikos ir transformacijos, kurios padeda kurti ir palaikyti programinę įrangą. Kai organizacijos branda kyla, procesas tampa labiau apibrėžtas ir aiškiai vykdomas visos organizacijos lygyje. To dėka kyla programinės įrangos kokybė, auga produktyvumas, mažiau dubliuojamas darbas, pagerėja projekto planai ir valdymas. (Paulk, 2009)

Dabar praktikoje retai pasitaiko CMM naudojimas, kadangi jo pagrindu sukurta nemažai naujesnių, patobulintų versijų, kurios geba geriau patenkinti rinkos poreikius. Atlikti tyrimai rodo, jog vienas populiariausių testavimo modelis yra būtent TMMI (angl. *Test Maturity Model Integration*), sukurtas 2012 m. (Afzal et al., 2016).

Kuriant šį modelį, buvo atrinkta grupė testavimo ekspertų iš lyderiaujančių IT įmonių Brazilijoje, kurie pasidalino savo patirtimi bei gerosiomis praktikomis. Taip atsirado TMMI, kurio struktūra susideda iš 5 brandos lygių. (Camargo et al., 2015)

3 paveikslas

TMMI modelio lygiai



Šaltinis: sudaryta autoriaus, remiantis Camargo et al., 2015.

3 paveiksle pavaizduoti TMMI modelio brandos lygiai. Žemiausias lygis (1) – chaotiškas, nevaldomas testavimo procesas, aukščiausias - kontroliuojamas bei optimizuotas testavimas (5). Šis modelis yra pakopinės (angl. *staged*) architektūros, kas reiškia, jog norint pereiti į aukštesnį lygį, turi būti sukurtas geras pagrindas prieš tai esančiame lygyje (įgyvendinti visi žemesniojo lygio kriterijai). (TMMi Foundation Reference Model, 2012)

Kiekvienas lygis turi apibrėžtus tikslus, kurie yra išskaidyti į konkrečias praktikas. Jose aprašytos veiklos, kurios padeda pasiekti tikslus.

TMMI modelio lygiai:

1 lygis – organizacijai pradėjus įgyvendinti TMMI modelį, daroma prielaida, kad ji yra pirmame lygyje, kuriame nėra jokio apibrėžto proceso, tik chaotiškas darbas.

2 lygis – naudojamos fundamentalios testavimo proceso praktikos. Veiklos valdomos, yra testavimo strategija. Joje nurodyta, koks yra testų dizainas, kada ir kaip jie turi būti vykdomi. Yra testavimo aplinka.

3 lygis – visi organizacijoje vykdomi projektai laikosi tų pačių standartų ir procedūrų. Komandos darbas gerai organizuotas, vykdomos mokymo programos. Testavimo fazė integruota į visą projekto gyvavimo ciklą nuo pat pradžių. Planuojami ir vykdomi nefunkciniai testai.

4 lygis – veikla bei jos rezultatai matuojami, ko dėka kontroliuojamas defektų kiekis nuo pačios projekto pradžios.

5 lygis – visos susijusios veiklos yra optimizuotos, siekiant nuolatinės defektų prevencijos. Produkto kokybė nuolat kontroliuojama, prižiūrima.

Literatūroje minima, kad TMMI modelis labiau tinka didelėms įmonėms, kadangi reikalauja nemažai investicijų tiek piniginiu, tiek žinių atžvilgiu. (Veenendaal, 2018)

Mažesnėms įmonėms tinkantis, taipogi, labai populiarus rinkoje modelis, yra TPI (angl. *Test Process Improvement*), sukurtas 1999 m. Pagal 2014 m. atliktą sisteminę literatūros apžvalgą, TMMI ir TPI yra dažniausiai naudojami modeliai, kurių pagrindu bandyti kurti daugelį kitų, neprigijusių, modelių. (Garcia et al., 2014) Priešingai negu TMMI, TPI yra ne pakopinis, o nuolatinis (angl. *continues*) modelis. Jame lengva sekti gaires naudojantis testavimo brandos matrica (aiškus 4 lygių aprašymas, kontroliniai punktai), kas padeda įsivertinti proceso brandą ir susidaryti planą sekantiems pagerinimams. (Afzal et al., 2016)

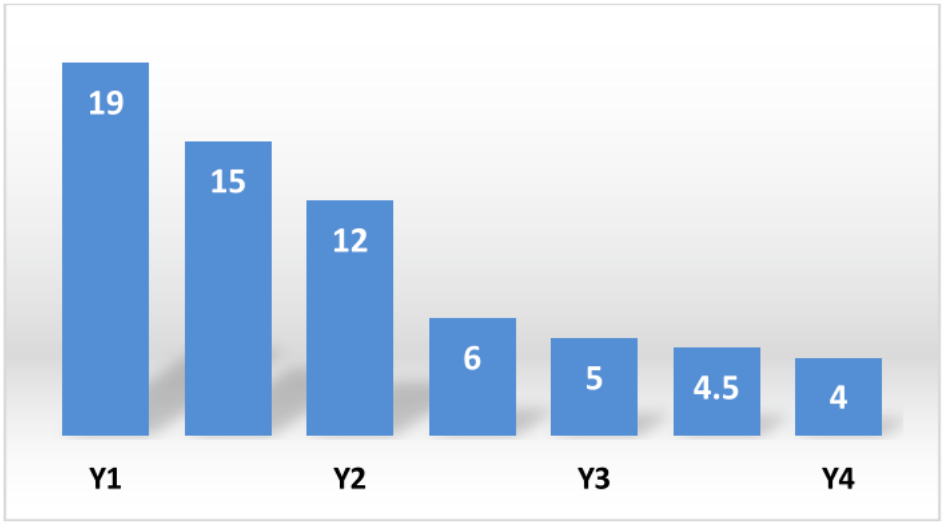
1.1.3. TMA/TPI veiklų nauda

Kad ir koks kokybės gerinimo modelis bus pasirinktas, tai kainuos laiko bei pinigų. Sakoma, jog modelio suteiktą naudą sunku pamatuoti skaičiais. Daugeliui organizacijų yra ganėtinai lengva paskaičiuoti, kiek pinigų išleido, bet sunku paskaičiuoti gautą naudą.

Tiesioginė nauda aiškiausiai pasimato, kai lyginama buvusi situacija su dabartine, po atlikto kokybės gerinimo. Netiesioginė nauda, tokia kaip „padidėjęs klientų pasitenkinimas“ arba „padidėjusi darbuotojų motyvacija“ gali būti išmatuoti atliekant apklausas ar interviu. (Veenendaal, 2018)

Siekiant iliustruoti galimą naudą, bus pateikta vienos IT įmonės rezultatai, kuri įgyvendino TMMI modelio veiklas ir pasiekė 3 lygį. Anot įmonės, ryškiausias pasiekimas buvo akivaizdžiai sutrumpėjusi testų vykdymo fazė. Tai atsispindi 4 paveiksle. Po 4 metų laikas sutrumpėjo daugiau nei 4 kartus.

4 paveikslas
Sisteminių testų vykdymo laikas pateiktas savaitėmis



Šaltinis: Veenendaal, 2018.

Kitos įmonės dalinasi, jog laikui bėgant, galima sutaupyti net iki 40% testavimo kaštų. Taip pat kokybė tampa labiau nuspėjama, ko dėka būna mažiau vėlavimų atiduoti galutinį produktą klientui. Defektų atsiradimo tikimybė produkcinėje aplinkoje sumažėja iki 50%, dėl ko mažėja palaikymo kaštai. Pastebėta, kad kyla darbuotojų pasitikėjimas savo jėgomis, didėja motyvacija (Veenendaal, 2018).

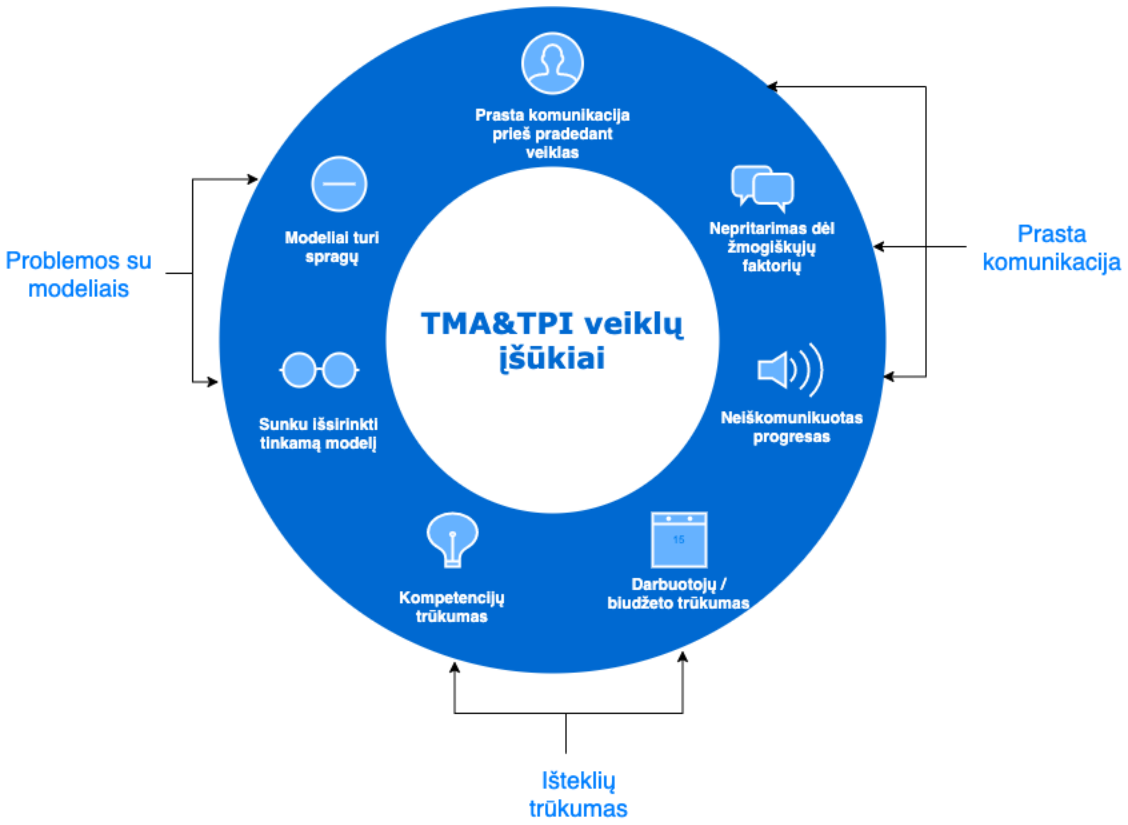
1.1.4. TMA/TPI veiklų trūkumai

Nors TMA/TPI teikia daug naudos, tačiau įmonės, bandančios pagerinti testavimo procesą besiremiant tam tikru modeliu, susiduria su įvairiais iššūkiais. Jie atvaizduoti 5 paveiksle. Atlikus analizę buvo išskirtos 3 grupės:

1. Problemos su modeliais
2. Prasta komunikacija
3. Išteklių trūkumas

5 paveikslas

Modelis, rodantis su kokiais iššūkiais susiduriama, norint atlikti TMA/TPI veiklas



Šaltinis: sudaryta autoriaus, remiantis Garousi, Felderer, & Hacaloglu, 2017.

Kaip išsirinkti tinkamą modelį

Pats pirmas sunkumas atsiranda, kai reikia išsirinkti tinkamą modelį. Jų yra virš 60, taigi kaip rasti sau tinkamą? Praktikai teigia, jog vienas modelis negali tikti visoms skirtingos situacijoms aprėpti. Tačiau tikslesnis galimas paaiškinimas yra tas, jog originalūs modeliai dažnu atveju buvo grindžiami ne atsiradusiais poreikiais, o hipotetiškai argumentuota motyvacija. Taip pat panašu, jog tyrinėtojai ne visada pilnai įsigilindavo į tai, kas jau buvo padaryta kituose modeliuose, dėl ko atsirado nemažas dubliavimasis tarp modelių. Dėl šių priežasčių išsirinkti tinkamą iš didelio kiekio tarpusavyje persipynusių modelių yra nemenka užduotis (Garousi, Felderer, & Hacaloglu, 2017)

Cognizant (viena didžiausių IT įmonių pasaulyje) dalinasi savo patirtimi apie tinkamo modelio pasirinkimą. Anot jų, nėra vieno modelio tinkančio visiems. Visų pirma reikia atsakyti į šiuos klausimus:

1. Kokie pagrindiniai jūsų verslo tikslai?
2. Kokio tipo aplikacija yra testuojama?
3. Centralizuota ar išskirstyta IT infrastruktūra?

Atsakant į šiuos klausimus, modelio pasirinkimas bus pagrįstas tiek strategiškai, tiek veiklos požiūriu. Būtina, jog sprendimas būtų suderintas su ilgalaikiais organizacijos tikslais. (Karthikeyan et al., 2014)

Svarbu paminėti, kad modelio pasirinkimas ne vienintelė problema. Patys modeliai turi spragų. Itin nemažai kritikos sulaukia pakopiniai modeliai, tokie kaip anksčiau aptartas TMMI. Pakopiniai - reiškia, jog organizacija turi patenkinti visus reikalavimus tam tikram modelio lygiui, kad galėtų gauti sertifikatą. Net jei viena veikla iš sąrašo neatlikta, aukštesnio lygio negalima pasiekti. Nors, galbūt, ta veikla nėra aktuali organizacijos kontekste. Praktika rodo, jog šis sprendimas labiau tinka didelėms organizacijoms (Camargo et al., 2015). Dauguma nenori aklai sekti visų nurodymų. Ypač mažesnės organizacijos nori tobulinti būtent tas vietas, kurios joms yra svarbios bei tenkina jų verslo poreikius, o ne atlikti tam tikrus reikalavimus vien dėl to, kad gautų tame lygyje sertifikatą. Dėl šios priežasties, pakopiniai modeliai sunkiai parodo esamą organizacijos testavimo proceso vaizdą bei padeda siekti verslui naudingų gerinimo tikslų. Neišskiria stiprių/silpnų vietų, kas leistų identifikuoti konkrečias sritis, kurių pagerinimas atneštų didžiausią naudą. Praktikai pataria rinktis tuos modelis, kurie, visų pirmą, atsižvelgtų į organizacijų verslo poreikius (Balaraman, 2018)

Kiti autoriai pastebėjo, kad nors modeliai išsprendžia nemažai testavimo proceso spragų, tačiau skiria mažą dėmesį būtent automatiniam testavimui, kurio svarba kiekvienais metais vis didėja. Modeliai aiškiai nurodo kaip spręsti bendras testavimo problemas, bet palieka mažai detalių automatizavimo iššūkiams spręsti (Eldh et al., 2014). Taip pat praktikai teigia, jog išskirta mažai matų, skirtų išmatuoti automatinio testavimo brandą, bei trūksta konkrečių žingsnių, kurie padėtų tobulėti šioje srityje (Wang, 2018).

Daugelis modelių neprigijo, nes keičiantis aplinkai, jie nesikeitė. Modeliai turi būti atnaujinami tam, kad neatsiliktų nuo vyraujančių programų kūrimo tendencijų. (Park et al., 2018)

Prasta komunikacija

Kitas atliktas tyrimas atskleidė, jog vienas didžiausių trukdžių yra per silpnai iškomunikuotas TMA &TPI veiklų poreikis ir prasmė tarp IT komandos narių bei visos kompanijos atžvilgiu. Dėl to šios veiklos nėra prioritizuojamos, kas lemia prastą pasiruošimą/planavimą ir atitinkamą veiklų vykdymą, negaunama jokios naudos, ir net priešingai, be reikalo išeikvojami resursai (Garousi, Felderer, & Hacaloglu, 2017).

Dėl prastos komunikacijos atsiranda kitas žmogiškasis faktorius – testuotojai patiria nemaža stresą dėl pokyčių, nežinomybės. (Woodruff, 2003) Jei darbo kultūra yra konservatyvi, atsiranda bendras pasipriešinimas pokyčiams, į ką būtina atsižvelgti, kai komunikuojama apie norimą atlikti proceso gerinimo veiklą (Burnstein, 2003).

Labai svarbu, kad organizacijoje visi susiję asmenys gerai suprastų proceso gerinimo veiklų svarbą bei prasmę. Tam gali padėti aiškus esamų problemų suvokimas, kas pagrįstai įrodys, jog būtina imtis tam tikrų veiksmų, norint gerinti proceso kokybę. Pradėjus pačią gerinimo veiklą, taip pat svarbu sekti progresą bei juo nuolatos dalintis su kolegomis. Tokiu būdu susiję asmenys aiškiais matys, jog resursai naudojami ne veltui, taip yra gerinamas testavimo procesas, ko dėka ateityje bus sutaupyta lėšų. (Garousi, Felderer, & Hacaloglu, 2017).

Išteklių trūkumas

Kalbant apie išteklius, svarbu paminėti, jog pajaučiamas darbuotojų trūkumas, nes kartu su proceso gerinimu atsiranda daug papildomų darbų, kuriems padaryti reikia laiko. Šis trūkumas itin pastebimas mažose ar vidutinio dydžio organizacijose. (J. García , A. de Amescua , M. Velasco, 2008)

Kiti praktikai pabrėžia, kad kolektyvas dažnai neturi tinkamų kompetencijų vykdyti veiklas, nurodytas proceso gerinimo modelyje. Dėl to pataria, prieš pradedant veiklas, iš pradžių suteikti reikiamus apmokymus testuotojams. Pabrėžiama, jog svarbu apmokyti ir vadovus tam, kad suprastų proceso gerinimo esmę bei gebėtų sekti metrikas. (Kasurinen, 2012)

Šie faktoriai susideda į vieną – norint atlikti proceso gerinimo veiklas, reikia skirti papildomo biudžeto. Tai yra investicija į ateitį, kurią atliktus teisingai, galima tikėtis daug didesnės grąžos.

1.2. *Agile* ir TMA/TPI veiklos

1.2.1. *Agile* metodologija

Agile metodologija išpopuliarėjo dar 2001 m., kai buvo išleistas taip vadinamas „Agile Manifesto“, kuriame yra aprašyti svarbiausi šios metodologijos principai. Ji iškilo kaip priešprieša krioklio (angl. *Waterfall*) modeliui, kuriam sunkiai sekėsi suvaldyti greitai besikeičiančius aplinkos, o tuo pačiu ir reikalavimų, poreikius. (Gaurav & Pradeep, 2012)

Agile metodologija yra sudaryta iš rinkinio programinės įrangos kūrimo metodų, kurie grindžiami iteracijomis. 4 esminės charakteristikos, kurios yra pagrindas visiems *agile* principams (Mordinyi et al., 2010):

1. Prisitaikantis (angl. *adaptive*) planavimas
2. Iteratyvūs kūrimas
3. Greitas bei lankstus požiūris į pokyčius
4. Komunikavimas

Pagrindinis principas – „lengvas, bet efektyvus" bei orientacija į žmones, o viso ko centras – gera komunikacija. Kadangi ši metodologija pabrėžia lengvumą, dėl to ji labiau tinkama mažoms organizacijoms, tačiau laikui bėgant, vis daugiau didelių organizacijų taipogi pasirenka *agile* kelią.

Ši metodologija numato, jog komandos turi pradėti savo darbą nuo paprastų aiškiai suprantamų užduočių dar prieš baigiant formuluoti galutinius reikalavimus, kurių detalių daugės kiekviename ateinančiame cikle. Šis iteracinis procesas apima visas produkto kūrimo dalis – dizainą, programavimą bei testavimą. Tokiu būdu bandoma išspręsti krioklio metodologijos problemą – prisitaikymą prie nenuspėjamos verslo bei technologijų aplinkos. (Mordinyi et al., 2010)

Metodologija pabrėžia kokybės svarbą, savarankiškos komandos naudą, bendravimą su klientais, mažesnį kiekį dokumentacijos. Visa tai padeda sutrumpinti produkto ar atnaujinimo laiką iki produkcijos.

Agile principai skaido užduotis į mažesnius tikslus, kuriuos galima įgyvendinti per tam tikrą iteraciją. Iteracijos yra trumpi laiko tarpai, dažniau nuo vienos iki keturių savaičių. Kiekvienoje iteracijoje apimamas pilnas programinės įrangos kūrimo ciklas, įskaitant planavimą, reikalavimų analizę, dizainą, programavimą bei testavimą. Tai padeda sumažinti riziką bei lengvai prisitaikyti prie besikeičiančių sąlygų.

1.2.2. *Agile* testavimo procese

Viena iš *agile* teikiamų naudų, yra produkto kokybės pagerėjimas. Ši metodologija skatina į testavimą žiūrėti per kliento perspektyvą, kas padeda greičiau identifikuoti defektus ar kitus netikslumus. Testavimas įterpiamas ne pačiam gale (kaip krioklio modelyje), bet nuo pat projekto pradžios. Tai yra nuolatinė veikla, kuri padeda programuotojams kurti kokybiškesnį kodą dėl nuolat gaunamo grįžtamojo ryšio. (Razak & Fahrurazi, 2011)

Knygoje „Agile Testing: A Practical Guide for Tester and Agile Teams" (liet. *Agile* testavimas – praktinis gidas testuotojui bei *agile* komandai) autorė išskyrė pagrindinius principus, kuriuos perėmė daugelis *agile* metodologijos praktikų. Apibendrinus jos mintis, galima teigti, jog itin pabrėžiamas glaudus ryšys tarp testuotojų ir programuotojų bei kliento. Svarbu po kiekvienos iteracijos turėti bent dalinai veikiantį produktą, kurį galima parodyti klientui. *Agile* testuotojai niekada nenustoja mokytis – kiekviena iteracija atneša naujų žinių. Itin svarbus lankstumas, gebant prisitaikyti prie reikalavimų pokyčių. Testuotojai neturi laukti, kol jiems bus duota darbo. Privalu rodyti iniciatyvą, savarankiškai organizuoti savo darbą. Atsiranda poreikis gilintis į techninę pusę, nes norint spėti su greitu kūrimo tempu, tenka rašyti automatinius testus.

Šie principai suformuoja tam tikrą idealų *agile* testuotojo profilį. Visų pirma, jis privalo turėti puikius bendravimo įgūdžius bei mokėti laviruoti tarp programuotojų ir verslo žmonių.

Nebeužtenka vien rankinio testavimo žinių, todėl *agile* praktikas privalo nuolat gilinti savo žinias iš techninės pusės. Jis turi būti lankstus dažnų pokyčių atžvilgiu bei orientuotas į verslo poreikių tenkinimą. Dar vienas svarbus punktas, jog testuotojas privalo būti orientuotas į rezultatą, teikti nuolatines ataskaitas apie esamą produkto situaciją.

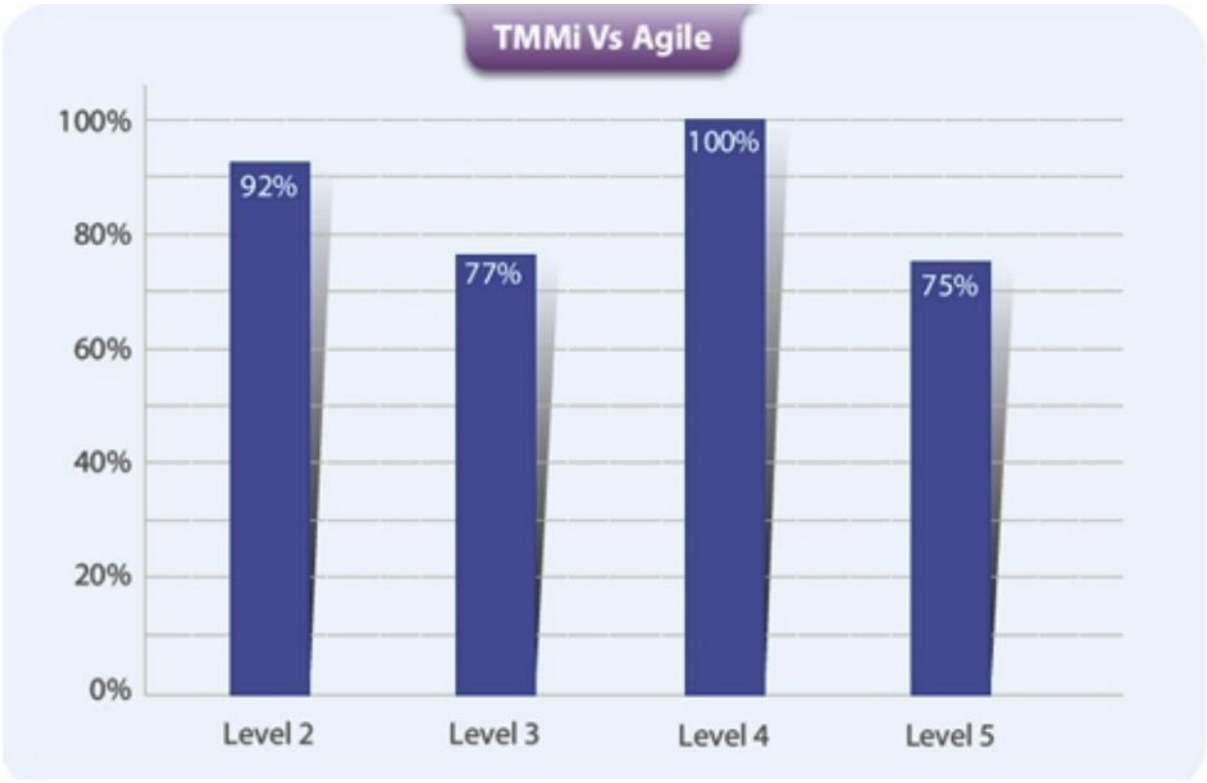
1.2.3. *Agile* bei testavimo brandos modeliai

Aptarėme dvi praktikas/būdus, kurie padeda siekti aukštesnės produkto kokybės – *agile* metodologijos laikymasis bei TMA/TPI vykdymas. Kyla klausimas, ar TMA/TPI veiklos tinkamai įsipina į *agile* darbo metodologiją?

Vieni praktikai teigia, jog *agile* ir TMA/TPI veiklos yra dvi atskiros pusės tos pačios monetos. Siekiant įrodyti TMMI tinkamumą *agile* aplinkai, buvo atliktas kriterijų atitinkamumo tyrimas. Kiekvienas brandos modelio lygis buvo įvertintas pagal *agile* metodologijos atitinkamumą – neatitinka/dalinai atitinka/pilnai atitinka. Pirmasis chaotiškas lygis praleistas, nes jis sunkiai suderinamas su *agile*. (Services, 2016).

6 paveikslas

TMMI lygių atitikimas agile metodologijoje



Šaltinis: Services, 2016

Rezultatai 6 paveiksle rodo, kad TMMI gali gyvuoti kartu su *agile* (jų atitikimas yra 86%), nors ir yra smulkių niuansų. Tai leidžia daryti išvadą, kad *agile* ir TMMI yra suderinami, siekiant bendrų verslo bei technologinių tikslų. Tam pritaria ir straipsnio „Test Process Improvement and Agile: Friends or Foes?“ (liet. *Testavimo proceso pagerinimas ir agile: draugai ar priešai*?) autorius, jis teigia, jog TMMI ir *agile* gerai dera kartu. (Veenendaal, 2014) *Agile* skatina greitesnį produkto išleidimą, lankstumą pokyčiams, kai tuo metu TMMI fokusuojasi į geresnę produkto kokybę. TMMI modelio kūrėjai netgi išleido dokumentą, kuriame paaiškino, kaip kiekvieną lygį suderinti su *agile* metodologija. (Foundation, 2019)

Literatūroje galima rasti ir specifinių, skirtų būtent *agile* metodologijos praktikams testavimo gerinimo modelių. Venas iš pavyzdžių yra 2010 m. sukurtas ATMM (angl. *agile testing maturity model*). Autoriai teigia, kad lūkesčiai testavimo praktikoms turi būti suderinti pagal komandos galimybes juos vykdyti. Šis modelis, kaip ir TMMI, yra pakopinis, turi 5 lygius: krioklio, formavimo, *agile* derinimo, atlikimo, keitimo. (Ronen, 2011) Tačiau literatūroje informacijos apie jį yra mažai, dėl ko galima teigti, jog praktikoje šis modelis neprigijo.

1.3. DevOps ir testavimo branda

DevOps įvardijama kaip mąstymo būdas, tam tikrų praktikų laikymasis, o kartais minima kaip atskira metodologija. Kad ir kaip bepavadintum, DevOps esmė nesikeičia – pagrindinis tikslas yra pagreitinti produkto atnaujinimą produkcinėje aplinkoje kartu užtikrinant aukštą kokybę. Šio tikslo įgyvendinimas turi įtakos darbo procesams, naudojamoms technologijoms, netgi organizacinei struktūrai bei verslo praktikoms. (Zhu et al., 2018)

Vienas pagrindinių DevOps bruožų yra CI/CD - nuolatinis integravimas bei nuolatinis diegimas. Dėl to programuotojai gali atnaujinti kodą keletą kartų per dieną, naudojant automatinius konstruktus (angl. *build*) bei automatinį testavimą, kas iškart parodo rastus defektus. Jei nerasta jokių defektų, atnaujintas kodas yra perkeliamas į produkciją. Pagal 2017m, atliktą tyrimą, aukšto lygio kompanijos, tokios kaip Amazon arba Netflix, atnaujina savo kodą produkcinėje aplinkoj keletą tūkstančių kartų per dieną. Konfigūracija leidžia sugrąžinti kodą (angl. *rollback*) iš produkcijos, aptikus klaidas. Deja, ne visos organizacijos geba pasiekti tokį aukštą lygį. Viena to priežasčių išskiriama automatinio testavimo trūkumas. (Agarwal et al., 2020)

Per mažai skiriamą dėmesį automatiniam testavimui pastebėjo praktikai, atliekantys testavimo proceso pagerinimą. Anot jų, esami TMA/TPI modeliai nepakankamai aprašo pagerinimus, reikalingus būtent automatiniam testavimui. Žiūrint į anksčiau aptartą TMMI modelį, jame atnaujinimai padaryti atsižvelgiant į naujus procesus, atėjusius iš *agile* metodologijos, bet realių pasiūlymų, kaip pagerinti būtent automatinį testavimą nepateikta. (Veenendal, 2012)

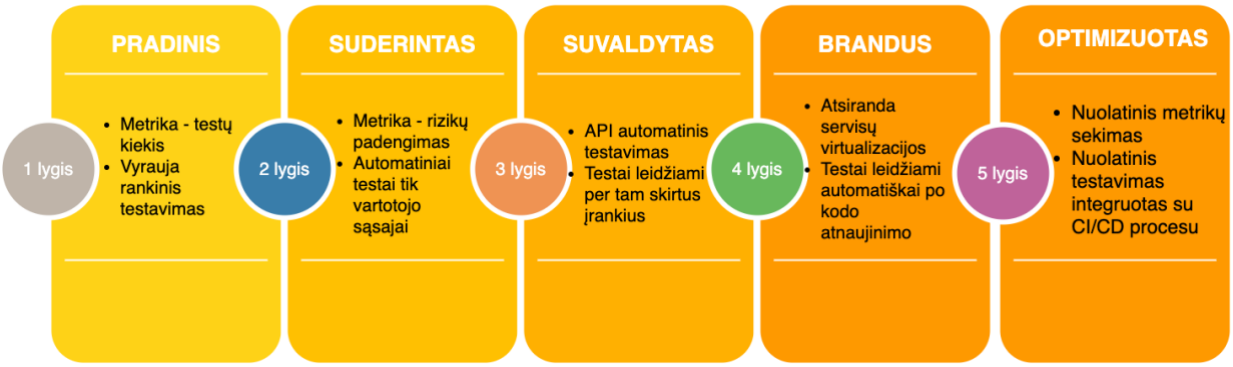
Dėl aukščiau išvardintų priežasčių, 2014m. buvo pradėtas kurti TAIM (angl. *Test Automation Improvment Model*) – automatinio testavimo pagerinimo modelis, kurio didžioji dalis vertinimo turėjo būti pagrįsta patikrintomis metrikomis. Tai įgalintų apsibrėžti, ką galima vadinti geresniu, efektyvesniu kainos atžvilgiu, objektyviu ir brandžiu būdu, siekiant atlikti automatinio testavimo veiklą. Su TAIM norėta apibrėžti 10 pagrindinių sričių (angl. *KA - key areas*) ir vieną bendrą (angl. *GA -general area*). Kiekvienai KA būtų skirti matavimai, kurie padėtų sekti progresą, pastebėti grėsmes arba galimybes. Deja, nors iniciatyva sukurti šį modelį susilaukė nemažai dėmesio, tačiau taip ir nebuvo įgyvendinta (Eldh et al., 2014).

1.4. Testavimo branda *agile* ir DevOps aplinkoje

Lankstus programinės įrangos kūrimo procesas, kurį įgalina *agile* metodologija, sumažina rizikas bei pagreitina produkto išleidimą. Šią metodologiją praplečia DevOps mąstymo būdas, kuris pagerina sąsają tarp verslo ir IT žmonių, padeda automatizuoti darbą. Toks bendradarbiavimas pagerina bendrą darbo jėgos produktyvumą. Su šiais privalumais, *agile* ir DevOps praktikos stipriai įsitvirtina IT verslo srityje. (Services, 2016)

Anot Tricentis įmonės, kuri teikia nuolatinio testavimo platformą, 81% organizacijų bando prisijaukinti DevOps praktikas, o 91% siekia įgyvendinti *agile* metodologiją. Tokioje aplinkoje nuolatinis testavimas tampa ne išimtiniu, o privalomu dalyku. Tam, kad būtų pasiekti verslo lūkesčiai, organizacijos privalo optimizuoti nuolatinio testavimo praktikas, kurios leidžia čia ir dabar daryti įžvalgas apie atnaujinimus į produkciją, keliančius riziką verslui. Tai nėra vien tik automatinio testavimo poreikis. Tai reikalauja gilesnės transformacijos, kuri paveikia žmones, procesus bei technologijas. Tricentis, atsižvelgdamas į savo ilgametę patirtį, pasiūlė nuolatinio testavimo brandos modelį (angl. *Continuous Testing Maturity Model - CTMM*), kuris, anot jų, gebą atlaikyti tiek *agile,* tiek DevOps keliamus iššūkius testavimo procese (Tricentis, 2017). Pagrindiniai lygiai pateikti 7 paveiksle.

7 paveikslas
Nuolatinio testavimo brandos modelio lygiai



Šaltinis: Tricentis, 2017

Tačiau Tricenčio įmonės pagrindiniai klientai yra itin didelės korporacijos, kas leidžia daryti išvadą, jog šis modelis tinka būtent joms, bet ne vidutinio ar mažesnio dydžio įmonėms. Taipogi pabrėžiama, kad CTMM modelis skirtas dideliems transformaciniams pokyčiams, kas kartu reiškia atitinkamai didelius kaštus.

Praktikas, DevOps konsultantas, M. Hornbeek kuria labai panašų modelį, kurį pavadino nuolatiniu automatinio testavimo brandos modeliu. (angl. *Continuous Test Automation Maturity Model - CTAMM*) Jo pagrindinis tikslas – nuolatinis automatinis testavimas. Nors CTMM bei CTAMM stipriai persidengia, skirtumas tas, jog Hornbeek į lygius žiūri per tris perspektyvas – žmones, procesus bei technologijas. (Hornbeek, 2020) Tačiau kaip ir TAIM modelis, CTAMM dar nėra pabaigtas ir praktikoje naudojamas modelis.

1.5. Testavimo proceso problematikos literatūros šaltiniuose išvados

Literatūros analizėje, visų pirma, buvo išnagrinėta TMA/TPI esmė bei svarba. Šios veiklos aktualumą gali pagrįsti tai, jog nuo 1985 m. nuolat kuriami nauji ar atnaujinami esami TMA/TPI veiklų modeliai, nors ir mažai jų sulaukia pripažinimo. Analizėje pristatyti du populiariausi - TMMI ir TPI. Svarbu pabrėžti, kad šis procesas yra nuolatinė, o ne vienkartinė veikla.

Dažniausiai paskata imtis TMA/TPI veiklų atsiranda, kai keletą projektų iš eilės nepavyksta pasiekti užsibrėžtų tikslų, kai klientai palieka neigiamus atsiliepimus apie produkto kokybę arba komanda nesugeba atlikti darbų laiku. Kartais tai ateina tiesiai iš verslo / departamento pusės, kai tenka mažinti išlaidas.

Buvo aptarta, kokią nauda teikia modeliai. Ji aiškiausiai pasimato, kai lyginama situacija prieš ir po TMA/TPI veiklų. Tačiau bandant pagerinti testavimo procesą, taipogi, susiduriama su

nemažais iššūkiais. Atlikus analizę, buvo išskirtos trys problemų grupės – modelių spragos, prasta komunikacija tarp susijusių asmenų, išteklių trūkumas.

Trečioje dalyje buvo atskleista *agile* metodologijos esmė ir kaip ji suderinama su TMA/TPI veiklomis. *Agile* grindžiama iteratyviu procesu bei mokėjimu prisitaikyti prie nuolat kintančių sąlygų. Vienas iš jos pliusų, yra produkto kokybės pagerinimas. Analizė taip pat parodė, jog *agile* metodologiją ir TMA/TPI veiklas galima suderinti. Nors yra sukurta specialiai *agile* metodologijai skirtų pagerinimo modelių, tačiau jie ne itin prigijo. Praktikai teigia, jog populiarusis TMMI modelis neblogai papildo *agile* metodologijos siekius.

Taipogi išnagrinėtas DevOps mąstymo būdo vaidmuo gerinant testavimo procesą. DevOps pagrindinis siekis – greitas atnaujinimų diegimas į produkcinę aplinką. Tam pasiekti reikalingi kokybiški automatiniai testai. Deja, brandos modeliuose mažai dėmesio skiriama testų automatizavimui. Dėl šios priežasties buvo pradėtas kurti TAIM modelis, skirtas būtent testų automatizavimui, tačiau praėjus daugiau nei 8 metams jis taip ir nebuvo pabaigtas.

Galiausiai siekta išsiaiškinti, kaip derinama *agile* metodologija kartu su DevOps praktikomis, siekiant pagerinti testavimo procesą. Teigiama, jog agile ir DevOps praktikos stipriai įsitvirtino IT verslo srityje. Siekiant suderinti šias dvi sritis, buvo pasiūlytas nuolatinio testavimo brandos modelis - CTMM. Jis geba susidoroti su *agile* metodologijos keliamai iššūkiais bei DevOps mąstymo keliamais techniniai sunkumais. Taip pat aptartas CTAMM modelis, kuris yra panašaus į CTMM, tačiau į lygius žiūri per skirtingas perspektyvas. Deja, bet CTMM modelis yra orientuotas tik į itin didelis korporacijos bei didelius transformacinius pokyčius, o CTAMM modelis dar nėra pabaigtas kurti.

Siekiant susisteminti išanalizuotą literatūra, buvo sudaryta lentelė esanti 1 priede, vaizduojanti esmines aptartų TMA/TPI modelių ypatybes. Taip pat, 2 priedo lentelėje sudėta pagrinde naudotų literatūros šaltinių santrumpos su pagrindiniais jų teiginiais, kuriais buvo remtasi darbe.

2. TESTAVIMO PROCESO PROBLEMATIKOS INTERVIU TYRIMO METODIKA

2.1. Tyrimo formuluotė

Nors nagrinėta problematika tyrimo erdvėje atskleidė, jog testavimo proceso gerinimas jau nuo seno buvo aktuali bei plačiai nagrinėjama tema, tačiau ir šiandien gausu pavyzdžių, kai dėl nekokybiškai ištestuotos PĮ, įmonės patiria milijoninius nuostolius. Dėl šios priežasties buvo nutarta atsiriboti nuo teorijos ir pažvelgti praktiškai, su kokiomis problemos susiduria testavimo procesą valdantys bei testuojantys PĮ specialistai. Ši analizė itin naudinga kiekvienam, susijusiam su PĮ kūrimu – padeda identifikuoti testavimo proceso problemas ir rasti būdus, kaip jas išspręsti.

Darbe buvo analizuojama praktinių problemų aibė, išskirta atliekant kokybinį tyrimo metodą – interviu, su IT įmonių testavimo proceso ekspertais. Klausimai buvo keliami pagal literatūros analizėje išskirtas šiandieną populiarias programų kūrimo praktikas – *agile* metodologiją bei DevOps mąstymo būdą, apėmė rankinį bei automatinį testavimus bei bendrai visą testavimo procesą.

Taip pat interviu metu buvo siekiama išsiaiškinti, kokios yra pageidautinos testuotojo kompetencijos. Ši dalis gali padėti suvokti, ką reikia ugdyti jau esamuose darbuotojuose ir į ką atsižvelgti, kai ieškoma naujo testuotojo į pastiprinimą. Tai itin svarbu siekiant aukštesnės testavimo proceso brandos.

2.2. Tyrimo siekiai

Probleminis pusiau struktūrizuoto interviu tyrimo klausimas – kokios problemos trukdo pasiekti aukštesnę testavimo proceso brandą PĮ kūrimo srityje?

Tyrimo tikslas – išsiaiškinti pagrindines testavimo proceso problemas bei jas išanalizuoti.

Tyrimo uždaviniai:

U1: Išskirti esmines testavimo proceso problemas bei jas išanalizuoti

U2: Išskirti pageidautinas testuotojo kompetencijas

2.3. Pasirinkto būdo tinkamumas

Šis metodas skatina problematizavimą – dėmesio sutelkimą į unikalumus, o ne bendrumą, padeda gerai suvokti pašnekovo situaciją. Pusiau struktūrizuotas tipas leidžia pašnekovui giliau atsakyti į užduotus klausimus, tačiau kartu nenukrypti nuo temos. Dėl to šis instrumentas padėjo gerai atskleisti ir suprasti esamas testavimo problemas, kaip bandoma su jomis susidoroti įvairiose įmonėse bei kokios yra pageidaujamos testuotojo kompetencijos. (Tamaševičius, 2015)

2.4. Tyrimo vykdymas

Kokybinis tyrimas vykdytas 2021m. Jame taikytas pusiau struktūrizuoto interviu metodas, su iš anksto pateiktomis temomis/klausimais. Siekta apklausti aukšto lygio testavimo ekspertus, dėl to pradžioje buvo kalbinami testavimo konferencijų vykdytojai/organizatoriai. Pagal jų rekomendacijas buvo renkami kiti respondentai – komandos/skyriaus vadovai bei keletas automatinio testavimo ekspertų. Apibendrinta informacija apie atliktus interviu ir respondentus pateikiama 1 lentelėje:

1 lentelė
Apibendrinta informacija apie atliktus interviu ir respondentus

Vidutinė interviu trukmė	50min.
Apklaustųjų skaičius	15
Patirtis PĮ testavime	~ 9.5m.
Aukštasis išsilavinimas IT sferoje	3
Nėra aukštojo išsilavinimo IT sferoje	11
Nėra jokio aukštojo išsilavinimo	1
Rašo automatinius testus	15
Komandos/skyriaus vadovas	10
Automatinio testavimo ekspertas	5

Šaltinis: parengta autoriaus, remiantis atliktu tyrimu

Apklausa buvo vykdoma per „Microsoft Teams" platformą. Siekiant sukurti glaudesnį ryšį, buvo prašoma įjungti kamerą. Respondentai į klausimus galėjo atsakinėti remdamiesi bendra testavimo patirtimi, o ne konkrečia darboviete. Tokiu būdu buvo galima lygiagrečiai lyginti keleto įmonių testavimo procesus – išryškėjo silpnosios/stipriosios pusės.

Anonimiškumas – tiek interviu metu kalbintas asmuo, tiek pokalbyje minėtos įmonės nebus skelbiamos viešai, minimos magistro darbe. Dalis apklaustųjų nesutiko, kad pokalbis būtų įrašomas.

Siekiant padidinti informacijos patikimumą, visiems pašnekovams buvo stengiamasi išlaikyti tą pačią klausimų struktūrą bei formuluotes. Interviu buvo suskirstytas į keturias dalis:

1. Keletas bendrinių aspektų apie respondentą:
    a) Išsilavinimas
    b) Patirtis
    c) Užimama pozicija
    d) Patirtis su TPI/TMA

Šioje dalyje buvo siekiama susipažinti su respondentu, surinkti pagrindinę informaciją apie jo patirtį.

2. Testavimo procesas:
    a) Naudojamos metodologijos/praktikos/mąstymo būdai
    b) Komandos sudėtis
    c) Kokią dalį testavimo siekiama padengti automatiniais testais

Ši dalis padėjo susidaryti bendrą vaizdą apie testavimo/programinės įrangos procesus, su kuriais yra susidūręs respondentas.

3. Iššūkiai testavimo procese:
    a) Problemos, su kuriomis susiduriama
    b) Galimi problemų sprendimo būdai

Šiuo klausimu buvo siekiama surinkti pagrindinę informaciją tyrimui – problemos, lemiančios žemą testavimo proceso brandą. Lygiagrečiai buvo užduodamas klausimas, kaip jas išsprendė/sprendžia, o gal problemos jaučiamos, tačiau nieko su jomis nedaroma.

4. 5 kompetencijos, būtinos šiandieniniam testuotojui.

Testavimo proceso kokybei didelę įtaką gali daryti komandoje dirbantys testuotojai. Dėl to buvo siekiama išsiaiškinti, kokį testuotoją galima laikyti geru, kokios kompetencijos labiausiai pageidaujamos. Buvo analizuojama tiek minkštieji įgūdžiai (angl. *soft skills*) tiek techniniai sugebėjimai.

2.5. Atlikto tyrimo apžvalga

Nors ieškant respondentų, nebuvo reikalavimo, jog jie dirbtų pagal *agile* metodologiją, tačiau visi 15 apklaustųjų dirbo bent iš dalies pagal ją. Taip pat visi iš apklaustųjų teigė, jog stengiasi naudoti ir ateityje sieks įvesti vis daugiau DevOps praktikų bei didžiąją dali testų padengti automatiniais. Tai pagrindžia literatūros apžvalgoje sudarytą šiandienės PĮ kūrimo aplinkos vaizdą – *agile* metodologija, populiarėjančios DevOps praktikos bei svarbus automatinio testavimo vaidmuo.

Taip pat atlikus visus interviu bei susisteminus duomenis, išskirta 13 pagrindinių problemų, kurios gali lemti žemesnę testavimo proceso brandą. Kiekviena problema detaliai išanalizuota - pateikiami praktiniai apklaustųjų pavyzdžiai, kurie iliustruoja pačią problemą ar jos sprendimo būdus. Kai kurios problemos bei jų išvados persidengia vienos su kitomis. Taip yra dėl to, nes testavimas apjungia į vieną vietą visų komandos narių darbą, pats procesas nėra nepriklausomai izoliuotas.

Taip pat pagal svarbą išrikiuota pageidautinos testuotojo kompetencijos. Į jas vertėtų atsižvelgti, kai siekiama aukštesnės testavimo kokybės ir ieškoma naujo komandos nario ar stengiamasi ugdyti esamų testuotojų kompetenciją.

3. INTERVIU TYRIMO REZULTATŲ APIBENDRINIMAS

3.1. Išskirtų testavimo proceso problemų aibė

Toliau darbe pateikiama 13 problemų, kurios buvo išskirtos apklausus 15 testavimo ekspertų. Kiekviena iš jų detaliai išanalizuota apjungus visų respondentų mintis. Po kiekvienos problemos analizės yra išskirta po keletą išvadų – tai vertingos įžvalgos/teiginiai, kurie išryškėjo kaip svarbūs aspektai sisteminant pačią problemą. Kai kurios išvados gali būti susijusios su PĮ kūrimu, o ne pačiu testavimo procesu, tačiau įtakojančios produkto kokybę.

1. Nesuprasta testavimo svarba tiek komandos, tiek valdžios lygmenyje:

Siekiant kurti itin greitai, dažnai taupoma testavimo sąskaita. Programinės įrangos kūrimo ciklo laikas yra suspaudžiamas praleidžiant, arba dalinai praleidžiant testavimo fazes. Jei testai turi būti automatizuojami, tai spaudžiant laikui jie neparašomi, o geriausiu atveju atliekamas tik rankinis testavimas. Tai lemia, jog rankinio testavimo kiekis vis auga, vis labiau vėluojama. Daugumos komandų tikslas yra pasidengti automatiniais testais didžiąją dalį regresijos, o rankomis testuoti tik naują funkcionalumą. Tačiau kai yra laiko spaudimas, gaunasi atvirkščiai, mažėja automatizavimo, daugėja rankinio testavimo. Dažnai ši problema kyla dėl to, kad verslo žmonės reikalauja produkto čia ir dabar. Parengimo data (angl. *deadline*) būna sugalvota nepasitarus su kūrimo (angl. *development*) komanda, verslo analitikai „prisižada“ prieš tai nepasitarę su programuotojais apie technines galimybes.
Taip pat buvo minėta, kad produkto vadovas nesugeba suvaldyti kliento lūkesčių. Tai lemia dėl skubėjimo krentančią komandos moralę, didėja jų nepasitikėjimas savimi, žmonės negali laiku kokybiškai atlikti savo darbo.

Vienas respondentas apibendrino, jog šiai problemai spręsti, turi būti suformuotas komandinis suvokimas apie testavimo svarbą, jog ji yra bendra, ne atskira, viso proceso dalis ir negali būti išmesta. Taupymas testavimo kaina yra tiesiogiai susijęs su prasta produkto kokybe, o tuo pačiu ir įmonės įvaizdžiu. Toks supratimas turi būti formuojamas pradedant nuo valdžios. Pats produkto vadovas turi suprasti testavimo svarbą, prioritetai turi būti aiškiai iškomunikuoti visai komandai. Svarbu aiški vizija ir gera komunikacija.

Komunikaciją stipriai pabrėžė ir kitas apklaustasis - „Jei komandoje nėra suvokimo apie testavimo svarbą, tai blogos komunikacijos ženklas“ -, teigė jis. Vienas svarbiausių aspektų – testavimo vadovas. Kaip jis geba komunikuoti ir parodyti testavimo vertę. Kaip pristato atliktus darbus ir jų rezultatus. Svarbu, kad komanda suprastų ką veikia testuotojai, kaip atlieka savo darbą.

Tačiau ne visi apklaustieji susiduria su šia problema. Vienoje įmonėje yra stipriai nusistovėjusi kultūra, kad testuoja visi. Dėl to kai kurios komandos netgi neturi paskirtų testuotojų. Programuotojai rašo ne tik atominius (angl. *unit*) testus, bet ir integracinius bei nuo pradžios iki galo (angl. *end-to-end*). Testuoja ir vartotojo sąsajos dizaineriai. Testuotojo rolė suprantama labiau kaip konsultanto, kuris gali peržiūrėti komandos testavimo procesą, suteikti pagalbą, patarti. Įmonėje yra keletas nededikuotų (nepriskirtų jokiai komandai) testuotojų, kurie padeda tam, kam tuo metu reikia – susirgus dedikuotam kolegai arba prireikus testavimo pagalbos komandai, kuri neturi paskirto testuotojo. Prieš didesnius pakeitimus į produkcinę aplinką, būna atliekami defektų medžiojimai (angl. *bug hunting*), kai didesnis kiekis testuotojų bando rasti likusius defektus.

Deja, bet tai tik vieno apklaustojo patirtis. Iš 14 likusiųjų apklaustųjų atsakymų, galima daryti išvadą, jog toks darbo būdas galėtų būtų jų siekiamybė, bet, kolkas, stiprios testavimo kultūros jų įmonėse nėra.

Išvados:

1. Spartėjant programinės įrangos kūrimo greičiui, tikimasi daugiau automatinio ir mažiau rankinio testavimo. Deja, dažnai gaunasi atvirkščiai – rankinio testavimo kiekis vis auga, automatinio mažėja. Tas lemia vis didesnius rankiniu būdu testuojamus regresijos kiekius, vis labiau vėluojama ir gaunasi tarsi užburtas ratas.
2. Taupymas testavimo kaina koreliuoja su galutine produkto kokybe – kuo daugiau taupoma, tuo žemesnė produkto kokybė.
3. Komunikacija – svarbiausias aspektas siekiant komandinio suvokimo apie testavimo svarbą. Testavimo rezultatai turėtų būti pateikiami rodant konkrečią jo naudą. Tam galima pasitelkti analitinį testavimo rezultatų vertinimą.

2. Aiškiai apsibrėžtų procesų trūkumas:

Svarbu pabrėžti, kad ši problema taikoma didesnėms komandos ir yra itin aktuali, kai dirbama per nuotolį.

Vienas apklaustasis teigė, jog testuotojas yra paskutinis taškas, kuris turi suvesti atskirų žmonių atliktus darbus į bendrą visumą**.** T. y. patikrinti, kaip skirtingų žmonių atlikti darbai sąveikauja kartu. Ar nesikerta tarpusavyje logika, ar kiekvienas aspektas apgalvotas. Šioje fazėje atsiranda nemažai analitinio darbo, nes dažnu atveju „nesueina galai“. Apklaustasis dalinosi pavyzdžiu, kai vartotojo sąsajos dizaineris nuolatos darė atnaujinimus savo reikalavimuose, kuriuos pristatydavo programuotojui, o pastarasis tai įgyvendindavo. Šiam darbui nebuvo kuriamos vartotojo istorijos (angl. *user story*), kuriose būna aprašyti reikalavimai, nebuvo

pristatoma visai komandai. Kai testuotojas pradeda savo darbą, dažniausiai *agile* aplinkoje jo pagrindinis darbo įrankis yra atliktos vartotojo istorijos. Šiuo atveju, sistema veikė ne pagal specifikaciją. Pradėjus testuoti išaiškėjo, kad vartotojo sąsajos pakeitimai taipogi iššaukė ir tam tikrus logikos pakeitimus, už kuriuos atsakingas kitas programuotojas, tačiau šie aspektai nebuvo apgalvoti. Kol testuotojas atsekė , kur yra problema, buvo išeikvota nemažai laiko. Galiausiai ši situacija buvo išspręsta tiksliai apsibrėžus procesą, kaip turi būti pristatomi nauji vartotojo sąsajos pakeitimai – privaloma sukurta naują vartotojo istoriją, pristatyti ją susijusiems asmenims, aptarti ir tik tada įgyvendinti pokytį.

Aptartas pavyzdys iliustruoja, kaip dėl aiškiai neapsibrėžtų procesų kyla nesusikalbėjimas, kažkas praleidžiama, pamirštama. Taip pat keletas apklaustųjų minėjo, jog per anksti uždaromos vartotojo istorijos. Dažnu atveju tai atsitinka dėl neaiškaus išbaigtumo apibrėžimo (angl. *definition of done*), t.y. kokios veiklos privalo būti atliktos, jog būtų galima vartotojo pasakojimą uždaryti. O jei apibrėžimas visgi yra, jame nėra įtraukta testavimo veiklų - tiek rankinio, tiek automatinio. Vienas apklaustasis svarstė – „Jei būtų griežti apibrėžimai, tai galbūt sumažintų produkto kūrimo greitį, tačiau žiūrint į perspektyvą, pats produktas taptų tvaresnis, kokybiškesnis“.

Kitas paliestas aspektas - testavimo strategijos nebuvimas, kuris veda prie chaotiško darbo - trūksta apibrėžtų procesų kaip apskritai testavimas turi vykti. Pavyzdžiui, kada ir kokio tipo testavimas turi būti atliktas, kaip jį reikėtų atlikti. Kas atliekama rankiniu būdu, o ką stengiasi padengti automatiniais testais. Ši problema itin pastebima, kai ateina naujas testuotojas. Jam sunku įsilieti, sunku suprasti, kas vyksta projekte. Nereikėtų rašyti didelių dokumentacijų su itin smulkiomis ir griežtomis taisyklėmis, tačiau bendrinės gairės turėtų būti aprašytos ir patalpintos visiems pasiekiamoje vietoje.

Visgi, vieno apklaustojo įmonėje šią problemą sprendžia kitaip. Jie neturi jokių dokumentacijų, netgi vartotojo istorijų. Yra nusistovėjusi taisyklė, jog visa komanda dalyvauja visuose susitikimuose. Todėl supratimas apie tai, kaip ir kas turi būti kuriama yra pas visus komandos atstovus. Turimas tik finalinio dizaino dokumentas, kuriuo dažniausiai remiamasi. Anot apklaustojo, esmė yra bendras kolegų suvokimas, atsakomybė ir daug komunikacijos.

Itin svarbu turėti vieną aiškiai apsibrėžtą informacijos vietą (angl. *single source of truth*). Kur pažiūrėjus, iškart galima suvokti sprendžiamą problemą, ką reikia padaryti ir koks turi būti rezultatas. Tokiu atveju testuotojas gali iškart imti ir testuoti, o ne dar keletą dienų rinkti duomenis ir informaciją, kaip turi atrodyti galutinis variantas. Apklaustasis pasakojo, kad verslo žmonės nusprendė dokumentuoti reikalavimus „Confluence“ aplikacijoje (ji skirta organizuoti bei komunikuoti projekto klausimais) bei „Jira“ įrankyje (jis skirtas projekto valdymui). Atsiradus pakeitimams, atnaujinimai turėdavo būti daromi keliose vietose. Tačiau dažnu atveju vienoj vietoj atnaujindavo, kitoj pamiršdavo.

Išvados:

1. Siekiant aukštesnės testavimo kokybės, neišvengiamai gali tekti keisti ir kitus programinės įrangos kūrimo procesus.
2. Aiškiai nustatyti konkretūs proceso apibrėžimai gali sulėtinti patį produkto kūrimą, tačiau žiūrint į perspektyvą, jis bus tvaresnis ir kokybiškesnis.
3. Į testavimą turi būti atsižvelgta jau projekto planavimo stadijoje – nustatyta ir gerai iškomunikuota aiški testavimo strategija.
4. Jei nuspręsta turėti minimaliai dokumentacijos, komunikacija turi būti itin aukšto lygio. Visi komandos nariai privalo dalyvauti visuose susitikimuose.
5. Turi būti apibrėžta konkreti viena vieta, kur kaupiama informacija apie sprendžiamas problemas.

3. Testų dokumentacija bei jų atnaujinimas atima daug laiko:

Keletas žmonių šią problemą išsprendė apskritai atsisakydami detalių testų rašymo tekstu, tačiau su sąlyga, jog didžioji dalis regresijos yra padengta automatiniais testais. Jie turi vieną vietą (pavyzdžiui - „Confluence“ puslapyje), kur yra aprašytas visas testų ūkis aukštame lygmenyje. Pvz.: nurodyta, kur padėti posistemės (angl. *backend*) testai, koks jų padengimas automatiniais testais, yra nuoroda kur patalpintas kodas, nuoroda, kur leidžiami testai kaip dalis nuolatinio integravimo. Visi automatiniai testai parašyti suprantama kodo kalba, kuriuos perskaičius aišku, kas jais testuojama.

„Geriausias testų aprašymas yra su *gherkin* sintakse“ -, teigė keletas apklaustųjų. „Gherkin“ sintaksė padeda apsirašyti testus pagal elgsenos dizainą (angl. BDD – *behavior driven design*) – struktūruotai bei aiškiai. Vienu metu yra kuriama dokumentacija, kuri, tuo pačiu metu, testuoja sistemos elgseną. Beje, tie testai gali būti pateikti verslo žmonėms, kurie dažniausiai neturi techninių žinių, tačiau gebės lengvai perskaityti ir suprasti testų esmę.

Rašant/dokumentuojant bet kokį testą, visada reikia atsižvelgti į jo universalumą, pritaikomumą. Svarbu nepamiršti, kad jis gali būti keičiamas dar ne vieną kartą. „Rašant bet kokį testą, visada reikia turėti mintyse DRY principą – *don't repeat yourself*“ -, teigė vienas apklaustas. Tai reiškia, kad informacijos rašoma tik tiek, kiek tikrai reikia. Ji nėra kartojama ar dubliuojama skirtingose vietose. Jei tie patys/panašūs testai bus keliose vietose, atsiradus pakeitimams, didelė tikimybė, kad bus atnaujinama tik vienoje vietoje, nes kita – pasimirš. Kils klausimas – kur yra tiesa, kaip turi veikti sistema?

Anot *agile* metodologijos, testų scenarijai yra būtini, tačiau lyginant su krioklio metodologija, jie turi būti daug *lengvesni*. Nereikia aprašyti visko, reikia dokumentuoti tik svarbias

esmines vietas. Dėl to itin naudinga naudoti „gherkin“ sintaksę – tas pats tekstas yra dokumentacija bei kodas, testuojantis sistemą.

Išvados:

1. Turi būti aiškiai apibrėžta vieta, kur pažiūrėjus galima matyti bendrą visų testų aprašų vaizdą.
2. Jei dauguma testų automatizuojama, galima atsisakyti testų dokumentacijos tekstu, tačiau patartina naudoti „gherkin“ sintaksę.
3. Reikia vadovautis DRY principu. Kuo mažiau dubliuojama informacija, tuo mažiau reikia daryti pakeitimų, pakitus reikalavimams.

4. Nuolat kintantys reikalavimai:

Šią problemą minėjo daugiau nei pusė apklaustųjų. Ji dažniausiai kyla tuomet, kai klientai ar verslo analitikai turi mažai patirties/kompetencijos arba skiria per mažą dėmesį reikalavimų išgryninimui – kažką pamiršta, neįvertina, neapgalvoja. Taip pat vienas iš veiksnių, kuris gali lemti nuolat kintančius reikalavimus, yra neišbaigti tobulinimo (angl. *refinement*) susitikimai. Dažnai juose neskatinama diskusija, o siekiama greičiau pristatyti naują funkcionalumą, praeiti tik paviršių. Kad ir kokia priežastis bebūtų, reikalavimai yra pagrindas kuriamai sistemai. Todėl privalu suprasti, jog nekokybiški reikalavimai prilygsta *Pandoros* skrynios atvėrimui pačioje projekto pradžioje. Vienas neišdirbtas reikalavimas reikalaus pakeitimo, tada reikės keisti nuo jo priklausantį kitą funkcionalumą ir gausis kaip užburtas ratas.

Prasti reikalavimai atneša daugybę problemų - išauga kaina dėl nuolat kintančio kodo, tiek pagrindinio, tiek automatinių testų. Darbas pradeda priminti *ad-hoc* procesą. Kadangi dauguma dirba pagal *agile* metodologijoje naudojamas iteracijas (darbas planuojamas kas tam tikrą apibrėžtą laikotarpį), per dažnas reikalavimų keitimas neleidžia pasiekti užsibrėžtų tikslų, nes tai, ką suplanavo daryti iteracijos pradžioje, viduryje jau gali būti radikaliai pasikeitę. *Agile* metodologija būtent ir iškilo kaip priešprieša krioklio metodologijai, kuri nesugebėjo valdyti nuolat kintančių reikalavimų. *Agile* aplinkoje pakeitimus valdyti lengviau, tačiau ir čia itin didelis nepastovumas kelia daug sunkumų. Pavyzdžiui, kai iteracijos laikas yra savaitė, o per ją reikalavimai pasikeičia pora kartų.

Anot vieno apklaustojo, jų projektas yra labai sudėtingas, turi daug priklausomybių. Dėl to itin sunku planuoti ir įgyvendinti MVP (angl. *minimal viable product*) – anot *agile* metodologijos, tai minimali produkto dalis, kurią galima ištestuoti ir pristatyti klientui. Jiems sunku padaryti net vieną nedidelį funkcionalumą nuo pradžios iki galo, kuriam galėtų būti atliktas pilnas testavimas ir demonstracija užsakovams. Apklaustasis tai mato kaip didelę riziką, nes nėra

gaunamas nuolatinis atsiliepimas iš kliento pusės. Jų MVP yra itin dideli, todėl klientai gali išsakyti savo nuomonę ir prašyti pakeitimų tik tada, kai didelės susijusio dalys jau yra sukurtos, ir bet koks pakeitimas reikalauja itin daug darbo.

Apklaustasis, kuris nekėlė šios problemos, teigė, kad turi labai stiprų planavimą tiek iš įmonės tiek iš kliento pusės. Visada yra pateikiami aiškūs, gerai išdirbti planai (angl. *roadmaps*) ir konkrečiai nustatyti artimiausi tikslai. Svarbiausias akcentas – geras planavimas.

Šis nuolatinis reikalavimų keitimas prideda labai daug darbo testuojant sistemą. Dažnai keičiami reikalavimai pasimeta, pamirštama juo užrašyti. Dėl to testuojant praleidžiama nemažai laiko siekiant suprasti, kas iš tiesų yra reikalaujama. Taipogi tai liečia ir automatinius testus. Juos patartina rašyti, kai pagrindinis kodas jau yra nusistovėjęs. Esant dažnai reikalavimų kaitai, neverta rašyti automatinių testų, nes daug laiko bus sueikvota juos keičiant. Visgi, jei jie rašomi, jų kaina dėl eikvojamo laiko nuolatiniams pakeitimas, stipriai išauga.

Išvados:

1. Testavimo kokybė stipriai priklauso nuo kitų žmonių darbo kokybės. Jei visa ko pamatai – reikalavimai, nėra gerai išdirbti, užtikrinti kokybę bus sunku.
2. Svarbu ne ignoruoti nuolat kintančių reikalavimų problemas, o ieškoti įvairių sprendimo būdų, metodų.
3. Simptomus gydyti reikia (stengtis atlaikyti kintančius reikalavimus), tačiau dar svarbiau sutvarkyti pačią problemą. Komunikuoti ją verslo žmonėms, pabrėžti kylančias rizikas, kartu ieškoti sprendimo būdų. Suprasdami pasėkmes, verslo žmonės yra linkę tobulinti savo procesus.

5. Didelis kiekis integracijų:

Tai rimta problema daugumai. Siekiant vis plačiau naudoti DevOps praktikas bei automatinį testavimą, tenka rinktis, ar trečiųjų šalių integracijas imituoti (angl. *mock*) ir turėti ramesnį gyvenimą su stabilesniais testais, ar naudoti realias integracijas, tačiau dirbti su nestabiliais testais, nuolat aiškintis, kodėl testai griūna, bandyti atskirti, ar klaida iš mūsų pusės ar visgi ne. Pasirinkus imitavimą, negalima visiškai pasitikėti savo kuriamu produktu, atsiranda abejonės, ar tikrai viskas gerai veiks su realiomis integracijomis. Optimalus pasirinkimas priklauso nuo aplinkybių kiekvienu konkrečiu atveju.

Apklaustasis teigė, jog šią problemą bando spręsti pagal vieną iš DevOps praktikų leidžiant testus keletą kartų po naujo kodo atsiradimo – pirmą kartą su imituojamais duomenis, antrą kartą su tikrais. Jei testai su imituotais duomenimis praeina be klaidų, daroma prielaida, kad naujas

funkcionalumas nieko nesugriovė. Jei testai nepraeina tik su tikromis integracijos, ieškoma problemos su mintimi, jog defektas ne pas mus.

Dar vienas apklaustasis teigė, jog testų leidimas su realiomis integracijomis užima daug laiko. Šią problemą išsprendė padarę automatinį testų leidimą ant realių integracijų naktį, o ne po kiekvieno naujo kodo atsiradimo.

Nusprendus imituoti integracijas, atsiranda papildomas kompleksiškumas – sudėtingesnis kodas, didesnis jo kiekis. Svarbu suprasti, kad šiuo atveju turime iš esmės tą patį kodą tik dvejose skirtingose vietose – produkciniame kode ir testuose. Todėl atsiradus pakeitimams, kodas turės būti keičiamas dvejose vietose.

„Jei integracijos yra vidinės, mes jas sekame *Dynatrace* arba *Datadog* sukurtame skydelyje, kuris rodo, ar susiję servisai veikia“ - , teigė vienas apklaustųjų. Minėtos aplikacijos yra skirtos stebėti bei optimizuotu IT infrastruktūrą, kuri remiasi dirbtiniu intelektu. Jei sekamas servisas dėl neaiškios priežasties nustoja veikti, galima sukonfigūruoti, kad iš karto ateitų pranešimas į el. paštą reikiamiems asmenims.

Kai kurios įmonės integracijų keliamas problemas stengiasi spręsti itin gera komunikacija. Kiekviena integracija turi savo konkretų atstovą, į kurį galima kreiptis iškilus klausimams. Jei naudojama vidinė integracija, atstovai stengiasi suderinti integracijas bei planus. Jei integracija išorinė ir turi rimtų defektų, kartais priimamas sprendimas patiems sutvarkyti kodą – tokiems darbams kas ketvirtį yra skiriama savaitė laiko.

Išvados:

1. Jei integracijos yra vidinės, pravartu sekti servisų „sveikatą“ (anlg. *service health*) – tai skydelis, kuriame sekami visi susiję servisai ir jei nors vienas jų nustoja veikti, atitinkami asmenys yra informuojami. „Dynatrace“ ir „DataDog“ yra rinkoje populiariausi įrankiai, leidžiantys kurti skydelius bei sekti įvairias aplikacijos metrikas. Tai užtikrina saugumą ir leidžia integracijas imituoti bei testuoti tik savo kodą.
2. Jei integracijos yra išorinės, tą suvaldyti gali būti sunkiau. Patartina testuoti ant imituojamų integracijų, tačiau kartas nuo karto leisti testus ir su tikromis integracijomis – pasižiūrėti, ar tikrai nebuvo jokių pakeitimų, ar nereikia atnaujinti testų.
3. Ar tai būtų vidinės ar išorinės integracijos – reikia stengtis nustatyti aiškų komunikavimą procesą, kuriuo bus skelbiama pakeitimai ar sprendžiamos atsiradusios problemos.

6. Prastas ryšys su klientu:

Apklaustasis teigė, jog ištestavus visą programą ir perdavus klientui ateidavo blogas grįžtamasis ryšys - kažkas neveikia arba veikia ne taip, kaip turėtų. Pradėjus aiškintis ateidavo suvokimas, jog nėra aišku kaip klientas naudojasi sistema, ko jis tikisi. Šią problemą gerai išsprendė tiesioginis testuotojo kontaktas su klientu, dažniausiai elektroniniu paštu. Itin padeda testavimas naudojant konfigūraciją, kuri yra identiška kliento aplinkai, suvokimas, ką konkrečiai klientas daro, ką bando pasiekti. Pažvelgus iš PĮ kūrimo proceso pusės, šiuo atveju testuotojas prisiėmė dalį verslo analitiko arba produkto vadovo darbo. Tačiau, jei toks procesas veikia šioje komandoje, ar reikėtų jį keisti vien tik dėl to, kad jis neatitinka standartų?

Dar vienas respondentas paantrino, kad būtent testuotojas privalo geriausiai suvokti verslo sritį. Jis turi būti lyg konsultantas programuotojams, turint klausimų apie patį produktą. Dažnai programuotojai dirba prie savo siauros dalies ir jiems sunkiau pamatyti bendrą sistemos vaizdą. O testuotojai turi įsitikinti, kad visos atskiros dalys veikia tvarkingai kaip viena bendra visuma. Dėl to jiems privalu matyti produktą iš aukščiau, suprasti, kaip integruojasi atskiros dalys.

Vienas iš apklaustų testuotojų teigė, jog nuolat kintančius reikalavimus bandė spręsti taikant įvairias praktikas/metodikas - pavyzdžiui vietoj *scrum* praktikos naudojo *kanban* (*scrum* bei *kanban* yra vienos populiariausių *agile* praktikų). Tačiau tai problemos išspręsti nepadėjo. Kiti teigė, jog stipriai komunikuoja tobulinimo susitikimo svarbą ir matant poreikį, jį kartoja dar kartą iteracijos viduryje. Treti teigė, jog šitą problemą spręsti padeda itin detalūs šablonai, kaip turi atrodyti vartotojo pasakojimas. Vienas respondentas minėjo, kad viską bando vaizduoti ant vartotojo sąsajos prototipų, kartu pridedant papildomą reikiamą informaciją (posistemio reikalavimus, diagramas). „Vizualiai vaizduojant reikalavimus pasidaro aiškesnė veiksmų seka, lengviau matosi, kur yra loginės klaidos“ -, teigė respondentas.

Kitas apklaustasis teigė, jog šią problemą bandė spręsti įtraukiant klientą į rytinius susitikimas. Deja, šis variantas pasirodė netinkamas, nes klientas turi per mažai techninių žinių, jog suprastų visą programinės įrangos kūrimo ciklą. Dėl to, išgirdęs, jog kažkas neveikia, iškart pradėdavo panikuoti.

Komanda, dirbanti pagal *agile* metodologiją ir turinti dviejų savaičių sprintus, prieš kiekvieną naujo sprinto planavimą daro *demo* sesijas klientams. Per jas užsakovui yra parodoma dažnu atveju tik dalinai veikianti versija, bet susirenkami komentarai, norai ir į tai atsižvelgiama planuojant tolimesnį darbą.

Išvados:

1. Kliento norų patenkinimas yra svarbiausias tikslas kuriant bet kokią sistemą. Svarbu surasti tinkamiausią bendravimo būdą abejoms šalims.

2. Vienas iš variantų – glaudesnis testuotojo ir kliento ryšys. Tokiu atveju testuojant lengviau įsijausti į pačio kliento norus – pagal tai fiksuojami atitinkami defektai, teikiami pagerinimo pasiūlymai.
3. Norint nenukrypti nuo kliento norų, itin svarbu demonstracijos. Kuo dažniau bus rodoma, kas iki šiol padaryta, tuo lengviau bus derintis prie kliento lūkesčių ir pakeitimų kainos nebus didelės. Geriausiu atveju reikia duoti klientui *pačiupinėti* sistemą pačiam, tačiau jei sistema veikia tik dalinai, privalu parodyti bent demonstracines sesijas, kas jau yra padaryta.
4. MVP – vienas esminių *agile* bruožų, kuris gali itin gerai pasiteisinti, siekiant turėti nuolatinius kliento atsiliepimus. Tačiau tam įgyvendinti, itin svarbu geras planavimas ir verslo pusės.

7. Skiriamas mažas finansavimas testavimui:

Kai kurių įmonių politika yra užsiauginti testuotoją pačiam, o ne iškart ieškoti aukštos kvalifikacijos žmogaus, dėl ko paprastai nukenčia testavimo proceso kokybė. Samdomi žmonės dažnai būna praktikantai, neturintys, arba turintys tik nedidelę patirtį testavime. Piniginiu atžvilgiu į žmones investuojama minimaliai, tačiau jie yra apmokomi. Tikimasi, kad nauji žmonės pasiliks ilgesniam laikui. Tai pasiteisina iš dalies, nes nemažai testuotojų, pajutusių turintys pakankamą kiekį patirties, susiranda darbovietę, kur gali gauti daugiau patirties bei didesnį atlygį. Vėl tenka ieškoti darbuotojo, jį apmokyti. Taip pat atsiradusi didelė darbuotojų kaita mažina pasitikėjimą darbdaviu, nelieka saugumo jausmo.

Šią problemą iš dalies bandoma spręsti padengiant internetinių kursų išlaidas. Stengiamasi, kad būtų bent vienas aukštesnio lygio testuotojas, kuris atlieka mentoriaus veiklas.

Taip pat stengiamasi turėti itin gerą dokumentaciją, kad atėjus naujam žmogui nereikėtų praleisti daug laiko supažindinant su įmonės procesais. Gerosiomis praktikomis stengiasi dalintis per specializuotus kanalus socialinėse platformose.

Kitas apklaustasis teigė, kad pas juos yra kiek kitokia situacija. Įmonė mažiau rūpinasi esamais testuotojais, bet daug investuoja į naujus. Nesvarbu kaip gerai esamas darbuotojas dirba, kokius gerus rezultatus rodo – pasikelti algą atitinkamai bus labai sunku. Bet jei į įmonę ateis panašaus lygio testuotojas, prašantis didesnės algos negu esami jo lygio darbuotojai uždirba, jis lengvai ją gaus. „Dėl to vyrauja nuostata – jei jauti, kad pakankamai subrendai didesnei algai gauti, metas keisti įmonę“ -, teigė apklaustasis. Anot jo, neaišku, ar čia yra įmonės taktika, ar tai taikoma vien testuotojams, ar tiesiog susiformavęs mąstymas, kad naujas darbuotojas vertesnis didesnės algos. Tačiau tai veda link vieno – didesnės testuotojų kaitos įmonėje.

Išvados:

1. Didelė testuotojų kaita atneša chaosą į projektą ir gali lemti žemesnę testavimo kokybę.
2. Jei kaita neišvengiama, patartina turėti gerą įtraukimo (anlg. *onboarding*) procesą, kad naujas testuotojas kuo greičiau įsilietų į darbą.
3. Vertėtų peržiūrėti įmonės strategiją darbinant testuotojus. Aiškiai sudėlioti pliusus minus, apskaičiuoti naudą. Gali būti, jog taupant ant testuotojų algų įmonė nukenčia kitose vietose – klientų pasitenkinimas mažėja, krenta įmonės įvaizdis.
4. Reikėtų stengtis išlaikyti esamus testuotojus – prarandamos ne tik darbo rankos, bet ir sukauptas žinių bagažas. Šiuo atžvilgiu naujas darbuotojas kainuoja daugiau – praeina nemažai laiko, kol jis įsivažiuoja kaip pilnavertis darbuotojas.

8. Trūksta geras technines žinias turinčių testuotojų:

Atsižvelgiant į apklaustųjų atsakymus, šią problemą galima skirstyti į keletą dalių:

1. Geras automatinių testų rašymo specialistas kainuoja palyginus brangiai.
2. Gerų automatinių testų specialistų tiesiog trūksta, nes pasiekus aukštesnį lygį, jie dažnai pasirenka programuotojo kelią.
3. Vyrauja supratimas, kad vienas lengviausių būdų patekti į IT sferą nebaigus atitinkamų mokslų yra būtent testavimas.

Dėl paskutinės priežasties dauguma testuotojų neturi atitinkamo išsilavinimo, techninių žinių pagrindo. Anot kelių apklaustųjų, jie dažniausiai geba atlikti juodos dėžės (angl. *black box*) tipo testavimą, kai testuojama tik paviršius, tačiau baltos dėžės (angl. *white box*) tipo, kas reikalauja nemažai techninių žinių, kodo skaitymo ir kita, jiems atlikti sunkiau. Tačiau ties šiuo klausimu apklaustųjų testuotojų nuomonės išsiskyrė. Vieni teigė, kad techninis išsilavinimas yra būtinas, siekiant aukštos testavimo proceso brandos. Kiti, jog visų pirma, būtina motyvacija ir noras mokytis, ko pasekoje galima visko išmokti. Taip pat buvo minėta, jog žmogus, atėjęs iš kitos darbo sferos, į projektą atsineša naują požiūrį, jam lengviau atsitraukti nuo projekto kūrimo ir pažvelgti kliento akimis. Taip pat šią pusę gali palaikyti ir interviu statistika – tik penktadalis apklaustųjų turi techninį išsilavinimą, tačiau iš likusiųjų apie pusė apklaustųjų užima vadovaujančias pozicijas – skyriaus ar komandos vadovai.

Kai kurios įmonės netechninių testuotojų problemą bando spręsti organizuojant vidinius kursus, kai aukšto lygio techniniai testuotojais apmoko rankiniu būdu testuojančius žmones. Kitos organizacijos stengiasi išlaikyti itin konkurencingą atlyginimą su mintim, jog geras automatinių testų specialistas gali pakeisti keletą rankiniu būdu testuojančių žmonių. Taip pat buvo minėta, kad stengiamasi daugiau investuoti į gerą darbuotojų paieškos sistemą. Jei sunku rasti gerą

automatizavimo specialistą, stengiamasi kruopščiai atrinkti testuoją, rašantį testus tik rankiniu būdu, bet su perspektyva, jog automatizavimą jis išmoks.

Vienas iš vadovų, turintis nemažą interviu patirtį renkantis į komandą naujus testuotojus teigė, jog į diplomą visiškai nekreipiamas dėmesys. Dažnai pasitaiko techninį išsilavinimą turinčių asmenų, kurie itin blogai pasirodo programavimo užduotyse ir atvirkščiai – nesusijusį išsilavinimą turintys žmonės pasirodo puikiai. Kas yra svarbu, tai turimos žinios. O kaip jos įgytos, čia jau kiekvieno asmeninis reikalas.

Išvados:

1. Rinkoje yra trūkumas stiprių testuotojų, gebančių rašyti automatinius testus. Norint juos pritraukti, pravartu turėti stiprius įdarbinimo specialistus bei išlaikyti konkurencingą atlygį.
2. Viena iš priežasčių, kodėl trūksta stiprių techninių testuotojų – pasiekus tam tikrą ribą, testuotojai persikvalifikuoja į programuotojus.
3. Siekiant užsiauginti, apmokyti žmones, svarbu turėti gerą atranką, kurios metu būtų įvertinama žmogaus asmenybė, analitiniai įgūdžiai bei motyvacija. Tokį žmogų galima laikyti perspektyviu ir imliu tobulėti siekiant techninių žinių.
4. Aukštasis išsilavinimas IT srityje yra pliusas. Tačiau esmė – ką žmogus iš tikrųjų geba. Dėl to neturėjimas diplomo neturėtų būtų kaip ženklas iškart atmesti testuotojo kandidatūrą.

9. Prastas projekto darbų valdymas

„Šios problemos buvimas gali būti silpno produkto vadovo ženklas“ -, teigė vienas apklaustasis. Produkto vadovas privalo matyti bendrą projekto vaizdą iš aukščiau. Jei jis mato, kad sunku valdyti projekto darbus, turėtų imtis iniciatyvos spręsti šią problemą. Galbūt neaiškūs procesai, kaip sukurtas reikalavimas virsta į realiai sukurtą produkto dalį. Tokiu atveju reikėtų išdiskutuoti ir konkrečiai apibrėžti procesus, priskiriant atsakingus žmones. Jei vis tiek nepavyksta išspręsti šios problemos, produkto vadovas turėtų pats apsiimti šių darbų, arba, esant galimybei, paskirti žmogų, atsakingą už komandos vedlio rolę.

Dauguma apklaustųjų minėjo, jog darbe naudoja „Jira“ programinę įrangą, kuri padeda valdyti *agile* projektą, jo procesus. Tačiau pusė pridėjo, jog ten didžiulė netvarka – nėra aiškiai apsibrėžtų procesų, netvarkingai sudėliotos užduotys, prastai aprašytos vartotojo istorijos, nesudėlioti ryšiai. Šioje vietoje svarbu paminėti rastą koreliaciją tarp netvarkos projekto valdymo sistemoje ir komandos vedlio (angl. *scrum master*) nebuvimo –visi apklaustieji, paminėję šią problemą kaip itin svarbią, neturi paskyrę konkretaus asmens, kuris turėtų valdyti šiuos procesus.

Problemą iš dalies bandoma spręsti perduodant dalį vedlio veiklų kitam komandos nariui – dažniausiu atveju tai būna testuotojas. Tačiau tai gali atnešti kitas problemas - vedlio rolė yra atskira profesija, kuri reikalauja daug žinių ir įgūdžių, todėl atsiradusios papildomas veiklos testuotojui atima nemažai laiko, kuris eikvojamas pagrindinės veiklos sąskaita. Vienas apklaustasis pabrėžė, kad toks sprendimo būdas gali tikti mažai komandai, tačiau didesnei – pravartu apsvarstyti atskira komandos vedlio rolę.

Kitas apklaustasis įsitikinęs, jog nuo tvarkos projekto valdymo sistemoje, stipriai priklauso ir tvarka visame projekte, kas lemia ir pačio produkto kokybę. Komandos vedlys bendrauja tiek su verslo žmonėmis, tiek su kūrimo komanda. Jis visada turi gebėti matyti bendrą viso projekto vaizdą, padėti komandos nariam suprasti tiek verslo lūkesčius tiek techninius pajėgumus. Tai įveda atsiskaitomumo kultūrą ir padeda komandai pasiekti užsibrėžtus tikslus laiko atžvilgiu.

Išvados:

1. Netvarka projekto darbų valdyme – ženklas, kad reikia imtis veiksmų.
2. Ši problema yra produkto vadovo atsakomybė. Jis turėtų rasti veiksnius, įtakojančius šią bėda ir sudaryti planą, kaip spręsti problemas.
3. Šią spragą dažnai užpildo vedlio rolė– tai gali būti visiškai atskiras žmogus arba papildomos atsakomybės jau esamam komandos nariui.

10. **Nesilaikoma gerųjų programavimo praktikų**

Tai itin pastebima iš kodo kokybės pusės. Programuotojai atiduoda netestuojamą kodą testavimui, su itin daug defektų. Tas lemia užtemptą ciklą - testuotojas švaisto laiką bandydamas rasti defektus, bet galiausiai kodas vis tiek gražinamas programuotojui, jis bando ištaisyti klaidas, vėl gerai nepratestuoja ir grąžina kodą atgal. Ciklas prasideda iš naujo.

Keletas apklaustųjų teigė, kad kodo kokybė stipriai skiriasi tarp komandų, kuriose yra testuotojų ir kuriose nėra. Tai gali lemti tai, jog programuotojas net nesistengia minimaliai prasitestuoti savo kodo, nes tai laiko tik testuotojo atsakomybe. Toks požiūris apie kokybę veda prie laiko švaistymo ir kitų kokybės problemų.

Vieni respondentai šiai problemai spręsti naudoja privalomą kodo patikrinimą. Tai galioja ne tik programuotojų naujam kodui, bet ir testuotojų kuriamiems automatiniams testams. Dažniausiai privaloma gauti bent dviejų programuotojų sutikimus prieš įdedant naują kodą. Taip pat naudojami įrankiai tokie kaip „Sonarqube“, kurie seka kodo padengimą testais ir kitas metrikas. Yra tam tikros taisyklės, kurių neatitikus negalima dėti naujo kodo.

Taip pat šią problemą padeda spręsti viena iš DevOps praktikų - testai, esantys automatinėse procesų struktūrose (angl. *pipelines*). Kai programuotojas įdeda naują kodą, pakeitimai perduodami testuotojui tik tada, kai sėkmingai praeina visos veiklos bei testai. Kokio tipo ir koks testų kiekis turi būti leidžiami po kiekvieno naujo kodo įdėjimo, yra komandinis susitarimas.

Keletas apklaustųjų minėjo, kad pas juos naudojama TDD (angl. *Test Driven Design*). Tai reiškia, jog programuotojai pirma pasirašo automatinį testą ir tik tada jam rašo kodą. Tam, kad prirastum prie šio proceso, reikia šiek tiek laiko. Pradžioje programuotojai nebūna itin laimingi dėl šios technikos, tačiau laikui bėgant pripranta, ir tai stipriai atsiliepia bendrai produkto kokybei.

Išvados:

1. Prasta kodo kokybė itin apsunkina testuotojų darbą ir prailgina PĮ kūrimo procesą.
2. Tiek programuotojo tiek testuotojo kodas privalo būti patikrintas kito žmogaus pieš keliant atnaujinimus į bendrą kodo vietą.
3. Privalu dar projekto planavimo stadijoje susitarti, kokie testai bus leidžiami automatiškai po kiekvieno naujo kodo įdėjimo.
4. TDD – pasiteisinęs būdas, kuris programuotojams padeda nepamiršti testavimo svarbos. Prieš rašant kodą, iš karto pagalvojama apie jo testavimo scenarijus.

11. **Automatinių testų infrastruktūros nebuvimas**:

Visi 15 apklaustųjų teigė, kad savo darbe siekia naudoti daugiau DevOps praktikų. Tačiau vienas testuotojas teigė, kad jiems išgauti maksimalią naudą iš testavimo trukdo geros infrastruktūros nebuvimas, kaip, pavyzdžiui, nėra sukurta aiškios automatinių procesų struktūros, kuri turi vykti po kiekvieno naujo kodo įdėjimo. Nesukonfigūruota, kad testai galėtų pasileisti automatiškai, kaip nuolatinio integravimo dalis. Testuotojai galėtų konfigūracijas padaryti/pataisyti patys, tačiau tai reikalauja nemažai žinių. Dažniausiai tam būna skirti DevOps inžinieriai, tačiau testavimo užduotys nėra jų prioritetas.

Kitas apklaustasis teigė, kad susiję su konfigūracija klausimai turi būti išspręsti dar prieš pradedant naują projektą ir atliekami reikalingi pasiruošimo darbai. Jis susidūrė su problema, kai viduryje projekto buvo nuspręsta į automatinių procesų struktūrą įdėti automatinius testus, tačiau dėl didelės projekto apimties ir kitų priklausomybių jų galimybės buvo itin ribotos. Iš dalies šią problemą lėmė kita problema, jog testuotojas buvo įtrauktas į projektą ne nuo pradžių, o jau stipriai įsivažiavus darbams. Dėl to buvo praleisti testavimo planavimo ir pasiruošimo darbai pačioje projekto pradžioje.

Išvados:

1. Testuotojas privalo būti įtrauktas į projektą nuo planavimo fazės.
2. Planavimo fazėje turi būti sudaryta aiški testavimo strategija – kokio tipo testai reikalingi, kur jie bus leidžiami, kokiu dažnumu.
3. Jei projekte bus vadovaujamasi nuolatinio integravimo principais – dar planavimo fazėje turi būti sukonfigūruota galimybė leisti ten įvairaus tipo testus.
4. Nuolatinio integravimo darbus gali daryti komandoje esantys programuotojai. Tačiau ne visada jie turi reikiamų DevOps inžinierių žinių. Tokiu atveju reikia užtikrinti ir šį resursą.

12. Testavimo proceso problemos ignoruojamos:

Tik keletas apklaustųjų yra apskritai girdėję apie testavimo proceso įsivertinimo bei gerinimo modelius. Iš apklaustųjų, tik dvejose įmonėje buvo užsiimama šiomis veiklomis. Vykdantys pagerinimo veiklas respondentai teigė, jog tai privertė pažvelgti į testavimo procesą iš kitos perspektyvos ir susisteminti visas matomas problemas. Anksčiau problemos buvo jaučiamos, tačiau niekas to garsiai neišreikšdavo. Respondentai pastebėjo, kad atlikus įsivertinimą, rastos problemos kartojasi ne tik jų komandose, bet ir kitose, dėl ko galima prieiti prie bendrų sprendimo būdų.

Vienas respondentas teigė, jog netgi šis interviu privertė kruopščiai išanalizuoti esamą testavimo procesą ir padėjo rasti problemas, apie kurias nebuvo susimąstęs anksčiau. „Įsivažiavus į rutiną, labai gerai kartas nuo karto atitolti ir pažvelgti į visą procesą iš aukščiau“ -, teigė apklaustasis.

Respondentas, turintis daugiau nei 10 metų patirties testavimo srityje, yra įsitikinęs, kad kiekvienas save gerbiantis vadovas ar aukšto lygio testavimo specialistas privalo domėtis proceso gerinimo būdais. Esant galimybei, pritaikyti juos savo komandoje/įmonėje, nuolat ieškoti spragų ir būdų, kaip jas pašalinti.

„Mano tikslas testavimą sudėti į materiją, kad jis būtų apskaičiuojamas, išmatuojamas bei nuolat gerinamas“ -, teigė vienas apklaustasis. Anot jo, tai padeda keliais atvejais:

1. Siekiant parodyti testavimo teikiamą naudą - ją sunku įvilkti į skaičius. Dėl šios priežasties, jis dažnai yra nuvertinamas, nes nesukuria konkretaus apčiuopiamo rezultato, kaip, pavyzdžiui, programavimas.
2. Analizuojant patį procesą, identifikuojant jo spragas (angl. *bottle neck*s). Apklaustasis komandoje yra sukūręs vidinį įrankį, kuris rikiuoja gautas užduotis pagal jų atsiradimo laiką. Matuojama, kiek laiko atskirose stadijose (paruošta testavimui, testuojama, užblokuota) užduotis išbūna. Vadovas seka šias metrikas,

identifikuojama strigimo vietas, analizuoja priežastis ir ieško sprendimo būdų, kaip to išvengti. Taip pat sekama defektų kiekio atsiradimas, kas leidžia nuspėti naujų kiekį. Tai padeda planuojantis darbą, nusistatant tikslus.

3. Motyvuojant testuotojus. Metrikų sekimas leidžia nusistatyti įvairius tikslus. Pastarųjų pasiekimas veikia kaip motyvacija siekti dar geresnių rezultatų, dar aukštesnės produkto kokybės.

Trumpai tariant, analitika yra puikus įrankis testavime, leidžiantis identifikuoti spragas procese ir tuo pačiu jas gerinti. Deja iš visų apklaustųjų, tik pastarasis paminėjo, besinaudojantis šiuo būdu.

Išvados:

1. Mažai testuotojų apskritai yra girdėję apie TMA/TPI modelius. Tai, galbūt, vienas iš faktorių, lemiančių žemą testavimo brandą. Dėl to itin svarbu skleisti šias žinias, skatinti testavimo proceso gerinimą.
2. Dažnai testavimo proceso problemos jaučiamos, bet su jomis tiesiog susigyvenama ir net nepagalvojama, kad reikėtų jas išspręsti.
3. Testavimas neatneša konkrečios lengvai išmatuojamos naudos. Todėl itin svarbu tai sudėti į materiją, pateikti skaičių pavidalu, iliustruoti diagramomis.
4. Testavimo procesas privalo būti kontroliuojamas. Tam itin gerai gali pasitarnauti analitika, įvairių metrikų sekimas. Tai leidžia identifikuoti silpnąsias vietas ir ieškoti sprendimo būdų.
5. Remiantis visais atliktais interviu – analitika testavime sutinkama itin retai.

13. **Tobulinimo susitikimai neišnaudojami iki galo:**

Tai dar viena *agile* praktika, kuri gali palengvinti bei paspartinti darbą, jei yra tinkamai atliekama. Šio susitikimo esmė peržiūrėti turimus darbus (angl. *backlog*), aptarti vartotojo istorijas, jas išsiaiškinti ir suskaidyti. Tai turėtų būti diskusija.Keletas apklaustųjų teigė, kad dažnai kūrėjų komanda į šį susitikimą žiūri atsainiai, neįsigilina į naują funkcionalumą, nekelia klausimų. Testuotojai neįsigilina į priėmimo (angl. *acceptance*) kriterijus, neieško klaidų ar trūkumų. Dėl to pradėjus kurti naują funkcionalumą, atsiranda daug neaiškumų, dažnu atveju tenka grįžti prie pačios vartotojo istorijos aiškinimosi, ją perdaryti.

Kai vartotojo istorija atiduodama testavimui, ji gali atitikti visus priėmimo kriterijus, tačiau galiausiai paaiškėja, kad yra tam tikrų loginių netikslumų, kažkas neapgalvota iki galo. Tada vėl grįžtama prie vartotojo istorijos perdarymo, vėl keičiamas funkcionalumas, eikvojamas laikas.

Vienas apklaustasis teigė, jog kelios dienos prieš tobulinimo susitikimą yra atsiunčiamos naujos vartotojo istorijos, kurios turėtų būti įterpiamos sekančioje iteracijoje. Yra griežtas reikalavimas, jog kūrėjų komanda peržiūrėtų istorijas prieš susitikimą ir pasižymėtų neaiškumus, o esant reikalui, pasiieškotų papildomos informacijos. Per susitikimą kiekviena vartotojo istorija pristatoma iš verslo pusės ir ją turi pakomentuoti kūrėjų komanda – tiek iš programavimo pusės, tiek iš testavimo. Jei viskas aišku, istorija keliama į sekantį sprintą. Jei ne, verslo analitikas daro gilesnę analizę ir keičia reikalavimus.

Išvados:

1. Pasiteisinusi praktika rodo, kad keletą dienų prieš tobulinimo susitikimus reikėtų atsiųsti naujai aptariamus reikalavimus. Su jais privalu susipažinti bei pasižymėti kilusius neaiškumus visoms susijusiomis šalims.
2. Šiuose susitikimuose turi būti skatinama diskusija, aptariamos visos išimtys.
3. Turint gerus tobulinimo susitikimus, mažėja reikalavimų pasikeitimo/atsiradimo tikimybė vėlesnėse projekto stadijose.

3.2. Pageidautinų testuotojo kompetencijų aibė

Atliekant interviu, buvo užduotas dar vienas klausimas, kuris gali padėti suprasti, kokios testuotojo kompetencijos yra labiausiai vertinamos šiandieninėje programų kūrimo aplinkoje. Šie atsakymai leidžia lengviau atlikti įdarbinimo procesą, kai siekiama surasti pastiprinimą į testuotojų komandą ar tiesiog norima ugdyti jau esamus testuotojus. Klausimas skambėjo taip: „Kokios 5 kompetencijos, jūsų nuomone, yra privalomos geram testuotojui?“

Sisteminant kompetencijas, buvo išskirtos trys grupės:

1. Asmenybės kompetencijos, kurios kyla iš pačio žmogaus – iš to, koks jis yra. Asmenybės kompetencijos sunkiai pakeičiamos.
2. Analitinės kompetencijos, kurios iš dalies priklauso nuo pačio žmogaus, tačiau praktika padeda jas išugdyti.
3. Techninės kompetencijos – visų pirma žmogus turi mėgti techninę sritį. Tuomet šią dalį reikia atidirbti įdedant daug praktinio darbo.

Lentelėje 2 pateikiama grupės pavadinimas, skaičius apklaustųjų, kurie paminėjo tam tikrą kompetenciją, bei trumpas aprašymas. Kiekvienoje grupėje kompetencijos išdėstytos pagal svarbumą – pradedama nuo dažniausiai minimų.

2 lentelė

Pageidautinų testuotojo kompetencijų lentelė, suskaidyta pagal grupes bei išrikiuota pagal svarbumą

Grupė	Atsakiusiųjų skaičius	Kompetencija
Asmenybės bruožai	15	Mokėjimas komunikuoti – dažniausiai minėtas atsakymas. Testuotojas yra suvokiamas kaip tarpininkas tarp verslo žmonių ir programuotojų. Jis turi pakankamai techninių žmonių, todėl supranta programuotojų „kalbą“. Tačiau privalo suprasti ir verslo pusę, jų poreikius, norus. Jei žmogus uždaras, negebantis išsakyti savo nuomonės – jį vargu ar galima pavadinti geru testuotoju.
	10	Proaktyvumas – „Testuotojas negali laukti, kol kas nors jam duos darbų. Jis pats privalo eiti, ieškoti, padėti“ - , teigė vienas apklaustasis. Jis turi būti nepatiklus, nuolat kelti klausimus. Gavus vartotojo istoriją su nustatytais reikalavimais, jis negali aklai pratestuoti visų priėmimo kriterijų ir baigti darbą. Testuotojas privalo apgalvoti, ar tikrai reikalavimai parengti gerai, ar apgalvoti tam tikri kritiniai atvejai.
	9	Lankstumas (angl. *agility*) – pokyčiai neturėtų stebinti. Jie turi būti priimami kaip natūralus bei neišvengiamas dalykas. Testuotojas privalo į juos reaguoti lanksčiai ir greitai prisitaikyti.
	9	Nuolatinis mokymasis – dauguma apklaustųjų teigė, kad atitinkamas bakalauro laipsnis nėra labai svarbu. Svarbu, kad testuotojas būtų linkęs mokytis, kai pastebi savo žinių trūkumą. Įvairūs kursai internete, knygos, konferencijos gerai praplečia testuotojo suvokimą apie patį programinės įrangos kūrimą, jos testavimą.
	3	Geras humoro jausmas. Anot apklaustųjų, programinės įrangos kūrimas sukelia nemažai streso. Todėl itin svarbu mokėti nekelti panikos, baimės, kai kas nors atsitinka ne

		pagal planą. Nežiūrėti į viską per daug rimtai ir palaikyti gerą atmosferą komandoje.
Analitiniai aspektai	12	Detalumas (angl. *eye for details*). Testuotojas privalo būti itin kruopštus, negali praleisti nei vienos detalės pro akis.
	12	Analitiškumas – dažniausiai čia pabrėžiamas mokėjimas surasti defekto priežastis (angl. *root cause*). Tam, kad tai sugebėtum, reikia mokėti analitiškai žingsnis po žingsnio surasti galimus klaidos atsiradimo kelius, eiti gilyn (angl. *drill down*). Kuo geriau bus išanalizuotos priežastys, tuo greičiau programuotojas galės ištaisyti defektą.
	6	Bendro vaizdo (angl. *big picture*) matymas – gebėti matyti vaizdą iš aukščiau. Suprasti, kaip atskiros dalys veikia kartu, kaip jos integruojasi. Gebėti numatyti vienu žingsniu toliau. „Tai skamba lyg savaime suvokiamas dalykas, tačiau testuotojai dažnai *paskęsta* mažose problemose ir nesugeba pamatyti, kaip ta problema atsispindi bendram vaizde“ -, teigė apklaustasis. Tai padeda prioretizuoti darbus, esant laiko spaudimui.
	3	Mokėjimas naudotis „Google“ paieškos sistema – anot kelių apklaustųjų, šis bruožas yra privalomas, tačiau retai kada minimas. Išgirdus naują sąvoką ar dokumentacijoj perskaičius neaiškią vietą – svarbu pirma paieškoti atsakymo internete, susidaryti bendrą vaizdą ir tik tada, esant reikalui, klausti pas komandos narius. Taip yra gerbiamas bei taupomas kitų laikas.
Techniniai aspektai	11	Kodo skaitymas – geras testuotojas privalo tai mokėti. Duomenų bazių SQL eilučių skaitymas, mokėjimas išsitraukti duomenis. Registro (angl. *log*) skaitymas bei naršyklės programavimo įrankių juosta turi būti įprastas savaime suprantamas dalykas.
	9	Naujovių siekimas bei naudojimas – kadangi informacinių sistemų srity technologijos itin greitai keičiasi, geras testuotojas stengsis nuolat atnaujinti savo žinias, neatsilikti nuo rinkos naujovių. Itin didelis pliusas, jei testuotojas

		domisi naujais atsiradusiai įrankiais, kurie gali palengvinti ir pagerinti testuotojams darbą. Esant galimybei, juos išbando praktiškai, o pamačius naudą, pristato komandai ir, galbūt, nutaria naudoti savo kasdieniniame darbe.
	4/6	Programavimo įgūdžiai – čia apklaustųjų nuomonės išsiskyrė. Vieni teigė, kad testuotojo negali vadinti geru, jei jis nemoka rašyti automatinių testų (4). Kiti teigė, kad šis įgūdis yra bevertis be geros rankinio testavimo patirties. Techniniai aspektai išmokstami – svarbiausia geras pagrindas. (6)

Šaltinis: parengta autoriaus, remiantis atliktu tyrimu

Apjungus lentelės duomenis, grupes galima apibrėžti taip:

1. Asmenybės kompetencijos – gebantis puikiai komunikuoti, lengvai priimantis dažnus pokyčius. Mėgstantis nuolat gilinti savo žinias bei proaktyviai atliekantis paskirtus darbus. Mokantis valdyti stresą, nekeliantis panikos komandoje, bet atvirkščiai, palaikantis gerą atmosferą.
2. Analitinės kompetencijos – jos persipina su pačio žmogaus savybėmis, tačiau joms lavinti reikia praktikos. Gebėjimas matyti kiekvieną detalę padeda lengviau surasti problemos šaltinius, analitiškai apibrėžti klaidų atsiradimo priežastis. Tačiau tuo pačiu, testuotojas privalo mokėti atsitraukti nuo visiško detalumo, ir pažvelgti į produktą lyg iš paukščio skrydžio, matyti, kaip skirtingos dalys veikia kartu. Ir kitas retai minimas, bet privalomas aspektas – puikus gebėjimas naudotis „Google“ sistema.
3. Techninės kompetencijos – visų pirma, testuotojui turi patikti techniniai aspektai, jam turi būti įdomi ši dalis. Itin gerai, jei testuotojas proaktyviai domisi naujovėmis šioje srityje. Iš praktikos turi būti suformuotas PĮ aplinkos suvokimas, mokėjimas skaityti kodą bei naudotis įvairiais įrankiais, kurie palengvina testavimą.

3.3. Atnaujinta TMA/TPI apibendrinto proceso diagrama

Atliekant interviu, išryškėjo, kaip glaudžiai testavimo procesas yra susijęs su kitais PĮ įrangos procesais. Testuotojo darbas yra apjungti visus kitų komandos narių nuveiktus darbus į vieną vietą ir užtikrinti, kad viskas veikia sklandžiai. Jei kitoje vietoje buvo padaryta klaidų – ne iki galo išdirbti reikalavimai, tarpusavyje nesuderinti procesai, prastas projekto darbų valdymas ir t.t., tai gali stipriai paveikti testuotojų darbą ir rezultatą, o tuo pačiu viso produkto kokybę.

Apibendrinta TMA/TPI proceso diagrama (1 paveikslėlis), gali kelti klaidingą supratimą, jog testavimo proceso gerinimas yra pilnai izoliuota veiksmų seka, t.y. visiškai nepriklauso nuo kitų, greta gyvuojančių PĮ procesų. Diagramoje nėra užuominos apie komunikaciją visoms susijusioms šalims, nepaminėta, kad pokytį gali tekti vykdyti ne tik testuotojams, bet ir greta esantiems procesų vykdytojams, tokiems kaip verslo žmonės, programuotojai ar vartotojo sąsajos dizaineriai. Svarbu suvokti, jog testavimo procesas yra sistemos dalis – tiek jis veikia aplinką (kitus PĮ procesus), tiek aplinka veikia jį. Siekimas gerinti vien tik testavimo procesą ignoruojant šalia esančius procesus, didelės naudos neatneš.

Siekiant išvengti klaidingo supratimo, nuspręsta pateikti atnaujintą TMA/TPI apibendrintą proceso diagramą. Tai buvo daroma remiantis gauta informacija iš literatūros apžvalgos bei atlikto kokybinio interviu tyrimo. Svarbiausi aspektai šie:

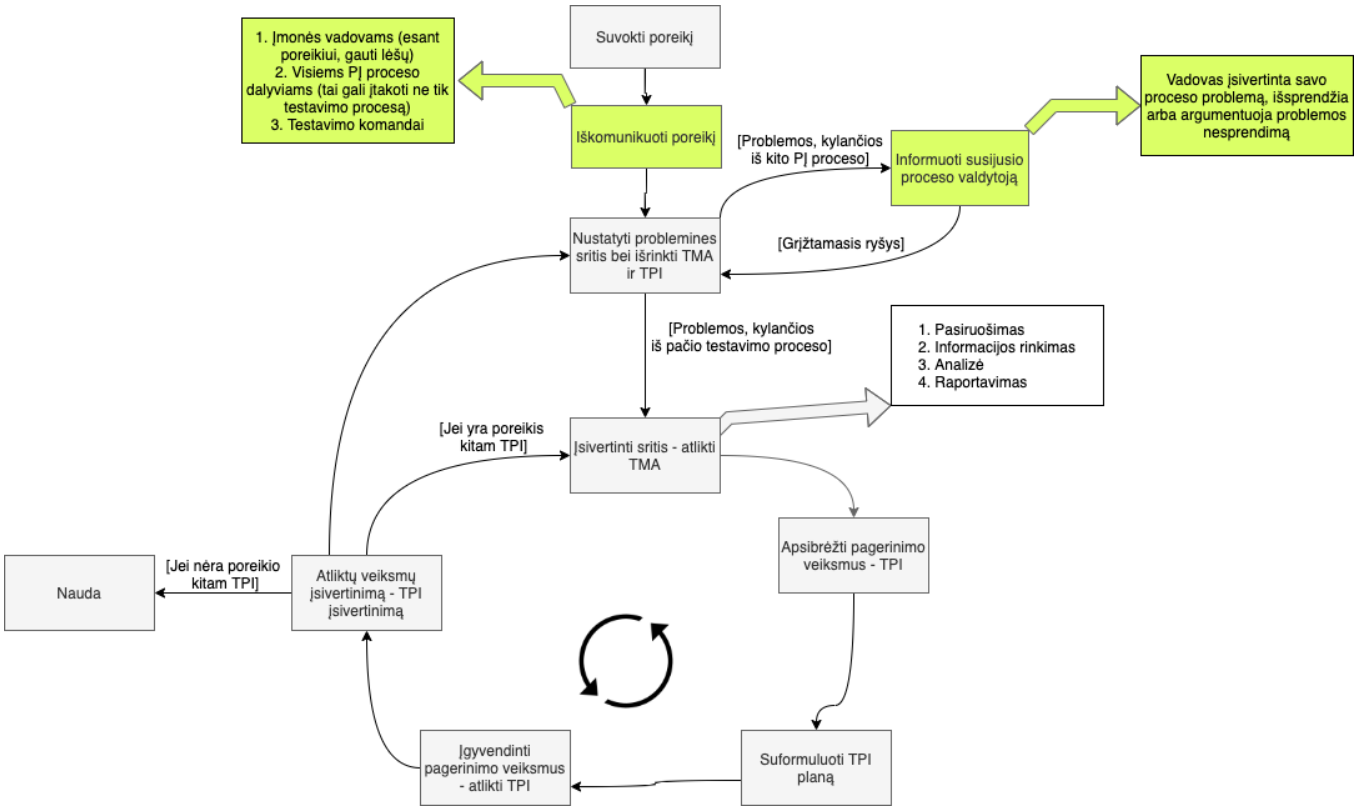
1. Suvokus poreikį gerinti testavimo procesą, itin svarbi dalis – gerai iškomunikuoti poreikį susijusiems asmenims:
    - Įmonės vadovams. Siekiant įgyvendinti pokytį testavimo procese, būtų sunku apsieiti be papildomų lėšų. Dėl šios priežasties, poreikis gerinti procesą privalo būti aiškiai iškomunikuotas aukščiausiai valdžiai. Šioje vietoje turi būti stiprus poreikio pagrindimas. Veiksmingiausia pateikti informaciją naudojantis analitika – vaizdžiai parodyti esamą situaciją bei kokio pagerėjimo tikimasi įvykdžius TMA/TPI veiklas. Informacija turi būti apčiuopiama ir lengvai supranta. Tam tinka įvairios diagramos, statistikos. Tik gavus valdžios paramą, galima tikėtis kokybiško pokyčio.
    - Visiems PĮ proceso dalyviams. Testavimo procesas nėra izoliuotas - atvirkščiai, itin glaudžiai susijęs su visais kitais PĮ procesais. Dėl šios priežasties, gavus valdžios sutikimą, privaloma informuoti apie galimus pokyčius kitų procesų savininkus.
    - Testavimo komanda – didžiausią pokytį pajunta būtent jie. Žmonės, įpratę prie esamų procesų, gali priešintis naujiems pokyčiams. Dėl to itin svarbu, kad pokyčio iniciatorius (dažniausiu atveju testavimo proceso vadovas) aiškiai iškomunikuotų esamas problemas ir kodėl jas reikia spręsti. Vadovas turi motyvuoti komandą ir pokytį įvesti žingsnis po žingsnio, nereikalaujant čia ir dabar tobulo pokyčio bei rezultatų.
2. Nustatant problemines sritis itin svarbu suvokti, ar jas įtakoja spragos pačiam testavimo procese, ar jos visgi kyla iš kito susijusio proceso. Esant pastarajam variantui, turi būti informuotas susijusio proceso vadovas. Preliminariai jo žingsniai turi būti šie:

- Įsivertinti problemą, kylančią iš jo valdomo proceso. Apsvarstyti, ar tikrai problema egzistuoja, jei taip, kaip būtų galima ją išspręsti.
- Problemą ištaisyti arba nuspręsti netaisyti.
- Pateikti grįžtamąjį ryšį – jei problema ištaisyta, paaiškinti kaip tai pavyko pasiekti. Jei problemos nuspręsta netaisyti – argumentuotai paaiškinti, kodėl buvo priimtas toks sprendimas.

Atsižvelgiant į aukščiau pateiktus argumentus, buvo sudarytas atnaujintas TMA/TPI veiklas apibendrintas modelis:

8 paveikslas

Autoriaus atnaujinta TMA/TPI apibendrinta proceso diagrama



Šaltinis: autoriaus sudaryta, remiantis Tim Koomen, 1999.

IŠVADOS IR PASIŪLYMAI

Atlikus literatūros analizę bei kokybinį interviu tyrimą, buvo identifikuota keletas išvadų, kurios aprašytos 3 lentelėje. Pagal esmę, išvados suskirstytos į keturias grupes:

1. Bendras testavimo procesas
2. Automatinis testavimas
3. Bendras PĮ kūrimo procesas
4. Įmonės procesai

3 lentelė
Išvados, suskirstytos į grupes

Kategorija	**Išvados**
Testavimo procesas	Atsižvelgiant į literatūros analizėje pateiktą TMMI modelį, kuris turi 5 lygius - pradinis, valdomas, apibrėžtas, valdomas & matuojamas, optimizuotas, preliminariai galima teigti, kad 14 apklaustųjų įmonių testavimo proceso brandos lygis neviršija 2-3. Tik 1 respondento įmonės procesas galėtų pretenduoti patekti į 4 lygį (valdomas & matuojamas), nes tik ten yra naudojama analitika, įvairios metrikos, matuojami bei gerinami procesai. Svarbu suprasti, jog analitika yra kelias į aukštesnį testavimo proceso lygį, nes tai, ką galima išmatuoti, tą galima ir pagerinti. Taipogi, ilgainiui ji padeda prognozuoti ateities įvykius, tokius kaip defektų atsiradimo tikimybė bei jų ištaisymo laikas ar reikiamų resursų kiekis.
	Visi apklaustieji teigė, kad darbe yra susidūrę su *agile* metodologija. Tačiau pridūrė, kad *agile* praktikas prisiderino prie savo projekto – kiekvieno apklaustojo procesas atrodė skirtingai. Visiems bendros praktikos buvo dieniniai trumpi susitikimai ir darbas iteracijomis. Visos kitos *agile* praktikos buvo naudojamos ir priderinamos pagal projektą**.** *Agile* metodologija nėra vienas bendras visiems tinkantis šablonas. Itin svarbu suprasti *agile* praktikų poreikį būtent savo projektui. Kas tinka vieniems, nebūtinai tiks kitiems. Pravartu pradžioje pabandyti keletą praktikų, išsirinkti tinkamiausią ir ją priderinti prie konkrečių savo projekto poreikių. Ateity, projektui augant, vėl galima apsvarstyti praktikų tinkamumą ir esant reikalui jas pakeisti/priderinti.
	Visi apklaustieji minėjo, kad savo darbe siekia naudoti DevOps praktikas. Tačiau daugiau nei pusė apklaustųjų pridūrė, kad komandoje šių žinių trūksta, todėl

	DevOps praktikų vis dar nenaudoja. Likusieji teigė, kad jų įmonėse yra atskiros DevOps inžinierių komandos, kurios visą konfigūracinį darbą atlieka už juos. Nereikėtų tikėtis gilių DevOps žinių iš testuotojų ar kitų komandos narių, kurių rolė yra kita. DevOps inžinierius turi specifines kompetencijas bei žinias, todėl siekiant tobulinti CI/CD procesus būtina turėti atitinkamos profesijos žmogų.
	Interviu metu dauguma apklaustųjų buvo aukštesnes testavimo pareigas užimantys asmenys, kurių patirties vidurkis yra 9.5m. Tačiau žvelgiant į jų išsilavinimą, tik 3 iš 15 apklaustųjų turi IT bakalauro ar aukštesnį laipsnį. Tai leidžia daryti išvadą, kad testuotojui diplomas IT sferoje nėra būtinas. Svarbiau pats žmogus, jo motyvacija bei noras tobulėti.
	Vos 3 apklaustieji buvo girdėję apie TMA/TPI veiklas. Kiti teigė, kad būtent šis interviu padėjo pamatyti, kiek problemų egzistuoja jų testavimo procese, tačiau nieko nėra daroma. Tai veda prie suvokimo, jog testavimo procesą reikia gerinti, tačiau, visų pirma, reikia šviesti testuotojus apie galimus testavimo proceso gerinimo metodus.
	7 apklaustieji teigė, kad jų komanda nesupranta testavimo svarbos, bet tik vienas apklaustasis minėjo, jog savo darbe aktyviai naudoja analitiką, t.y. atliktus darbus bando sudėti į skaičių pavidalą. Testavimas neatneša aiškiai matomos naudos, kaip, pavyzdžiui, programavimas. Tam, kad komandos nariai pamatytų testavimo naudą ir ją pripažintų, reikia stengtis testavimą sudėti į materiją. Metrikų sekimas, jų gerinimas, diagramų rodymas – puikus būdas tai pasiekti.
	6 apklaustieji teigė, kad iš testuotojo reikalaujama itin gero suvokimo apie visą sistemos veikimą. Išryškėjo išvada, jog programuotojai į jį žiūri kaip į konsultantą, kuris turėtų puikiai suprasti kliento norus. Dėl to glaudesnis testuotojo ir užsakovo ryšis gali puikiai pasiteisinti, siekiant išgauti aukštesnę kokybę klientui.
Automatinis testavimas	Visi iš 15 apklaustųjų teigė, kad turi daugiau ar mažiau praktikos rašant automatinius testus. Taip pat minėjo, kad šias žinias yra privalu įgyti einant testuotojo karjeros keliu**.** Tai leidžia daryti išvadą, kad geram testuotojui anksčiau ar vėliau reikėtų prisiliesti prie techninės dalies – rašyti automatinius testus.
	Spartėjant programinės įrangos kūrimo greičiui, tikimasi daugiau automatinio ir mažiau rankinio testavimo. 13 apklaustųjų teigė, kad, visų pirma, automatiniais testais siekiama padengti regresiją. Tačiau, interviu rezultatai parodė, jog išeina atvirkščiai – rankinio testavimo kiekis vis auga, automatinio mažėja. Tas lemia vis

	didesnius rankiniu būdu testuojamus regresijos kiekius, vis labiau vėluojama ir gaunasi užburtas ratas.
	3 automatinio testavimo ekspertai teigė, jog dažnai testuotojų prašoma rašyti skirtingus automatinius testus tam pačiam funkcionalumui – nuo pradžios iki galo, integracinius, greitaveikos (angl. *performance*) ir t.t. Tai padeda greičiau aptikti atsiradus defektus. Tačiau dažnai pamirštama, kad atsiradus pakeitimams, tai iššauks daug darbo – keisti kodą visose skirtingose vietose. Patartina rinktis kokybę, o ne kiekybę ir nepamiršti DRY principo. Taipogi, visi 15 apklaustųjų sutiko, kad ne tik programuotojo kodas privalo būti kokybiškas. Testuotojo rašomi automatiniai testai taip pat yra kodas. Todėl privaloma, kad automatiniai testai būtų peržiūrimi programuotojų pagal bendrus programavimo standartus.
Bendras PĮ kūrimo procesas	Atlikti interviu parodė, kaip stipriai PĮ kūrime visi procesai yra persipynę. Dėl to, siekiamybė gerinti tik testavimo kokybę, didelio rezultato neatneš. Reikia suprasti, kad kokybė yra bendras visų komandos narių darbas, todėl gali tekti gerinti/keisti ir kitus procesus.
	"Nors *agile* metodologija teigia, kad reikia mažinti dokumentaciją, tačiau tai dažnai būna meškos paslauga komandai, jei neužtikrinama kita sąlyga – itin stipri komunikacija", - šiai minčiai pritarė daugiau nei pusė apklaustųjų (9). Reikia pasirinkti – arba turėti itin gerą komunikaciją visoje komandoje, kai visi susiję asmenys dalyvauja aktualiuose susitikimuose, arba turėti konkrečią vietą, kur bus kruopščiai dokumentuojama reikalinga informacija. Tai itin svarbu testuotojui, kai reikia tarti galutinį žodį dėl produkto kokybės.
	Neišdirbti ir nuolat kintantys reikalavimai lemia žemesnę produkto kokybę. Šiai problemai spręsti itin veiksminga turėti nuolatinį grįžtamąjį ryšį, dirbti pagal *agile* metodologijoje vadinamus MVP. Užsakovui bus paprasčiau išreikšti savo norus, matant bent dalinai veikiantį produktą. Kūrimo komanda bus rami, kad kuria tą produktą, kuris atitinka reikalavimus. 6 apklaustieji minėjo, kad stengiasi sukurti ir parodyti mažą MVP kas 2sav. 5 teigė, kad veikiantį naują funkcionalumą išeina parodyti tik kas kelis mėnesius. Likę teigė, kad jų projektai itin sudėtingi, dėl daug įvairių priklausomybių. Tai trukdo reguliariai kurti ir pateikti MVP, kas iššaukia kliento norų neatitikimą ir naujus reikalavimus.
	12 iš apklaustųjų teigė, kad tiek išorinių tiek vidinių integracijų nestabilumas – didelis skausmas testuotojams. Vidinėms integracijos sekti pasiteisina sukurti servisų sveikatos skydeliai, kurie informuoja, apie kintančias būsenas. Su išorinėmis

	integracijomis kiek sudėtingiau. Tačiau abejais atvejais – aiškiai nusistatytas komunikavimo būdas, padeda sutaupyti testuotojų laiką, aiškinantis, testų griuvimo priežastis.
Įmonės procesai	5 apklaustieji teigė, kad jų darbe itin taupoma testuotojų algoms, ieškoma praktikantų ir siekiama juos apmokyti. Taupymas testavimo sąskaitą yra stipriai susijęs su produkto kokybe. Ar būtų taupomi pinigai (žemi atlyginimai testuotojams, maža jų patirtis) ar taupoma laikas (skubama išleisti produktą, todėl skiriama mažai laiko testavimui), bet kokiu atveju kentės produkto kokybę.
	Anot 6 vadovų patirties, testuotojai, pasiekę pakankamai aukštą automatinio testavimo lygį, dažnai pasirenka programuotojo kelią. Tai viena iš priežasčių, dėl ko rinkoje trūksta gerų testavimo specialistų. Norint juos pritraukti, pravartu turėti stiprius įdarbinimo specialistus bei išlaikyti konkurencingą atlygį.

Šaltinis: parengta autoriaus, remiantis literatūros analize bei atliktu tyrimu

Taip pat darbe buvo identifikuotos pageidautinos testuotojo kompetencijos, pagrįstai suskirstytos į tris grupes: asmenybės, analitinės bei techninės kompetencijos. Asmenybės – tos, kurias pakeisti sudėtinga. Analitinės – iš dalies galima išugdyti, bet iš dalies priklauso nuo pačio žmogaus, koks jis yra. Techninės kompetencijos – jei yra noras, galima viską išmokti. Ieškant pastiprinimo į testavimo komandą, rekomenduojama, daugiausiai dėmesio skirti asmenybės kompetencijoms, jų atitikimui. Šiuos bruožus vargu ar galima išugdyti ir bėgant laikui pagerinti. Sekanti pagal svarbą grupė - analitiniai aspektai. Šias kompetencijas pakeisti sudėtinga, bet laikui bėgant įmanoma. Lengviausia yra su techniniais aspektais – jei matosi, kad žmogus nebijo techninės dalies, tai itin geras ženklas. Techninės kompetencijos gali būti pilnai išugdomos.

Paskutinis aspektas – darbe pateikiama pagrįstai atnaujinta TMA/TPI apibendrinto proceso diagrama. Itin svarbu suprasti, jog testavimo proceso izoliuoti negalima – jis yra stipriai priklausomas nuo kitų PĮ procesų. Pasiryžus vykdyti TMA/TPI veiklas, pokytis turės vykti ne tik testavimo procese, bet ir kituose. Tai supratus, pasidaro aišku, kad TMA/TPI veiklų poreikis turi būti gerai iškomunikuotas visiems susijusiems asmenims – vadovams, kitų PĮ procesų valdytojams bei pačiai testavimo komandai. Tik tai atlikus ir gavus leidimą, galima pradėti vykdyti TMA/TPI veiklas.

LITERATŪROS SĄRAŠAS

Afzal, W., Alone, S., Glocksien, K., & Torkar, R. (2016). Software test process improvement approaches: A systematic literature review and an industrial case study. *Journal of Systems and Software*, *111*, 1–33. https://doi.org/10.1016/j.jss.2015.08.048

Agarwal, A., Gupta, S., & Choudhury, T. (2020). Proceedings of the 2020 International Conference on Advances in Computing and Communication Engineering, ICACCE 2020. *Proceedings of the 2020 International Conference on Advances in Computing and Communication Engineering, ICACCE 2020*, *June*, 290–293.

Anand, R. V., & Dinakaran, M. (2016). International Journal of Scientific and Technical Advancements Popular Agile Methods in Software Development: Review and Analysis. *International Journal of Scientific and Technical Advancements*, *2*(4), 147–150.

Balaraman, R. (2018). PERSPECTIVE NEED FOR A COMPREHENSIVE TEST MATURITY MODEL. *Infosays*.

Burnstein, I. (2003). *THE TESTING MATURITY MODEL AND TEST*.

Camargo, K. G., Ferrari, F. C., & Fabbri, S. C. (2015). Characterising the state of the practice in software testing through a TMMi-based process. *Journal of Software Engineering Research and Development*, *3*(1). https://doi.org/10.1186/s40411-015-0019-9

Eldh, S., Andersson, K., Ermedahl, A., & Wiklund, K. (2014). Towards a Test Automation Improvement Model (TAIM). *Proceedings - IEEE 7th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2014*, 337–342. https://doi.org/10.1109/ICSTW.2014.38

Florea, R., & Stray, V. (2019). A Global View on the Hard Skills and Testing Tools in Software Testing. *Proceedings - 2019 ACM/IEEE 14th International Conference on Global Software Engineering, ICGSE 2019*, 143–151. https://doi.org/10.1109/ICGSE.2019.00035

Foundation, Tmm. (2019). TMMi in the Agile world Produced by the TMMi Foundation. *TMMi Foundation*, 1–44.

Garcia, C., Dávila, A., & Pessoa, M. (2014). Test process models: Systematic literature review. *Communications in Computer and Information Science*, *477*, 84–93. https://doi.org/10.1007/978-3-319-13036-1_8

Garousi, V., Felderer, M., & Hacaloglu, T. (2017). What We Know about Software Test Maturity and Test Process Improvement. *IEEE Software*, *35*(1), 84–92. https://doi.org/10.1109/MS.2017.4541043

Garousi, V., Felderer, M., & Hacaloğlu, T. (2017). Software test maturity assessment and test process improvement: A multivocal literature review. *Information and Software Technology*, *85*, 16–42. https://doi.org/10.1016/j.infsof.2017.01.001

Gaurav, K., & Pradeep, B. K. (2012). Impact of Agile Methodology on Software Development Process. *International Journal of Computer Technology and Electronics Engineering (IJCTEE)*, *2*(4), 46–50. https://pdfs.semanticscholar.org/b7c4/48b29363b6ea8c946ede6cab91de6673aa1f.pdf

ISTQB. (2018). *Worldwide Software Testing Practices Report Message from the President Executive Summary Survey Questions and Analysis*.

Hornbeek, M. (2020). *Continuous Test Automation Maturity Levels*. https://blog.testproject.io/2020/06/07/continuous-test-automation-maturity-levels/

J. García , A. de Amescua , M. Velasco, A. S. (2008). *Ten factors that impede improvement of verification and validation processes in software in- tensive organizations.*

Karthikeyan, S., Rao, S., & Technology Solutions, C. (2014). *Adopting the Right Software Test Maturity Assessment Model*. *june*. https://www.cognizant.com/whitepapers/Adopting-the-Right-Software-Test-Maturity-Assessment-Model-codex881.pdf

Kasurinen, J. (2012). Software Organizations and Test Process Development. In *Advances in Computers* (Vol. 85). Elsevier Inc. https://doi.org/10.1016/B978-0-12-396526-4.00001-1

Krasner, H. (2018). The Cost of Poor Software Quality in the US: A 2018 Report. *Consortium for IT Software Quality (CISQ)*, 1–44.

Kulkarni, S. (2006). Test Process Maturity Models–Yesterday, Today and Tomorrow. *Proceedings of the 6th Annual International Software Testing Conference*, 1–15. http://www.qaiglobal.com/Design/Newsletter/attachments/sep_09/white_paper.pdf

Landscape, D. (2019). *Is DevOps Actually Paying Off?* https://www.mabl.com/devtestops/survey-results

Mordinyi, R., Kühn, E., & Schatten, A. (2010). Towards an architectural framework for agile software development. *17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS 2010, section IV*, 276–280. https://doi.org/10.1109/ECBS.2010.38

Paulk, M. C. (2009). A History of the Capability Maturity Model for Software. *The Software Quality Profile*, *1*(1), 5–19.

Paulk, M. C., Curtis, B., Chrissis, M. B., & Weber, C. V. (2011). The capability maturity model for software. *Software Process Improvement*, 49–52. https://doi.org/10.1109/9781118156667.ch2

Rajkumar, M., Pole, A. K., Adige, V. S., & Mahanta, P. (2016). DevOps culture and its impact on cloud delivery and software development. *Proceedings - 2016 International Conference on Advances in Computing, Communication and Automation, ICACCA 2016*. https://doi.org/10.1109/ICACCA.2016.7578902

Razak, R. A., & Fahrurazi, F. R. (2011). Agile testing with Selenium. *2011 5th Malaysian Conference in Software Engineering, MySEC 2011*, 217–219. https://doi.org/10.1109/MySEC.2011.6140672

Ronen, S. (2011). *A practical view on Agile Testing Maturity Levels*. https://www.slideshare.net/AgileSparks/atmm-practical-view

Services, T. C. (2016). Implementing TMMi for Agile Projects A Rating Criteria and Mapping Framework Agile and TMMi: Two Sides of the Same Coin? *Tata Consultancy Services*.

Tamaševičius, V. (2015). *Tyrimų metodai*.

Tim Koomen, M. P. (1999). *Test process improvement: a practical step-by-step guide to structured testing*. Addison-Wesley Longman Publishing Co., Inc.75 Arlington Street, Suite 300 Boston, MAUnited States.

TMMi Foundation Reference Model. (2012).

Tricentis. (2017). *Continuous Testing Maturity Model - Tricentis*. 1–2. https://www.tricentis.com/resources/continuous-testing-maturity-model/

Veenendaal, E. Van. (2014). Test Process Improvement and Agile : Friends or Foes ? *Magazine for Professional Testers*, *June 2014*.

Veenendaal, E. Van. (2018). *TMMi in the Agile world*. *September*, 1–44.

Veenendal, E. van. (2012). *Test Maturity Model Integration TMMi Guidelines for Test Process Improvement*.

Wang, Y. (2018). Test Automation Maturity Assessment. *Proceedings - 2018 IEEE 11th International Conference on Software Testing, Verification and Validation, ICST 2018*, 424–425. https://doi.org/10.1109/ICST.2018.00052

Woodruff, W. D. (2003). *Introduction of Test Process Improvement and the Impact on the Organization*. http://rube.asq.org/pub/sqp/past/vol5_issue4/woodruff.html

Zhu, L., Csiro, D., & Champlin-scharff, G. (2018). *07458765*. 32–34.

Rugilė ŽIUKIENĖ

Magistro darbas

Strateginis informacinių sistemų valdymas

magistro programa

2 kurso studentė

Rugilė Žiukienė

Vilniaus Universitetas, Ekonomikos ir Verslo Administravimo fakultetas

Vadovas – lekt. A. Gavrilov

Vilnius, 2022

SANTRAUKA

60 puslapių, 8 diagramos, 5 lentelės, 37 nuorodų.

Pagrindinis šio magistrinio darbo tikslas yra įvertinti testavimo proceso problematiką šiandieninėje programinės įrangos kūrimo aplinkoje.

Darbas susideda iš trijų pagrindinių dalių: testavimo problematika tyrimo erdvėje, atlikto tyrimo metodika ir rezultatai bei išvados, pateiktos kartu su rekomendacijomis.

Literatūros analizėje buvo apžvelgiami testavimo proceso įsivertinimo bei gerinimo modeliai – TMA/TPI, kokie iki šiol yra sukurti, kas rodo, kad laikas gerinti testavimo procesą. Taip pat buvo apžvelgta, su kokias iššūkiais susiduriama, bandant vykdyti TMA/TPI bei kokia nauda gaunama, pagerinus procesą. Buvo vertinama, kaip TMA/TPI modeliai dera kartu su *agile* metodologija bei DevOps praktikomis.

Po apžvelgtos proceso problematikos tyrimo erdvėje, autorius atliko kokybinį interviu tyrimą. Jame buvo apklausta 15 testavimo ekspertų, siekiant išskirti praktines problemas, su kuriomis susiduriama testuojant programinę įrangą bei valdant testavimo procesą. Interviu metu, taipogi, buvo nagrinėjamas dar vienas aktualus klausimas – kokios yra pageidautinos testuotojo kompetencijos. Galiausiai buvo išskirta bei išnagrinėta 13 praktinių problemų, su kurioms dažniausiai susiduriama testavimo procese ir sudaryta lentelė pageidautinų testuotojo kompetencijų, išskaidytų į tris grupes (asmeninės, analitinės bei techninės), išrikiuotų pagal svarbumą.

Atliktas tyrimas padėjo išskirti daug naudingų įžvalgų bei paradoksų apie testavimo proceso problemas, kurios buvo susistemintos pagal keturias grupes (bendras testavimo procesas, automatinis testavimas, bendras PĮ kūrimo procesas, įmonės procesai) bei pateiktos kaip išvados lentelėje. Tyrimas taipogi atskleidė, į kokias testuotojo kompetencijas svarbiausia

kreipti dėmesį, ieškant naujo komandos nario į pastiprinimą – kokias kompetencijas galima išugdyti, o kokie žmogaus bruožai sunkiai pakeičiami. Galiausiai yra pateikta pagrįstai atnaujinta TMA/TPI apibendrinto proceso diagrama.

Išvadose bei rekomendacijos yra susisteminta pagrindinė informacija rasta atlikus analizę tyrimo erdvėje bei interviu su testavimo ekspertais. Šiame darbe pateikta informacija gali atnešti didelę naudą kiekvienam susijusiam su PĮ kūrimu, padėti pažvelgti į testavo procesą iš kitos perspektyvos – kaip į veiklą, kuri turi būti nuolat gerinama, matuojama bei supranta kaip kertinis taškas siekiant aukštesnės produkto kokybės. Kadangi interviu tyrimas atskleidė, jog TMA/TPI veiklos apskritai yra mažai kam girdėtos ir tuo labiau naudojamos, autorius tiki, jog žmonės privalo būti šviečiami apie testavimo gerinimo būdus, ką būtent ir buvo norėta pasiekti šiuo darbu.

Rugilė ŽIUKIENĖ

Master thesis

Strategic Information Systems Management master study programme

Vilnius University, Faculty of Economics and Business Administration

Supervisor – prof. A. Gavrilov

Vilnius, 2022

SUMMARY

60 pages, 8 diagrams, 5 tables, 37 references.

The main purpose of this master thesis is to evaluate testing process problematic in nowadays software development.

The work consists of three main parts: the analysis of literature, the research and its results, conclusion and recommendations.

The analysis of the literature reviews what kind of test process assessment and improvements models (TMA/TPI) exist, what factors show that it is time to improve the testing process. It also looks at the challenges of trying to implement TMA/TPI and the benefits of improving the process. It investigates how TMA/TPI models fits together with agile methodology and DevOps mindset.

After reviewing the process issues in the research space, the author conducted a qualitative interview research. It interviewed 15 testing experts to highlight the practical problems encountered in testing the software and managing the testing process. During the interviews, another relevant question was also discussed - what are the desirable competencies of the tester. Finally, 13 practical problems that are most commonly encountered in the testing process were identified and examined, and a table of preferred tester competencies was divided into three groups (personal, analytical, and technical), ranked in order of importance.

The performed research revealed a lot of useful insights and paradoxes about the problems of the testing process, which are systematized into four groups (common testing process, automated testing, common software development process, organization processes) and presented as conclusions in the table. The research also revealed which competencies of the tester are most important to pay attention to when looking for a new team member - what competencies can be developed and which human traits are difficult to change. Finally, a reasonably updated diagram of the TMA/TPI generalized process is provided.

The conclusions and recommendations systematize the main information found in the analysis in the research area and interviews with testing experts. The information provided in this paper can be of great benefit to anyone involved in software development, helping to look at the testing process from a different perspective - as an activity that needs to be continuously improved, measured and understood as a cornerstone for higher product quality. As the research of the interviews revealed that TMA/TPI activities are generally not well heard and used more, the author believes that people must be educated about the ways to improve testing, which is exactly what this work was intended to achieve.

PRIEDAI

1 Priedas. Testavimo brandos modelių, naudotų literatūros analizėje, trumpa apžvalga

Kategorija	**Modelis**	**Tipas**
Bendrinis	Testavimo brandos integracinis modelis (angl. *Test Maturity Model Integration - TMMI*) (Afzal et al., 2016)	Pakopinis Lygiai: 1. Pradinis 2. Valdomas 3. Apibrėžtas 4. Valdomas & matuojamas 5. Optimizuotas
	Testavimo pagerinimo modelis (angl. *Test Process Improvement - TPI*) (Garousi, Felderer, & Hacaloğlu, 2017)	Nuolatinis Sudarytas iš 20 sričių (angl. *Key Performance Areas – KPAs*)
Skirtas metodologijai	Agile testavimo brandos modelis (angl. *Agile Testing Maturity Model - ATMM*) (Ronen, 2011)	Pakopinis Lygiai: 1. Krioklio 2. Formavimo 3. Agile derinimo 4. Atlikimo 5. Keitimo.
Skirtas veiklai	Automatinio testavimo pagerinimo modelis (angl. *Test Automation Improvment Model - TAIM*) (Eldh et al., 2014)	Nuolatinis Sudarytas iš 10 pagrindinių sričių ir vienos bendros
	Nuolatinio testavimo brandos modelis (*angl. Continuous Testing Maturity Model – CTMM*) (Tricentis, 2017)	Pakopinis Lygiai: 1. Pradinis 2. Suderintas

		3. Suvaldytas 4. Brandus 5. Optimizuotas
	Nuolatinio automatinio testavimo brandos modelis (angl. *Continuous Test Automation Maturity Model - CTAMM*) (Hornbeek, 2020)	Pakopinis Lygiai: 4. Chaosas 5. Nuolatinis integravimas 6. Nuolatinis procesas 7. Nuolatiniai atsiliepimai 8. Nuolatiniai pagerinimai

2 Priedas. Literatūros analizėje naudotų šaltinių lentelė su trumpu aprašymu

Metai	Autorius	Pavadinimas	Pagrindinės mintys
2018	Herb Krasner	The Cost of Poor Quality Software in the US (Krasner, 2018)	Tirta, kiek kainuoja prastos kokybė programinė įranga Jungtinių Amerikos Valstijų ekonomikai.
2016	R. Vijay Anand, Dr. M. Dinakaran	Popular Agile Methods in Software Development: Review and Analysis (Anand & Dinakaran, 2016)	Palygintos 6 skirtingos agile metodikos pagal išskirtas charakteristikas.
2019	R. Florea, V. Stray	A Global View on the Hard Skills and Testing Tools in Software Testing (Florea & Stray, 2019)	Buvo nagrinėjama reikalingi testuotojo įgūdžiai bei technologijos, norint neatsilikti nuo nuolat kintančios aplinkos. Tyrime buvo įvertinta 500 darbo skelbimų iš 33 šalių.
2016	Rajkumar, M. Pole, Anil Kumar Adige, Vittalraya Shenoy Mahanta, Prabal	DevOps Culture and its impact on Cloud Delivery and Software Development (Rajkumar et al., 2016)	Šaltinyje atskleista DevOps praktikų esmė bei tyrinėta, kokią įtaka tai atneša programinės įrangos kūrimui

2015	K. Camargo, F. Ferrari, S. Fabiano	Characterising the state of the practice in software testing through a TMMi-based process (Camargo et al., 2015)	Atliekant apklausą tarp IT profesionalų Brazilijoje, siekta išsiaiškinti, kurios TMMI praktikos privalo būti testavimo procese.
2017	V. Garousi, M. Felderer, T. Hacaloğlu	Software test maturity assessment and test process improvement: A multivocal literature review (Garousi, Felderer, & Hacaloglu, 2017)	Atliekant plataus spektro analizę, siekta pateikti skaitytojui, kas skatina atlikti TMA/TPI, su kokiais išūkiais susiduriama ir kokia nauda gaunama.
2006	Shrini Kulkarni	Test Process Maturity Models - Yesterday, Today and Tomorrow (Kulkarni, 2006)	Straipsnyje nagrinėjama TMA/TPI istorija bei modelių aktualumas dabar ir ateityje. Pristatomas potencialiai naujas modelis.
2016	W. Afzala, S. Alonec, K. Glocksienc, R. Torkar	Software test process improvement approaches: A systematic literature review and an industrial case study (Afzal et al., 2016)	Naudojantis sisteminės literatūros analizės metodu, buvo tyrinėta testavimo proceso pagerinimas, rasta 18 modelių. Taip pat atliktas detalus TPI NEXT ir TMMI modelių palyginimas.
2018	Erik van Veenendaal	TMMi in the Agile Era (Veenendaal, 2018)	Knygoje pateikta TMMI modelio struktūra. Aptarta, kaip šis modelis gali būti pritaikomas agile kontekste bei kokią naudą tai atneša.
2014	C. Garcia, A. Dávila, M. Pessoa	Test Process Models: Systematic Literature Review (Garcia et al., 2014)	Atliekant sisteminę literatūros analizę, siekta ištirti, kurie TPI/TMA modeliai buvo apibrėžti, prigyje, pratęsti nuo 1990 m. Analizėje nagrinėta 23 modeliai.

2017	V. Garousi, M. Felderer, T. Hacaloğlu	What We Know about Software Test Maturity and Test Process Improvement (Garousi, Felderer, & Hacaloglu, 2017)	Buvo atlikti plataus spektro literatūros analizė, siekiant ištyrinėta, kas žinoma apie TMA/TPI modelis iki šiol ir taip padėti programinės įrangos inžinieriams lengviau spręsti iššūkius, susijusius su testavimo procesu.
2014	S. Eldh, K. Andersson, K. Wiklund	Towards a Test Automation Improvement Model (TAIM) (Eldh et al., 2014)	Straipsnyje yra pasiūlytas būtent automatinio testavimo modelis TAIM, kuriuos siekiama užpildyti kitų modelių spragas dėl per mažai dėmesio, skiriamo itin svarbiam aspektui – automatizavimui.
2018	Yuqing Wang	Test automation maturity assessment	Esami testavimo proceso gerinimo modeliai apima tik bendras testavimo idėjas. Dėl šios priežasties straipsnyje buvo nagrinėjama, kurie faktoriai yra esminiai, siekiant aukšto automatinio testavimo brandos lygio ir kaip tą lygį įsivertinti.
2012	K. Gaurav, K. Pradeep	Impact of Agile Methodology on Software Development Process (Gaurav & Pradeep, 2012)	Straipsnyje nagrinėta, kaip agile metodologija veikia programinės įrangos procesus, dėmesį kreipiant į produkto kokybę.
2020	A. Agarwal, S. Gupta, T. Choudhury	Continuous and Integrated Software Development using DevOps (Agarwal et al., 2020).	Atliktoje sisteminėje literatūros analizėje, nagrinėta metodologijos bei įrankiai, padedantys efektyviai įgyvendinti CD/CI praktikas.