Tallinna Ülikool

Informaatika Instituut

Testimisprotsesside parendamine firma Navionics näitel

Bakalaureusetöö

Autor: Sandra Saartok

Juhendaja: Inga Petuhhov

Autor: ..... „ ..... „ 2015

Juhendaja: ..... „ ..... „ 2015

Instituudi direktor: ..... „ ..... „ 2015

Tallinn 2015

Autorideklaratsioon

Deklareerin, et käesolev bakalaureusetöö, mis on minu iseseisva töö tulemus, on esitatud Tallinna Ülikooli lõpudiplomi taotlemiseks Informaatika erialal, ning tööd ei ole varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed, on viidatud.

……..	……..
(kuupäev)	(autor)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Sandra Saartok (sünnikuupäev: 02.08.1987)

1. annan Tallinna Ülikoolile tasuta loa (lihtlitsentsi) enda loodud teose ”Testimisprotsesside parendamine firma Navionics näitel”, mille juhendaja on Inga Petuhhov, säilitamiseks ja üldsusele kättesaadavaks tegemiseks Tallinna Ülikooli Akadeemilise Raamatukogu repositooriumis.
2. olen teadlik, et punktis 1 nimetatud õigused jäävad alles ka autorile.
3. kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid õigusi.

Tallinnas,

…….. ……..

(kuupäev) (allkiri)

Sisukord

Sissejuhatus

Navionics on Itaalia ettevõte, mis tegeleb merede, järvede ja jõgede elektrooniliste navigatsioonikaartide ning süsteemide arenduse ning tootmisega. Firma asutati aastal 1984, kui Giuseppe Carnevali ja tema partner Fosco Bianchetti tutvustasid ning tõid turule maailma esimese elektroonilise kaardiseade Geonav. (Tänavsuu, 2013)

Ettevõte on tõusnud selles valdkonnas maailma juhtivaks ning omab suurimat erakasutuses olevat mere- ja järvekaartide andmebaasi, mis katab kogu planeedi soolaseid veekogusid ning kümneid tuhandeid järvi ja jõgesid üle maailma. Navionicsi kartograafiline andmebaas koosneb enam kui 25 000 kaardist ja sadamaplaanist, millest suur osa on loodud Navionicsi enda tehtud uuringutega. (Postimees, 2013)

Navionicsi peakontor asub Itaalias, kuid omab harukontoreid ka USAs, Indias ning aastast 2013 ka Eestis. Peakontor juhib kogu teadus- ning arendusprotsessi ning ülemaailmset tootmist, USAs paikneb turundusosakond ning Indias ja Eestis on avatud arenduskeskused. Eesti üksus tegeleb põhiliselt veebi- ning mobiilirakenduste arendamisega.

Elektroonilisi kaarte luuakse endiselt GPS kaardiplotterile, kuid oluliseks arengusuunaks on veebi- ning mobiilirakenduste arendamine samavõrd detailsete kaartidega kui plotteril. See võimaldab saada üha rohkem kasutajaid, kuna tehnoloogia areng on turule toonud väga palju erinevaid mobiilseid seadmeid, mille peal rakendus töötab, ning enam ei pea kasutajad tingimata endale kaardiplotterit soetama. Lisaks annab see arengusuund ka võimalusi pakkuda kasutajatele üha rohkem lisafunktsionaalsust, mis võib merel viibides kasulikuks osutuda. Mobiilirakenduste loomine algas 2008. aastal esialgu vaid iOSi platvormile, aastast 2010 algas arendus ka Androidil. Lisaks merekaartide mobiilirakendustele hakati samast koodibaasist arendama ka suusatamise rakendust Ski, iOSi jaoks 2010. aastal ning Androidil alles 2013. Aastal 2013 algas iOSi arendus matkamise rakendusele Hike&Bike, Androidis sellega veel algust tehtud ei ole.

Ettevõte plaanib veel jõudsalt kasvada ning tulevikusuunana nähakse just mobiilirakenduste arendamist. Sellega seoses on tekkinud vajadus ühtse tööprotsessi järele, mille juurutamist alustati selle aasta alguses. Paika pandi rollid, vastutusalad ning üldine töövoog. Ettevõtte teiseks suuremaks eesmärgiks sel aastal on rakenduste kvaliteedi parandamine, mis tänaseni on olnud teisejärguline. Siiani ei ole oluliselt

pööratud tähelepanu testimisele ning kvaliteeti on mõõdetud vaid regressioonitestide põhjal, mida hallatakse MS Excelis ning mis on iganenud või rakenduse funktsionaalsuse katvuse mõistes puudulikud. Selle aasta alguseni polnud iga meeskonna liikmete seas kvaliteediinseneri, kes tegeleks igapäevaselt rakenduse testimisega ning aitaks tagada kvaliteeti. Täna on jõutud arusaamisele, et senine strateegia pole jätkusuutlik ning on vaja juurutada ka ühtne testimisprotsess, mida rakendada meeskondades.

Töö autor on tänaseks töötanud ettevõttes mobiilirakenduste arendusmeeskonnas kvaliteediinsenerina veidi üle aasta. Kuna autor on varemgi samal ametikohal töötanud, siis alates esimestest päevadest Navionicsis on ta olnud töös esitatud probleemidega tuttav ning üritanud neile lahendusi leida. Kuigi on olnud nii positiivseid kui ka negatiivseid kogemusi oma igapäevatöös, on töö autor siiski suutnud rakendada erinevaid testimismeetodeid edukalt ühe meeskonna piires ning aidanud sellega rakenduste kvaliteeti tõsta. Toetudes kogemustele, keskendub autor nüüd ühtse testimisprotsessi juurutamisele kõikide meeskondade piires.

Käesoleva töö põhieesmärgiks on ühtse testimisprotsessi loomine ja juurutamine. Eesmärgi saavutamiseks analüüsib autor testimise hetkeseisundit mobiilirakenduste arendusmeeskondades ja leiab selle kitsaskohad. Lisaks analüüsib töö autor erinevaid testihaldussüsteeme eesmärgiga asendada MS Excel, loob testidele struktuuri ning uued testilood, mis katavad funktsionaalsuse. Töös kajastatakse analüüs, kirjeldatakse loodud testide struktuur ning viis, kuidas valmisid testilood.

Töö tulemusena liigutatakse komplekt testilugusid MS Exceli arvutustabelist testihaldussüsteemi, leitud lahendused katsetatakse läbi meeskondade peal ning tulemused kajastatakse käesolevas töös. Lisaks valmib ka lühike testimisjuhis mobiili arendusmeeskondade jaoks, mis samuti katsetatakse läbi meeskondades. Kuna ettevõte on rahvusvaheline ning ametlikuks töökeeleks on inglise keel, on töö tulemid ingliskeelsed.

Töö on jaotatud neljaks peatükiks, kus esimeses peatükis antakse ülevaade testimisest, teises kirjeldatakse Navionicsi tööprotsessi ning põhilisi probleeme. Kolmas peatükk keskendub põhiliselt olemasolevate regressioonitestide kvaliteedi hindamisele ning parendamisele ning neljandas peatükis tuuakse välja olulisemad järeldused.

1 Ülevaade testimisest

Peatüki eesmärk on anda ülevaade testimisest, et edaspidine töö lihtsamini arusaadav oleks. Autor selgitab lahti, mis on testimine, miks see vajalik on ning toob välja testimise liigid. Kuna valdkond on väga lai, ei ole eesmärgiks anda põhjalikku ülevaadet, vaid keskenduda sellele, mis on edaspidise tööga seotud.

1.1 Testimine

ISTQB (*The International Software Testing Board*) definitsiooni järgi on testimine protsess, mis koosneb kõigist tarkvara elutsüklis sisalduvatest tegevustest, mis tegelevad tarkvaraprodukti ja seotud töötulemite hindamise planeerimise, ettevalmistamise ja läbiviimisega, et selgitada välja, kas ja mil määral nad vastavad nõuetele, näidata nende sobivust eesmärgi saavutamiseks ning leida vigu. (Graham, Van Veenendaal, Black, & Evans, 2008, lk 18)

Olemuselt paikneb testimine tarkvaraarendusprotsessis nõuete formuleerimise, analüüsi ning realiseerimise vahel. Testimise sisendiks on ühelt poolt süsteemile esitatavad nõuded ning teiselt poolt testitav objekt. Põhiliseks väljundiks on leitud vigade kirjeldused, millele toetudes süsteemi ja vajadusel ka nõudeid parandatakse ning ka testide kirjeldused, testiplaanid ja aruanded. (Markvardt, 2011)

Tarkvara mittevastavust nõuetele nimetatakse veaks ning vigu on võimalik leida testimise käigus. Vead on tarkvarasüsteemide keerukuse tõttu tarkvara loomise kaasnähtus. Nende tekkimist ei ole võimalik täielikult vältida, kuid efektiivse testimisprotsessi olemasolul on võimalik suur osa olulistest vigadest leida. Vigade tekkimisel on mitmeid põhjuseid, sealhulgas nõuete ebatäpsus, arendusprotsessi üldine keerukus, varasemate vigade parandamine ning paranduste käigus uute tekitamine. (Markvardt, 2011)

Vigade leidmise aeg on väga oluline: praktika on näidanud, et mida varasemas tarkvara elutsüklis viga leitakse, seda odavam on üldjuhul selle parandamine. Sellest lähtuvalt on testimise üheks põhieesmärgiks kindlasti ka oluliste vigade leidmine võimalikult varajases arengufaasis. (Markvardt, 2011)

Testimine on tarkvaraarendusprotsessi lahutamatu osa ning kvaliteedi tagamise valdkonna (*quality assurance*) peamine alamvaldkond. Testimine ise kvaliteeti ei tõsta,

vaid annab infot selle hindamiseks. Et kvaliteeti tõsta, on vajalik testimise käigus leitud vead parandada, neist õppida ning neid edaspidi ennetada. (Markvardt, 2011)

Oluline on märkida ka, et kuigi testimine on oluline, on täielik testimine (*exhaustive testing*) sisuliselt võimatu, kuid sellegipoolest tuleks testimist alati alustada nii varakult kui võimalik. (Graham, Van Veenendaal, Black, & Evans, 2008, lk 13)

1.2 Testimise liigid

Vastavalt eesmärgile on võimalik eristada erinevaid testimise liike. Liigitused võivad olla omavahel kattuvad vastavalt sellele, millest testimismeetodi puhul lähtutakse. Testimisvajadused sõltuvad süsteemile püstitatud nõuetest, seega ei ole vaja kõiki testimise liike alati rakendada. Järgnevalt on toodud definitsioonid mõnele enamlevinud testimisliigile.

Koodi läbivaatus (*code review*)

Koodi ei käivitata, vaid testimine piirdub koodi inspekteerimisega testija poolt (ISTQB, 2007, lk 33). Läbivaatus eeldab küsimustiku olemasolu, mille alusel hinnatakse koodi ning otsitakse vigu, mis isegi koodi käivitamisel ei pruugi avalduda. (Markvardt, 2011)

Moodultestimise (*component testing*)

See on eraldiseisva tarkvaramooduli testimine. Mooduli iga meetodi jaoks koostatakse test, mis on eraldiseisev mooduli teistest testidest. Moodultestimine on tarkvaraarendajate vastutusel ning selle eesmärk on vigade leidmine, mitte nõuetele vastavuse kontroll. (Markvardt, 2011)

Integratsioonitestimise (*integration testing*)

Testitakse erinevate moodulite/komponentide koostööd. Kontrollitakse, ega liidestamisel ei esine vigu. Läbiviijaks on enamasti sõltumatu testija või teine arendaja. (Markvardt, 2011)

Funktsionaalne testimine (*functional testing*)

Testimise eesmärgiks on kontrollida süsteemi vastavust funktsionaalsuse nõuetele. Üldjuhul on loodud testid, mille põhjal kontrollimine käib. Läbiviijaks on testija. (Markvardt, 2011)

Suitsutestimine (*smoke testing*)

See on pealiskaudne testimine, millega tehakse kindlaks, kas põhifunktsionaalsus töötab (ISTQB, 2007, lk 31). Tavaliselt teostatakse suitsutest enne põhjalikumat testimist. Automatiseeritud suitsutestid võiksid olla esimene samm testimise automatiseerimise suunal.

Uuriv testimine (*exploratory testing*)

See on testimine, kus testilood tulenevad testimise tulemusest ning kus testija kasutab saadud informatsiooni, et luua testimise käigus uued testid (ISTQB, 2007). Alusena võib kasutada eelnevalt dokumenteeritud testlugusid, mida testija laiendab vastavalt oma mõtetele. Kõik testijad praktiseerivad mingil määral uurivat testimist.

***Ad hoc* testimine (*ad hoc testing*)**

See on programmi mittesüstemaatiline testimine juhuslike, mittedokumenteeritud testilugudega (ISTQB, 2007). Seda ei ole soovitav pikka aega kasutada, kuna sellise testimise puhul pole võimalik hinnata testide katvust, sest mingit dokumentatsiooni läbiviidud testide kohta ei säili. Eelnevalt mainitud põhjustel sõltub testimise efektiivsus oluliselt testija kompetentsusest.

Regressioonitestimine (*regression testing*)

Tarkvara testimine pärast iga muudatust eesmärgiga teha kindlaks, kas muudatused ning funktsionaalsuse lisamine pole tekitanud vigu varem toiminud programmi osas. Regressioonitestimist on kasulik automatiseerida üldise töökoormuse vähendamiseks. (Markvardt, 2011)

Vastuvõtutestimine (*acceptance testing*)

Testitakse süsteemi vastavust kasutaja nõuetele, kontrollitakse tüüpilisi kasutaja tegevusi. Üldjuhul on teostajaks klient. (ISTQB Exam Certification)

2 Tarkvaraarendusprotsess Navionicsis

Peatükis kirjeldab töö autor põhilisi probleeme, mis siiani arendusprotsessis on esinenud ning uut, aasta alguses juurutatud protsessi. Lisaks toob ta välja peamised rollide mõisted

ja kirjeldused ettevõttes ning annab ülevaate arenduses kasutusel olevatest tööriistadest ning töövoogudest.

2.1 Probleemid arendusprotsessis

Kõige suuremaks probleemiks arenduse juures saab pidada ühtsete tööprotsesside puudumist. Arenduses ei ole defineeritud rollid ja nende peamised vastutusalad ning arenduse üldine teekaart (*road map*) ei ole paika pandud. Järgnevalt on toodud välja põhilised probleemid, mis ajendasid ühtset protsessi looma.

Nõuded

Puuduvad täpselt defineeritud nõuded ning nende täielik muutumine arenduse käigus on tavaline nähtus. Nõuete muudatused kuskil ei kajastu, parimal juhul levivad suulisel teel asjaosalistele.

Planeerimine

Puudub arendustegevuste planeerimine. Järgmise väljalaske sisu on üldjoontes teada, kuid enamasti muutub ka see üsna tihti. Muutuse põhjuseks on enamasti see, et lisatakse ootamatult funktsionaalsust juurde või tuleb kõrgemalt otsus, et muu funktsionaalsus on tähtsam. Toote õigeaegne väljastamine muutub sellega seoses praktiliselt võimatuks. Planeerimisel ei osale ka kogu meeskond, hinnangud antakse kõhutunde järgi ning ei arvestata juurde funktsionaalsuse testimist.

Lisaks puudub pikemaajaline ülevaade, mida ning millal arendama hakatakse.

Arendusfaas

Arendusülesanded ei ole enamasti kuskil kirjas. Kasutusel on küll vea- ning ülesannetehaldussüsteem JIRA, kuid tihti sinna uusi ülesandeid kirja ei panda. On teada, et kogu funktsionaalsuse areng peaks seal kajastuma, kuid keegi otseselt seda ei kontrolli ning kasutatavus on olematu. Kuna puudub dokumentatsioon, siis tuleks enne arenduse alustamist luua kasutajalood täpse kirjeldusega, milline on oodatud käitumine ning arendus peaks käima selle põhjal.

Testimine

Kõige suuremaks probleemiks võib pidada seda, et testimist ei peeta arendustegevuseks ning arendusfaasi osaks, mida ta tegelikult on. Siiani ei olnud kõikides meeskondades

kvaliteediinseneri ning selle töö pidid ära tegema arendajad ise. Kuidas ning kui põhjalikult testima peaks, määratud ei olnud, ning üldjuhul piirdus see sellega, et arendaja vaatas kiiresti enda töö üle.

Kvaliteediinsener oli olemas vaid ühes meeskonnas, keda tihti isegi ei kaasatud arendusse algusest peale, vaid alles siis, kui funktsionaalsuse arendus oli sisuliselt juba valmis. Seegi piirdus sellega, et talle öeldi täpselt ette, mida ning kui palju testida ning enamasti oli ees ka ajaline piirang. Põhjaliku testimise jaoks vajadust ei nähtud. Lisaks puudus konkreetne versioon testimise jaoks, tihti olid vaid lokaalselt ehitatud versioonijärgud, millel puudusid versiooninumbrid, mille põhjal neid eristada. Ülevaade vigadest levis tihti suuliselt või e-kirja teel.

Lisaks on probleemiks ka see, et siiani on kasutatud põhiliselt *ad-hoc* testimist, mille puhul ei ole peale vearaportite mingit väljundit. Tänaseks on selge, et on vaja testimistegevusi dokumenteerida, et rakenduste üleüldine kvaliteet paraneks.

Olulise probleemina võib välja tuua ka testihaldussüsteemi puudumise.

Tegelikkuses tuleks testimine koos arendusega planeerida ning seda tuleks alustada võimalikult varakult, et suuremad probleemid arenduse alguses kiiremini välja tuleksid. Lisaks on kasulikum, kui kvaliteediinsener on meeskonna liige, sest siis on ta projekti üldise käigu ning eripäradega kursis ning omab ülevaadet muudatustest. Selle põhjal saab ta koostada testiplaanid ning saab otsustada, kuidas ning kui põhjalikult funktsionaalsust testima peaks.

Ainukese kohustusliku testimistegevusena on paika pandud, et enne igat väljalaset tuleb teha regressioonitest kogu funktsionaalsusele. Selle põhjal hinnatakse versiooni kvaliteeti, kuid sellega on omaette probleemid:

- Regressioonitestimisel puudusid eeldused, seda alustati tihti siis, kui tähtaeg hakkas lähenema, hoolimata sellest, kas arendus on lõpetatud või ei. Enamasti on funktsionaalsuse arendus regressioonitestimise alguseks lõpetamata.
- Regressioonitesti jaoks tuleks kasutada stabiilset versiooni, kus enam ei käi aktiivne arendustegevus. Kuna enamasti algab regressioonitest siis, kui arendus on veel poolik, on üsna tihti olukord, kus regressioonitestimiseks kasutatakse mitut erinevat versiooni. Kuna samal ajal samadel versioonidel toimub vigade parandamine ja vahel ka funktsionaalsuse arendus, siis tekib üldjuhul vigasid veel

rohkem juurde ning regressioonitesti tulemused ei anna mingit infot üldise kvaliteedi kohta.

- Regressioonitesti tehakse tegemise pärast (kuna see on ainuke paika pandud protsess), mitte kontrollimaks funktsionaalsust.
- Regressioonitestid on iganenud ja testid ei kata kogu funktsionaalsust. Testid on kirjutatud ammu ja neid ei ole muudatustega uuendatud, testimise käigus tuleb enamasti üle kontrollida, kas viga on testis endas või arenduses.
- Regressioonitestimise organiseerimine on võrdlemisi keeruline: testilugusid hoitakse MS Excelis, kuhu tuleb märkida testimise tulemused. Testi teostab kogu meeskond, mis omakorda tähendab väga suurt lisatööd, et jagada dokument kõikide asjaosalistega ning pärast tulemused ühte tabelisse kokku koondada.
- Testimise käigus leitud vigade puhul tuleb kontrollida, kas need esinesid ka eelmises versioonis. Kui viga esines ka eelmises versioonis, siis ei ole see parandamiseks piisavalt tähtis. Kui ei esinenud, tuleb see käesoleva väljalaske jaoks parandada.
- Testimise käigus leitud vead hoitakse tekstidokumendis, veahaldussüsteemi neid ei märgita. Testimise lõpus vaadatakse kõik leitud vead üle ning olulisemad liigutatakse veahaldussüsteemi.

2.2 Arendusprotsess mobiilirakenduste meeskondades

Eelmises peatükis toodud probleemide lahendamiseks juurutati aasta alguses uus tööprotsess ning defineeriti rollid ja täpsed vastutusalad. Järgnevalt on toodud nende kirjeldus.

Arendusprotsessi parendamisel ei lähtutud kindlast tarkvaraarendusmeetodist, kuid võeti eeskujuks agiilsed metoodikad. Agiilne tarkvaraarendusmetoodika on tarkvara arenduse meetodite grupp, mis baseerub tsüklilisel arendusel, kus nõuded ja lahendused arenevad läbi iseorganiseeruvate meeskondade koostöö (Subramaniam & Hunt, 2006).

Scrum on enim tuntud agiilse arenduse raamistik, mis sisaldab reegleid, rolle ja aja haldamist. Arendus on jagatud tsükliteks, mida kutsutakse sprintideks. Sprint on enamasti kaks kuni neli nädalat pikk ning sellele on seatud kindel lõppeesmärk. Scrumis on täpselt paika pandud rollid ja strateegilised koosolekud. Rollide hulka kuuluvad Scrumi meister, tooteomanik ning meeskond. Scrumi meister vastutab põhiliselt Scrumi protsesside

toimimise eest ega sekku meeskonna töösse. Tooteomanik vastutab toote visiooni ning meeskond visiooni realiseerimise eest. Strateegiliste koosolekute hulka kuuluvad sprindi planeerimine (*sprint planning*), sprindi ülevaatus (*sprint review*), sprindi tagasivaate koosolek (*sprint retrospective*) ja igapäevased lühikoosolekud (*daily standups*). Planeerimiskoosolekul määratakse arendustööd sprindi jaoks ning seatakse lõppeesmärk. Ülevaatus on informatiivne koosolek kliendiga, kus arutatakse, mis läks hästi ning mis halvasti. Tagasivaatekoosolek on meeskonnasisesene, kus arutatakse sprindi parendamisvõimalusi. Igapäevased koosolekud ei kesta üle 15 minuti ning seal selgitatakse välja tehtud ja planeeritavad tööd ning arutletakse võimalike takistuste üle.

Navionicsis küll ei rakendatud konkreetset raamistikku, kuid võeti enim arvesse Scrumi elemente. Rollid defineeriti ettevõttespetsiifiliselt, kuid kasutusele võeti sprindi planeerimine ning igapäevaste koosolekute pidamine. Arendus jagati kahenädalasteks sprintideks ning iga meeskond vastutas, et vähemalt kuue nädala arendustegevused oleks ette planeeritud.

2.2.1 Rollid arendusprotsessis

Järgnevalt on kirjeldatud olulisemad rollid ettevõtte arendusmeeskondades, juurde on lisatud kirjeldused ning tähtsamad vastutusalad.

Platvormi tootejuht (*Platform Product Owner*) on tootejuht, kes vastutab kogu platvormi arendustegevuste eest. Ta defineerib arendatava funktsionaalsuse koos programmijuhiga ning määrab prioriteedid. Ta loob funktsionaalsuse jaoks kõrgema taseme ülesande (*epic*) JIRAsse ning hoolitseb kõikide meeskondade tegemata tööde nimekirja eest. Lisaks hoolitseb ta ka selle eest, et meeskonnajuhid looksid ülesannete põhjal detailsemad kasutajalood (*user stories*), mille põhjal algab arendus. Kui funktsionaalsus ei ole piisavalt arusaadav, on tema ülesandeks täpsemate nõuete väljaselgitamine, et kasutajalood saaks piisavalt arusaadavalt kirja. Kohustuste hulka kuuluvad veel väljalaskeplaanide (*release plans*) kinnitamine kogu platvormi piires ning ärivajaduste edastamine meeskondadele. (Navionics, 2015)

Programmijuht *(Feature Product Owner)* on sisuliselt tootejuht, kes vastutab konkreetse funktsioonaalsuse eest. Tema peamiste ülesannete ja kohustuste hulka kuuluvad funktsionaalsete nõuete defineerimine ja prioriteetide määramine, koostöös platvormi tootejuhiga. Ta hoolitseb konkreetse funktsionaalsuse eest, hindab vajadusel tööde loetelu ümber ning vastutab, et meeskondadele jõuaks vajalik informatsioon. Ta aitab

defineerida väljalasete sisu ning kinnitab nende plaanid funktsiooni skoobi piires. Ta ei juhi inimesi ega sekku arendusmeeskonna igapäevasesse töösse, vaid vastutab ainult konkreetse funktsionaalsuse eest. (Navionics, 2015)

Tehniline juht (*Technical Lead*) on ühtlasi ka meeskonnajuht. Ta hoolitseb tegemata tööde nimekirja eest koostöös platvormi tootejuhiga. Tema kohustuste hulka kuuluvad kasutajalugude loomine ning sprindi ja väljalasete sisu kinnitamine. Ta haldab tehnilisi ülesandeid JIRAs ning omab õigust otsustada, kuidas need tuleb lahendada. Ta kontrollib väljalasete arendusprotsessi ning jälgib, et arendustegevused püsiks graafikus. Vajadusel toetab ta teisi meeskonnaliikmeid ning vastutab igapäevaste tegevuste eest. (Navionics, 2015)

Kvaliteediinsener (*Quality Engineer*) on professionaal, kes on kaasatud süsteemi või komponendi testimisse. Tema ülesanneteks on toetada meeskonda, verifitseerides tehtud tööd, testides süsteemi vastavust nõuetele. Lisaks peab ta omama ülevaadet tehtavatest töödest ning koostama testidokumentatsiooni. Samuti valmistab ta ette regressioonitestimise testiplaanid, osaleb ise regressioonitestimises ning annab hinnangu väljalaskekandidaatide kohta. (Navionics, 2015)

Meeskond (*Team*) koosneb arendajatest, kelle ülesanneteks on arendada süsteemi komponente vastavalt prioriteetidele ning tööde loetelu eest hoolitsemine koostöös tehnilise juhiga. Arendajad keskenduvad vaid konkreetse sprindi ja/või väljalaskega seotud tegevustele enda meeskonnas ega saa ülesandeid väljastpoolt. Abi saamiseks pöörduvad teiste meeskonnaliikmete või -juhi poole. (Navionics, 2015)

2.2.2 Koodi töövoog

Mobiilirakenduste arendus toimub ühes koodibaasis ja kvaliteedi tagamiseks on juurutatud ühtne protsess ka koodi jaoks, mida peavad kõik meeskonnad järgima. Arendus toimub pidevalt ja oluline on hoida stabiilsed versioonide harud eraldi teistest harudest.

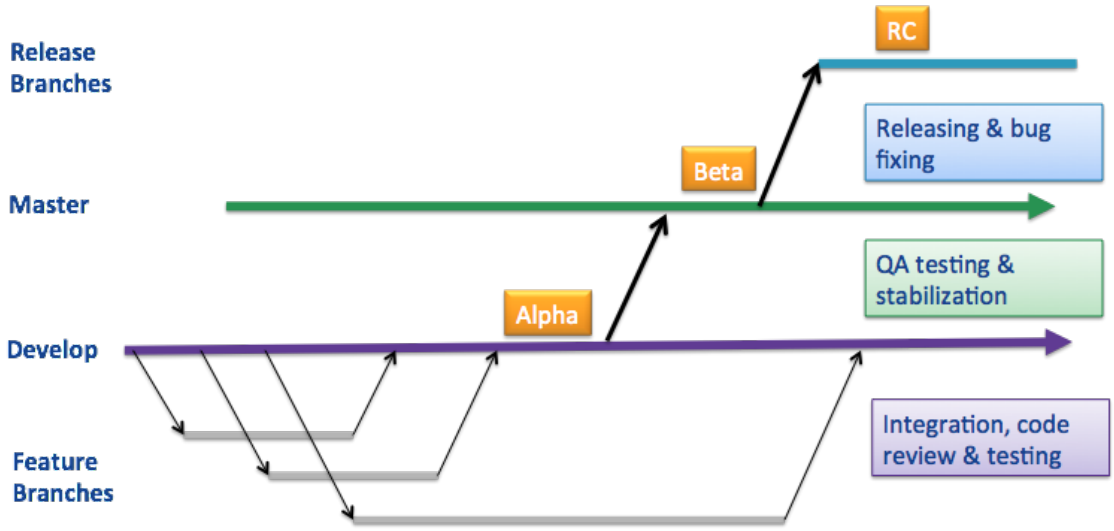
Kasutusel on versioonihaldussüsteem Git, mis võimaldab luua korraga mitmeid harusid. Kohustuslikud harud on defineeritud järgmiselt:

- **Põhiharu (*master branch*)**: selles harus on ainult rakenduste versioonid, mis on valmis viimseks testimiseks enne väljalaset. Funktsioonid integreeritakse arendusharus ning mestitakse seejärel põhiharuga. Selles harus on aktiivne

arendus lõpetatud ning kvaliteediinsenerid alustavad regressioonitestiga. Võivad toimuda väiksemad veaparandused.

- **Arendusharu (*develop branch*)**: selles harus integreeritakse ning testitakse uued funktsioonid. Siin on enamasti veel poolik töö, aga nõue on, et see ei ole kunagi katki ning rakendused peavad kompileeruma. Enne integratsiooni funktsiooniharust peab arendaja üle testima enda muudatused ning veenduma, et muudatused ei lõhu midagi ära. Sellest harust luuakse automaatselt igapäevased versiooni järgud (*daily builds*).
- **Funktsiooniharu (*feature branch*)**: siin toimub uute funktsionaalsuste aktiivne arendus. Iga haru on ühendatud kasutajalooga, viimase ID on märgitud haru nimetuseks. Harud eksisteerivad lühikest aega (kuni kolm päeva) ja mestitakse arendusharuga nii kiiresti kui võimalik. Enne mestimist tuleb teha koodiülevaatus (*code review*) ja testida, võimalusel kaasata kvaliteediinsener.
- **Väljalaskeharu (*release branch*)**: selles harus on ainult rakenduste versioonid, mis on valmis turule laadimiseks. Siin on alati stabiilne versioon rakendusest. Parandusi siin enam ei tehta.

Tavaliselt algab uue kasutajaloo arendus uue funktsiooniharu loomisega arendusharust. Arenduse käigus funktsiooniharu uuendatakse pidevalt arendusharuga. Kui arendus on funktsiooniharus lõppenud, siis arendaja on kohustatud enda töö lokaalselt üle testima, koodile tehakse ülevaatus teise arendaja poolt ning viimaks integreeritakse muudatus arendusharuga. Kogu tiim vastutab selle eest, et versiooni järk (*build*) ei läheks katki arendusharus. Kui rakenduse ehitamine ebaõnnestub, on muudatuse teinud arendaja kohustus probleem koheselt kõrvaldada. Eesmärgiks on hoida rakendus selles harus kogu aeg töötavana. Sellesse harusse integreeritakse mitmed erinevad funktsioonid kuni väljalaske sisu on koos ja arendus jääb seisma. Kui suuremaid vigu ei esine, mestitakse arendusharu põhiharuga, kus algab regressioonitest. (vt Joonis 1) (Navionics, 2015)



Joonis 1. Koodi töövoog (Navionics, 2015)

2.2.3 Tööde haldamine ja ülesannete töövoog

Tööde haldamise süsteem on iga projekti jaoks hädavajalik. Seal kajastatakse kõik tööd, mida projektiga seoses tehakse ning see annab hea ülevaate projekti seisust. Seal on selge ülevaade, mis funktsionaalsus mis versioonis peaks valmima ning juba töös olevate ülesannete puhul on selgelt näha, millises staatuses need on.

Navionicsis kasutatakse tööde haldamiseks kõikides meeskondades Atlassian JIRAt. JIRA on ühendatud ka Gitiga ning tööprotsess näeb ette, et iga koodimuudatuse salvestamisega (*commit*) tuleb lisada JIRA ülesande ID. Kuna üheks suuremaks probleemiks meeskondades on olnud ka see, et JIRAsse üldiselt ülesanded ei jõua, siis uue tööprotsessiga määrati täpne viis, kuidas ülesandeid hallata ning kirjeldus töövoogude kohta.

Uue funktsionaalsuse arendamine peab alati algama kasutajalugude loomisega JIRAsse. Selle eest vastutab meeskonnajuht ning kõik kasutajalood peavad olema seotud funktsionaalsuse kõrgema taseme ülesandega (*epic*). Loodud kasutajalood on esialgu tööde nimekirjas *Open* olekus. Samamoodi on defineeritud ka vigade loomine JIRAsse. Vigasid võib raporteerida igaüks, kuid enne töösse võtmist tuleb neile määrata prioriteet ja versioon.

Tehnilise juhi ülesandeks on aeg-ajalt üle vaadata kõik lisatud ülesanded ja vead ning määrata neile prioriteedid. Kui ülesanded on analüüsitud, tuleb neile määrata versioon ja

seejärel saab need planeerida sprinti. Iga meeskonna liige saab ise sobiva ülesande töösse võtta, vajadusel abistab neid selle juures meeskonnajuht.

Arendusfaasis töötab kogu meeskond sprindi/väljalaske ülesannetega. Kui ülesandega alustatakse, siis muudetakse selle staatus *In Progress* olekusse. Kui ülesanne saab tehtud, peab ülesande täitnud arendaja muutma staatuse *Code Review* olekusse. Koodi ülevaatuse teostab üldjuhul meeskonna tehniline juht või tema poolt määratud arendaja. Kui ülevaatus on edukas, siis mestitakse funktsiooni haru arendusharuga ning ülevaataja liigutab ka JIRAs ülesande *QA Review* olekusse. Selles staatuses ülesanne tuleb kvaliteediinseneri poolt üle testida. Kui testimise käigus vigu ei leita, pannakse ülesanne *Closed* olekusse. (Navionics, 2015)

Sprindi eesmärk on üldjuhul kõikide planeeritud ülesannete liigutamine *Closed* staatusesse, kuid tihti see ei õnnestu. Kui sprindi lõpuks jääb midagi tegemata, liigutatakse see järgmisse sprinti.

Ülesannete liigutamine ja seos harudega on esitatud tabelis (vt Tabel 1).

Tabel 1. Ülesannete liigutamine (Navionics, 2015)

Issue state	Branch				Definition
	Feature	Develop	Master	Release	
Open					Issue added to backlog, enough information added to start development as soon as issue assigned to release and sprint
In progress	X				Development has started, branch name with issue ID has been created in Git.
Code review	X				Development is complete, pull request for a code review after which issue will be merged to develop branch.

Issue state	Branch				Definition
	Feature	Develop	Master	Release	
QA review		X			Branch has been merged on develop, integration is complete and quality engineer must review the changes.
Closed			X		Changes have become part of the platform release, development and QA testing is complete.

2.3 Testimisprotsess

Uue arendusprotsessi juurutamisega jõuti arusaamisele, et kvaliteediinseneril on meeskonnas tähtis roll. Ta peab osalema arenduse juures algusest peale ning andma hinnangu väljalasetele. Sellest tulenevalt värvati ettevõttesse kvaliteediinsenerid, kes paigutati konkreetsetesse meeskondadesse. Uute värbamiste ning arusaamisega, et kvaliteeti ei saa hinnata vaid regressioonitestimise põhjal, on vaja defineerida ka ühtsed testimistegevused kogu arendusfaasi jooksul kõikides meeskondades.

Töö autor vastutas suures osas testimistegevustele parendamisettepanekute leidmise ning nende rakendamise eest. Kuna autor nägi suurt probleemi testihaldussüsteemi puudumises, siis sai parendamine alguse testihaldussüsteemi valikuga. Peatüki eesmärk on seda kirjeldada. Lisaks kirjeldab autor peatükis uut testimisprotsessi, mille autor lõi, toetudes uuele arendusprotsessile ning enda kogemusele antud ettevõttes kvaliteediinsenerina töötades. Protsessi põhjal valmib ka juhis, mida testimisel silmas pidada. Dokument laetakse üles ettevõtte siseveebi ja on leitav antud töö lisades (vt Lisa 1).

2.3.1 Testihaldussüsteemi valik

Kuna üks suurematest probleemidest testide puhul seisneb selles, et neid hallatakse MS Excelis, siis oli vaja analüüsida erinevaid testihaldussüsteeme ning valida kasutuseks sobivaim. Kuigi antud töö keskendub ainult mobiili platvormi projektidele, tuli

testihaldamissüsteemi valikul silmas pidada ka seda, et sinna oleks võimalik lisada erinevaid projekte ka teistel platvormidel, millel olid osaliselt teistsugused nõuded süsteemile.

Autor toob välja põhilised kriteeriumid, millele tööriist vastama pidi. Need olid valdavalt määratud autori enda poolt, lähtudes projekti vajadustest. Vastavalt nõuetele analüüsis töö autor erinevaid tööriistu ning analüüsi põhjal tehti otsus, milline tööriist kasutusele võtta.

Põhilised kriteeriumid olid järgmised:

1. Hind. Eelistatud on kindlasti odavam või tasuta tööriist, kuid see ei ole määrav.
2. Seadistamine ja kasutajaliides. Tööriist peab olema lihtne üles seada, kasutajaliides peab olema võimalikult mugav ja lihtsasti arusaadav.
3. Kasutajad. Mitme kasutaja lisamine ja nende haldamine (võimalus lisada grupid ning määrata erinevad õigused).
4. Projektid ja nende haldamine. Võimalus kasutada erinevate projektide (nii mobiili kui veebi) puhul ning määrata erinevad seadistused vastavalt projektide eripäradele.
5. Testide haldamine ning käivitamine. Testplaanide, -lugude ning -pakettide loomine. Testilugude põhjal peab saama liigutada või kopeerida sama või erineva testiplaani piires ning peab olema võimalus testisammud eraldi välja kirjutada. Testpakettide puhul peab olema võimalus valida, milliseid testilugusid soovitakse sinna lisada. Kindlasti ei tohi olla võimalik vaid ühe testiplaani komplekti paketti lisamine.
6. Raportid. Raportite loomine testimistulemustest, nende eksportimise võimalus.
7. Integratsioon veahaldussüsteemidega (vähemalt JIRA). Võimalik peab olema testiloo sidumine JIRA ülesande või veaga. Siin tuleb arvesse võtta Navionicsi JIRA seadistust. Navionicsis on kasutusel JIRA OnDemand, kus JIRA asub Atlassiani enda serveris, mis võib seada mõned piirangud integratsiooniks.

Analüüsimiseks võeti erinevad tööriistad, nii tasuta kui ka tasulised. Tasuta tarkvara paigaldati katsetamiseks lokaalselt masinasse, tasuliste puhul kasutati demoversioone või ajaliselt limiteeritud prooviversioone. Katsetamiseks oli mitu erinevat tööriista: TestLink, Fitnesse, TestRail, Testia Tarantula, QABook, RTH, Trac, Quality Spy, Zephyr ning Testopia. Enamus valikus olevatest tööriistadest olid liiga lihtsad ega vastanud seatud

nõuetele, puudus võimalus luua testilugudest testipakette ning ei olnud võimalik saada testitulemusi. Mõned olid raskesti kasutatavad ja omasid liiga palju funktsionaalsust, millele ei leia Navionicsi projektide puhul kasutust. Lõplikusse valikusse jäid Testia Tarantula ja TestRail. TestRail on võimalik ka pilveteenusena, mis on Navionicsile eelistatum variant.

Hind

Testia Tarantula on tasuta.

TestRaili litsentsi hind sõltub kasutajate arvust, Navionicsile piisab 21-40 kasutajast, mis tähendab, et maksimaalne summa on €449 kuus. (TestRail)

Seadistamine ja kasutajaliides

Paigaldamine oli mõlema süsteemi puhul lihtne. TestRaili on võimalik seadistada rohkem vastavalt vajadustele ning lisada vajalikke väljasid.

Testia Tarantula puhul on kasutajaliides arusaadav, kuid kohmakas. Lisaks on see väga aeglane ning jookseb tihti lehekülje kerimise peale kokku.

TestRaili kasutajaliides on väga intuitiivne ning mugav. Seal on lisatud kohtspikrid (*tooltips*), mis on abiks algajale kasutajale.

Kasutajad

Mõlemas tööriistas on võimalik lisada kasutajaid ning määrata gruppe.

Testia Tarantula puhul on võimalik luua ülesandeid (*task*) ning määrata neile teostajad, samuti ka testipakettide puhul, kuid see on suhteliselt ebamugav. Testia Tarantulas on kasutusel sildid (*tags*) ja kui need on määratud, on üsna lihtne testipakette koostada. Siltide puudumisel on testipakettide loomine keeruline ning eeldab ükshaaval testide liigutamist paketti. See kujuneb probleemiks kui testilugusid on testiplaanis palju.

TestRailis on võimalik määrata testpaketile teostaja ja see on oluliselt lihtsam ja arusaadavam kui Testia Tarantula puhul. Kasutaja lisamine on lihtne, kustutamine ei ole võimalik, aga saab kasutaja mitteaktiivseks teha. See tähendab aga seda, et litsents on ostetud, kuid pole kasutusel. Maksma selle eest peab ikka. Olukorra lahendab uue kasutaja lisamine.

Projektid ja nende haldamine

Kummaski programmis on võimalik lisada projekte ning määrata erinevaid seadistusi erinevatele projektidele.

Testide haldamine ning käivitamine

Testilugusid on võimalik mõlemas tööriistas kirja panna sammudena.

Testia Tarantulas on kasutusel sildid (*tags*), millega seotakse omavahel testilood ning -plaanid, kuhu nad kuuluvad. Edaspidine töö testilugudega on võrdlemisi lihtne ja arusaadav, vaid siis, kui sildid on määratud.

Testia Tarantula puhul on testide liigutamine ja kopeerimine võimalik, kuid keeruline.

TestRailis on teste väga lihtsalt võimalik liigutada ning kopeerida ühest testiplaanist teise.

Raportid

Testimistulemused on TestRailis kujutatud visuaalselt diagrammidena, mida on võimalik ka eksportida.

Testia Tarantula raportid jäävad visuaalselt TestRaili omadele alla, kuid info on seal olemas. Võimalik ka eksportida.

Integratsioon veahaldussüsteemidega

Testia Tarantula puhul on integratsioon JIRAga teoreetiliselt võimalik. Probleemid tekivad OnDemand versiooniga, mis Navionicsis on kasutusel. Seadistamine võib põhjustada häireid JIRA töös, mis on liiga suur risk.

TestRailil on integratsioon JIRAga olemas. Testilugu on võimalik siduda ühe või mitme JIRA ülesande või veaga. Lisaks on võimalik JIRA vearaporti koostamine otse TestRailist, mis ühendatakse testilooga ka kohe ära.

Eelneva põhjal osutus sobivamaks tööriistaks TestRail, eelkõige mugava kasutajaliidese, paremate seadistamisvõimaluste ning JIRA integratsiooni tõttu. Kuigi tööriist on tasuline, sobib see meeskondade vajadustega paremini.

2.3.2 Testimisprotsessi parendamine

Testimistegevusi ajendas parendama uue arendusprotsessi juurutamine. Nagu varem öeldud on ülesannete ning vigade haldamiseks kasutusel Atlassian JIRA. Testide

haldamiseks on siiani kasutusel olnud MS Excel, kuid alates aasta algusest on paralleelselt kasutusel ka eelnevas alampeatükis kirjeldatud TestRail. Eesmärk on edaspidi täielikult loobuda MS Excelist ning jätkata vaid TestRailiga.

Testimistegevustega tuleb algust teha võimalikult varases arendusfaasis ning kvaliteediinsenerid peavad osalema ka planeerimiskoosolekul, et olla kursis eesmärkidega. Aasta algusest on värvatud juurde kvaliteediinsenere, kes on lisatud konkreetsetesse meeskondadesse. Nad osalevad kõikidel koosolekutel, mis mõjutavad tervet meeskonda ning funktsionaalsuse arendamist.

Kuna üldjuhul puudub arendustegevuseks nõuete põhjalik dokumentatsioon, on olukord lahendatud nii, et ülesande loomisel JIRAs tuleb lisada võimalikult täpne kirjeldus, mille põhjal on võimalik arendada valmis funktsionaalsus. Juba sealt algavad ka testimistegevused.

Tegevused arendusfaasi alguses

Uue funktsionaalsuse arenduse alguses tuleb luua TestRaili uus testiplaan. Planeerimiskoosolekul saab selgeks, mis on vajalik ja testimistegevused saavad alata juba enne arenduse algust. Kõikidele kasutajalugudele tuleb luua võimalikult vara testilood ning need omavahel siduda. Seda saab nimetada ka nii-öelda nõuete testimiseks. Juba selle käigus on võimalik leida esimesed mittevastavused ning funktsiooni täpne käitumine välja selgitada.

Kõikidel testilugudel on unikaalne tunnusnumber, kirjeldav nimi, kirjeldus ning oodatav tulemus. Igal testilool on ka omanik ning ajalugu, mille põhjal on võimalik jälgida muudatusi. Kui nõuetes peaks tulema muudatus, siis luuakse selle kohta uus kasutajalugu, mis seotakse eelmisega. Juba sealt saab selgeks, et vajalik on muuta ka loodud testilugu.

Tegevused arendusfaasi ajal

Arendusfaasi ajal jätkub funktsionaalsuse katmine testilugudega, lisaks tuleb alustada ka versioonide testimisega ning valmis ülesannete verifitseerimisega. Üldjuhul on seadistatud igapäevane versiooni järk (*daily build*), mida tuleb testimiseks kasutada. Versiooni järk uueneb iga uue koodisalvestusega, mis tagab selle, et viimased muudatused on alati kättesaadavad.

Kõik kasutajalood peavad olema kaetud testilugudega. Võib juhtuda, et kasutajalooks on olemasoleva funktsionaalsuse ümbertegemine. Sellisel juhul tuleb otsida funktsionaalsusele vastavad testilood ning need ümber muuta vastavalt uutele nõuetele. Samuti tuleb testilood siduda uue kasutajalooga.

Kõik raporteeritud vead, mis on *QA Review* staatuses, tuleb sprindi jooksul kvaliteediinseneri poolt üle testida. Kui probleeme ei esine, tuleb viga panna *Closed* olekusse, märkides juurde testitava versiooni, seadme ning rakenduse nime. Kui probleemid endiselt esinevad, tuleb ülesanne uuesti avada, märkides juurde kommentaari testitava versiooni, seadme ning rakenduse nimega.

Ka kasutajalood, mis on *QA Review* staatuses, tuleb üle testida. Tuleb kontrollida, et funktsionaalsusele oleks loodud testilood ning need oleks antud kasutajalooga seotud. Kasutajalugusid uuesti ei avata, see tähendab, et kui nende testimise käigus ilmneb viga, tuleb luua JIRAsse viga ning see siduda kasutajalooga, mille käigus viga ilmnes.

Vea leidmisel testitavas versioonis tuleb kõigepealt otsida, ega viga ei ole juba varem raporteeritud. See välistab duplikaatide tekke. Kui viga on juba varem raporteeritud, aga teise versiooniga, tuleb viga uuendada versiooni numbriga. Kui otsing tulemusi ei anna, tuleb raporteerida viga JIRAsse ning märkida juurde testitava versiooni, seadme ning rakenduse info. Lisaks tuleb märkida sammud, mille tulemusel viga ilmnes, kuna see lihtsustab oluliselt arendaja tööd. Kui vea kohta on olemas ka testilugu, siis tuleb siduda selle ID leitud veaga. Kui viga ei ole ühegi testilooga kaetud, kuid tulevase testimise vajadus on olemas, tuleb luua uus testilugu, mis seotakse JIRA vea IDga.

Regressioonitest

Tarkvara versioon on väljastamiseks valmis, kui kõik selle versiooni vead ning ülesanded on lahendatud ning suletud ja regressioonitest edukalt läbitud. Regressioonitest on vajalik, et kontrollida, et uus arendus ei oleks eelnevat ära lõhkunud.

Kui siiani on regressioonitesti alustatud ka siis, kui funktsionaalsuse arendamine on veel poolik, siis nüüd on regressioonitestimise eelduseks, et kogu arendustegevus on lõpetatud ning kogu arendatud funktsionaalsus on ka kvaliteediinseneri poolt üle testitud ning JIRAs kinni pandud.

Kui varasemalt oli regressioonitesti puhul üsna tavaline see, et versioone oli mitu, siis nüüd on kindlalt defineeritud, et test toimub ühe konkreetse versiooniga. Seda kontrollib ühelt poolt kvaliteediinsener ning teiselt poolt ka platvormi tootejuht. Regressioonitestis

on kohustuslik käia läbi kõik testilood, mis on rakendusele loodud. Vigade ilmnemisel raporteeritakse need JIRAsse, meeskonnajuht koostöös platvormi tootejuhiga otsustab, kas viga on piisavalt oluline, et vajab kiiret parandamist. Kui viga on oluline, tehakse vastav parandus, mis omakorda tähendab seda, et regressioon on vaja vähemalt osaliselt läbi käia uue versiooniga.

Kuna kvaliteediinseneride hulk kasvas aasta alguses, oli vajalik, et igas meeskonnas toimiks töö ühtemoodi, seetõttu lõi töö autor eelnevalt kirjeldatu põhjal igapäevatöös abistava juhise kvaliteediinsenerile. See laeti üles ettevõtte sisekeskkonda ning on leitav ka lisades (vt Lisa 1).

3 Regressioonitestid

Kuigi ettevõttel on plaanis regressioonitestid suuremas osas automatiseerida, on vaja nad enne korrastada. Antud peatükis annab töö autor ülevaate regressioonitestide hetkeseisust, põhilistest probleemidest ning enda leitud lahendustest. Autor kirjeldab ka testidele üldise struktuuri ning testiplaanide ja -lugude loomist.

3.1 Testilood MS Excelis

Hetkel on MS Excelis kogu funktsionaalsuse kohta loodud 23 testiplaani, milles on kokku ligi 3000 testilugu. Selle läbimine võtab viiel meeskonnaliikmel aega seitse päeva.

Testidega seoses on mitu probleemi, suurimaks võib nimetada üldise struktuuri puudumist. Ühe funktsionaalsuse testid võivad asuda mitmes erinevas testiplaanis. Probleemiks kujuneb see testimise käigus, kuna testide läbimisel puudub loogiline järjestus. Näitena võib tuua, et testiplaanis esinevad testilood, mis kontrollivad rakenduse sisese ostu sooritamist ning sellega seonduvate tegevuste kontrollimist. Kui need testid läbitud ja minnakse järgmise funktsionaalsuse testiplaani juurde, siis on suur tõenäosus, et seal on testid, mis eeldavad, et rakenduses ei ole sooritatud ühtegi ostu. Ostude tühistamine on mõnes testserveris üsna keeruline, seega olukorda parandaks ühe funktsionaalsusega seotud testilugude liigutamine ühte testiplaani. Samuti on palju teste, mis:

- sisuliselt ei testi midagi;
- dubleerivad teisi testlugusid osaliselt või täielikult;
- kontrollivad vaid ühe elemendi olemasolu.

Lisaks on rakenduses palju funktsionaalsust, mis on testidega täiesti katmata, kuid mida peaks kindlasti regressioonitesti käigus kontrollima. Testid on kirja pandud ammu ning enamus testidest on uuendamata vastavalt muudatustele arenduses. Väiksema probleemina võib veel välja tuua, et testid on kirja pandud kohati vigases inglise keeles, mistõttu on vahel raske aru saada, mida mõeldud on ning mida test täpselt kontrollib.

Kuna nende testide põhjal hinnatakse kvaliteeti, on väga oluline testid parandada ning luua ka uued veel katmata funktsionaalsuse jaoks.

3.2 Regressioonitestide struktuuri loomine

Autor alustas testide parendamist struktuuri loomisega, kus võttis arvesse eelnevalt toodud probleemid. Kuna autor on läbinud MS Excelis kirjeldatud teste viimase aasta jooksul väga palju kordi, oli tal olemas väga hea ülevaade sealsest struktuurist ning selle puudustest.

Struktuuri moodustamisel oli oluline näha kogu rakendust tervikuna. Põhifunktsionaalsustest ülevaate saamiseks loodi nimekiri koos alamfunktsioonidega. Põhifunktsionaalsus määrati olemasolevate testiplaanide ning rakenduse enda baasil. MS Excelis olevaid testiplaane analüüsiti, et selgitada välja, milliseid testiplaane ning -lugusid saab sealt kasutada. Rakendus oli oluline käivitada, et võrrelda selle käitumist ning MS Excelis olevate testide kirjeldusi. Kasutuslugude puudumise tõttu tuli infot hankida ka arendusmeeskondadelt. Vastavalt saadud infole kaardistati alampunktid funktsionaalsuse kohta.

Nagu varasemalt öeldud, on MS Exceli arvutustabelis rakenduse kohta 23 testiplaani. Pärast struktuuri loomist ning kõikide testiplaanide läbi töötamist jäi alles 18 testiplaani, mille autor lõi kohe ka TestRaili. Loodud struktuur laeti üles ettevõtte siseveebi ning on nähtav antud töö lisades (vt Lisa 2).

3.3 Testilugude loomine testiplaani

Siin kirjeldatakse testide parandamist ühe funktsionaalsuse (*Basic Route*) näitel. Olemasolev versioon MS Exceli testidest ning ka sama funktsionaalsus TestRaili testlugudena on allalaetavad: https://charlie.atlassian.net/wiki/display/EXCEL/Regressioonitestid.

Basic Route testiplaanis on kirjeldatud 23 testilugu. Põhifunktsionaalsus on küll kaetud, kuid on teada, et testiplaanist puuduvad põhjalikumad testid arhiivi kohta. Arhiivi salvestamine ning sealt salvestatud andmete vaatamine on rakenduses põhjustanud mitmeid vigu, sealhulgas ka krahhe (*crash*), mida peetakse Navionicsis üheks kriitilisemaks veaks. Samuti on teada, et marsruudi muutmise testid on muutunud, kuid regressioonitabelis muudatused ei kajastu.

Töö algas TestRailis *Basic Route* testiplaani avamisega. Testplaani loodi jaotised (*section*), mis defineeriti vastavalt põhilistele alamosadele, mida on vaja testidega katta:

- põhifunktsioonid:
    - loomine;
    - muutmine;
    - salvestamine;
    - kustutamine;
- arhiivitestid:
    - üldine kontroll (avamine ja kasutajaliidese testid);
    - kustutamine;
    - detailide vaade;
    - jagamine;

Antud funktsionaalsuse puhul tuleb kontrollida selle käitumist teiste funktsionaalsustega. Seega loodi jaotis *Interaction with other features,* kuhu loodi test sihikujoonestiku (*Crosshair*) kohta, kuna on teada, et nende kahe koostöös on varem vigu esinenud.

Järgnevalt pandi testilood kirja rakenduse käitumise ning koodi uurimise järgi. MS Excelis olid kaetud põhifunktsioonid, kuid palju oli korduvat infot. Olemasolevad testid võeti aluseks, kuid töö autor vähendas nende hulka oluliselt, kirjutades osaliselt kattuvad testid üheks.

Testide kirjapanekul võttis töö autor arvesse, et tulevikus võiks testid suuremas osas automatiseerida ning pani testid kirja võimalikult täpselt ning lühidalt. Autor parandas ka funktsionaalsuse testide katvust, lisades juurde arhiivitestid. Tulemuseks saadi 14 testilugu.

Sarnaselt hakati kirjutama testilugusid teistele funktsionaalsustele. Suuremate puhul, kus puudus terviklik pilt funktsionaalsusest endast ning seotusest teistega, võeti abiks meelekaardi koostamine (vt Lisa 3).

Meelekaart (*mind map*) võimaldab ideed ning mõtted kujutada graafiliselt sellisel viisil, et peamine idee on kantud keskele, ning sellega seotu hakkab sealt hargnema. Meelekaardiga on võimalik informatsiooni paremini struktureerida ning saavutada seostest terviklik pilt (Kanchyani, 2015).

Tänaseks on sarnasel viisil lisatud testilood 11 testiplaani jaoks, kokku loodud testilugusid on 116. Ülejäänud seitsme testiplaaniga töö jätkub.

4 Tulemite katsetamine meeskondades ja järeldused

Testimisprotsessi ning testilugude kirjutamisel toetus autor põhiliselt enda varasematele kogemustele ning lähtus sellest, mida isiklikult mõistlikuks pidas. Töö autor andis tehtud töö kasutamiseks meeskondadele, et selgitada välja, kas toodud lahendus parendab olukorda testimises. Tulemite testimisel osales viis inimest, kellel paluti läbi katsetada loodud testiplaanid ja -lood TestRailis ning töö autori poolt koostatud testimisjuhis. Lisaks palus töö autor võrrelda TestRaili ning MS Exceli kasutamist testimisel ning tuua välja nii negatiivsed kui ka positiivsed omadused.

Positiivsena toodi välja, et testilugude katvus on parem ning nende arv märkimisväärselt vähenenud, tehes läbimise kiiremaks. Siin võib võrdluseks tuua, et kui varem võttis *Basic Route* testiplaani läbimine aega 2,5 tundi, siis nüüd võtab see 1 tunni. Enam ei esine üherealisi testilugusid, mis midagi ei testi ning testilood on järjestatud vastavalt kasutaja päristegevustele. Testijatele meeldis, et testiplaanid olid korrastatud: need ei sõltu enam ükstseisest ja funktsionaalsus on tervenisti kaetud talle vastavas testiplaanis.

TestRaili kasutuselevõtuga toodi välja, et testimise koordineerimine on lihtsam ning puuduvad lisategevused nagu eraldi testimisülesannete loomine JIRAsse, MS Exceli lehtede jagamine ning hilisem kombineerimine testide tulemustega ja käsitsi raportite koostamine, mis varem olid regressioonitestimise paratamatu osa.

Testimisprotsessi juures nähti üleliigsena testiloo sidumist JIRA ülesandega, kuid hiljem leiti, et see on siiski vajalik parema ülevaate saavutamiseks. Seda eriti regressioonitestimise puhul, kus nüüd saab JIRA vearaporti koostada TestRailist, mis seotakse automaatselt testilooga.

Leiti, et uued testilood on kirjutatud täpsemalt ning kirjas on vaid vajalik. Lisaks meeldisid testijatele TestRailis seadistatud eraldiseisvad testisammud, sest nii oli lihtsam testi jälgida testimise käigus.

Ainukese negatiivse asjana toodi välja, et kuna paralleelselt on kasutusel MS Excel ja TestRail, peab lisatööna veel osaliselt koostama käsitsi raporteid testimistulemuste kohta. Samas nähakse, et ka see probleem laheneb hiljemalt sügiseks.

Testimisjuhisest said kõige enam kasu aasta alguses liitunud kvaliteediinsenerid, kuna nende jaoks oli vajalik info ühes kohas ning aitas töö iseloomust paremini aru saada.

Parimaks testimise parenduseks arvati testilugude loomine arenduse alguses kasutajalugude põhjal. Selle tulemusena olid testilood iga funktsionaalsuse kohta tehtud juba kas enne arendust või arenduse algusfaasis. See aitas leida ka ebatäpsusi nõuetes ning defineerida paremini kasutajalood.

Juhise parandusettepanekuna paluti lisada juhend testilugude loomise kohta TestRaili. Testimisjuhis sai vastavalt ettepanekutele täiendatud.

Kokkuvõte

Käesoleva bakalaureusetöö põhieesmärgiks oli luua ning juurutada ühtne testimisprotsess mobiilirakenduste arendusmeeskondadele ning testide korrastamine. Eesmärgi saavutamiseks analüüsis töö autor testimise hetkeseisundit, kirjeldas suuremad probleemid ning leidis neile lahendused. Lisaks analüüsis autor erinevaid testihaldussüsteeme eesmärgiga asendada praegu kasutusel olev Excel, lõi testidele struktuuri ning uued testilood.

Töö tulemustena valmis ühtne testimisprotsess meeskondadele, loodi abistav testimisjuhis ning liigutati komplekt testilugusid MS Excelist testihaldussüsteemi. Tulemid katsetati läbi meeskondades, mille liikmed leidsid, et autori pakutud lahendused lihtsustavad töö tegemist ning tõstavad rakenduste kvaliteeti.

Käesoleva bakalaureusetöö edasiarendusena oleks võimalik töötada välja süsteem regressioonitestide automatiseerimiseks.

Summary

This thesis “Improvement of Testing Processes: the Case of Navionics” main goal was to create and establish an integrated testing process for mobile application development teams. Another set goal was to improve the quality of tests used in regression testing.

To accomplish the set goals, the author analysed current condition of testing in the teams, described bigger issues within the process and also came up with solutions to improve the current method. Even more, the author analysed various test management tools in attempt to find a substitute for MS Excel that is currently being used for storing testcases. The author created a structure for tests, new testplans and testcases.

As a result, an integrated testing process and guidelines for testing were created for the teams and a set of testcases were moved from MS Excel to selected test management tool. The members of the teams tried the results themselves and found that author’s solutions simplify their everyday work and also improve the quality of the applications under development.

The further development of this thesis is to improve the testing process by starting to automate regression testplans.

Kasutatud kirjandus

Graham, D., Van Veenendaal, E., Black, R., & Evans, I. (2008). *Foundation of Software Testing: ISTQB Certification.*

ISTQB Exam Certification. (kuupäev puudub). *ISTQB Exam Certification*. Loetud aadressil: What is acceptance testing: http://istqbexamcertification.com/what-is-acceptance-testing/

ISTQB. (2007). *Standard glossary of terms used in Software Testing* .

Kanchyani, N. (2015, veebruar). *Software Testing Help.* Loetud aadressil: Mind Mapping in Software Testing – Ways to Make Testing More Fun!: http://www.softwaretestinghelp.com/mind-mapping-software-testing/

Markvardt, M. (2011). *Tarkvara testimise alused - loengumaterjalid.* Loetud aadressil: http://www.e-ope.ee/repositoorium?@=7gxj

Navionics. (2015, jaanuar). Loetud aadressil: Roles and Responsibilities - Navionicsi sisemine dokument.

Navionics. (2015, jaanuar). Loetud aadressil: Software Development Process - Navionicsi sisemine dokument.

Postimees. (2013, 13. aprill). *Navionicsi asutaja: näen Eesti üksusel suurt kasvuvõimalust.* (K. Traks, Toim.) Loetud aadressil: Postimees: http://majandus24.postimees.ee/1201142/navionicsi-asutaja-naen-eesti-uksusel-suurt-kasvuvoimalust

Subramaniam, V., & Hunt, A. (2006). *Practicies of an Agile Developer.* USA.

Tänavsuu, T. (2013). World famous chart producer Navionics moves to Estonia. *Life in Estonia* , 40-42.

TestRail. (kuupäev puudub). *TestRail*. Loetud aadressil: TestRail Hosted Pricing: http://www.gurock.com/testrail/pricing/hosted/

LISAD

Lisa 1
Guidelines for Testing

Reporting an issue

- Before reporting an issue, run a quick search to ensure issue has not been reported already. This is to prevent duplicates;
- If issue is found while testing some other issue, the issues must be linked together (in JIRA issue view, press on 'More'>'Link', add issue number and 'discovered while testing');
- When reporting an issue add steps to reproduce, device information, also application and build versions used for testing;
- When reporting crash issues, also provide Crash Log.

Issue verification during development

- All issues in sprint must be verified with daily build. This applies for issues that are in state QA Review;
- If issues are still present, reopen JIRA issue and add comment about application version and device information used for testing. Also assing the issue to developer who already worked on the issue;
- If issues are fixed, close the issue and add a comment about application version and device information used for testing;
- User Stories (also check Epic for covering all the positive flows for entire feature) must be covered with Test Cases. If Test Cases already exist, should update them according to User Story. If Test Case does not exist, create it and link it to the User Story;
- When verifying a User Story, the behaviour must be verified, Test Case must be created or updated and a comment added to user story with testcase ID;
- If issues are found while testing a User Story, separate bugs must be reported and linked to the User Story ('Found while testing'), User Story itself should be closed but a comment should be added that Test Cases were created and issues were found;
- Important bugs (mostly critical issues) should also be covered with Test Cases to recheck with every regression test;

- User Stories and linked issues need to be tested and verified during sprint before starting regression test.

Testing on feature branch:

This usually applies for a work that is still in progress, i.e. issue is still in progress but Quality Engineer help is needed to verify behaviour partially. Build usually is a local build provided by a developer, not an automatic daily build.

When issues are found in feature branch, would be advisable to add them as a comment or create separate issue with information about what build was used for testing.

Use common sense with that: when it is known what part of the feature is implemented and what not, and issues found are NOT part of either or are only part of the already implemented part, then it would make sense to create a separate issue. Otherwise it is sufficient to add a comment under task in progress what problems occurred and what to take into account when implementing feature.

Adding tests to TestRail

Guidelines for adding tests:

- Start with covering positive flows mentioned in User Story, corner cases can be added later on while doing actual testing on a feature;
- Keep in mind logical flows from a user point of view, that means, when writing down an entire feature into separate Test Cases, try to keep them intact: when one test ends, the other one begins right where previous left off;
- Do not add Test Cases that are only one step. Also try to keep tests shorter than 10, otherwise it is hard to follow while testing;
- When adding steps to Test Cases, keep as simple as possible: provide enough detail but leave some room for imagination;
- Follow the same style as already used for tests;
- Keep descriptions as short and accurate as possible (think of maintenance and how often specification can change);
- Keep in mind the structure: everything related to a specific feature should be in the same Test Suite;
- If Test Suite has not been created for a feature, create it yourself to start adding Test Cases.

Lisa 2

Testide struktuur

Archive

- Includes only general archive tests (look and feel).

Autorouting

- New from ‘Route’ after purchase:
    - Include actionsheet items: manual route and automatic route;
    - Edit and delete functionalities;
        - Include automatic edit;
- Archive;
- Interaction with other features:
    - Advanced Map Options;
    - S57;
    - SonarCharts.

Basic Route

- Basic route from ‘Route’ (new, edit, save, delete);
- Archive management (general, details, save, delete, share);
- Interaction with other features (crosshair).

Charts&Upgrades

- General tests for access;
- Tests for sections and menu items;
- Purchase flows.

Empty Boating (include all tests from all spreadsheets)

- First time startup;
- Menu look and feel (S57 vs. ROW);
- Invitation to purchase a chart from ‘Search’;
- Weather&Tides search management;
- N+ trial management:
    - N+ trial activation and active period;

    - Advanced map options trial not possible;
- Community edits (with and without N+ trial);
- Invitations/banners to purchase charts;
- Cartoselector behaviour and displaying of banners;
- Interaction with SCL layer;

Enhanced Navigation Module

- Trial management;
    - Always possible (with or without charts);
- ENM purchased
    - Edit mode;
    - Navigation mode;
    - Archive management.
- Interaction with other features.

GPS & Distance

- GPS
    - Disabled/enabled from device settings;
    - Pan/zoom;
- Course up;
- Compass;
- Distance.

Magazines&Guides

- First access & tutorial;
- Download;
    - Top;
    - Categories;
    - Magazines;
    - Nearby;
        - Map preview;
    - Search.

Map Options

- Trial management;

- Map preview;
- S57 availability.

PlotterSync

- Connection;
- Sync;
- Regions;
- Settings;
- Archive.

Search

- Access;
- General tests.

Settings

- Boat settings,
- Units;
- Ranking;
- Data Sync.

SonarPhone

- Details page and demo;
- SonarPhone;
- SonarPhone Settings;
- Interaction with other features (mainly route and track consoles).

SonarCharts Live

- Invitation banner;
- Demo;
- SonarCharts Live;
- S57.

Startup & General tests

- Splash screen;
- Startup (1st time + following times);

- Map download;
- Main map view;
- Menu items;
- Submit feedback;
- Help file;
- Camera;

Tracks

- General track functionality;
- Track playback;
- Archive management.

UGC

- Community Edits;
- Categories and Sub-categories;
- Cartographic data.

Weather&Tides

- Map preview;
- Tides&Currents.

Lisa 3

Empty Boating meelekaart