TESTAUSAUTOMAATION TULOSTEN TUOMINEN ASIAKASPALVELUN SAATAVILLE

Jaakko Mäntysaari

Tampereen yliopisto
Viestintätieteiden tiedekunta
MDP in Human-Technology Interaction
Pro gradu -tutkielma
Lokakuu 2018

TAMPEREEN YLIOPISTO, Viestintätieteiden tiedekunta
MDP in Human-Technology Interaction
MÄNTYSAARI, JAAKKO: Testausautomaation tulosten tuominen asiakaspalvelun saataville
Pro gradu -tutkielma, 49 s.
Lokakuu 2018

Työssä tarkasteltiin testausautomaation välineiden käyttöä järjestelmän valvonnan työkaluna ja näiden työkalujen tulosten saattamista asiakasrajapinnan hyödynnettäviksi. Työssä testattiin kahta eri lähestymistapaa: chatbot ja jaettu näyttö. Näiden avulla pyrittiin lisäämään asiakasrajapinnassa työskentelevien ymmärrystä järjestelmän tilasta, eli järjestelmän tilan tuntemusta.

Järjestelmän tilan tuntemus on malli, jossa järjestelmän käyttäjä pyrkii useista eri lähteistä yhdistelemällä muodostamaan toimivan kokonaiskuvan järjestelmän toiminnasta, vaikka järjestelmän tila olisikin ennalta arvaamaton ja kompleksinen.

Keskeisiä prototyyppien suunnittelussa havaittuja haasteita on testausautomaation osalta löytää oikea tasapaino testien laajuuden, tarkkuuden, suoritusajan ja luotettavuuden välillä.

Chatbotin todettiin toimivan hyvin ChatOps-mallin työkaluna, mutta sopivan huonosti pyrkimykseen lisätä järjestelmän tilan ymmärrystä. ChatOps-mallissa työkalut yhdistetään viestikanaviin, pyrkimyksenä lisätä kommunikaatiota ja dokumentoida työkalujen käyttöä.

Jaettu näyttö todettiin toimivaksi ratkaisuksi informaation jakamiseen tässä tarkoituksessa. Vaaditaan kuitenkin tarkkaa suunnittelua ja yhteistyötä käyttäjien kanssa, jotta esitetty informaatio on oikea-aikaista ja soveltuvaa. Tämä myös avaa mahdollisuuksia esittää muuta palvelun toimintaan liittyvää informaatiota.

Koska testausautomaatiota hyödynnetään muissa toiminnoissa kehittäjien ja järjestelmäasiantuntijoiden toimesta, on hyödyllistä laajentaa tätä hyötyä myös organisaation muihin osiin, jotka informaatiosta potentiaalisesti hyötyvät.

Avainsanat: testausautomaatio, chatbot, jaettu näyttö, järjestelmän tilan tuntemus, asiakasrajapinta

Esipuhe

Ilman Aliisan tukea ja Taimin kannustusta tämä työ olisi jäänyt tekemättä. Vaikka Taimi on nuorempi kuin tämä projekti.

Tampereella 30.10.2018

Jaakko Mäntysaari

Sisällysluettelo

1 JOHDANTO

IT-maailma on siirtynyt vauhdilla verkkoon. Yhä suurempi osuus kehittäjistä työskentelee verkkopalveluiden parissa. Samalla uudet menetelmät valtaavat alaa: A*gile, DevOps, ChatOps, SCRUM, MVP* ynnä muut termit vilisevät aiheen parissa käytävässä keskustelussa. Kuitenkin organisaation muut toiminnot jäävät helposti uudistuksissa paitsioon. Asiakaspalvelun työ ei valtavasti muutu, oli kyseessä sitten verkkopalvelu tai perinteisempi ala. Asiakaspalvelija käsittelee saapuvan kontaktin, kirjaa kontaktin tiedot ylös ja tarvittaessa kontaktoi toista tahoa saadakseen tehtävän hoidetuksi, oli tämä sitten talous- tai laskutusosasto, tekninen asiantuntija tai muu vastaava erikoistunut taho.

Verkkopalveluiden tapauksessa informaation määrä on paitsi suurempi, myös nopeasti muuttuva. Mikäli fyysisessä tuotteessa on virhe, saman asian selvittely saattaa kestää pitkäänkin ja yleensä asian käsittelyyn muodostuu tai muodostetaan prosessi, eli ennalta määritelty toimintatapa. Verkkopalveluiden ollessa kyseessä kehitys on kuitenkin jatkuvaa ja ongelmatilanteita saattaa syntyä ja poistua nopeassa tahdissa.

Kehittäjät ja muu tekninen organisaatio käsittelee tätä haastetta muun muassa yksikkötestaamisella, monitoroinnilla, testiympäristöillä ja testausautomaatiolla. Usein tämän informaation hyödyntäminen näyttää kuitenkin jäävän lähinnä kehittäjille ja muille asiantuntijoille. Hyvin rakennetun testijärjestelyn hyödyt voitaisiin kuitenkin ulottaa suoraan asiakasrajapintaan saakka.

Asiakaspalvelijoilla on yleensä varsin syvällinen tuotteen tuntemus, sillä keskeinen osa heidän työtehtäviään on asiakkaiden neuvominen ja mahdollisten virhetilanteiden tunnistaminen. Ongelmien selvittäminen yhdessä asiakkaan kanssa, mahdollisesti useita sähköposteja vaihtaen, on kuitenkin melko aikaa vievä prosessi. Mikäli järjestelmän tilasta kertovaa informaatiota saataisiin nopeammin ja vähällä vaivalla suoraan asiakasrajapintaan, saattaisi tämä lyhentää kontaktin käsittelyä melkoisesti ja ohjata asiakaspalvelijaa kysymään asiakkaalta suoraan oikeita kysymyksiä tilanteen selvittämiseksi.

Tämän työn tarkoituksena on tutkia mahdollisuuksia hyödyntää testausautomaation tulosten jaettavuutta ja hyödyntämistä asiakasrajapinnassa vertailemalla kahta eri menetelmää: Slack-viestipalvelussa toimivaa chatbottia ja infotaulua, joka näyttää koosteen edellisen testiajon tuloksista.

Etuovi.com ja Vuokraovi.com -palveluissa on toteutettu laajasti järjestelmän tilaa valvovia automaattitestejä. Näitä käytetään pääasiallisesti kolmessa eri tilanteessa: järjestelmän tilan valvomista ajastetuin testein, kehittäjien avuksi varmistamaan koodimuutosten turvallisuus ja testaajien avuksi etsimään palveluun hiipinyttä regressiota. Järjestelmän tilan seuranta on kuitenkin paljolti hyödyntämättä asiakasrajapinnassa. Asiakasrajapinta asettaa testeille ja niiden tulosten raportoinnille joitakin erikoisvaatimuksia, jotta niistä saataisiin täysi hyöty irti. Testien tulosten on oltava nopeasti luettavissa ja niistä on kyettävä tekemään oikeat ja riittävät johtopäätökset nopeasti.

Keskeisiä tässä työssä esiintyviä termejä ovat testausautomaatio, järjestelmän tilan ymmärrys (eng. s*ystem state awareness*) chatbot, ChatOps, ympäristönäyttö (eng. *ambient display*) ja jaettu näyttö. Termit esitellään tarkemmin myöhemmissä luvuissa.

Testausautomaation tehokas hyödyntäminen vaatii ratkaisujen integroimista olemassa olevaan kehitysympäristöön ja sen välineisiin. Tässä työssä toteutetut prototyypit pohjautuvat testausautomaatioon, joka on toteutettu Ruby-pohjaisella RSpec-testauskirjastolla. Työssä toteutettiin rajatumpi testijoukko tämän laajemman testausjoukon pohjalta.

Yrityksessä käytetty viestintäväline sekä kehittäjien kesken että myös kehittäjien ja asiakaspalvelijoiden välillä on Slack (“Where work happens | Slack”). Tämä rajasi chatbotin alustavalintaa sellaisiin, jotka tukevat kyseistä viestintä. Vaikka testausautomaatio onkin Ruby-kielisellä kirjastolla toteutettu, pääasiallinen yleiskäyttöinen järjestelmäkieli on Python. Tämä rajasi entisestään sopivan chatbot-alustan valintaa. Yksi suosittu Python-pohjainen, Slackiä tukeva chatbot on Errbot (“Errbot — Err 5.1.3 documentation”). Kyseinen alusta löytyy myös Slackin suosittelemista kirjastoista (“Slack Platform: Community | Slack”).

Jaettu näyttö toteutettiin Python-pohjaisella Flask-webkehyksellä ja Javascriptillä. Vaikka NodeJS on yleisesti käytetympi kevyiden verkkopalveluiden rakentamisessa, saavutettiin taustajärjestelmän kehityksessä yhtenäisyydestä hyötyä. Python-moduuleja kyettiin käyttämään suoraan keskenään prototyyppiä rakentaessa.

Järjestetyn prototyyppien testausten ja tätä seuranneiden haastatteluiden myötä kävi ilmi, että lopullisen järjestelmän suunnittelussa tulee ottaa erityisesti huomioon näytettävän informaation määrä, laatu ja ajantasaisuus. Chatbot koettiin hyväksi työkaluksi ylipäätänsä, mutta suoraan tähän tarkoitukseen sopimattomaksi. Jaettu näyttö sen sijaan herätti paljon

mielenkiintoa ja positiivisia kokemuksia. Molempia järjestelmiä ryhdytään ajamaan eteenpäin ja jatkokehittämään, mutta hieman eri näkökulmilla.

Laajemmassa kontekstissa tämän työn tulokset osoittavat, että yleensä kehittäjien ja järjestelmäasiantuntijoiden käyttöön tarkoitettu tieto järjestelmän tilasta on hyödyllistä myös organisaation muissa osissa. Koska työn luonne on erilainen eri osissa organisaatiota, vaativat sekä informaation raportointi että sen esittämisen käyttöliittymät kuitenkin erityistä suunnittelua, jotta informaatio saadaan myös hyödynnetyksi.

Tämä työ alkaa luvussa 2 teorian ja aiempien toteutusten kuvauksella. Esittelen lyhyesti testiautomaatiota, järjestelmän tilan tuntemusta, jaettuja näyttöjä ja chatbotteja sekä niihin liittyvää ChatOps-mallia. Luvussa 3 kuvaan prototyyppien suunnittelua ja testausta. Esittelen luvussa ympäristön, prototyyppien suunnittelun sekä toteutuksen ja miten prototyyppeihin suhtauduttiin testauksen jälkeen järjestetyissä keskusteluissa. Luvussa 4 käyn läpi testauksessa ja työssä yleisesti heränneitä ajatuksia ja arvioin miten testauksessa onnistuttiin. Luvussa 5 esitän työstä vetämäni johtopäätökset.

2 TEORIA JA AIEMPIA TOTEUTUKSIA

Testausautomaatio on monisyinen aihepiiri, johon liittyvää tutkimusta on paljon olemassa, mutta jossa riittää yhä tutkittavaa, sillä testauksen automatisoinnin merkitys vaikuttaa pikemminkin kasvavan kuin vähenevän. Tässä työssä keskityn kuitenkin testausautomaation tulosten esittämiseen ja hyödyntämiseen. Seuraavassa esittelen testausautomaation yleistä teoriaa ja siihen liittyviä haasteita tästä näkökulmasta.

Tämän jälkeen esittelen lyhyesti *järjestelmän tilan tuntemuksen* käsitteen siinä roolissa, kun se työhön liittyy. Monitorointi ja testaus pyrkivät samaan asiaan: siihen, että järjestelmän tila olisi tunnettu ja varmistettu.

Testausautomaatiosta saadut tulokset, vaikka olisivat hyvin muotoiltuja ja oikeanlaista tietoa sisältäviä, pitää tuoda jollakin menetelmällä käyttäjiensä saataville. Keskeinen haaste tulosten hyödynnettävyydessä on saada tulokset esitetyksi tavalla, joka ei aiheuta lisää kuormitusta tulosten hyödyntäjälle. Tästä syystä työn suunnittelussa lähtökohtana oli valita menetelmiä, joissa kyetään hyödyntämään joko olemassa olevia kommunikaatiotyökaluja ja mahdollisuuksia jakaa informaatio passiivisen seurannan avulla jaettua näyttöä hyväksi käyttäen.

Työssä toteutettiin kaksi prototyyppiä testiautomaation tulosten tueksi. Kohderyhmänä toimii verkkopalvelun asiakaspalvelu, joten suunnittelussa tuli ottaa huomioon sekä testausautomaation ja kohderyhmän että jo käytössä olevien prosessien asettamat vaatimukset. Tämän työn teoreettinen viitekehys nojaa yllä esitettyihin kolmeen aiheeseen, joita yhdistelemällä pyrin pohjustamaan prototyyppien suunnittelussa tehtyjä valintoja ja esiin tulleiden haasteiden ratkaisuja.

Luvussa 2.1 alustan testausautomaatiota ja luvussa 2.2 järjestelmän tilan tuntemusta. Luvussa 2.3 esittelen jaetun näytön ja luvussa 2.4 chatbotin teoriapohjaa järjestelmän tilan tuntemuksen välineinä.

2.1 Testausautomaatio

IT-maailmassa tuotantojärjestelmien automaattinen testaus ennen julkaisua on jo pitkään ollut sekä ohjelmoijien että testaajien työkaluna, mutta graafisten käyttöliittymien yleistyttyä testien merkitys ja toisaalta vaativuus kasvoi huomattavasti. Tämä johtaa kompromisseihin testien kattavuuden, luotettavuuden ja ajantasaisuuden suhteen. Tästä syystä Pettichord artikkelissaan *Success with Test Automation* puhuukin tyytymisestä "riittävän hyvään" automaattitestaamiseen (Pettichord, 2001).

Pettichordin mukaan keskeisiä haasteita testiautomaatiossa on ylläpidettävyys ja luotettavuus. Onnistuakseen testien pitää noudattaa samoja menetelmiä kuin muukin ohjelmistokehitys ja testin kirjoittajien tulee työskennellä läheisessä yhteistyössä muiden kehittäjien kanssa. Ylläpidettävyys rajaa Pettichordin mukaan pois graafiset työkalut, joilla testien kirjoittamista pyritään helpottamaan: tällöin pienikin muutos järjestelmän ulkoasuun aiheuttaa virheitä ja testien ylläpitotyö moninkertaistuu. Mikäli testit eivät ole ajan tasalla, niiden luotettavuus kärsii. Hän suositteleekin yleisluontoisempaa testausta, joka on lähempänä ohjelmointia, joka selviää helpommin testattavan järjestelmän ulkoisen käyttöliittymän muutoksista testattaessa ydintoiminnallisuuksia (Pettichord, 2001).

Pelkästään testiympäristöt eivät enää ole testaamisen kohteena. Vaikkakin tuotannon testaamista on pidetty aiemmin turhana, muun muassa Microsoft on lähtenyt ajamaan käytäntöä omassa ympäristössään.

Microsoftin Exchange-tiimi on listannut asettamansa tavoitteet tuotannon testaamiselle:

- Löytää proaktiivisesti ongelmia tuotantopalvelusta
- Aktiivisesti testata kehittämiään ja itse käyttämiään palveluita
- Uusien ominaisuuksien ja päivitysten validointi
- Varmistua ettei konfiguraatiomuutos tai päivitys riko ajossa olevaa palvelua
- Auttaa kumppanien Microsoft Exchangestä riippuvia palveluja toteuttamaan hyväksymistestejä
- Järjestelmän valvonnan avustusta ja valvontaan tehtäviä parannuksia
- Latenssin eli viiveen havaitseminen eri palvelun osissa
- Kehittäjien ohjaaminen ymmärtämään loppukäyttäjän käyttökokemusta testien avulla

(Johnston & Deschamps, 2012).

Tuotannon testaamisesta seuraa se hyöty, että samalla automaatiolla voidaan havaita paitsi testiympäristöjen, niin myös tuotantojärjestelmän ongelmia. Tämä auttaa ylläpitoa, mutta mikäli informaatio saadaan leviämään, myös muita organisaation osia. Asiakaspalvelu ei työssään juuri hyödy löydöksistä, jotka koskevat vain testiympäristöjä.

Testauksen tuloksissa on usein kohinaa eli vääriä positiivisia tuloksia. Kohina johtuu muun muassa siitä, että tuotannon testaaminen usein vaatii ympäristön asettamien haasteiden kiertämistä. Haasteita saattaa olla esimerkiksi tavassa, jolla ympäristöön yhdistetään tietoturvaan liittyvät vaatimukset tai tuotantopalvelun suojaaminen testaamisen tekemiltä muutoksilta. Tämän vuoksi yksittäisen testin virheily ei kerro kaikkea, vaan tarvitaan pidempiaikainen trendi testien tuloksissa. Tämä auttaa hahmottamaan onko ongelma vain yksittäinen virhetilanne kuten verkkovirhe vai onko ongelma perustavampaa laatua (Johnston & Deschamps, 2012). Kuitenkin järjestelmän tilaa tarkkaillessa tulee herkemmin puuttua yksittäisiinkin virheisiin. Tässä syntyy ero järjestelmän tilan tarkkailun ja ongelmien paikallistamisen välillä. Järjestelmän tilan ymmärrystä lisää huomiot virhettä näyttävästä testistä, kun taas ongelman tarkempi selvittely vaatii syntyvän trendin hahmottamista.

Mitä sitten tulisi testata automatisoiden? Hyvä nyrkkisääntö on automatisoida tavanomaisia, toistuvia toimenpiteitä, jättäen monimutkaisemmat ja arviointia vaativat tapaukset manuaalista testaamista varten. Testausautomaatio ei korvaa osaavia testaajia, vaan pyrkii poistamaan yksinkertaiset ja puuduttavat testitapaukset manuaalitestauksesta (Kohl, 2012). Mikäli halutaan testata lomakkeen toimivuus, ei testaajan ole järkevää jokaisella kerralla käydä jokaista kenttää yksittäin läpi testaten kenttien validoinnin toimivuutta. Tämän voi hyvin automatisoida, jolloin testaaja voi keskittyä monimutkaisempiin ja potentiaalisesti järjestelmää rikkoviin tapauksiin.

Tämän vuoksi on tärkeä huomata, että läpi mennyt testi ei ole tae siitä, ettei järjestelmässä ole mitään vialla. Testit testaavat vain järjestelmän tiloja, joita testin on ohjelmoitu havaitsevan. Tämä vaatii sitä, että järjestelmää läpikäy testimielessä myös ihminen ja monitorointia tekee monitorointiin dedikoitu järjestelmä. Testit yleensä tarkastelevat järjestelmää käyttäjän näkökulmasta, joten järjestelmän sisäinen ongelma ei välttämättä testissä näy (Kohl, 2012).

Kaikki edellä mainittu asettaa haasteita testien suunnittelulle. Testit tulisi suunnitella siten, että tulokset on helppo ymmärtää ja analysoida. Pelkkä onnistuminen tai epäonnistuminen ei useinkaan riitä (Benet et al., 2012). Suunnitellessa tulisi myös miettiä, mitä

testituloksilla halutaan saavuttaa: järjestelmän monitoroinnissa yksinkertaisuus on tärkeää, toiminnallisuuksia syväluotaavissa testeissä taas tarvitaan mahdollisimman tarkkaa informaatiota siitä, minkälaisen ongelman testi havaitsee. Joka tapauksessa testien tulosten raportointia tulisi muokata sellaiseksi, että se sisältää tarkastelijalle hyödyllistä informaatiota (Johnston & Deschamps, 2012).

2.2 Järjestelmän tilan tuntemus

Keskeisiä käsitteitä työssä on järjestelmän tilan tuntemus eli englanniksi *system state awareness*. Tällä tarkoitetaan yleistä tuntumaa järjestelmän tilasta. Tuntuman saavuttaminen ei välttämättä tarvitse aktiivista selvittämistä, vaan useasta lähteestä tulevan informaation yhdistelyä. Termi juontaa juurensa vahvasti Nielsenin heuristiikkoihin, joista ensimmäiseksi usein listataan *visibility of system status*, järjestelmän tilan näkyvyys (Nielsen, 1994). Kun järjestelmän tilan näkyvyys on jatkuvaa, saavutetaan suurempi järjestelmän tilan tuntemus.

Järjestelmien suunnitteluun liittyy termi *situation awareness*. Tämä tilannetietoisuus on kyky havaita ongelma ja toimia sen ratkaisemiseksi. Mikäli virheitä tapahtuu, johtuvat virheet ihmisestä: tilannetta ei havaittu ajoissa tai toimittiin väärin tilanteen kohdatessa. Tämä ei kuitenkaan taivu monimutkaisten järjestelmien (eng. *complex system*) käsittelyyn, sillä mikäli mahdolliset järjestelmän tilat eivät ole ennalta tiedossa, ei niiden käsittelyyn ole myöskään kehitetty toimintaa. Rinnalle kehitetty termi system state awareness määrittelee kompleksisen järjestelmän tilan ymmärrystä, kun järjestelmä saattaa olla monenlaisessa ennalta määrittelemättömässä tilassa (Kasdaglis, Newton, & Lakhmani, 2014).

Käsitteen keskiössä on ajatus ympäristön tarkkailusta. Käytännössä tämä tarkoittaa usean tietolähteen samanaikaista seuraamista järjestelmän tilan havaitsemiseksi. Tässä järjestelmän tilan tunnistus on tärkeässä osassa. Vastakohtana tälle on kontrolliorientoitunut lähtökohta, joka vaatii ylläkuvattua tilanteen havaitseminen, tilanteen ratkaisu -kulkua. (Kasdaglis et al., 2014).

Järjestelmän tilan tuntemus on siis tapa, jolla käyttäjä yhdistelee useista lähteistä tietoja saadakseen kuvan ympäristöstään, jotta kykenee toimimaan odottamattomissa tilanteissa.

2.3 Jaetut näytöt

Englanninkielinen termi *ambient surface* kuvaa läsnä olevaa, informaatiota sisältävää näyttöä. Näytöllä pyritään tuomaan informaatiota jatkuvasti näkyväksi informaation visualisointityökalujen avulla. Tämänkaltaisen toteutuksen on todettu olevan hyödyksi sekä aktiivisessa että passiivisessa käytössä ohjelmistokehityksen tiimityöskentelyssä (Schwarzer et al., 2016). Vaikkakin suomenkielinen termi vaihtelee, käytetään tässä jatkossa termiä *jaettu näyttö* ja prototyypin ollessa kyseessä termiä *infonäyttö.*

Jaetut näytöt ovat olleet pitkään teollisuuden käytössä niin tuotantolaitoksissa kun ydinvoimaloissakin (Burns, 2000). *Ecological interface design* (EID) on lähtöisin 1960-luvulta (Vicente, 2002). Oleellista EID:n avulla suunnitelluissa järjestelmissä on ympäristön ottaminen suunnittelun lähtökohdaksi käyttäjän sijaan. Vaikka syntyvä mielikuva on helposti teolliseen prosessivalvontaan liittyvä, on samoja suunnitteluperiaatteita testattu myös pienemmillä näytöillä kuten kameroiden runkojen LCD-näytöissä (Mazaeva & Bisantz, 2014). Käytännössä suunnittelussa otetaan huomioon informaation esittäminen käyttäjän toimintojen mahdollistamisen edellä.

On kuitenkin huomioitava näytön sisältämän informaation ja sen fyysisen sijainnin vaikutus käyttäjän kognitiiviseen kuormaan. Kun yhteiskäyttöisiä näyttöjä on tutkittu luokkahuoneessa, on havaittu, että perifeerisen informaation lisääntyminen saattaa aiheuttaa negatiivisia vaikutuksia keskittymiseen. Tätä voidaan kuitenkin vähentää sijoittelulla ja käyttöliittymän suunnittelulla (Beserra et al., 2014). Vaikka Beserra et al. paperissaan *Measuring cognitive load in practicing arithmetic using educational video games on a shared display* (Beserra et al., 2014) käsittelevät järjestelmiä, joissa on yksi yhteinen näyttö, mutta jokaisella käyttäjällä oma syöttövälineensä, kuten hiiri, on tutkimuksesta johdettavissa muutamia hyödyllisiä seikkoja. Keskeisiä huomioita on, että ihminen käsittelee visuaalista informaatiota pääosin näkökenttänsä keskeltä, mutta myös näkökentän rajoilta. Tämä saattaa aiheuttaa kognitiivista ylikuormitusta. Kognitiivinen kuorma kertoo kuinka paljon tietoista keskittymistä yksittäisestä tehtävästä suoriutuminen vaatii (Beserra et al., 2014).

Jaetut näytöt ovat, huolimatta suuresta kiinnostuksesta sekä kaupallisessa että teknisessä mielessä, yleensä käyttäjien ylenkatsomia: mikäli näyttö ylipäätään nähdään, sitä katsotaan vain hetken aikaa ja yleensä varsin kaukaa. On esitetty havaintoja näyttösokeudesta samalla tavoin kuin verkkosivujen bannerisokeudesta. Dalton et al. (2009) aiheesta

tekemä tutkimus ei suoraan täysin tyrmää tätä väittämää, jos ei varsinaisesti sitä tuekaan. Tutkimuksessa he seurasivat katseenseurantatyökaluilla kauppakeskuksen asiakkaiden huomion kiinnittymistä näyttötauluihin. He esittävät huomioita nyansseista, joita asiaan liittyy. Tutkimuksen perusteella ensinnäkin ihmiset todella vain nopeasti vilkaisevat näytön sisältöä. Toiseksi, näytön sijainti arkkitehtonisessa kontekstissa vaikuttaa kiinnittyykö katse näyttöön. Kolmanneksi, mikäli näyttö on pieni, kannattaa sen olla silmien korkeudella. Suurempi näyttö voi olla korkeammalla ja saada ihmiset silti kiinnittämään siihen yhtä paljon huomiota. Neljänneksi, tasainen valaistus on suuria kirkkauden vaihteluita parempi vaihtoehto. Suoranaisesta näyttösokeudesta, kuten verkkosivujen tapauksessa, ei siis voida puhua, vaan käy ilmi, että näyttöjen sijoittelu ja suunnittelu auttavat käyttäjiä kiinnittämään näyttöjen informaatioon huomiota (Dalton, Collins, & Marshall, 2009). Vaikkakin nämä huomiot on tehty kaupallisessa asetelmassa, jossa ohikulkijoiden huomio halutaan kiinnittää mainos- ja infotauluihin, voidaan tuloksista tehdä johtopäätöksiä myös toimistoympäristöön. Ajatus näytön informaation vertauskuvallisesta myymisestä käyttäjille pitänee paikkaansa.

2.4 Chatbotit, DevOps ja ChatOps

Tähän työhön rakennetun chatbotin toiminnallisuus nojaa *DevOps*-malliin. Perinteisessä kehitysmallissa kehittäjät työstävät ohjelmistoa ja järjestelmäasiantuntijat olivat vastuussa ohjelmiston tuomisesta palvelimille ajoon. Tämä lokerointi aiheuttaa tilanteen, jossa vastuun siirtyminen ohjelmistoversiosta vaatii aikaa ja ennalta määrättyjä prosesseja, samalla lisäten kommunikaatioon liittyviä haasteita. Tämä puolestaan aiheuttaa kehityksen hidastumista, sillä jokaisen muutoksen pitää kulkea tämän putken läpi, etenkin jos eteenpäin päästettyyn versioon pitää tehdä nopeassa aikataulussa muutoksia. DevOps -malli pyrkii vastaamaan tähän ongelmaan.

DevOps on yhdistelmä kahdesta sanasta: *development* ja *operations.* Kuten nimestä voi päätellä, DevOps on tapa yhdistää nämä kaksi eri maailmaa: kehittäjien työ ja järjestelmäasiantuntijoiden työ. Tavoitteena on näiden kahden ryhmän töiden tuomista lähemmäksi toisiaan tavalla, joka edistää tiimien välistä kommunikaatiota ja joustavuutta. Näiden kahden toiminnon tuominen lähemmäksi toisiaan on kulttuurinen muutos, mutta keskiössä muutoksessa ovat työkalut. Tarkoitus on antaa molemmille osapuolille työkaluja selviytyä kumpaankin liittyvistä tehtävistä, jotta henkilö voi suoriutua haluamastaan tehtävästä alusta loppuun saakka (Ebert, Gallardo, Hernantes, & Serrano, 2016).

DevOps jakaa kehityksen vaiheet karkeasti kolmeen: *build* (koontivaihe), *deployment* (käyttöönotto) ja *operations* (käyttövaihe). Nämä eivät kuitenkaan ole peräkkäisiä vaiheita projektissa, vaan yhtä aikaa käynnissä olevia projektin toimintoja, joiden läpi muutokset kulkevat. Jokaiseen vaiheeseen liittyy työkaluja, jotka pyrkivät rakentamaan siltaa eri tiimien ja toimintojen välille. Tärkeässä osassa kaikkien työkalujen osalta on automaatio, oli kyseessä sitten koontivaiheen sovellusversioiden automaattinen siirtyminen testipalvelimille ajoon *continuous-integration* (jatkuvan integraation) työkalujen avulla tai tarvittavien resurssinen määritteleminen koodina (*infrastructure as code)* (Ebert et al., 2016).

Miten tämä malli liittyy chatbotteihin? Valtaosa chatbotteihin liittyvästä tutkimuksesta on luonnollisia kieliä käyttävää, assistentinomaista chatbottia koskevaa. Tämä jatkaa varhaisten keskustelubottien perinnettä tietotekniikan alkutaipaleelta. Keinoälytoteutukset jatkavat kasvuaan etenkin mobiilialustoilla. Tästä esimerkkeinä toimivat laajalle levinneet Applen Siri ja Googlen Assistant. Tämä saattaa chatbotteja laajemman yleisön tietoisuuteen (Zamora, 2017).

Tähän tutkimukseen liittyy DevOps-mallissa käytetty käsite *ChatOps*. Yleisimmät kuvaukset ChatOps työkaluista toteuttavat DevOps-ajattelua, usein varsin suoraan. Näissä kuvataan DevOps-työkalujen sijoittamista chatbotin taakse, jolloin chatbot toimii pikemminkin käyttöliittymänä olemassa olevaan toteutusputkeen, kuin että botti yrittäisi luonnollisesta kielestä tulkita käyttäjän aikomusta. ChatOps onkin DevOps-työkalujen tuomista ryhmäkeskustelujen sisään (Hand, 2016).

Tutkiessaan chatbotteihin liittyviä mielikuvia ja odotuksia Zamorra artikkelissaan *I'm Sorry, Dave, I'm Afraid I Can't Do That: Chatbot Perception and Expectations* (2017) havaitsi, että käyttäjille on usein epäselvää mitä kaikkea chatbot pystyy tekemään, jolloin odotuksia oli vaikea määritellä. Hän toteaakin, että on tärkeää pystyä osoittamaan toiminnallisuuksia ja auttaa käyttäjiä asettamaan odotuksia botin toiminnalle. Tämä vaikuttaa suoraan siihen, kuinka käyttäjä kokee botin suoriutuvan annetussa ympäristössä ja siten kuinka paljon ja millä tavoin käyttäjä bottia hyödyntää (Zamora, 2017).

Näiden asetettujen odotusten täyttäminen ei kuitenkaan ole triviaalia. Odotuksiin vaikuttaa käyttäjän kokemustaso, jolloin aiemmat kokemukset vaikuttavat siihen, kuinka nopeasti ja tarkasti käyttäjä odottaa palvelun toimivan. Tämä osaltaan vaikuttaa siihen, että Zamorra raportoi joidenkin käyttäjien kokeneen, ettei chatbotin käyttö nopeuttanut annetusta toimenpiteestä suoriutumisessa vaan tuntui ylimääräiseltä työvaiheelta.

Esimerkkinä Zamorra antaa tilanteen, jossa tavallisen hakukoneen käyttö on tehokkaampaa ja informatiivisempaa kuin chatbotin käyttäminen sekä pelkässä tiedonhaussa että monimutkaisemmissa tapauksissa, joissa vaaditaan tiedonhakua yhtenä osana tehtävästä suoriutumista (Zamora, 2017).

Zamorra huomauttaa, että yksinkertaiset, suoraviivaiset ja toistuvat tehtävät näyttävät sopivan chatboteille parhaiten. Käyttäjät haluavat työkaluja, jotka säästävät aikaa ja lisäävät tehokkuutta. Parhaiten tutkimuksessa toimineet tehtävät olivat päivittäisiä rutiininomaisia toimenpiteitä, jotka eivät juurikaan vaatineet ihmisen varmistusta. Tämä tarkoittaa, että luottamus tehtävästä suoriutumiselle on melko matala ja ongelmatilanteen syntymisen seuraukset ovat kohtalaisen pienet (Zamora, 2017).

Toxtili, Monroy-Hernández ja Cranshaw artikkelissaan *Understanding Chatbot-mediated Task Management* (2018) tuovat esiin ongelman tiimien työskentelyssä. Vaikka tehtävienhallinta ja kommunikaatio nähdäänkin keskeisinä tekijöinä toimissa tiimissä, jostakin syystä tässä tapahtuneet kehitysaskeleet ovat kehittyneet tiimien pääasiallisen kommunikaatiokanavan ulkopuolelle. Mikäli tiimin pääasiallinen yhteydenpitoväline on pikaviestin tai ryhmäkeskustelu kuten Slack, tulisi myös tehtävänhallinnan hyödyntää samoja viestikanavia. He myös toteavat osuvasti informaatiotyöläisten jatkuvasti joutuvan vaihtamaan kontekstia siirtyessään työkalujen ja viestimien välillä (Toxtli, Monroy-Hernández, & Cranshaw, 2018).

ChatOps vastaa omalta osaltaan näihin huomioihin. ChatOps on ryhmäkeskustelutyökalujen käyttämistä muuhunkin kuin vain keskusteluun. Tällöin voidaan tuoda keskustelu toimenpiteestä lähelle työkalua, jolla toimenpide tehdään, samalla dokumentoiden sekä aiheeseen liittyvä keskustelu että työkalun käyttö ja sen tuottama tuloste. Näin saadaan yhdistetyksi kommunikaatio, toimenpide ja informaation näkyminen tavalla, joka on kaikkien tiimin jäsenten saavutettavissa (Hand, 2016).

Yllämainitut haasteet tulee kuitenkin ottaa huomioon ChatOps-työkaluja ja työkulkuja suunnitellessa. Erityisesti tämän työn osalta kiinnostavia DevOps-mallin osuuksia ovat käyttövaiheeseen liittyvät lokien käsittely ja monitorointi, joihin testausautomaation tulosten esittäminen voidaan laskea (Ebert et al., 2016).

2.5 Yhteenveto

Edellä esittelin viitekehyksen, jonka perusteella seuraavassa luvussa esiteltäviä prototyyppejä lähdettiin suunnittelemaan. Tavoitteena on yllä kuvatun järjestelmän tilan tuntemuksen lisääminen testausautomaation tulosten avulla yhdistämällä tähän niiden esittämiseen rakennetut chatbot ja jaettu näyttö.

Jaetun näytön keskeiseksi suunnitteluhaasteeksi aiempien tutkimusten perusteella nousee paitsi sisältö, myös näytön sijoittelu. Kaikkiaan informaation määrän ja näytön sijoittelun tulee olla sellaista, että kyetään minimoimaan näytön seuraamisesta aiheutuva kognitiivisen kuorman kasvaminen.

Chatbottien suunnittelussa voidaan seurata ChatOps-ajattelun luomaa mallia, jossa työkalut pyritään tuomaan lähelle käyttäjien kommunikaatiokanavaa. Tällöin työkalun käyttäminen toimii samalla sen käytön ja tulosten dokumentointina muita tiimin jäseniä varten.

Seuraavassa luvussa kuvataan ympäristö johon prototyypit rakennetaan ja sen esittämät vaatimukset, sekä prototyyppien suunnittelu ja testaus.

3 PROTOTYYPPIEN SUUNNITTELU JA TESTAUS

Tässä luvussa esitellään työn kohteena oleva järjestelmä ja toteutetut prototyypit. Aliluvussa 3.1 kuvaan ensin asiakasrajapinnan toimintaa. Aliluvussa 3.2 esittelen nykyisen palvelussa käytössä olevan testausautomaation. Aliluvussa 3.3 kerron prototyyppien suunnitteluvaiheesta ja aliluvussa 3.4 lopulta testausprosessista ja sen tuloksista.

3.1 Asiakasrajapinta

Työn kohteena olevan järjestelmän kohderyhmä on Alma Mediapartners Oy:ssä toimiva Etuovi.com ja Vuokraovi.com -palveluiden asiakaspalvelu. Asiakasrajapinnassa työskentelee kuusi henkeä, käsitellen kontakteja puhelimitse, sähköpostitse ja palveluissa näkyvän chat-palvelun kautta. Kyseisten palveluiden asiakaspalvelun työtehtävät vaihtelevat yksinkertaisesta käyttöneuvonnasta ja asiakassuhteen hoidosta aina kohtalaisen vaativaan, teknistä osaamista vaativaan ongelmien ratkomiseen. Koska asiakaspalvelu on ensimmäinen kontaktipinta asiakkaisiin, he huomaavat muuhun organisaation verrattuna melko nopeasti, mikäli palvelussa on jotakin vialla. Tämä näkyy suoraan asiakaskontaktien määrässä ja laadussa: kun palvelu tuottaa vain valkoista sivua asiakkaat ottavat herkästi yhteyttä. Kun tämänkaltaisia yhteydenottoja alkaa tulla, normaali työnkulku on seuraava:

1. Asiakaspalvelija pyrkii toistamaan ongelman omalla koneellaan asiakkaan ollessa yhä yhteydessä, mikäli kontakti on tullut puhelimitse. Hän myös pyrkii selvittämään onko ongelma korjattavissa käyttäjän toimintaa muuttamalla, toisin sanoen, onko kyseessä käyttäjävirhe?

2. Asiakaspalvelija pyrkii arvioimaan ongelman laajuutta: onko kyseessä yksittäisen asiakkaan ongelma vai koskeeko kaikkia käyttäjiä? Mitä osiota palvelusta ongelma koskee?

3. Asiakaspalvelija tiedustelee ympärillä olevilta kollegoiltaan, onko heille tullut vastaavia kontakteja.

4. Asiakaspalvelija konsultoi palveluiden ylläpitotiimiä, että onko asiakkaan kohtaamaan ongelmaan liittyviä laajempia virhetilanteita tiedossa.

Vaikka tätä työnkulkua ei ole formalisoitu viralliseksi prosessiksi, normaali kontaktin eskaloituminen tilanteessa noudattaa toistuvasti samaa, yllä esitettyä kaavaa. Tämä havainto perustuu omaan kokemukseeni työskentelystä asiakaspalvelussa ja ylläpitotiimissä.

Asiakasrajapinta itsessään on tekniseltä osaamistasoltaan varsin korkealla, mutta järjestelmäylläpidon ja ohjelmistotuotannon osaamista heillä ei yleisesti ottaen ole. Käytännössä tämä tarkoittaa, että he kykenevät selviämään useimmista asiakkaan teknisistä ongelmista, mutta mikäli ongelma vaatii datan korjaamista suoraan tietokantaan tai sovelluksen lokien lukemista he eskaloivat tapausta seuraavalle tasolle. Heillä kuitenkin on osaamista ja kokemusta tunnistaa ja luokitella asiakkaan ongelma kontaktin perusteella joko käyttäjän tai käyttäjän laitteen ongelmaksi tai yleisemmäksi palvelua koskevaksi ongelmaksi.

Ylläpitotiimi on asiakaspalvelusta seuraava taso ja luonteeltaan huomattavasti teknisempi. Ylläpitotiimillä on pääsy palvelinten lokitiedostoihin, itse palvelimiin ja tietokantoihin. He kuitenkin tarkastelevat tilannetta hieman samalla tavalla: asiakaspalvelun kuvauksen perusteella, itse järjestelmän tilaa tarkastelemalla, etsimällä saapuneita virheilmoituksia tai lokeja sekä katsomalla onko palveluun liitetty testausautomaatio törmännyt virhetilanteeseen. Palveluihin on yleisellä tasolla liitetty testausautomaatiota neljällä tasolla: kehitystyön mukana tuomaa yksikkötestausta, järjestelmän avainparametrien valvontaa, automaattihälytyksiin liitettyjä testejä ja lopulta laajempia regressiotestejä.

Tämän työn keskeisiä tavoitteita on tuoda ketjun alkupää, eli asiakaspalvelijan ensimmäinen kontakti asiakkaan ongelman kanssa lähemmäs ketjun loppupäätä, eli testausautomaation tuomaa lisähyötyä järjestelmän tilan arvioinnissa. Nykyisellään testausautomaatio suorittaa järjestelmän valvontaa, mutta tasolla, jossa virheeseen törmääminen aiheuttaa kiireellisissä tapauksissa hälytyksen päivystäjälle tai vähemmän kriittisissä tapauksissa ilmoituksen notifikaatioille tarkoitetulle Slack-kanavalle. Kuitenkin tieto tästä valuu useamman käden kautta asiakaspalveluun saakka. Lisäksi valvontaa suorittavan automaation tulee olla hyvin karkeatasoista, jotta siitä saadaan täysi hyöty suhteessa vaivaan: sivun latautumisen hidastuminen ei ole yhtä kriittinen vika kuin koko palvelun käytön estyminen, mutta saattaa silti aiheuttaa huomattavan määrän kontakteja asiakaspalveluun.

On siis tarvetta löytää tapa lisätä asiakaspalvelun järjestelmän tilan tuntemusta jo ensimmäisen kontaktin aikana. Mikäli asiakaspalvelija saa reaaliaikaista informaatiota ympäristön tilasta, hän kykenee myös hoitamaan kontaktin nopeammin ja laadukkaammin:

mikäli järjestelmän tilassa automaatiotestien perusteella näkyy ongelma, jää kontaktin hoitamisessa ongelman laajuuden selvittely vähemmälle. Täten kontaktissa voidaan keskittyä ongelman mahdolliseen kiertämiseen ja korjaamista varten tarvittavan informaation keräämiseen. Tarkoitus ei kuitenkaan ole korvata järjestelmän tilaa valvovaa automaatiota, joka hoitaa hälytykset ylläpitäjille tai muodostaa aina virheilevää testipatteria, johon ei reagoida. Tarkoitus on tuoda ajantasaista mittarointia nopeasti omaksuttavassa muodossa asiakaskontaktien äärelle.

Työtä varten kävin keskusteluja aiotun käyttäjäryhmän, eli asiakaspalvelun henkilöiden kanssa järjestelmän tarpeellisuudesta ja mahdollisista ominaisuuksista. Oma arvioni ennen keskusteluja oli, että järjestelmä on käyttökelpoinen mikäli:

1) sen käyttö ei ole käyttötavaltaan liian kuormittava normaalin työn ohessa

2) se antaa tulokset riittävällä varmuudella

Alustavissa keskusteluissa tuli ilmi paitsi innostus esitettyä ajatusta kohtaan, mutta myös huoli nimenomaan siitä, että onko järjestelmän käyttö työn aikana liian kuormittavaa suhteessa siitä saatavaan hyötyyn.

3.2 Nykyinen testausautomaatio

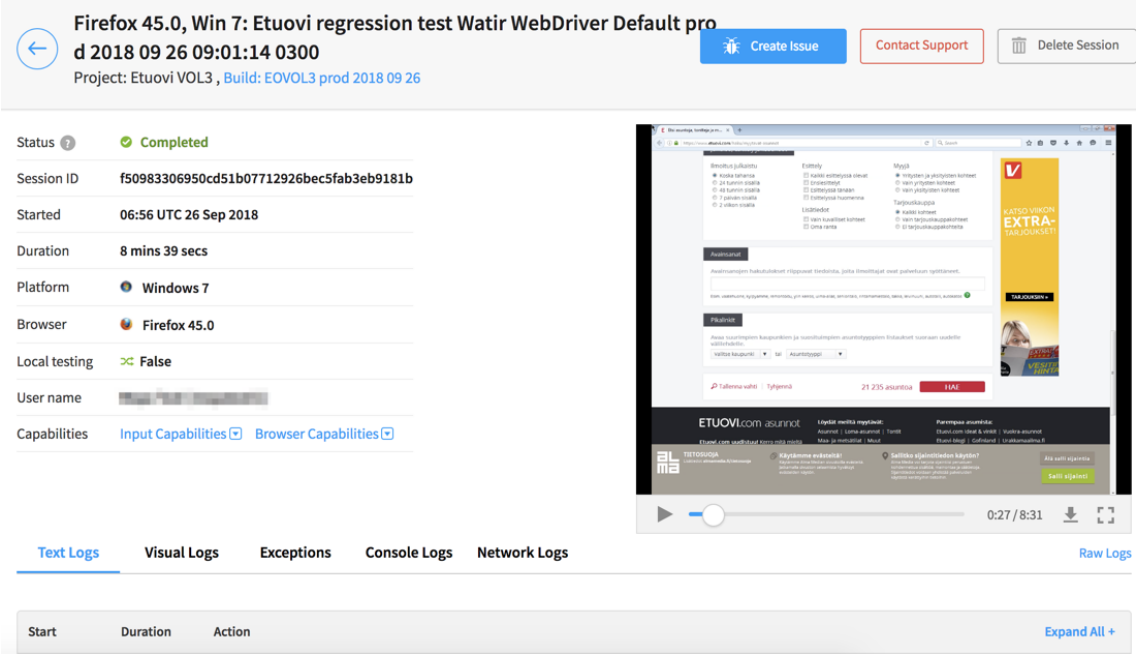
Tässä luvussa esitellään tiiviisti nykyinen käytössä oleva testausautomaatio ja sen käyttötarkoitukset. Ensin tekniseltä ratkaisuiltaan, sitten käyttötarkoituksiltaan.

3.2.1 Eri menetelmät

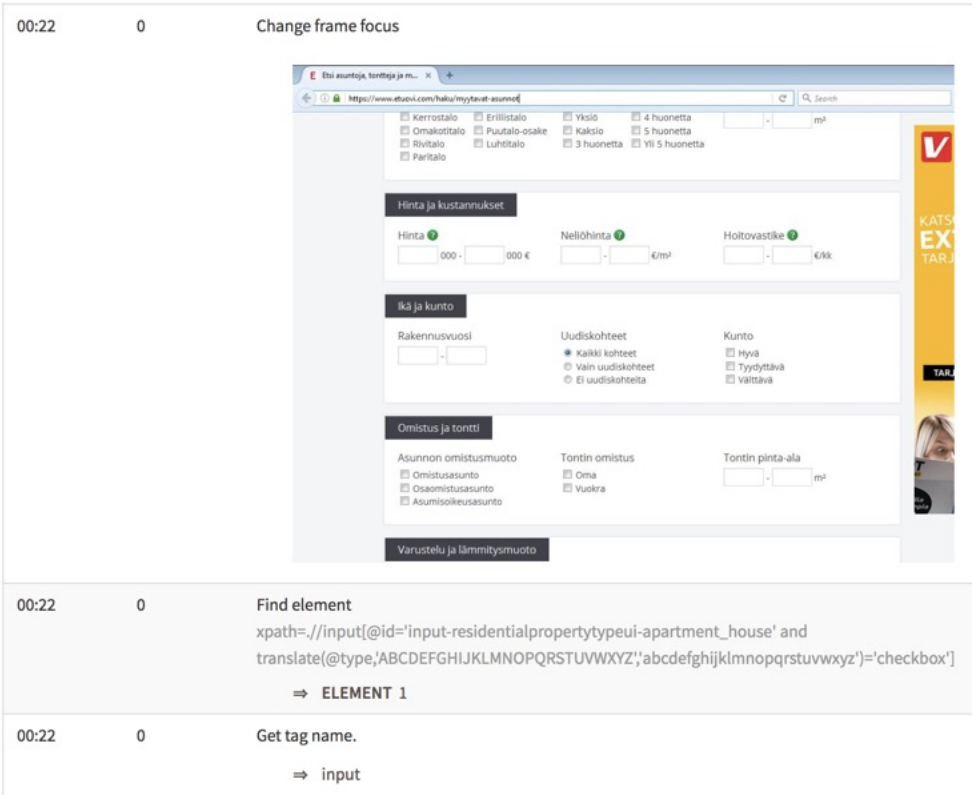
Testausautomaatio, johon tämä työ perustuu, on kahdessa eri käytössä hieman eri laajuudessa. Suppeampi testijoukko ajetaan automaattisesti kolmasti päivässä ja laajempi, regressiotestaukseen apuna toimiva testijoukko ajetaan manuaalisesti sovellusten päivitystä valmisteltaessa. Näiden lisäksi on käytössä koko ajan ajossa olevat, eri tekniikoilla toteutetut järjestelmät: varsin suppea järjestelmän toimivuutta seuraava järjestelmä, joka varmistaa lähinnä, että sivut latautuvat ulkoverkosta tarkistellessa ja laajempi, palvelun tilaa monitoroiva järjestelmä, joka seuraa muun muassa kantalukkojen ja viestijonojen pituuksia.

Testausautomaatiojärjestelmä on toteutettu Ruby-ohjelmointikielellä Rspec-testauskirjaston (“RSpec: Behaviour Driven Development for Ruby”), Watir-selainohjauskirjaston

(“Watir Project,”) ja Browserstack-palvelun selainvirtualisoinnin avulla (“Browserstack”). Käytännössä RSpec hoitaa testien varsinaisen muotoilun ja ohjaa Watir-kirjaston avulla selainta, joka varsinaisesti ajetaan Browserstack-palvelussa (kuva 1). Browserstack tuottaa selaimen toiminnasta lokia ja videon, jota voidaan myöhemmin tarkastella (kuva 2). Koko järjestelmää ajetaan Jenkins-palvelussa, jonka tehtävänä on ajastetusti suorittaa testejä, tallentaa tuloksia ja välittää niistä muodostettuja raportteja Slack-kanaville.



Kuva 1. Browserstack-palvelussa ajettava testi. Palvelu käynnistää selaimen, joka ohjelmallisesti ohjattuna suorittaa testiä.

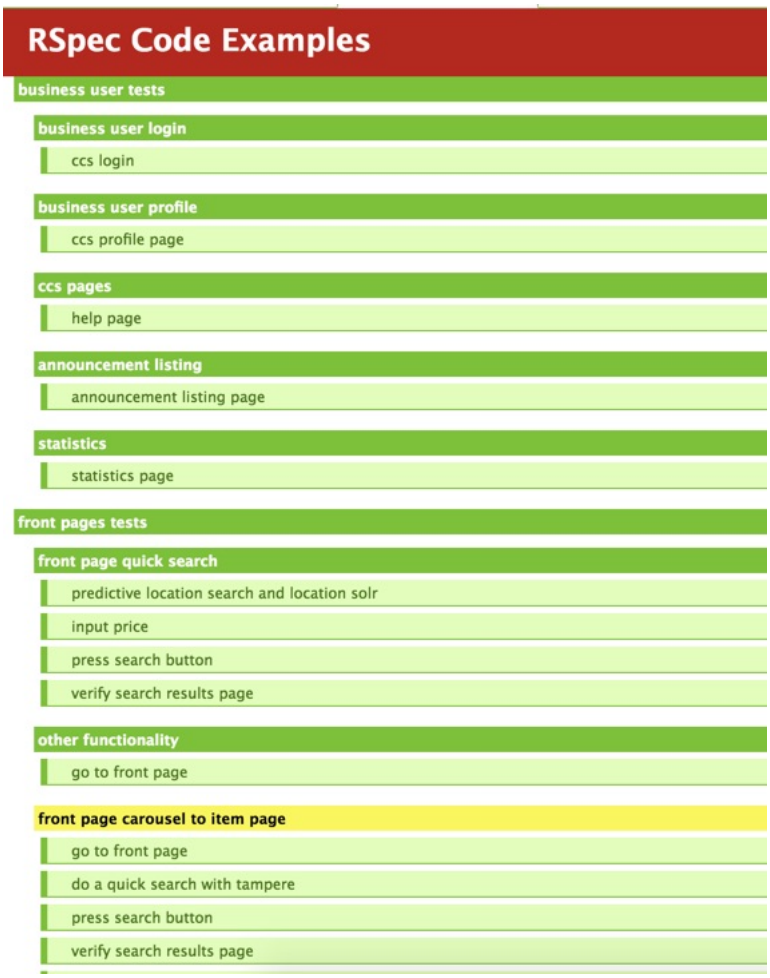


Kuva 2. Browserstack-palvelussa ajettavan testin lokia. Kuvassa näkyy kuvakaappaus selaimeen renderöidystä sivusta ja sitä seurannut input-elementin etsiminen sivulta.

Alla on esimerkki rspec-testauskirjastolla kirjoitetusta testistä, jossa aiemmin alustettu selainta ohjaava Watir-olio etsii sivulta tekstikentän ja syöttää siihen merkkijonon “tampere” ja painaa esiin tulevaa elementtiä suorittaakseen haun:

```
@browser.text_field(:name,
'locationSearch:locationInput').set!("tampere")

@browser.element(:xpath,
'//label[@searchcode="FI_PIRKANMAA_TAMPERE"]')
       .wait_until_present

@browser.element(:xpath,
'//label[@searchcode="FI_PIRKANMAA_TAMPERE"]').click

print "Searching with location \n"

end
```

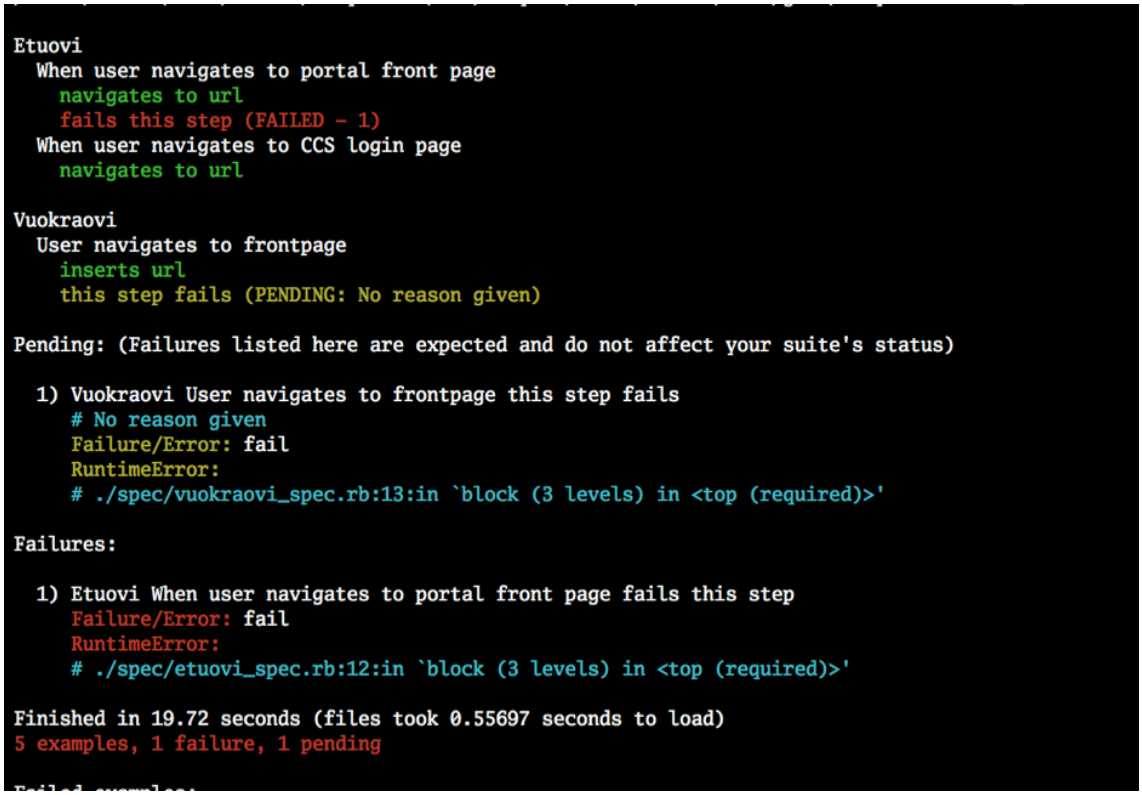
Esimerkki rspec-kirjaston ruby-koodista



Kuva 3. Esimerkki rspec-testausraportista tuotantopalvelussa. Yläpalkki muuttuu punaiseksi mikäli kaikki testiaskeleet eivät mene läpi. Keltaisella on merkitty pending-tilassa oleva testi, jonka tiedetään epäonnistuvan.

RSpec antaa suoritetuista testeistä itsessään kahta erityyppistä raporttia: yksi tekstimuotoinen, ajonaikainen raportointi RSpecin ”—documentation” vivulla, toinen ajon lopuksi muodostuva html- tai json-muotoinen raportti RSpecin ”--output html” tai ”–output json” vivulla. Kuvassa 3 esitellään RSpecin tuottama html-raportti. Kuvassa 4 esitellään esimerkki RSpecin ajonaikaisesta raportoinnista. Kuvassa valkoisella näkyy testiaskelten kuvaukset, vihreällä läpi menneet testiaskeleet, punaisella epäonnistuneet ja keltaisella pending- eli odottavassa tilassa olevat testiaskeleet.

Pending-tila testeissä tarkoittaa testiaskelta, jonka suoritus epäonnistuu, mutta on merkitty tiedossa olevaksi. Näin voidaan osoittaa, että jokin ominaisuus on joko toteuttamatta tai jonka tiedetään olevan rikki, mutta korjaus on työn alla.



Kuva 4. RSpecin ajonaikainen raportointi esimerkkitesteillä

Lyhyempi testijoukko tässä yhteydessä on suhteellinen käsite, sillä koko joukon ajo kestää yhteensä noin tunnin. Testijoukko sisältää palvelun tyypillisiä käyttöskenaarioita niin sanottua *happy pathia* läpi käyden. Happy path tässä yhteydessä tarkoittaa normaalia, oletettua kulkua järjestelmän toimintojen läpi jonkin toimenpiteen suorittamiseksi. Näitä ovat esimerkiksi:

- Käyttäjä kirjautuu palveluun ja lisää kohteen suosikiksi
- Käyttäjä tallentaa ja poistaa hakuvahdin
- Käyttäjä käy verkkopankissa maksaakseen ilmoituksen

Näitä toimenpiteitä läpikäydessä ei ole tarkoituskaan etsiä mahdollisia rajatapauksia, vaan varmistaa järjestelmän normaali toimivuus. Tämä kuitenkin antaa verrattain hyvän kuvan siitä, että mikäli jokin perusominaisuus hajoaa, nähdään testien tuottamasta lokituksesta milloin se on edellisen kerran toiminut.

Laajempi testijoukko taas on tarkoitettu testaajien tueksi, kun palveluun tehdään tuotantopäivitys. Tällöin testaajien tehtäviin kuuluu paitsi uusien ominaisuuksien testaaminen, mutta myös sen varmistaminen, ettei järjestelmään hiivi päivityksen myötä vanhoja

ominaisuuksia rikkovaa regressiota. Tämän vuoksi keskeisistä toiminnoista on rakennettu testejä, jotka ovat jossakin yksikkötestien ja käyttöliittymätestauksen välimaastossa. Nämä testit pyrkivät varmistamaan, että myös harvinaisemmat tapaukset tulevat testatuksi. Tämän merkitys korostuu erityisesti tapauksissa, joissa manuaalisesti testaaminen on raskas prosessi. Lisäksi osa testeistä on järjestelmälle tuhoisia, joten ne voidaan suorittaa vain testiympäristöissä. Tuhoisaksi testiksi voidaan laskea ilmoituksen julkaiseminen tai poistaminen sekä yhteydenottopyynnön lähettäminen, sillä tätä ei haluta tehdä tuotantopalvelussa. Näihin testeihin kuuluvat esimerkiksi:

- Käyttäjä lähettää yhteydenottopyyntöjä, kunnes käytetty osoite asetetaan estolistalle
- Käyttäjä täyttää uuden ilmoituksen lomakkeen jokaisen kentän satunnaisilla arvoilla ja julkaisee ilmoituksen
- Käyttäjä ostaa kohteelle lisänäkyvyystuotteita

3.2.2 Regressiotestaus ja valvonta

Palvelua seuraavia järjestelmiä on siis useita. Karkeasti nämä menetelmät voidaan kuitenkin jakaa kahteen: testaus ja valvonta. Mihin tässä työssä tarkasteltu järjestelmä jaottelussa asettuu? Rspec ja Watir -yhdistelmää käytetään yleensä testaukseen pikemminkin kuin suoranaiseen valvontaan. Näin saadaan hyötyä siitä, että järjestelmä avaa selaimen ja vuorovaikuttaa palvelun kanssa sen käyttöliittymän kautta, ei suoraan rajapintoja testaamalla. Tällä tavoin saatetaan tavoittaa ongelmia, joita asiakkaat palvelua käyttäessään kohtaavat, mutta joita ei välttämättä yksittäisiä taustajärjestelmiä monitoroidessa havaitse.

Normaali työnkulku kehittäjillä uusia ominaisuuksia tai muita muutoksia tehdessä on kehittää ensin paikallista kehitysympäristöä vasten. Kun kehitystyö on täällä saatettu riittävän pitkälle, siirtää kehittäjä työnsä erilliseen testiympäristöön. Etuoven tapauksessa näitä testiympäristöjä on kaksi: QA1 on lähempänä versionhallinnan päähaaraa ja QA2 on lähempänä tuotannossa olevaa versiota. QA1-ympäristöön tehty muutos käy läpi testaajien manuaalisen testausvaiheen varmistukset, kun ominaisuus tai muutos on valmis. Tämän jälkeen muutos poimitaan julkaisukandidaattiin, joka käynnistetään QA2-ympäristöön. Tässä vaiheessa testaajat yleensä vielä varmistavat, että uuden ominaisuuden ympärillä kaikki toimii ja ajavat regressiotestaukseen liittyvät automaattitestit. Tässä

yhteydessä suoritetaan myös manuaalinen regressiotestaus, jossa testaajat käyvät palvelua läpi eri selaimilla varmistaen etteivät muutokset ole rikkoneet olemassa olevia toimintoja. Iso osa regressiotestaamisesta on melko monotonista lomakkeiden ja kenttien toimivuuden testausta, jota on voitu ulkoistaa automaattitesteille. Näitä testejä ajetaan eri selain ja käyttöjärjestelmäkombinaatiolla, jotta saadaan mahdollisimman laaja kattavuus.

Nämä testit ovat yksityiskohtaisia ja raskaita ajaa, eivätkä yli tunnin suoritusajat ole mitenkään poikkeuksellisia. Tämän lisäksi testien tulosten tulkitseminen vaatii harjaantumista. Testausautomaatiossa yleinen ongelma on herkät testit, jotka saattavat antaa vääriä virhetuloksia. Tämä vaatii testien käyttäjältä testien tiettyjen osien tai testijoukkojen uudelleen ajamista, manuaalista varmistamista ja tulosten analysointia.

Edellä mainittujen lisäksi järjestelmää valvoo kourallinen pienempiä, eri tekniikoin toteutettuja testejä, jotka lähinnä varmistavat, että palvelu on ajossa ja sen osaset toimivat teknisesti oikein. Näitä ovat muun muassa Applications Manager (“Application Monitor Tools | Application Performance Monitoring (APM),” Zoho Corp.), joka valvoo palvelun toimintaa aina url-osoitteiden toimivuudesta tietokantojen kantalukkoihin ja dedikoitu, ulkoverkosta sivujen lataamista testaava hälytysjärjestelmä.

3.3 Suunnittelu

Tässä luvussa kuvaan prototyyppien suunnittelun vaiheita. Toteutin asiakasrajapintaa varten optimoidun testijoukon ja kaksi käyttöliittymäprototyyppiä, joiden kautta tuloksia tarkastellaan. Tavoitteena ei ollut tehdä määrällistä evaluointia käyttöliittymien käytöstä, vaan tarkastella ja pyrkiä ymmärtämään esitettyjen käyttöliittymien mahdollisuuksia asiakasrajapinnan työskentelyn tukena.

3.3.1 Alustavat haastattelut

Työtä varten kävin useita lyhyempiä vapaamuotoisia keskusteluja asiakaspalvelun työntekijöiden ja ylläpitotiimin kanssa sekä yksin että suuremmalla ryhmällä. Näissä keskusteluissa korostui etenkin kannatus yleisellä tasolla ajatukselle tuoda informaatiota lähemmäs asiakkaita, mutta myöskin huoli siitä, että kyseessä on jälleen yksi työkalu, jota pitää seurata kontaktin aikana jo ennestään huomattavan työkalumäärän lisäksi. Lisäksi useassa keskustelussa pohdittiin, mitä dataa he kokivat järjestelmän tilasta tarvitsevansa.

Muutamassa tapauksessa huoli jälleen yhdestä seurattavasta järjestelmästä ja toiveet tarvittavan informaation määrästä ja luonteesta tuntuivat olevan jonkin verran ristiriidassa keskenään: samanaikaisesti osa toivoi järjestelmän toimintojen avainlukuja näkyville, kun osa arvioi, etteivät todennäköisesti alun kokeilun jälkeen tietoja katsoisi. Lisäksi osa toivotusta informaatiosta oli joko tämän työn kattavuuden ulkopuolella tai erittäin vaikeasti saatavissa.

Prototyyppien suunnitteluvaiheessa kävin lisää keskusteluja asiaan liittyvien tahojen kanssa. Keskustelut olivat vapaamuotoisia tilaisuuksia, joissa pyrin hahmottamaan eri toimijoiden näkökulmia työhön. Keskeisessä roolissa keskusteluissa olivat asiakaspalvelun työntekijät, joiden kanssa tulisin myöhemmin prototyypin testauksen suorittamaan. Keskusteluita käytiin asiakaspalvelun tekijöiden kanssa kahdenkeskisissä tilanteissa sekä lyhyemmin laajempien palavereiden yhteydessä, joissa oli läsnä suurin osa asiakaspalvelusta samaan aikaan.

Viimeiset näistä keskusteluista käytiin muutamia viikkoja ennen prototyyppien testauksen aloittamista. Näissä keskusteluissa tuli uudelleen ilmi muutamia seikkoja, joista oli keskusteltu jo aiemmin. Ensinnäkin ajatus koettiin yhä yleisellä tasolla varsin hyväksi. Yleisesti ottaen kaikki informaation tuominen asiakaspalvelun lähelle koettiin oikeaksi kehityssuunnaksi. Esitetyt toiveet eivät kuitenkaan olleet aina täysin linjassa tämän työn aihepiirin kanssa. Kommenteissa toivottiin sekä työkaluja ohjata järjestelmän toimintaa, kuten työkalua virheilevien palveluiden uudelleenkäynnistämiseen, että yksityiskohtaista informaatiota järjestelmän avainluvuista. Avainluvuista esille nousivat palvelujen kävijämäärät, kuvankäsittelyjärjestelmien työjonojen pituudet ja aineistosiirtojen toiminta. Koska näiden toteuttaminen vaatisi uusia teknisiä ratkaisuja tai organisaation työnjaon muutoksia, rajasin tämän suuntaisia toiveita pois, mutta välitin huomioita eteenpäin. Testauksen alettua asiakaspalvelu saikin käyttöönsä muun muassa työkalun erään virheilevän sisäisen työkalun uudelleenkäynnistämiseksi ilman ylläpitotiimin puuttumista asiaan.

Vastaavia työkaluja toivottiin eritoten chatbotin kohdalla. Kokeeksi toteutin chatbotille yhden toiminnallisuuden toiveisiin liittyen. Prototyypin testaamisen yhteydessä tämä toiminnallisuus näkikin jonkin verran käyttöä. Toiminto haki palveluun jätetystä ilmoituksesta lisätietoja, joita usein joutuu tarkastelemaan ongelmatilanteita selvittäessä.

Alustavissa haastatteluissa kävi myös ilmi, joskaan ei kovin yllättäen, ettei asiakaspalvelussa oltu kovin laajasti tietoisia nykyisistä ratkaisuista testausautomaation parissa. Myöskään palveluiden seuraamiseen käytetty Applications Manager ei ollut heille suoraan tuttu. Applications Managerin käyttöliittymän monimutkaisuus kuitenkin oli luotaantyöntävä, kun palvelua lyhyesti keskustelun aikana esiteltiin.

Nämä kuitenkin herättivät keskustelua siitä, että asiakaspalvelijat eivät uskoneet käyttävänsä ylimääräistä palvelua muutaman mielenkiinnosta tehdyn alustavan kokeilun lisäksi. Monet totesivat, että kontaktin hoitamisen aikana heillä on jo useita työkaluja, joita seurata samanaikaisesti. Tämän vuoksi yhden hyvänkin työkalun lisääminen työnkulkuun on huomattavan vaikeaa. Tässä korostui prototyyppien suunnittelua silmällä pitäen vaivattomuuden vaatimus: mikäli järjestelmän käyttö on edes kohtalaisen vaivalloista, jää järjestelmästä saatu hyöty saamatta, koska sitä ei ehditä tai voida käyttää silloin kun informaatiosta olisi hyötyä. Tämä liittyy suoraan luvussa 2 esiteltyyn kognitiivisen kuorman ongelmaan.

Itse tein huomion, että saadun tiedon oikeellisuuden vaatimus kasvaa huomattavasti tuodessa tuloksia lähemmäs asiakkaita: kun järjestelmä näyttää ongelman, on monella refleksinomainen tapa käyttää informaatiota kontaktin ratkaisemiseksi. Esimerkiksi väärä positiivinen tulos ongelmasta saattaa pahentaa käyttäjän ongelmaa, koska asiakaspalvelu kuittaa ongelman laajempana, eikä enää välttämättä huomaa käsitellä asiakkaan ongelmaa erillisenä.

Edellä mainitut havainnot tulee ottaa suunnittelussa huomioon, joten haastattelujen perusteella tiivistin suunnittelun avuksi kolme teesiä:

- Näytettävän informaation määrä on rajallinen
- Näytettävän informaation on oltava ajantasaista: ajon on mentävä läpi maksimissaan viidessä minuutissa
- Näytettävän informaation on oltava luotettavaa

3.3.2 Prototyyppien suunnitteluratkaisujen kuvaukset

Ratkaisuja suunnitellessa työhön valikoitui kaksi eri menetelmää: chatbot ja jaettu näyttö. Keskeistä suunnittelussa oli luoda välineitä, jotka auttavat käyttäjiä saamaan paremman ymmärryksen järjestelmän tilasta luvussa 2.2 esitetyn viitekehyksen mukaan. Näiden välineiden tulisi olla jatkuvasti saatavilla ja toimia yhtenä keskeisenä informaationlähteenä tuottamaan tuota ymmärrystä muiden käyttäjien käytössä olevien työkalujen lisäksi.

Chatbot valikoitui testattavaksi siitä syystä, että yrityksessä oli käynnissä laajempaa ChatOps-kehitystä, jossa ylläpidon ja asiakaspalvelun toimintojen avuksi suunniteltiin työkaluja. Lisäksi ylläpidon ja asiakaspalvelun tärkein yhteydenpitokanava on jaettu Slack-kanava, joten järjestelmän tilan tarkkailun työkalulle tämä oli luonteva valinta.

Jaettu näyttö taas valikoitui koska, kuten alustavissa haastatteluissa tuli ilmi, asiakaspalvelun käyttämien järjestelmien määrä on jo kohtalaisen suuri. Yksi selainikkuna tai työpöytäsovellus lisää olisi lisännyt liikaa kognitiivista kuormaa ollakseen käyttökelpoinen. Jaetun näytön keskeinen hyöty on olla läsnä ja antaa informaatiota ilman jatkuvaa, spesifistä käyttöä. Tämä tukee järjestelmän tilan tuntemuksen tavoitetta kuormittamatta liiaksi käyttäjäänsä.

Tällä tavoin jaoteltuna vastakkain on siis aktiivinen ja passiivinen työkalu saman informaation saamiseksi.

Eräs keskeinen haaste prototyyppien suunnittelussa oli valita sopiva testijoukko. Liian laajan ja yksityiskohtaisen testijoukon ongelmia ovat liika herkkyys virheille, mikäli järjestelmän tilassa tapahtuu pienikin muutos. Lisäksi yksittäisen ajon kestäminen liian pitkään ja liian tarkan tiedon esittämiseen liittyy haasteita. Liian suppea testijoukko taas ei anna riittävästi informaatiota järjestelmän tilasta. Toinen haaste on löytää sopiva testijoukon suoritustiheys. Kolmas haaste on informaation esittämiseen liittyvä: pitäisikö esille tuoda tiivistelmä järjestelmän tilasta vai tarkempi tieto yksittäisten testien onnistumisesta tai epäonnistumisesta?

Alkuperäinen ajatus käyttää olemassa olevia, laajempia testijoukkoja suoraan ei siis onnistuisi: näiden suorittama testaus on joko hyvin spesifiä tai ajo kestää huomattavan pitkään. Lisäksi laajempien ja tarkempien testijoukkojen herkkyys virheille on suurempi kuin pienen dedikoidun testijoukon. Suoraan testin tuloksia tarkastellessa on helppo päätellä, ettei selaimen aikakatkaisusta johtunut virhe välttämättä tarkoita koko järjestelmän

vikaantumista, mutta kun informaatio esitetään tiivistetyssä muodossa, ei toistuvaa, normalisoituvaa virheilyä saa esiintyä.

Näiden haasteiden ja yllä kuvattujen vaatimusten perusteella rakensin Etuovi.com-palvelulle 11 kohtaisen testijoukon, joka on riittävän yksinkertainen, jotta suoritus on nopeaa ja jonka tulokset ovat luotettavat. Testijoukko sisälsi:

1. Etusivun ja sen elementtien latautumisen
2. Etusivun haun latautumisen
3. Etusivun haun toiminta
4. Listasivulla on uusia kohteita
5. Listasivun kohteilla on kuvia
6. Etusivun kirjautumislinkin latautumisen
7. Etusivun kirjautumisikkunan latautumisen
8. Käyttäjän kirjautumisen ja kirjautuneen käyttäjän tietojen latautumisen
9. Käyttäjän uloskirjautumisen
10. Yritysasiakkaan palvelun latautumisen
11. Asiakaspalvelun ja ylläpidon käyttämän ylläpitoportaalin toiminnan

Huolimatta verrattain yksinkertaisesta perustoimintojen testaamisesta, aiheutti testien kirjoittaminen työtä, sillä dynaamisen verkkosivun ollessa kyseessä pitää jokaisessa testissä olla rakennettuna logiikkaa muuttuvan sisällön, mainosten, selaimen eväste-dialogien ynnä muiden käsittelylle.

Prototyyppien rakentamisessa käytettiin yhtä yhteistä testijoukkoa ja niitä ajavaa taustajärjestelmää. Prototyyppien erot tuleva esille tavassa, jolla nämä tulokset esitetään. Tämä valinta liittyy keskeisesti yllä mainittuihin haasteisiin, etenkin testien herkkyyden osalta. Jotta testeistä on hyötyä, tulee niitä ajaa riittävän usein. Testien rinnakkainen ajaminen aiheuttaa kuitenkin huomattavasti enemmän työtä, joka menisi tämän tutkimuksen laajuuden ulkopuolelle.

Sekä tekstimuotoinen että HTML-muotoinen raportointi suoraan RSpecistä tarjoaa paljon hyödyllistä informaatiota, mutta ollakseen suoraan käyttökelpoista asiakaspalvelun tapauksessa sen pitäisi olla nopeammin sisäistettävissä. Vaikkakin järjestelmän yleistä tilaa voi nopeasti päätellä punaisen ja vihreän värin keskinäisellä suhteella, tai jopa ylipäätänsä siitä onko punaista väriä näkyvissä, tarkemman ongelmakohdan hahmottaminen tuloksesta vaatii silmän harjaantumista ja testien sisäisen rakenteen tuntemusta.

Testiin käytetty taustajärjestelmä rakentui oleellisesti kolmesta osasta:

1. Skripti, joka ajaa testit, kopioi tulokset yhteiseen kansioon ja odottaa 60 sekuntia. Tämän jälkeen sama skripti ajetaan uudelleen.

2. RSpec testijoukko, jonka yllä oleva skripti ajaa. Tämä tuottaa JSON-muotoisen raportin tuloksista.

3. Parseri, joka sekä infonäytön että chatbotin tapauksessa lukee JSON-muotoisen raportin ja palauttaa tulokset esitettäväksi käsitellyssä muodossa.

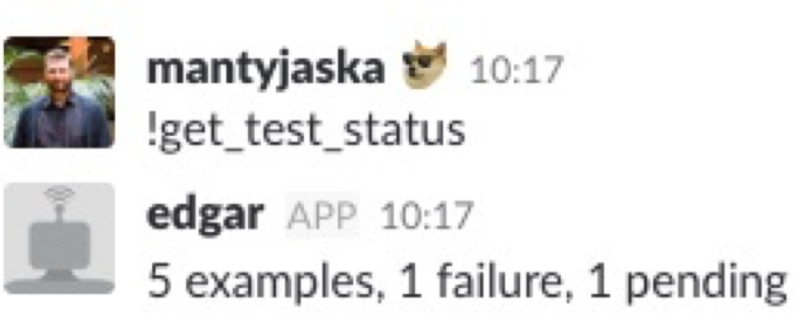
Sama taustajärjestelmä palveli molempia käyttöliittymiä prototyyppejä testatessa. Taustajärjestelmää ajettiin paikallisesti, sillä prototyyppien testauksen rajoitetun keston vuoksi ei ollut tarpeen käynnistää dedikoitua palvelinta. Tämä myös auttoi testien nopeatahtisessa muokkauksessa, kun niihin piti tehdä muutoksia.

3.3.3 Chatbot

Chatbotin ensimmäinen hahmotelma oli vastata yhteen pääkysymykseen järjestelmien tilasta, jota voisi tarkentaa parametrillä, joka rajaa tuloksen vain yhteen palveluun. Kuva 5 esittelee esimerkkinäkymän chatbotin käskyistä ja niiden esimerkkitulosteet.

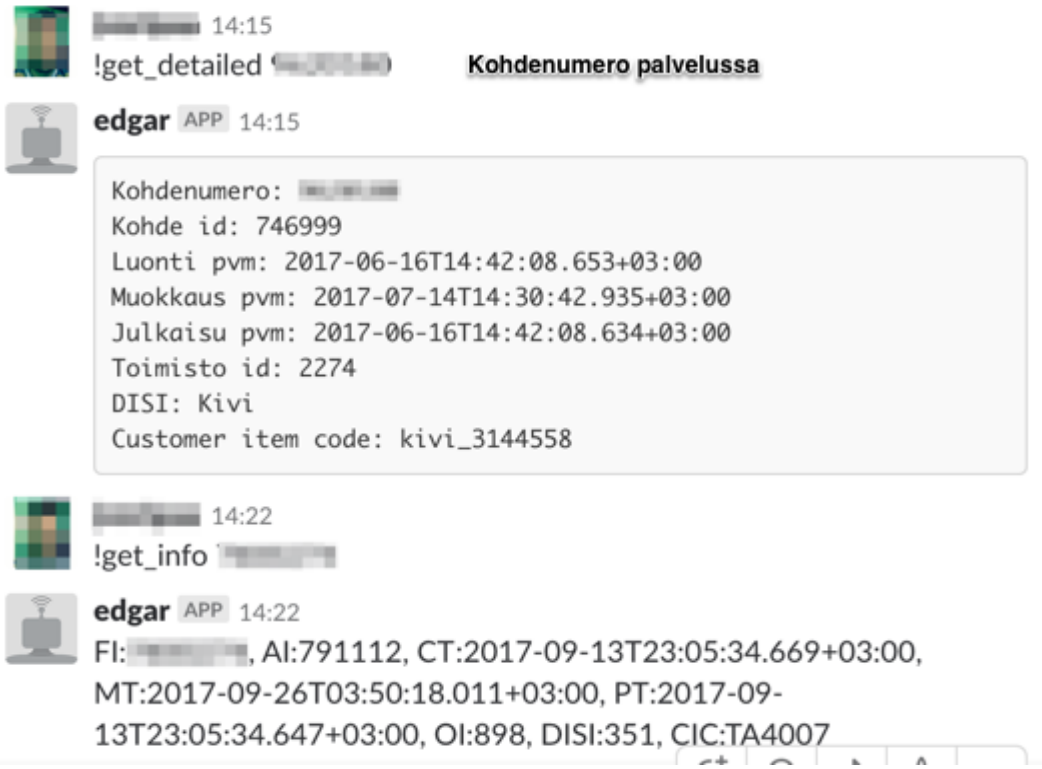
Kuva 5. Alustava hahmotelma chatbotin käskyistä ja tulosteesta

Kuvassa näkyvä *@edgar* on chatbotin nimi Slackissä. Tämän nimen mainitseminen keskustelussa mahdollistaa käskyjen antamisen. Nimi on viittaus *Men in Black* -elokuvaan vuodelta 1997. Prototyyppiä rakennettaessa kuitenkin vakiintui kohtalaisen yleisessä käytössä oleva huutomerkki (!) chatbotille annettavan käskyn merkiksi. Toinen vaihtoehto olisi ollut käyttää kenoviivaa antamaan Slack-mallinen komento, mutta prototyypin suunnittelua helpotti käyttää huutomerkkiä tämän sijasta. Slack-mallisia käskyjä käyttävät sovellukset vaativat jonkin verran enemmän työtä toimiakseen, sillä tällöin tulisi rekisteröidä chatbot erikseen Slack-applikaationa organisaation tilille. Olisi ollut myös mahdollista parsia sopivat käskyt kaikesta viestiliikenteestä, mutta tässä tapauksessa ei kuitenkaan haluttu botin reagoivan jokaiseen viestiin, jonka se saattaisi tulkita käskyksi, vaan haluttiin eksplisiittisesti kutsua toimintoa tarvittaessa. Tämä on yleensä ero tekoälyä hyödyntävien, luonnollista kieltä parsivien virtuaalisten avustajien ja ChatOps-työkalujen välillä, kuten luvussa 2.4 käsiteltiin. Kuvassa 6 näkyy esimerkki Slackissa annetusta komennosta ja esimerkkitulosteesta prototyypin kehitysvaiheessa. Esimerkissä chatbotin antama vastaus on vielä pelkkää tekstiä ilman muotoilua.



Kuva 6. Kehitysvaiheessa oleva chatbot reagoi Slack-kanavalla annettuun komentoon.

Eräs chatbot-ratkaisun puolesta puhuva seikka on, että siihen on verrattain helposti mahdollista lisätä muuta toiminnallisuutta. Tästä esimerkkinä kuvassa 7 kaksi komentoa, jolla voi pyytää lisätietoja palvelussa olevasta kohteesta, kuten luonti- ja julkaisupäivä ja aineistonsiirron tietoja. Nämä toiminnot lisättiin chatbottiin prototyypistä käytyjen keskustelujen perusteella.

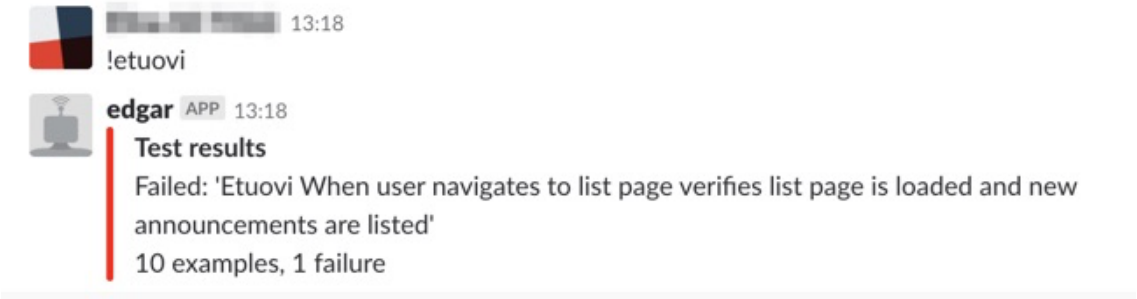


Kuva 7. Esimerkki chatbottiin toteutetusta lisätoiminnallisuudesta. Käyttäjän ja myynti-ilmoituksen julkiset tunnistetiedot tiedot peitetty.

Tämä myös oli eräs syy chatbotin valintaan prototyyppejä rakennettaessa, sillä käynnissä on laajemminkin kartoitusta ChatOps-tyylisten työkalujen tuomista sekä kehittäjien että asiakaspalvelun käyttöön. Näin chatbotin kehitys ja testaaminen edistäisi tätäkin tavoitetta.

Prototyypin lopullisessa versiossa käskyn muodoksi vakiintui "*!etuovi*", sillä tuloksia haluttiin rajata palveluittain ja prototyypin testausvaiheeseen valikoitui vain Etuoven testejä. Vastaavasti tarkoituksena oli myöhemmin lisätä vastaava käsky "*!vuokraovi*" ja tätä nimeämiskäytäntöä edelleen muihin palveluihin jatkaen.

Kuva 8 näyttää tilanteen, jossa käyttäjä pyytää Etuoven testien tilannetta ja yksi testi on kohdannut virheen. Viesti sisältää punaisen palkin, kuvauksen mikä testi on kohdannut virheen ja viimeisenä koko testijoukon tulokset. Tuloksen teksti muodostuu RSpec-testin sisäkkäisen rakenteen otsikoista, joten se ei ole kieliopillisesti oikein. Kuvassa 9 näkyy vastaava tilanne, mutta kaikki testit ovat menneet läpi.



Kuva 8. Slackissa toimiva bot vastaa kyselyyn järjestelmän tilasta. Yksi testi antaa virheen. Käyttäjän nimi peitetty.

Kuva 9. Slackissa toimiva bot vastaa kyselyyn järjestelmän tilasta. Kaikki testit menevät läpi. Käyttäjätunnus on kirjoittajan.

Teknisesti chatbot toteutettiin Errbot-alustalla (http://errbot.io). Errbot on valmis chatbot, joka pystyy käsittelemään useita eri taustajärjestelmiä, kuten Slack, IRC ja Telegram ja

jonka keskeisenä ominaisuutena on mahdollisuus kehittää omia liitännäisiä (*plugin*) suorittamaan toimintoja kohdepalvelussa. Python-pohjaisena Errbotin kautta pystyy käyttämään suoraan projektia varten tehtyä *parseria*, joka palauttaa testien tulokset esitettäväksi.

Slack tarjoaa käytettäväksi tavallisten viestien lisäksi "kortteja", jotka mahdollistavat jonkin verran enemmän muotoilua. Tätä ominaisuutta hyväksikäyttäen botin vastaukseen saatiin otsikko "*Test results"* ja testien statuksesta kertova pystypalkki. Tämä auttaa erottamaan chatbotin antamat vastaukset muusta viestiliikenteestä kanavalla. Lisäksi värikoodaus helpottaa yhdellä vilkaisulla päättelemään, onko testituloksissa virheitä vai ei.

Chatbot pystyi vastaamaan kysymykseen testituloksista melko nopeasti: testijoukon ajaminen kesti noin minuutin, jonka jälkeen testejä suorittava skripti odotti toisen minuutin. Koska botille käytettäväksi kopioitiin vain valmis testituloksia sisältävä tiedosto, oli informaatio pahimmillaan noin kaksi tai kolme minuuttia vanhaa.

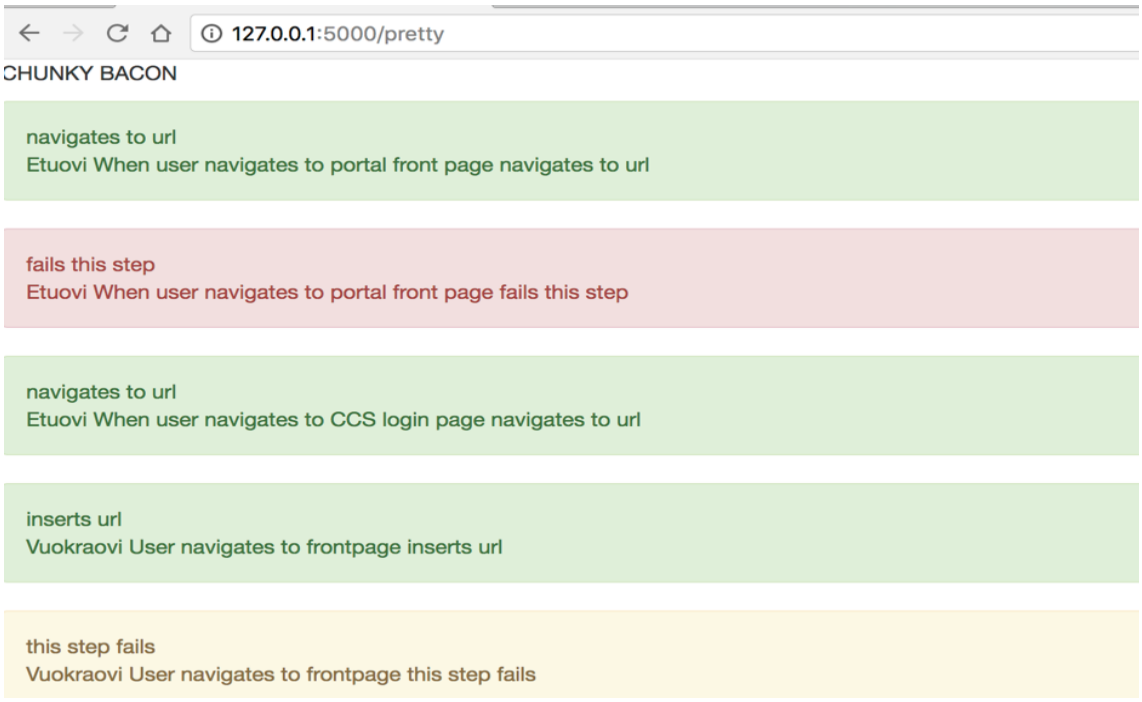
3.3.4 Infonäyttö

Infonäytön suhteen korostui chatbottia enemmän informaation visuaalinen esittäminen, mutta myös tiedon oikea-aikaisuus. Osittain nimenomaan infonäytön asettamien rajoitusten vuoksi testijoukon laajuus asettui lopulliseen muotoonsa.

Keskeisenä elementtinä näytöllä on testit yksinkertaisina vaakasuuntaisina palkkeina. Palkeissa vihreä merkitsee läpi mennyttä testiä ja punainen virheeseen törmännyttä testiä. Kuvan 10 työvaiheessa olevan prototyypin keltainen on asetettu merkitsemään odottavassa tilassa olevaa testiä, jonka törmääminen virheeseen tiedetään ennalta. Tätä ei kuitenkaan lopullisessa prototyypissä käytetty, sillä saatu hyöty on kyseenalainen: mikäli testi osoittaa virhettä pitäisi ongelma tai testi korjata, ei vain osoittaa, että virhe on tiedossa. Tätä käytetäänkin yleensä tilanteessa, että testi tai ominaisuus on työn alla. Kun testi on saatettu valmiiksi, muuttuu keltainen punaiseksi, sillä testi ei enää odotetusti törmää virheeseen.

Kuva 11 esittää lopullisen prototyypin ulkoasun. Näyttö asetettiin asiakaspalvelun työpisteiden läheisyyteen siten, että suurimmalla osalla oli vaivaton tai melko vaivaton näköyhteys. Asiakaspalvelun istumajärjestyksen vuoksi oli kuitenkin mahdotonta sijoittaa näyttöä paikkaan, johon olisi ollut täysin keskeytyksetön näköyhteys. Näyttö asetettiin

noin seisovan ihmisen silmien tasolle, jotta näyttöön olisi seisovalla ja istuvalla käyttäjällä paras näköyhteys. Jotta samalle sivulle mahtuisi kaikki relevantti informaatio, käytännössä siis testien tulosrivit, ei yksittäisen testin otsikkoa voinut asettaa kovin suureksi. Käytännössä tämä tarkoitti sitä, että kauempana näytöstä olevat asiakaspalvelijat eivät kyenneet erottamaan tekstiä. He kuitenkin kertoivat nopeasti oppineensa muistamaan mikä rivi vastasi mitäkin osaa palvelussa.



Kuva 10. Varhainen versio näytön toiminnasta.

Prototyypin toteutus tehtiin Python-pohjaisella Flask web-kehyksellä. Koska prototyypin testaus oli ajallisesti kohtalaisen lyhyt jakso ja koska näyttö tarvitsi joka tapauksessa tietokoneen, ajettiin palvelua paikallisesti näyttöön liitetyltä koneelta.

Kuva 11. Inforuudun lopullinen ulkoasu. Kaikki testit menevät läpi.

Sivu ajastettiin päivittämään itsensä 60 sekunnin välein. Kun keskimäärin testijoukon ajaminen kesti noin minuutin, oli viive pahimmassa tapauksessa noin kaksi tai kolme minuuttia. Tämä ratkaisu näytti prototyypin rakennusvaiheessa toimivan hyvin.

Ulkoasun suhteen pitäydyttiin mahdollisimman yksinkertaisissa ratkaisuissa. Sivu esittää riveittäin testitulosten JSON-tiedostosta kuvausrivin ja laajemman kuvauksen. Kuvausrivi varustettiin *strong* HTML-tunnisteella tehden siitä otsikon.

Kuten chatbotin kohdalla, Python-pohjaisena Flask pystyy käyttämään samaa taustajärjestelmän parseria suoraan. Parseri lukee testien tuloksen ja muokkaa sitä hieman esittämistä varten. Sivupohjassa asetetaan class-attribuutti riville tuloksista löytyvän statuksen mukaan: mikäli tarkasteltavan testin status on “passed”, tulkitaan se arvoksi “success” ja esitetään rivi vihreänä. Statuksella “failed” tulkitaan se arvoksi “danger” ja esitetään rivi punaisena.

3.4 Prototyyppien testaus

Käyttöliittymien testaus toteutettiin kolmessa eri vaiheessa. Ensimmäisessä pyysin asiakaspalvelijoita tarkkailemaan kontaktejaan ja Slack-keskustelujaan ja merkitsemään ylös testijakson aikana tapahtumia, milloin he joutuivat konsultoimaan ylläpitotiimiä saadakseen kuvan järjestelmän tilasta.

Toisessa vaiheessa käynnistin ja pyysin käyttämään vastaavissa tilanteissa Slack-bottia, joka pyrkii vastaamaan samantapaiseen kysymykseen, kuin mitä he esittäisivät ylläpitotiimille järjestelmän tilasta. Tämän jälkeen pyysin heitä arvioimaan 1) järjestelmän hyödyllisyyttä yleisellä tasolla ja 2) järjestelmän kuormittavuutta työn ohessa.

Kolmannessa vaiheessa toimitin paikalle tietokoneen ja näytön, jonka pyrin asettamaan sellaiseen paikkaan, että testiin osallistuvat asiakaspalvelijat näkivät sen helposti kontaktin aikana. Myös tämän jakson jälkeen pyysin heitä arvioimaan järjestelmän hyödyllisyyttä ja kuormittavuutta kuten chatbotinkin kohdalla.

Seuraavassa käyn läpi testausten jälkeisten keskustelujen tulokset ja huomiot.

3.4.1 Chatbot

Asiakaspalvelun kanssa suoritettiin kahden päivän mittainen testaus. Kahden päivän jälkeen pidettiin lyhyt vapaamuotoinen purkutilaisuus, jossa keskusteltiin laajasti kokeilun herättämistä ajatuksista.

Kukaan asiakaspalvelussa ei nähnyt suoraan hyötyä järjestelmälle sellaisenaan. Vaikkakin chatbot itsessään koettiin hyvänä työkaluna, he totesivat sen toimivan paremmin spesifimpiä työkaluja tarjoavana palveluna. Yleistä järjestelmän tilan tietoisuutta ei sen koettu juurikaan lisäävän.

Osa koki testin tulosten kyselyn ruuhkauttavan turhan paljon tärkeää viestintäkanavaa ja ehdottikin, että botin toiminnalle olisi oma dedikoitu kanava. Botin toteutus nykyisellääkin pystyisi antamaan tulokset yksityisviestinä, mutta tällöin jäisi saamatta se hyöty, että muutkin asiakaspalvelussa saavat samat tiedot. Tämä myös on ChatOps-ajatuksen vastaista, sillä tällöin katoaa saavutettu läpinäkyvyys. Huoli on kuitenkin aiheellinen ja tulee ottaa suunnittelussa ja sovituissa prosesseissa huomioon vastaavaa järjestelmää käytettäessä.

Paljon keskustelua käytiin muista tavoista hyödyntää botteja ja sivujuonteena toteutettu ominaisuus näyttää kohteen lisätietoja otettiin ilolla vastaan (kuva 7). Testaamisen aikana asiakaspalvelulle avattiin pääsy työkaluun, joka mahdollistaa erään normaalisti ylläpidon suorittaman toimenpiteen. Tämän yhdistäminen tulevaisuudessa chatbottiin nähtiin hyvänä ajatuksena, sillä nykyisellään työkalua käyttävä asiakaspalvelija joutuu erikseen raportoimaan Slack-kanavalle työkalun käytöstä itse. Tämä saattaa pahimmillaan aiheuttaa saman toimenpiteen suorittamisen moneen kertaan, kun eri käyttäjät suorittavat saman toimenpiteen, mutta eivät ole vielä ehtineet raportoimaan sen suorittamisesta. Chatbottia hyödyntämällä saisi yhdistetyksi suorituksen ja siitä tiedon välittämisen relevantille kanavalle, luvussa 2.4 esitettyjen periaatteiden mukaisesti.

Positiivisena puolena nähtiin chatbotin antaman tiedon mahdollistama oman työn priorisointi. Mikäli saadussa vastauksessa näkyy virheitä, voi asiakaspalvelija eskaloida kontaktia nopeammin. Tämän lisäksi ylläpidolla on suoraan näkyvyys kanavalle ja siten samoihin testituloksiin.

Kontaktin aikana asiakaspalvelijalla on kuitenkin monta järjestelmää auki: jonopalvelu, jonka kautta kontakti tulee, asiakastiedot, palvelu johon kontakti liittyy, tikettijärjestelmä ja niin edelleen. Tässä tilanteessa Slackin tuominen etualalle ei ole triviaalia. Prototyypin chatbotin toteutuksen keskeinen päämäärä oli nopea vastauksen saaminen. Kontaktin jälkeen asiakaspalvelijalla on aikaa tehdä syvempää tarkastelua järjestelmän tilasta. Tähän tarkoitukseen chatbotin tuottama vastaus taas on liian suppea. Tässä palvelisi paremmin esimerkiksi jokin järjestelmää monitoroiva verkkopalvelu. Näin työkalu jää herkästi käyttämättä.

Osa käyttäjistä ilmaisi myös kokevansa testien selitetekstit hankaliksi ymmärtää. Jälkikäteen ajatellen tämä onkin ymmärrettävä ongelma, joka olisi pitänyt huomioida suunnittelussa. Vakiintunut käytäntö on kirjoittaa testit siten, että koodiin muodostuu loogisesti ehjiä lauseita ja kulkuja, mutta tämä käytäntö ei suoraan siirry tuloksiin. Tällä tavalla kirjoitettuna saadaan hyötyä virheitä testin lähdekoodia apuna käyttäen etsiessä, mutta ei silloin kun tuloksia esitetään ilman pääsyä lähdekoodiin. Tämän lisäksi toinen vakiintunut käytäntö on kirjoittaa testit englanniksi, mutta jälleen tämä ei suoraan siirtynyt helppotulkintaiseksi testien raportoinniksi. Tämä on selkeä epäkohta, johon tulisi tulevaisuudessa kiinnittää huomiota.

Mikäli järjestelmää lähdettäisiin kehittämään tässä muodossa eteenpäin, käyttäjät toivoivat laajempaa, palvelukohtaista testijoukkoa, jolla saa mahdollisimman tarkan kuvan

järjestelmän avainkohtien toimivuudesta, kuten kuvakäsittelijöiden työjonojen pituuksista tai aineistosiirtojärjestelmien aineistoajoista. Tämä vaatii tiedon yhdistelyä eri paikoista ja siihen liittyvien työkalujen kehittämistä.

Ylläpito ja asiakaspalvelu eivät kokeneet keskinäisen kommunikaationsa vähentyneen testin aikana.

3.4.2 Infonäyttö

Samaan tapaan kuin chatbotin kanssa, järjestettiin asiakaspalvelun kanssa kaksi päivää kestänyt testi, jossa sama informaatio esitetään jaetulla näytöllä.

Keskeiseksi keskustelunaiheeksi nousi tiedon ajantasaisuuden, tarkkuuden ja luotettavuuden suhde. Lisäksi keskusteltiin laajasti siitä, minkälaisia tietoja näytöllä pitäisi näyttää. Asiakaspalvelussa oikea-aikaisuus on merkittävässä roolissa, jotta he pysyvät järjestelmän tilasta tietoisina. Keskusteluissa tuli esimerkkinä mittasuhteet, jossa noin 5 minuuttia on ehdoton maksimi yksittäisen testijoukon ajoajaksi. Toinen merkittävä, ja myös oikea-aikaisuuteen liittyvä huomio, oli tiedon luotettavuuden merkitys.

Asiakkaan ollessa yhteydessä ei asiakaspalvelulle ole suurtakaan hyötyä siitä, minkälainen järjestelmän tila on ollut aiemmin. Vaikka reaaliaikaisuudelle ei suoraan nähty tarvetta, asiakaspalvelijat painottivat etenkin, että mikäli testijoukon ajo kestää kymmeniä minuutteja, ei se vastaa kysymykseen järjestelmän tilasta sillä hetkellä.

Jaettu näyttö koettiin huomattavasti paremmaksi ratkaisuksi kuin chatbot järjestelmän tilan tuntemuksen lisäämiseen. Testin aikana ylläpitoportaali oli ongelmissa, joten sen tilaa seurattiin tarkasti. Koska ajo kesti noin minuutin, oli portaalin tila hyvin pienellä vaivalla luettavissa näytöltä. Asiakaspalvelijat, joiden työpiste oli kauempana näytöstä, kertoivat nopeasti oppineensa alitajuisesti katsomaan oikeaa riviä arvioidakseen, oliko portaali toiminnassa vai ei. Tämä koettiin varsin hyväksi. Normaalissa tilanteessa jokainen yritti vuorollaan avata portaalia. Vasta tämän jälkeen he yhdessä pyrkivät varmistumaan, että se todella on saavuttamattomissa, eikä kyse ole yksittäisen koneen tai yhteyden ongelmasta. Nyt tämä sama informaatio saatiin näytöltä.

Toinen virhettä näyttänyt testi oli listasivulla oleva uusien kohteiden testaus. Oletus oli, että testattavalla listasivulla olisi aina kohteita, jotka on merkitty 24 tunnin sisällä palveluun tulleiksi. Tämä on oikea testi, jota asiakaspalvelu satunnaisesti itsenäisesti suorittaa nähdäkseen, että uusia kohteita tulee palveluun. Tämä kuitenkin törmäsi tilanteeseen,

jossa kohde, jolle on ostettu lisänäkyvyyttä, ei enää saanut tuota uuden kohteen merkintää ja siten testi törmäsi virheeseen. Tämä aiheutti nopeasti vääränlaisen tottumisen siihen, että kyseinen rivi on punaisella, ennen kuin virhe korjattiin. Käyttäjät kokivat tämän tärkeäksi huomioksi, sillä mikäli kyseessä olisikin ollut oikea vikatilanne, se olisi tällä tavoin jäänyt huomaamatta, kaikkien olettaessa, että sama rivi on usein punaisella.

Kuten chatbotin kanssa, ylläpito ja asiakaspalvelu eivät kokeneet keskinäisen kommunikaationsa vähentyneen testin aikana. Testi kuitenkin antoi suurempaa varmuutta reagoida nopeasti ylläpitoportaalin ongelmiin.

4 KESKUSTELU

Jatkokehitystä järjestelmään tarvitaan. Vaikkakin testit paljastivat järjestelmien hyötyjä, ne myös osoittivat puutteita, jotka täytyy huomioida, jotta testatut järjestelmät olisivat käyttökelpoisia. Suppean testijoukon myötä ei testien aikaan sattunut juurikaan todellisia ongelmatilanteita, jotka olisivat valvonnassa näkyneet. Täten asiakaspalvelun tekijöiden ei ollut aivan vaivatonta päätellä, olisiko järjestelmästä todellisuudessa ollut hyötyä. Tämä korostui etenkin chatbottia testatessa.

Infonäyttö koettiin hyödylliseksi ja sen paikalleen jättämistä jopa nykyisellään pyydettiin testauksen päätyttyä. Tämä kertoo siitä, että jonkinlainen järjestelmän tilaa kuvastava jaettu näyttö voisi asiakasrajapinnassa toimimista hyödyttää, mikäli esitetyn informaation sisältöön panostetaan. Tämän testauksen perusteella näyttäisi, että parhaan tuloksen saisi yhdistelemällä eri lähteistä saatavaa informaatiota yhdelle näytölle sekä sivusilmällä seurattavaksi että lähemmin tarkasteltavaksi. Lisäksi näytön koko tulisi olla riittävän suuri.

Chatbotin koettu hyödyllisyys oli heikompi, mutta senkin toiminnan jatkamista toivottiin. Tämä kuitenkin johtui työkalun muista lupauksista, kuten esimerkinomaisesti rakennetusta kohteen tietojen näyttämisestä ja muista ChatOps-työkaluista, joita työ tueksi kehitetään.

Voidaan siis kysyä, onko chatbotilla mahdollista saavuttaa ChatOps:n ja siten DevOps:n lupaamia hyötyjä testausautomaation hyödyntämisen osalta myös organisaation muissa osissa kuin vain kehittäjien parissa? Vaikkakin tämän tutkimuksen tulokset eivät täysin kaikilta osin vakuuttaneet, en sulje mahdollisuutta pois. Työkalu selvästi kiinnosti ja sen tuottama informaatio oli laajemmin hyödynnettävissä lisääntyneen näkyvyyden vuoksi. Toimenpiteen suorittaminen myös samalla dokumentoi suoritetun toimenpiteen ja sen tulokset myös muiden hyödynnettäviksi. Tässä tulee kuitenkin kiinnittää huomiota siihen, että informaatio on sekä oikeassa paikassa, ja että sitä on oikea määrä. Mitkä nämä oikeat paikat ja määrät ovat, riippuu luonnollisesti organisaatiosta. Mutta prototyyppejä testatessa esiin tullut huoli tärkeän viestikanavan tukkeutumisesta ja toisaalta tarve dokumentoida sovelluksen uudelleenkäynnistyksiä, jotta samaa toimenpidettä ei tehdä usealta taholta yhtä aikaa osoittavat, että balanssia tulee etsiä.

Alkuperäinen ajatus olemassa olevien testien hyödyntämisestä törmäsi prototyyppejä hahmotellessa ongelmiin, kuten aiemmin kuvasin. Jatkokehitys kuitenkin saattaisi löytää tasapainon testien laajuuden, herkkyyden ja nopeuden välillä olemassa olevastakin testijoukosta pilkkoen sopivia kokonaisuuksia. Tämä vaatii kuitenkin läheistä yhteistyötä kehittäjien, ylläpidon ja asiakaspalvelun kesken, jotta relevantti informaatio päätyy käyttöön.

Työn mittaavan osuuden suorittaminen venyi monestakin syystä, joten ensimmäisten alustavien keskustelujen ja varsinaisten testausten välillä ehti kulua aikaa useita kuukausia. Tämä saattoi olla sekä hyvä että huono asia: toisaalta asiakaspalvelun tekijöillä oli kohtalaisen runsaasti aikaa tottua ajatukseen, että moinen on tulossa, mutta toisaalta taas tämä saattoi aiheuttaa tuntemuksia, että heidän työtään edistävää toimintoa ei saatu edistetyksi. Tämä aiheutti myös ennakoimattoman tarpeen käydä samankaltaisia keskusteluja useamman kerran.

Palvelun eläessä ja päivittyessä tämän projektin aikana, osa testauksesta on siirtynyt pois yllä kuvatuista työkaluista käyttämään Javascript-pohjaisia, suoraan käytettyyn web-kehykseen liittyviä työkaluja. Näiden tuoma etu on muun muassa kehittäjien pysyminen samassa kielessä kehityksen ja testaamisen osalta, mutta myös kyseisen kirjaston avulla tuotettujen komponenttien testaaminen helpommin tätä varten kehitetyllä testauskirjastolla. Tämä tarkoittaa jatkossa testaamisen erikoistumista, mutta myös tulosten keskittämisen tarvetta. Jatkokehitystä tarvitaan siis etenkin tämän osalta.

Suunnittelun alkuvaiheessa kävin keskusteluja, joiden perusteella muodostin suunnitteluun kolme teesiä:

- Näytettävän informaation määrä on rajallinen
- Näytettävän informaation on oltava ajantasaista: ajon on mentävä läpi maksimissaan viidessä minuutissa
- Näytettävän informaation on oltava luotettavaa

Näiden teesien valossa voi tarkastella prototyyppien ja testauksen onnistumista.

Informaation rajallisuus nousi esille testien nopeuden, vakauden ja ylläpidettävyyden suhteen, mutta myös siinä minkä verran näytölle kannatti informaatiota laittaa kerralla näkyville. Mikäli informaation määrää olisi tästä merkittävästi nostettu, yleinen arvio

käyttäjien parissa oli, että näyttö olisi vaatinut huomattavasti enemmän huomiota. Vaikkakin monella oli toiveita ja ehdotuksia siitä mitä näytöllä voisi olla, kaikki myönsivät näytön käyttötarkoituksen vähintään muuttuvan informaation lisääntyessä. Testien määrässä lisäsuunnittelulla kyetään näiden kokemusten perusteella lähenemään optimaalista tilannetta, jossa testien määrä ja niiden testaamien toiminnallisuuksien laajuudet ovat tasapainossa. Tämän työn prototyyppien testeissä oli tässä suhteessa turhaan toistoa, jota eliminoimalla saadaan samaan määrään laajempi kattavuus.

Informaation ajantasaisuudessa prototyypeissä onnistuttiin hyvin. Testien laajuus mahdollisti nopean toiston, jolloin harvemmin tuli tilannetta, milloin testitulokset olisivat olleet yli 2-3 minuuttia vanhoja. Tässä tasapainon etsiminen on myös merkittävää, mutta näin nopea perustoteutus antaa tilaa kasvattaa testien kompleksisuutta ja siten suoritusaikaa tarpeen mukaan.

Informaation luettavuus on automaatiossa haastavaa, etenkin tämänkaltaisessa järjestelmässä, jossa tulokset ovat riippuvaisia paitsi testattavan järjestelmän toiminnasta, mutta huomattavasta määrästä ulkoisia tekijöitä, niin testijärjestelmän kuin yleisen infrastruktuurin osalta. Testien ajossa syntyikin muutamaan otteeseen tilanne, jossa testi osoitti virhettä, vaikka kyseessä ei palvelun kannalta ollutkaan virhetilanne. Tämä paljasti kaksijakoisen dilemman: joko testien tuloksiin luottamus laskee tai käyttäjät ryhtyvät toimenpiteisiin virheellisen informaation varassa. Tämä on seikka, johon tulee kiinnittää huomiota paitsi testejä suunniteltaessa, mutta myös testien tulosten seuraamiseen liittyvien prosessien suunnittelussa ja sopimisessa. Ensimmäinen reaktio on vastata asiakkaalle, että palvelussa on virhe, mutta mikäli tuota virheilyä ei ole varmistettu manuaalisesti, voi kyseessä hyvinkin olla tilapäinen häiriö muualla kuin itse testattavassa palvelussa.

Nämä seikat yhdessä osoittavat yhden tämänkaltaisten järjestelmien keskeisen haasteen: on tärkeää arvioida tarkkaan mitä valvotaan, millä tarkkuudella ja mitä tietoja valvonnasta näytetään. Mikäli valvonta on tarkkaa, tulee testien suoritusajasta pitkä ja informaatiota tulee paljon, jolloin sen näyttäminen tulee suunnitella tarkoin. Mikäli taas valvonta on suurpiirteisempää, ei järjestelmä välttämättä havaitse potentiaalisesti isojakaan ongelmia.

Tätä haastetta käsitellessä nousee keskeiseen osaan testauksen suunnittelu. Tämä on yleensä testaajien ja kehittäjien vastuulla, mutta tämän työn kokemusten perusteella on tärkeää ottaa suunnitteluun mukaan informaatiota hyödyntävä taho, eli tässä tapauksessa asiakaspalvelijat. Keskusteluissa heillä olikin varsin paljon ajatuksia siitä, mikä

informaatio heitä työssään hyödyttäisi. Informaation valintaan ja muodostukseen tarvitaan kuitenkin apua ja ohjausta kokeneemmilta testaajilta.

5 JOHTOPÄÄTÖKSET

Tässä työssä pyrittiin suunnittelemaan järjestelmä, jolla yleensä teknisille asiantuntijoille tarkoitettua testausraportointia voitaisiin hyödyntää myös organisaation muissa osissa. Tarve tähän syntyi olemassa olevan toteutuksen hyödyntämisen vaikeudesta ja selkeästä tarpeesta tuoda testituloksia lähemmäs välitöntä asiakasrajapintaa.

Toteutusratkaisut valikoituivat paitsi ympäristön asettamien vaatimusten myötä, mutta myös näytettävän informaation luonteen vuoksi. Keskeisiä suunnittelutavoitteita oli tarjota informaatio muodossa, joka ei kohtuuttomasti lisäisi kognitiivista kuormaa muun työn ohessa, mutta joka kasvattaisi yleistä järjestelmän tilan tuntemusta ja näkyvyyttä.

Chatbot suunniteltiin ChatOps-periaatteiden mukaisesti toimimaan pikemminkin käyttöliittymänä testitulosten hakemiseksi kuin tekoälyä hyödyntäväksi assistentiksi, joiksi chatbotit usein mielletään. Tämä lähestymistapa toimi hyvin, mutta käyttäjät kokivat sen hyötyjen tulevan paremmin esille muussa toiminnassa kuin testitulosten tarkastelussa. Täten chatbot ei suoraan lisännyt luvussa 2.2 esiteltyä järjestelmän tilan tuntemusta.

Infonäyttö asetettiin näyttämään testituloksia ja päivittymään automaattisesti minuutin välein. Informaatio esitettiin riveittäin yksinkertaisella värikoodauksella: vihreä rivi kertoi kyseisen testin suorituksen edenneen ilman virhettä, punaisen värin taas testin suorituksen törmänneen virheeseen. Vaikkakin näytön asemoimisessa paikkaan, jossa näytölle ei ollut etukäteen suunniteltua ja rakennettua paikkaa asetti omat haasteensa, olivat kokemukset pääosin positiivisia. Kognitiivisen kuorman ei koettu kasvavan liiaksi, sillä näytön hyödyntäminen ei vaatinut aktiivista toimintaa omalla työasemalla, vaan nopea vilkaisu näytön suuntaan aika ajoin riitti tilan järjestelmän tilan hahmottamiseen.

Sekä chatbotin että jaetun näytön osalta keskeisiksi opeiksi nousi informaation määrän hallinta: chatbotin koettiin tuovan ylimääräistä melua muutenkin aktiiviseen viestintäkanavaan. Jaetun näytön taas etäisyytensä vuoksi piti välittää informaatio siten, että se on nähtävissä yhdellä vilkaisulla ilman tarvetta joka kerta kävellä lähemmäs katsomaan näyttöä.

Informaation luotettavuus on testausautomaatiossa keskeisessä asemassa. Laajempia ja monimutkaisempia testejä suunnitellessa tulee ottaa huomioon ulkoisista tekijöistä johtuvat väärät positiiviset tulokset. Mikäli automaatiotestaus suoritetaan vain testausta varten, voidaan ongelma ratkaista tarkastelemalla tuloksia ja ajamalla testit tarpeen mukaan

uudelleen. Usein testien suorittaja on henkilö, jolla on kokemusta testauksesta ja siihen liittyvistä haasteista. Kun kyseessä on järjestelmän monitorointi ja testin tulosten raportointi kulkee aggregaation läpi, nousee vaatimus tulosten luotettavuudesta suurempaan rooliin, sillä tulosten käyttäjällä ei ole mahdollisuutta arvioida millä todennäköisyydellä testitulos on väärä positiivinen. Infotaulusta tai chatbotin antamasta tiivistelmästä ei ole mahdollista luotettavasti sanoa, mistä ongelma johtuu. Tämä asettaa rajoituksen testien kompleksisuudelle, sillä kompleksisuuden lisääntyessä lisääntyy myös väärien positiivisten mahdollisuus.

Tulosten ajantasaisuus on monitorointitarkoituksissa tärkeää. Mikäli järjestelmässä on häiriö, tulee aiheeseen liittyviä kontakteja potentiaalisesti varsin pian. Samoin ongelman poistumisesta on tärkeää saada tieto nopeasti. Laajan, järjestelmän tilaa tarkastelevan testin ajoaika voi nousta yli tuntiin. Käydyissä keskusteluissa tuli esille, että suurimmillaan noin viiden minuutin ajoaika on vielä käyttökelpoinen, sillä tällöin informaatio on hyödynnettävissä silloin kun asiaan liittyviä kontakteja joutuu hoitamaan.

Koska testausautomaation ylläpito ja kehittäminen on kallista, vaatien kehittäjien ja testaajien resursseja, on suositeltavaa hyödyntää testejä ja niistä saatavia tuloksia mahdollisimman tehokkaasti. Onkin suositeltavaa ottaa testejä suunnitellessa huomioon eri käyttötapoja ja pyrkiä siten varmistamaan, että testejä pystytään erottamaan eri kokoisiin setteihin ilman merkittävää refaktorointityötä. Tämä auttaa muodostamaan tarkoitukseen sopivia testijoukkoja ja etsimään tasapainoa testien tarkkuuden, suoritusaikojen ja luotettavuuden suhteen eri tilanteissa.

Tämän työn perusteella uskon, että testausautomaation hyödyntäminen on kannattavaa yleisemmässäkin kontekstissa. Infonäytön toteuttaminen olemassa olevista testeistä on toteutettavissa kohtalaisella vaivalla ja mikäli testituloksia ohjataan joka tapauksessa käytössä olevaan viestikanavaan, ei ChatOps-henkisen työkalun rakentaminen tulosten näyttämiselle yleensä ole mahdotonta. Tärkeää on kuitenkin kuunnella sitä tahoa, jonka tulisi informaatiota hyödyntää.

LÄHTEET

Application Monitor Tools | Application Performance Monitoring (APM). (n.d.). Noudettu 10.8.2018, osoitteesta https://www.manageengine.com/products/applications_manager/

Benet, A. F., Lujua, C. E., Grau, H. S., Jáimez, M. M., Pérez, F. M., & Bianco, C. (2012). Software for Medical Devices and Our Need for Good Software Test Automation. *Experiences of Test Automation: Case Studies of Software Test Automation,* s. 375–400.

Beserra, V., Nussbaum, M., Oteo, M., & Martin, R. (2014). Measuring cognitive load in practicing arithmetic using educational video games on a shared display. *Computers in Human Behavior*, *41* (s. 351–356). https://doi.org/10.1016/j.chb.2014.10.016

Browserstack. (n.d.). Noudettu 10.8.2018, osoitteesta https://www.browserstack.com/

Burns, C. M. (2000). Putting It All Together: Improving Display Integration in Ecological Displays. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, *42*(2), s. 226–241. https://doi.org/10.1518/001872000779656471

Dalton, N. S., Collins, E., & Marshall, P. (2009). Display blindness? Looking again at the visibility of situated displays using eye tracking. *SIGCHI Conference on Human Factors in Computing Systems*, s. 3889–3898. https://doi.org/10.1145/2702123.2702150

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. *IEEE Software*, *33*(3) (s. 94–100). https://doi.org/10.1109/MS.2016.68

Errbot — Err 5.1.3 documentation. (n.d.). Noudettu 11.3.2018, osoitteesta http://errbot.io/en/latest/

Hand, J. (2016). *ChatOps*. (B. Anderson & V. Wilson, Eds.) (First Edit). O'Reilly Media, Inc.

Johnston, K., & Deschamps, F. (2012). Moving to the Cloud: The Evolution of TiP, Continuous Regression Testing in Production. *Experiences of Test Automation: Case Studies of Software Test Automation,* s. 49–67.

Kasdaglis, N., Newton, O., & Lakhmani, S. (2014). System State Awareness: A human centered design approach to Awareness in a complex world. *Proceedings of the Human Factors and Ergonomics Society*. https://doi.org/10.1177/1541931214581063

Kohl, J. (2012). There's More To Automation Than Regression Testing: Thinking Outside the Box. *Experiences of Test Automation,* s. 355–373.

Mazaeva, N., & Bisantz, A. M. (2014). Ecological displays, information integration, and display format: An empirical evaluation across multiple small displays. *Journal of Cognitive Engineering and Decision Making*, *8*(2), s. 137–161. https://doi.org/10.1177/1555343414521424

Nielsen, J. (1994). Enhancing the explanatory power of usability heuristics. *Conference companion on Human factors in computing systems - CHI '94*. https://doi.org/10.1145/259963.260333

Pettichord, B. (2001). Success with Test Automation. Noudettu 1.10.2018, osoitteesta https://www.prismnet.com/~wazmo/succpap.htm

RSpec: Behaviour Driven Development for Ruby. (n.d.). Noudettu 10.8.2018, osoitteesta http://rspec.info/

Schwarzer, J., Draheim, S., von Luck, K., Wang, Q., Casaseca, P., & Grecos, C. (2016). Ambient Surfaces: Interactive Displays in the Informative Workspace of Co-located Scrum Teams. *Proceedings of the 9th Nordic Conference on Human-Computer Interaction - NordiCHI '16*, 1–4. https://doi.org/10.1145/2971485.2971493

Slack Platform: Community | Slack. (n.d.). Noudettu 11.3.2018, osoitteesta https://api.slack.com/community

Toxtli, C., Monroy-Hernández, A., & Cranshaw, J. (2018). Understanding Chatbot-mediated Task Management. https://doi.org/10.1145/3173574.3173632

Vicente, K. J. (2002). Ecological Interface Design: Progress and Challenges. *Human Factors: The Journal of the Human Factors and Ergonomics Society*, *44*(1), s. 62–78. https://doi.org/10.1518/0018720024494829

Watir Project. (n.d.). Noudettu 10.3.2018, osoitteesta http://watir.com/

Where work happens | Slack. (n.d.). Noudettu 11.3.2018, osoitteesta https://slack.com/

Zamora, J. (2017). I ’ m Sorry , Dave , I ’ m Afraid I Can ’ t Do That : Chatbot Perception and Expectations, s. 253–260. https://doi.org/10.1145/3125739.3125766