Onni Aaltonen

# Mukautuvien käyttöliittymien testauskäytäntöjen suunnittelu

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Mediatekniikan koulutusohjelma

Insinöörityö

27.4.2016

Metropolia

| | |
|---|---|
| Tekijä<br>Otsikko<br><br>Sivumäärä<br>Aika | Onni Aaltonen<br>Mukautuvien käyttöliittymien testauskäytäntöjen suunnittelu<br><br>67 sivua + 5 liitettä<br>27.4.2016 |
| Tutkinto | Insinööri (AMK) |
| Koulutusohjelma | Mediatekniikka |
| Suuntautumisvaihtoehto | Digitaalinen media |
| Ohjaajat | Yliopettaja Kari Aaltonen<br>Teknologiavastaava Juha Karonen |

Insinöörityön tarkoituksena oli löytää tehokas tapa testata responsiivisia verkkosivuja mainostoimistossa. Mainostoimistolla oli olemassa tapoja, joilla testataan verkkosivujen yhteensopivuus eri selaimien kanssa, mutta yhtenäinen testaustapa verkkokehittäjien keskuudessa puuttui. Kaikki verkkokehittäjät suorittivat testauksen omalla tavallaan, jolloin testausvaihe jäi helposti pintapuoliseksi.

Tavoitteena oli parantaa verkkokehittäjille tarjottuja testitaustyökaluja, jotta testaaminen tehtäisiin huolellisemmin ja yhtenäisesti projektityypin mukaan. Osana insinöörityötä verkkokehittäjille tehtiin kysely, jonka tarkoituksena oli saada selville vanhat testauskäytännöt, virhealteimmat laitteet, epäkohdat projektin hallinnassa sekä suunta, johon verkkokehittäjät itse tahtoisivat viedä testausvaihetta projekteissa.

Kerättyjä tuloksia analysoitiin ja niiden perusteella verkkokehittäjille ehdotettiin ratkaisua ilmenneisiin ongelmiin. Insinöörityön aikana tehtiin suunnitelma mainostoimiston testaustapojen syventämiseksi liittämällä verkkokehitysprojekteihin testipalvelin. Suunnitelman mukaan verkkokehitysprojekteille toteutettaisiin yhteensopivuus-, regressio- ja toiminnallisuustestit, joiden avulla verkkosivujen moitteeton toiminta varmistettaisiin säännöllisesti.

Insinöörityön tuloksena yrityksellä on yksityiskohtainen suunnitelma testaamisen automatisointiin. Tuloksena syntyi myös paljon testitapauksia, joita hyödynnetään testipalvelimen prototyypin rakentamisessa. Insinöörityön tekeminen valisti yrityksen henkilökuntaa testaamisen tärkeydestä ja auttoi heitä kiinnittämään huomiota automaattisen testausvaiheen kehittämiseen. Tulevaisuudessa kaikkien verkkokehitysprojektien on läpäistävä minimi vaatimukset, jotka testipalvelin varmistaa.

| | |
|---|---|
| Avainsanat | testaus, verkkokehitystyökalut, laitetestaus, front-end, verkkojulkaiseminen, laadunvalvonta, jatkuva integraatio |

| | |
|---|---|
| Author<br>Title<br><br>Number of Pages<br>Date | Onni Aaltonen<br>Responsive testing practices<br><br>67 pages + 5 appendices<br>27 April 2016 |
| Degree | Bachelor of Engineering |
| Degree Programme | Media Technology |
| Specialisation option | Digital Media |
| Instructors | Kari Aaltonen, Principal Lecturer<br>Juha Karonen, Chief Technology Officer |

The purpose of this bachelor's thesis was to find effective testing methods for test responsive layouts. The study was conducted in cooperation with a medium-size advertisement agency's web developers. The agency utilized some testing tools to confirm browser compatibility during various web development projects. The agency's web developers used their own diversified methods to complete the testing phase and thus there was no consistent testing process in place.

The goal of this work was to establish a reliable testing method and guarantee dependable test results during the company's web development projects. Two queries were conducted as a part of the study among the agency's Helsinki-based web developers in order to gain more insight into testing practices, most incompatible browsers, problems with project management, and new methods that the web developers would be willing to utilize.

The collected material was analysed and conclusions were made based on the findings. Plans were created on how to renew the testing methods by introducing new testing tools to the agency's developers. The first tool that was introduced included use of continuous integration through automatically validating the performance and functionalities of various web development projects. The second testing method used synchronised mobile browsers when theming the responsive layout.

The thesis showed there is a need for more efficient testing processes within the agency. As a result of this work the agency now has a general test plan and test cases that can be implemented to the test server prototype built in the near future. The goal is to set a minimum test standard for every web development project that the agency releases.

| | |
|---|---|
| Keywords | testing, Web development tools, device testing, front-end, Web publishing, quality assurance, continuous integration |

## Sisällys

Liitteet

Liite 1. Front-end testing methods -kyselyn tulokset

Liite 2. Front-end testing methods -kyselylomake

Liite 3. Testing survey results -diaesitys

Liite 4. Presentation debriefing -kyselylomake

Liite 5. Presentation debriefing -kyselyn tulokset

## Lyhenteet ja käsitteet

| | |
|---|---|
| HTML | Lyhenne sanoista Hypertext Markup Language, joka on verkkosivuilla käytetty on merkintäkieli. |
| Front-end | Loppukäyttäjälle näkyvä visuaalinen käyttöliittymä. |
| Backend | Ohjelmistokoodi, jonka avulla hallinnoidaan tietokannasta haetun informaation esittämistä loppukäyttäjälle. |
| JavaScript | Komentosarjakieli, jota käytetään suorittamaan toimintoja verkkosivun käyttäjän selaimessa. |
| Mobiiliselain | Älypuhelimessa tai tabletissa käytetty sovellus verkkosivujen selaamiseen. |
| Git | Versionhallintatyökalu, joka mahdollistaa ohjelmiston versioimisen koodimuutosten mukaan. |
| Avoin lähdekoodi | Yhteisöllinen ohjelmiston kehitysmenetelmä. |
| Mediakysely | Media query. Mahdollistaa sisällön räätälöimisen päätelaitteen ominaisuuksien mukaisesti. Ominaisuus voi olla esimerkiksi selaimenikkunan leveys. |
| NodeJS | Chrome-selaimen Javascript-moottoriin perustuva ohjelmistoympäristö, jonka avulla voidaan rakentaa palvelimen taustatoimintoja. |
| NPM | Node Package Manager. NodeJS-pohjaisen koodin jakelujärjestelmä. |

# 1 Johdanto

Insinöörityön tavoitteena on löytää tehokas tapa verkkosivujen laitetestaamiseen mainostoimisto Mirum Agencyssa. Mirum tuottaa suuryrityksille digitaalisia markkinointipalveluita. Mainostoimiston sisäisesti tuotetut verkkopalvelut pohjautuvat pääasiallisesti WordPress- ja Drupal-sisällönhallintajärjestelmiin. Mirumin Helsingin toimipisteessä on noin seitsemäntoista verkkokehittäjää. Mainostoimistolla on jo olemassa tapoja, joilla verkkosivuista testataan selaimien yhteensopivuus, mutta yhtenäinen testaustapa verkkokehittäjien keskuudessa puuttuu. Kaikki verkkokehittäjät tekevät testauksen omalla tavallaan, jolloin testausvaihe jää helposti pintapuoliseksi eikä tieto parhaista testaustavoista aina välity.

Insinöörityössä analysoidaan, kuinka verkkokehittäjien kehitysympäristöön saataisiin sovellettua kaupallisia ja/tai avoimeen lähdekoodiin perustuvia testausratkaisuja. Tavoitteena on kehittää työtapa, joka tehostaa älypuhelimien testausta, sekä tuoda esille eri työkaluja ja tapoja mobiiliselaimien testaamiseen. Testaamisen lisäksi keskitytään myös kehitystyyleihin, jotka painottavat enemmän älypuhelimella selaavan käyttäjän kokemusta.

Insinöörityössä keskitytään testaustyökaluihin, jotka perustuvat kaupallisiin ohjelmiin ja avoimen lähdekoodin ratkaisuihin. Läpikäytyjä työkaluja vertaillaan keskenään ja pohditaan, miten ne soveltuvat mainostoimisto Mirumin verkkokehitystapoihin ja tekniikoihin. Työn aikana tehdään palautekysely Mirumin verkkokehittäjien keskuudessa, ja sen odotetaan tuovan esille tärkeimmät epäkohdat nyt käytössä olevista testaustavoista.

Vertailuun valitut työkalut toimivat erilaisilla pohjilla. Tämän vuoksi tarkoituksena on tehdä vertailuja siitä, minkälaiset testauskäytännöt toimisivat parhaiten Mirumin tapauksessa. Vertailukohtina käytetään työkalujen helppokäyttöisyyttä, tuettuja laitteita, tuettuja kehitysympäristöjä, vian etsinnän mahdollisuuksia, ylläpidettävyyttä ja testausympäristön kuluja. Laitetestaaminen keskitetään lähinnä älypuhelimien ja tablettien selaimiin.

## 2 Verkkosivujen testaaminen

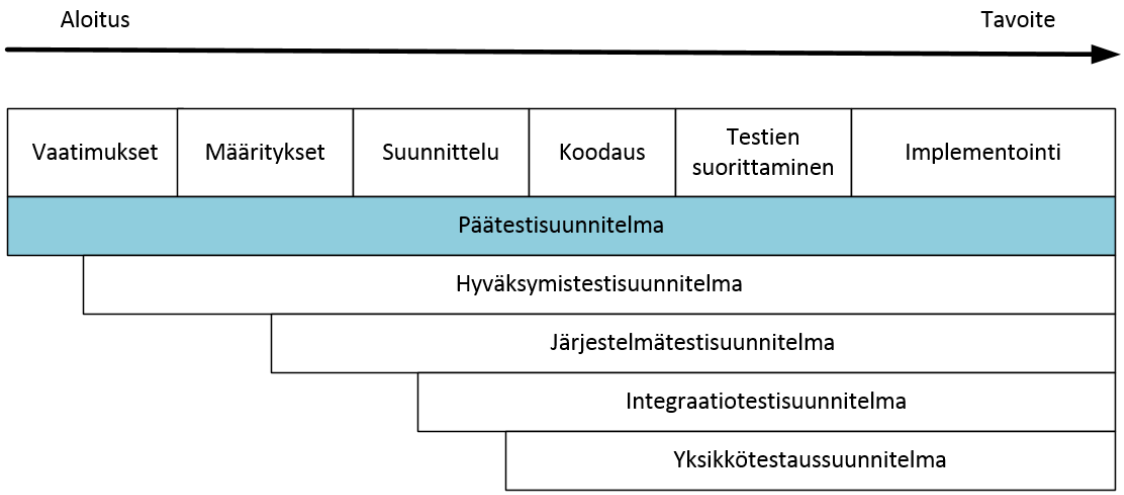
Verkkosivujen tarkoitus on informaation jakaminen graafisen käyttöliittymän avulla. Jos verkkosivu sisältää vikoja, informaatiota ei pystytä lukemaan. Informaation siirtyminen voi häiriintyä hitaan verkkoyhteyden, verkkosivuston sisäisten ongelmien tai selainyhteensopivuusongelmien takia. Ongelmana voi olla myös, ettei verkkosivuston selaaja löydä tarvitsemaansa informaatiota. Tällöin ongelmana on enemmänkin käyttöliittymän suunnittelu kuin ohjelmiston sisäinen ongelma. Tässä työssä keskitytään pääosin teknisten virheiden välttämiseen ja havaitsemiseen sekä yhteensopivuuden varmistamiseen suosituimmilla laitteilla.

Verkkosivustot edustavat aina omistajaansa. Verkkosivun sisältö on sivun omistajan vastuulla, mutta usein verkkosivun tekninen ylläpito on ulkoistettu. Teknisen ylläpidon ulkoistaminen on kustannustehokkaampaa kuin työllistää omia verkkojulkaisuun keskittyneitä työntekijöitä. Verkkosivun omistava yritys voi siten keskittyä omaan ydinosaamiseensa, eikä omia resursseja tarvita tekniseen kehitykseen. Säästöjä syntyy, kun asiantuntijoita ei tarvitse pitää omilla palkkalistoilla, vaan heidän työaikaansa ostetaan tarvittaessa. Verkkosivustojen tekninen vastuu on tällöin ulkoistettu ylläpitosopimuksella, joka sisältää lupauksen kuukausittain käytettävissä olevasta kehitystyöstä ja ylläpidosta tiettyä rahamäärää vastaan. Sopimus velvoittaa palveluntarjoajaa toimittamaan tukea määrätyksi ajaksi kuukaudessa. Käyttämätön tekninen tuki yleensä kerrytetään tuntisaldona vuoden loppuun, jolloin jo maksetuille kehitystunneille luodaan pieniä kehitysprojekteja, joihin tunnit voi sijoittaa. Sopimuksessa on määritelty kaiken kehitystyön minimivaatimukset. Näihin vaatimuksiin sisältyy selaintuki ja takuu. [1.]

Palveluntarjoajalle lankeaa vastuu siitä, että verkkosivusto on käytettävissä vuorokauden ympäri ilman katkoksia. Palveluntarjoaja vastaa myös siitä, että verkkosivu on käyttökelpoinen asiakkaan vaatimilla selaimien ja älypuhelimien yhdistelmillä. Tuettujen selainten lista voi olla yhteistyössä laadittu tai asiakkaan omien vaatimuksien mukainen. Palveluntarjoajalla on oltava olemassa oma sisäinen prosessi vaatimuslistan käyttöä varten. [2, s. 4.]

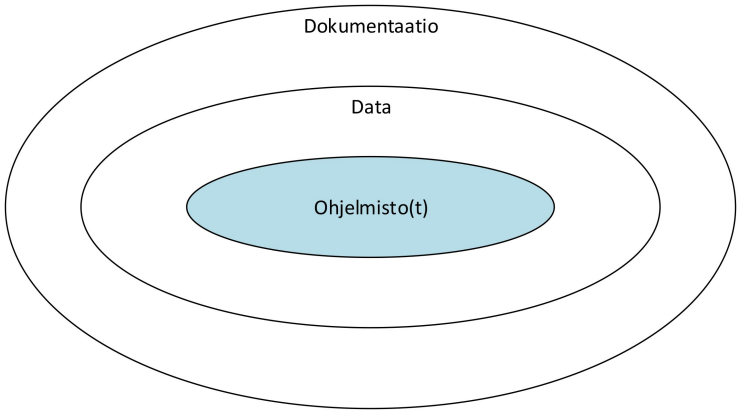
## 2.1 Ohjelmistotestausmallit ja -metodit

Asiakas ostaa palvelua, jonka laadun oletetaan olevan ensiluokkaista. Ensiluokkaista laatua syntyy vain lopputuotoksen huolellisen testaamisen, eli vikojen ja puutteiden etsimisen, tuloksena. Vikoja ei löydy, ellei niitä aktiivisesti etsi. Testaaminen on laadunvalvontaa, joka muodostuu kuvion 1 komponenteista, eli suunnittelusta, valmistelusta, toteuttamisesta ja vaatimusten analysoinnista. Ohjelmistotestaamisen tavoitteena on tuoda ohjelman aiheuttamat virheet esille ja luoda realistinen kuva ohjelmistontoimintavarmuudesta sen todellisessa toimintaympäristössä. [2, s. 2.]



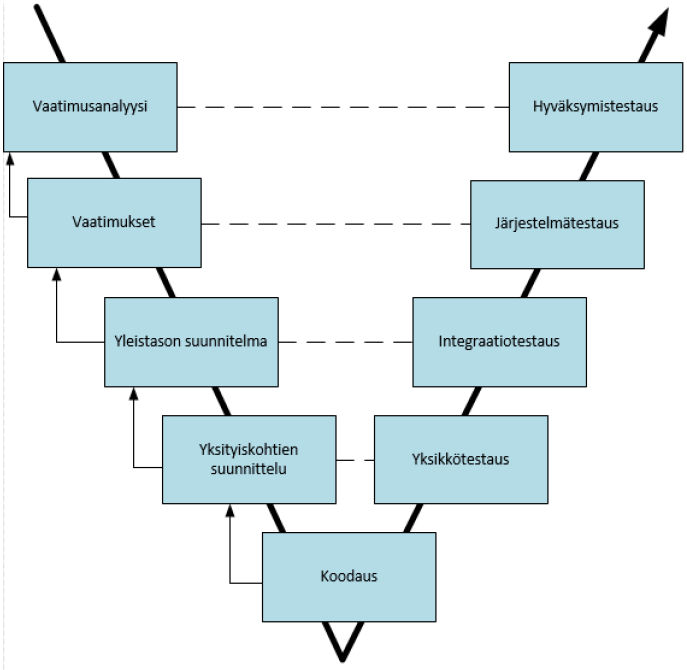

Kuvio 1. Testisuunnitelman aikataulu [2, s. 202].

Verkkosivun vaatimukset syntyvät asiakkaan kanssa sovitusta lopputuloksen määritelmästä. Kuten on aikaisemmin mainittu, verkkosivujen tulee olla luotettavia ja niiden tulee olla visuaalisesti yhteneväiset päätelaitteesta riippumatta, eikä verkkosivun laajennettavuus saa hankaloitua merkittävästi tulevaisuudessa. Hyvät käytännöt ja suunnittelu kattavat paljon verkkosivun laadun hallitsemisesta. Testisuunnitelman tavoitteena on luoda dokumentaatiota testauskäytännöistä, jotta testaaminen tehtäisiin johdonmukaisesti oikealla tavalla. Muut hyvät käytännöt ovat prosessien dokumentointi ja kirjoitetun koodin kommentointi, jotka auttavat muita ymmärtämään koodin logiikkaa tulevaisuudessa. Kattava dokumentointi auttaa tiimiä ja tarvittaessa myös uusia ihmisiä jatkamaan koodaamista. Koodin kommentointi nopeuttaa myös vianetsintää, mikä helpottaa ja halventaa virheiden korjaamisen kustannuksia. Koodin kommentoiminen kuitenkin unohtuu liian helposti kehitystyön tahdin kiihtyessä. [4, s. 4.] Kuvio 2 sisältää ohjelmiston pääkomponentit, jotka ovat pakollisia jokaiselle ohjelmistoprojektille.




Kuvio 2. Ohjelmiston komponentit [2, s. 3].

Ohjelmistotestaamisessa käytetään perinteisesti kuvion 3 V-mallia, joka on ollut pitkään testaamisen standardina. V-mallin nimi tulee sanoista verifiointi ja validointi. V-malli on kehitetty perustuen vesiputousmalliin, joka on ensimmäisiä ohjelmistokehityksen malleja. Khanin ja Mustafan [2, s. 181.] mukaan V-malli esittää loogisen järjestyksen verifikaatioille ja varmistusvaiheille. Sen käyttö vaatii testien suunnittelun ja dokumenttien laatimisen heti, kun vaatimukset ovat valmiita. Testidokumentaation aikainen laatiminen asettaa kehitystyön ja testaamisen samalle tärkeystasolle.




Kuvio 3. Ohjelmistotestauksen V-malli [2, s. 182].

V-mallia kohtaan on esitetty paljon kritiikkiä, sillä sen käyttäminen vaatii huomattavan määrän resursseja dokumentaation laatimiseksi. Kuviota 3 tulkitaan vasemmalta oikealle V-kuvion mukaisesti. Kuviosta voidaan havaita, että ennen koodaamisen aloittamista ohjelmistovaatimusten tulisi olla valmiina. Koodausvaiheessa suoritetaan yksikkötestejä, joita verrataan alimman tason vaatimuksiin. Yksikkötestausta käyttämällä varmistutaan, että ohjelmiston pienimmät rakennuspalat toimivat odotetusti. Virheet kumuloituvat helposti, jos komponentit ovat riippuvaisia toistensa luotettavuudesta. Yksikkötestit koodataan yleensä dynaamisesti toteutusvaiheen aikana, ja itse testien suorittaminen tehdään ohjelmiston testausvaiheessa. [2, s. 167.]

Kuvion 3 tasoilla rinnastetaan dokumentaatioita ja vaatimuksia testien tuloksiin, jolloin vaatimustaso on läpäisty tai hylätty. Yksikkötestaamisen jälkeen seuraava suoritettava taso on integrointiteostaustataso, jolla varmistetaan koodattujen moduulien toiminta kokonaisuuden kanssa. Integraation toiminta varmistetaan suorittamalla mustalaatikkotestejä järjestelmän ohjelmointitason kanssa. [2, s. 16.]

Integrointitason jälkeinen taso on järjestelmätestaustaso, josta käytetään myös termiä alpha-testi. Testaustavan tarkoituksena on tarkkailla ohjelmiston toimintaa samassa ympäristössä, jossa valmis ohjelmisto toimii. Toimintaa voidaan vertailla toiseen vastaavaan ympäristöön, jolloin ohjelmistoversioiden ominaisuudet erottautuvat. Eroavia ominaisuuksia kutsutaan regressioiksi, minkä takia järjestelmätestausta voidaan kutsua regressiotestaamiseksi. Testiympäristön tulisi olla mahdollisimman lähellä tuotantoympäristöä. Tällöin keskitytään koko järjestelmän toimintaan tietyssä ympäristössä. [2, s. 17.]

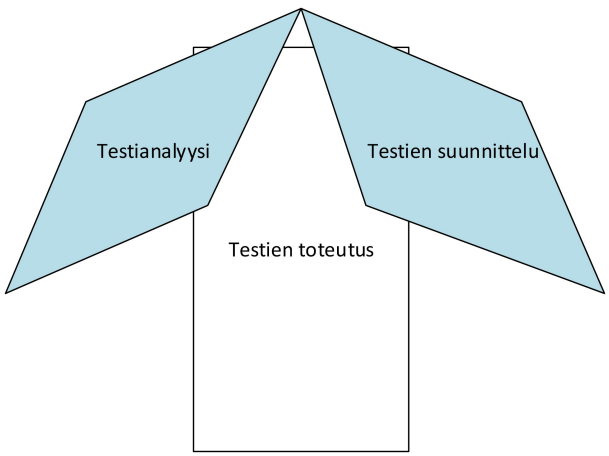
Kuvion 3 viimeinen testitaso on hyväksymistestaus, jolla varmistetaan tuotteen toimivan suunnitellusti. Testillä ei yritetä rikkoa testauksen kohdetta tai etsiä siitä virheitä. Toteutettua ohjelmaa tai komponenttia verrataan suunniteltuun ja arvioidaan, läpäiseekö ohjelmisto sille asetetut tavoitteet. Viimeisen testivaiheen aikana määritetään, onko tuote valmis jakeluun tai tuotantoon. Vaihe toteutetaan loppukäyttäjien avulla, jotka arvioivat järjestelmän suorituskykyä omasta näkökulmastaan. Verkkosivuja kehittäessä tämä loppukäyttäjäryhmä voi olla esimerkiksi jokin yrityksen sisäinen ryhmä tai tilaajana toimiva asiakas, joka varmistaa ohjelmiston toimivan odotetusti. [2, s. 17.]

Khanin ja Mustafan mukaan V-mallia kritisoidaan useasti siitä, että se on monimutkainen testaustapa. V-mallia voidaan tulkita monella tavalla, mikä voi johtaa harhakuvaan suoritetusta testauksen tasosta [2, s. 183]. Testausmallin väärät tulkinnat voivat viedä tes-

taustiimin ja projektipäälliköt outoihin tilanteisiin, joissa ohjelmisto ei toimi odotusten mukaisesti. Erehdyksiä tapahtuu helpoiten, kun projekti tiimin jäsenet tulkitsevat V-mallin vaiheita väärin puutteellisen ohjeistuksen takia. Tilanteelta voidaan välttyä sopimalla mahdollisimman tarkasti yhteneväisistä työtavoista ja luomalla ohjeistuksia jokaiseen työvaiheeseen.

Kevyempi B-malli

Testaamiseen varattu aika on rajallinen resurssi. Siksi testaamisen tulisi olla mahdollisimman tehokasta ja virheet tulisi löytää mahdollisimman helposti. Kuvion 4 B-malli on varteenotettavat vaihtoehto, kun vaatimusmäärityksiä ei ole. B-testausmalli on saanut nimensä englannin kielen sanasta *butterfly*. B-mallista käytetään myös nimeä perhosmalli. Kuviossa 4 perhosen vasen siipi koostuu suoritettujen testien tuloksista ja oikea siipi testin luonnoksesta. Perhosen ruumis esittää testin toteutusta. B-mallia voidaan pitää kevyempänä testausmallina, sillä siinä ei luoda turhia testejä. B-mallissa testausvastaavat perehtyvät testiraportteihin ja luovat niiden perusteella seuraavat testitapaukset. Seuraavat testit räätälöidään, jolloin varmistutaan myös vaatimuksiin pääsemisestä. [2, s. 185.]




Kuvio 4. Ohjelmistotestauksen B-malli [2, s. 184].

B-mallin testaamistapa on verkkosivuille sopiva testausmalli, sillä verkkosivun ominaisuuksia voidaan käydä läpi yksitellen. B-mallia voidaan käyttää esimerkiksi regressiotestauksessa, jolla tarkistetaan, miten ohjelmistoon tehdyt muutokset vaikuttavat alkuperäi-

seen ohjelmiston versioon. Verkkosivun toiminnallisuuksia laajennettaessa halutaan varmistua, ettei järjestelmän vanhat liitännäiset aiheuta uusia virheitä koodin muuttuessa. Yleensä testit tehdään automaattisesti. Regressiotestien tulokset esitetään joko prosenttilukuna muutossuhteesta tai indikaationa vanhojen testien epäonnistumisesta. Metodia käytetään erityisesti jo tuotannossa olevien ohjelmistojen kanssa, jolloin ei haluta rikkoa vanhoja ominaisuuksia. Regressiotestauksen tuloksena saadaan kartoitettua koodimuutosten vaikutukset suhteessa tuotantoversioon. [2, s. 17.]

Toinen B-mallilla suoritettava testi on suorituskykytestaus. Suoritusaikaa voidaan testata valmiilla verkkopalveluilla, jotka analysoivat verkkosivun latausaikaa. Esimerkki tällaisesta verkkopalvelusta on Googlen PageSpeed Insights -palvelu, joka ehdottaa, kuinka verkkosivun toteutusta tulisi ehostaa, jotta sen suorituskyky olisi ensiluokkaista. Selaimet sisältävät kehitystyökaluja, joilla voidaan tarkistaa verkkosivujen yhteydessä ladattavien resurssien kokoa ja latausaikaa. Selaimen työkalut eivät anna palautetta sivun teknisestä toteutuksesta. Ladattaviin resursseihin kuuluvat kaikki verkkosivulla olevat kuvat sekä JavaScript- ja CSS-tiedostot. Suorituskykytestaamista voidaan tehdä, kun uutta toiminnallisuutta testataan. Suorituskykytestaaminen varmistaa, että todellinen suoritusaika on tavoitteiden mukainen eivätkä verkkosivun latausajat kärsi uusien toiminnallisuuksien takia. [2, s. 17.]

Yleensä suorituskykytestauksen yhteydessä suoritetaan myös rasitustestaus, josta käytetään myös termejä kuormitus- ja stressitestaus. Rasitustestauksen tavoitteena on varmistua ohjelman suoriutumisesta myös silloin, kun käyttäjämäärä on suuri. Testaustavalla yritetään myös arvioida, miten ohjelma suoriutuu todellisessa tuotantoympäristössä, jossa ohjelmiston rasitusaste voi vaihdella paljon. Verkkosivuja testattaessa stressitestillä varmistetaan, kuinka paljon käyttäjiä verkkosivusto voi palvella samanaikaisesti vaikuttamatta verkkosivun latausaikoihin. [2, s. 17.]

Savutestillä tarkoitetaan ohjelmiston tai verkkosivun päätoiminnallisuuksien läpikäymistä. Verkkosivuilla tällä testaustavalla keskitytään kokonaisuuden toimivuuteen [6]. Savutestejä voidaan käyttää myös yhteensopivuustestaamisen kanssa, jolloin käyttöliittymän elementtien toimintaa tarkastellaan eri päätelaite- ja selainyhdistelmillä. Yhteensopivuustestaamisella varmistetaan vaatimuksien mukainen selaintuki. Selaimet toistavat HTML-dokumentteja eri tavalla. Siksi on huolehdittava verkkosivujen olevan yhteensopivia muillakin kuin verkkokehittäjän käytössä olevalla selaimella. Käytännössä eri toistamistavat johtuvat eri selainmoottorien ominaisuuksista. Yleisimmin virheet esiintyvät esimerkiksi

väärin toistetusta CSS-koodista, joka näkyy virheellisenä HTML-elementin ulkoasuna. Myös JavaScript voi käyttäytyä eri lailla selaimesta riippuen. Esimerkiksi valitsimet tai metodit eivät käyttäydy aina odotetusti. Selaintuki voidaan myös määritellä ennakkoon eri tasoihin, jolloin yleisimmin käytetyt selaimet tulee asettaa korkeimmalle prioriteetille. [3.]

## 2.2 Testaamisen haasteet

Testaamisen ongelma on se, miten suorittaa testejä niin, että kaikki mahdolliset virheet löytyvät, mutta samalla olla tuhlaamatta arvokkaita resursseja tehottomaan testaukseen. Kaikkia mahdollisia virheitä on mahdoton löytää, sillä testaukseen määrätyt resurssit ovat aina rajalliset. Perinteisesti testaaminen on se vaihe, jolle varattu aika kutistuu muiden työvaiheiden venähtäessä. Projektin aikataulun laahatessa viimeiset työvaiheet tehdään hätäisesti, jotta päästään tavoitteeseen. [4.]

Testaamisen tehokkaan onnistumisen kannalta on tärkeää luoda testitapauksia, jotka

- vähentävät testausvastaisuutta
- tuovat riittävän testien peitealueen
- hyödyntävät resursseja ja sallivat tehokkaan testaamisen [2, s. 12].

Toinen ohjelmistotestaamisen on ongelma se, ettei testaamisen edistystä voi mitata. Olemassa olevien virheiden määrä ei ole tiedossa testausta suorittaessa. Tämä tuo ongelman, miten resursoida virheiden korjaamista varten oikea määrä työvoimaa, jos ei tiedetä, kuinka paljon korjattavia virheitä on.

Yleisimpiä testaamisprosessia häiritseviä ongelmat ovat seuraavat:

- testisuunnitelman puuttuminen
- testauksen liian myöhäinen aloittaminen
- hankalat vaatimukset
- epätäydelliset suoritettavien testitapausten ohjeistukset
- tehtävään sopimattomien testaajien käyttö
- useampi kuin yksi samanaikainen päämäärä testaamisessa
- kokemattomat ohjelmoijat [2, s. 14].

Edellä mainittuja häiriötekijöitä voidaan välttää, kun seurataan testaamisen periaatteita. Optimaalisessa tilanteessa testivastaava olisi itsenäinen osapuoli. Pienessä kehitystiimissä samoja resursseja hyödynnetään koodaamiseen ja testaamiseen. Ohjelman koodaajan ei tulisi olla vastuussa testaamista, sillä työntekijä ei osaa arvioida tekemänsä työn laatua objektiivisesti. Testien huonoja tuloksia voidaan puolustaa, ja näin testien tulokset vääristyvät. Ihmisluonteeseen kuuluu myös omien virheiden huomiotta jättäminen. Testaamisen tulisi suunnitella ja toteuttaa kaikista pätevin henkilöstö, sillä testaaminen vaatii luovuutta ja huomattavan määrän erikoisosaamista. Pätevä henkilöstö osaa suunnitella valmiiksi testaustapauksia, josta saadaan informaatiota ohjelman luotettavuudesta. Hyvin suunnitellut testitapaukset ovat nopeita suorittaa ja niiden tulisi osoittaa, että testattava oleva ohjelma sisältää virheitä. [2, s. 21.]

Muita testauksen periaatteita on, että tiedetyt virhetapaukset tulevat esille testien tuloksissa. Silloin kysytään, miten odottamaton tai väärä sisääntuloarvo käyttäytyy testissä? Tällaisissa tapauksissa tarkastellaan palautettua virheviestiä ja varmistetaan, onko se oikea.

Ohjelmistoa ei tulisi muokata, kun testejä suoritetaan. Muokkaaminen voi aiheuttaa odottamattomia vääristymiä testituloksissa. Vääristyneet tulokset johtavat tehottomaan testaamiseen, ja jos virheitä ei saada toistettua uudelleen, aikaa tuhlataan olemattomien virheiden etsimiseen. [2, s. 21.]

Testaamisvaiheiden tuloksia tulisi dokumentoida. Tuloksista tulisi koostaa raportteja, joista käyvät ilmi läpikäydyt testitapaukset ja niiden tulokset. Raportteihin tulisi liittää mahdollisimman tarkkaa tietoa siitä, mitä ohjelmistoversioita on testattu ja kuinka monta kertaa testejä on suoritettu. Raportteihin tulisi myös liittää testien odotettu tulos. Odotettua tulosta verrataan saatuun tulokseen, jolloin voidaan määrittää, onko testi onnistunut vai epäonnistunut. [2, s. 21.]

Kaikki testit tulisi suunnitella etukäteen ja huolehtia siitä, että suunnitelmissa pysytään. Testien koodaamisen aikana ohjelmaan ei saa lisätä mitään ylimääräistä, kun testejä lisätään ohjelmistoon. Tällöin keskitytään vain testien kehittämiseen, ei testattavan ohjelman kehittämiseen. Jokaiselle testivaiheelle tulisi määritellä raja, jolloin testausvaihe on valmis. Raja voi olla testien onnistumisprosentti tai odotettu arvo. Jos testejä täydennetään, nykyinen testi tulee arvioida ja uusi testisuunnitelma lisätä alkuperäiseen suunnitelmaan. Käytettävät testausmetodit tulisi valita toteutettavan toiminnallisuuden tärkey-

den mukaan. Kaikilla metodeilla on omat ominaisuutensa, jotka määrittävät sen soveltuvuuden eri ohjelman eri osiin. [2, s. 21.]

## 2.3 Sisällönhallintajärjestelmien testaaminen

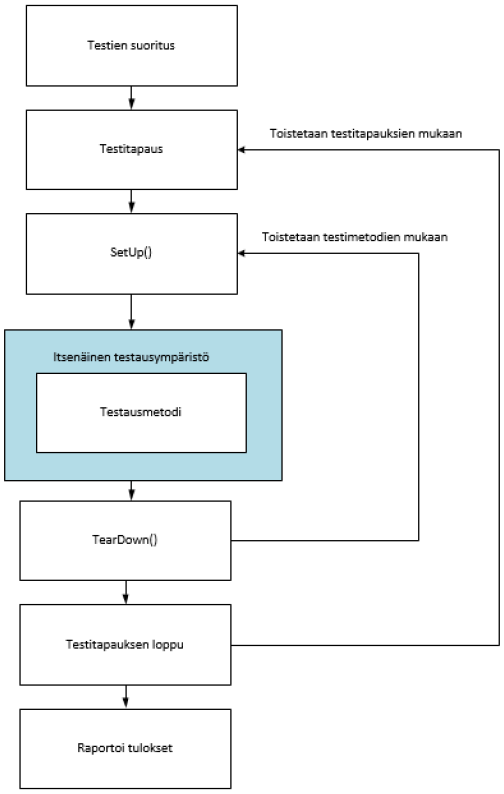
Verkkosivut ovat ohjelmistoja, joiden tehtävänä on jakaa informaatioita lukijoille. Itse selaimessa näytettävä verkkosivu ei juurikaan sisällä loogisesti tehtäviä vaativia tehtäviä. Sen sijaan verkkosivujen sisältöä käsittelevä järjestelmä hoitaa paljon automaattisia tehtäviä. Verkkosivujen hallintajärjestelmästä käytetään nimitystä sisällönhallintajärjestelmä. Englannin kielessä käytetään termiä CMS eli *Content management system*. Sisällönhallintajärjestelmässä lajitellaan ja hallitaan vierailijalle näytettävää sisältöä. Sen tarkoitus on helpottaa suurien alasivumäärien hallitsemista graafisen käyttöliittymän avulla. Sisällönhallintajärjestelmää käyttämällä vältytään verkkosivujen käsin koodaamiselta, sillä järjestelmä koostaa valmiiksi koodatun verkkosivun sinne syötetystä tiedoista. Käytännössä verkkosivun sisältöä editoidaan pitkässä verkkolomakkeessa, johon verkkosivun sisältö syötetään. Lomakkeessa on useita muokattavia kenttiä, jotka kontrolloivat julkisen verkkosivun ulkoasua ja sisältöä. [5.]

Sisällönhallintajärjestelmän toiminnallisuuksia voidaan laajentaa kahdella eri tavalla. Ensimmäinen vaihtoehto on käyttää valmiita liitännäisiä tai moduuleja, joiden ulkoasua muokataan niin, että uusi moduulin ulkoasu täsmää verkkosivun teemaan. Toinen vaihtoehto on koodata liitännäisen ominaisuudet itse, jolloin varmistutaan siitä, ettei turhaa koodia ladata. Valmiita liitännäisiä voidaan ladata sisällönhallintajärjestelmää ylläpitävän yhteisön kotisivuilta. Esimerkiksi Wordpress-sisällönhallintajärjestelmälle on 42 067 liitännäistä, joilla voidaan laajentaa järjestelmän mukana tulleita työkaluja. Drupalin vastaava liitännäisten määrä on 32 785. Suuri liitännäisten määrä selittyy osin sillä, että molempiin järjestelmiin on olemassa paljon vain vanhoihin versioihin yhteensopivia liitännäisiä. [6; 7.]

Liitännäiset ovat yleensä yhden tai muutaman kehittäjän koodaamia. Liitännäiset ovat yleensä painavin syy, minkä takia valita sisällönhallintajärjestelmä. Verkkosivun kehitysaika lyhentyy huomattavasti, kun kaikkia verkkosivun sisältämiä toiminnallisuuksia ei tarvitse itse koodata. Liitännäisen koodaajalla on ollut vastaava tarve laajentaa verkkosivustoa, ja hän on koodannut liitännäisen. Oleellista avoimen lähdekoodin ohjelmistoprojektille on liitännäisen jakaminen, jolloin muut sisällönhallintajärjestelmän käyttäjät

voivat arvioida sitä ja tarvittaessa myös parannella sitä. Liitännäisten parantelu tuo lisää ominaisuuksia liitännäisten käyttäjille. Liitännäisten säännöllinen päivittäminen tukkii mahdollisia tietoturvaukkoja. Liitännäisten päivittäminen kuuluu yleensä verkkosivujen ylläpitosopimuksiin, sillä se pitää mahdolliset tietoturva-aukot tukossa. Liitännäisen laadunvalvonta jätetään sen ylläpitävän kehittäjän ja yhteisön harteille. Molemmissa, Wordpress- ja Drupal-sisällönhallintajärjestelmissä on olemassa omat virheiden raportoimisprotokollat, jotka indikoivat liitännäisen laatua. Liitännäisten latausmäärät kerrotaan avoimesti, samoin kuin niitä käyttävien verkkosivujen määrä. Yhteisön jäsenet raportoivat avoimesti liitännäisten sisältävistä virheistä. Raporttien perusteella on mahdollista arvioida liitännäisen laatua, ennen kuin liitännäisen ottaa käyttöön.

Drupalille on olemassa lisämoduuleita, jotka on tarkoitettu erityisesti laadunvalvontaan. Drupal 7 -version ytimeen on lisätty SimpleTest-testauskehikko, joka lisää moduulien testaustasoa. Julkisesti jaossa olevia Drupalin moduuleja testataan automaattisesti verkossa olevalla SimpleTest-testauskehikolla, joka luo omat tietokantataulukot moduulia varten ja raportoi niistä eteenpäin Drupal.org-verkkosivulle. Testitulokset lisätään liitännäisen tietoihin. SimpleTest-testauskehikolla voidaan tehdä moduuleille toiminnallisuustestejä, yksikkötestejä ja päivitystestejä. Testauskehikko käy läpi testit puhtaalla Drupalin asennuksella, jolloin varmistetaan, että moduulien koodauskäytännöt seuraavat yhteisön standardeja, tekemättä vääriä oletuksia Drupal-pohjaisesta verkkosivustosta. Testille luodaan vasta-asennetut tietokantataulukot ja tiedostot, jolloin moduuli eristetään nykyisestä Drupal-ytimen asennuksesta. Kuviossa 5 nähdään, että yhteen testiin kerätään samantyyppiset testit, joilla on samat asetus- ja moduulivaatimukset. Drupal-yhteisön mukaan sisällönhallintajärjestelmän sisäänrakennettu testaus keskittyy enemmän toiminnallisuustestaamiseen, mikä johtuu Drupalin koodaustavasta. Toiminnallisuustestaamiseen sisältyy yleensä muutaman alasivun mekaaninen luominen ja niiden sisältöjen satunnainen luominen. Testitapauksien koodaamisella säästytään hitaalta käsin testaamiselta ja varmistetaan, että tarvittavat asetukset tehdään jo moduulissa oletuksena. [9; 8; 10]




Kuvio 5. SimpleTest-testausympäristön luonti [8].

Wordpress-sisällönhallintajärjestelmälle on olemassa testauskäytäntöjä, mutta ne eivät ole niin automaattisia kuin Drupalin testauskäytännöt. Wordpressin ytimeen ei ole valmiiksi liitetty vastaavaa testauskehikkoa niin kuin Drupalissa. Tämän takia Wordpressille testaustyökaluja on monenlaisia tapauksesta riippuen. Liitännäisille on mahdollista asentaa PHPUnit-testauskehikko, mutta tämä vaatii asetusten määrittelyä. PHPUnit on tarkoitettu funktioiden yksikkötestaamiseen. Liitännäisten toiminnallisuustestaamiseen löytyy myös omia apuja, esimerkiksi Wptest-verkkosivu, jossa on ladattava kokoelma liitännäisiä ja sivupohjia, joilla voidaan testata oman liitännäisen integraatiota muiden yleisimpien liitännäisten kanssa. Wordpressin liitännäisten laadunvalvonta perustuu yhteisön arvioihin ja palautteeseen. Wordpress.org-verkkosivuston liitännäisten arviointiosastolle raportoidaan, jos liitännäisessä on virheitä. Avoimien ongelmien määrästä voidaan arvioida liitännäisen laatua. Toinen arviointiin käytetty kriteeri on päivitystiheys, joka kertoo, päivitetäänkö liitännäistä myös tulevaisuudessa. Wordpressin testausmetodia voisi kuvailla enemmän beta-testaamiseksi, sillä liitännäisen loppukäyttäjät antavat palautetta sen sisältämistä ohjelmistovirheistä. [11.]

Toinen tapa sisällönhallintajärjestelmän laajentamiseen on kirjoittaa oma liitännäinen, joka toteuttaa vain vaadittavat toiminnallisuudet. Molempien, Wordpress- ja Drupal-sisällönhallintajärjestelmien liitännäisten ongelma on, että niitä on joskus monimutkaista käyttää. Liitännäisien kaikkia ominaisuuksia ei aina tarvita. Asennettujen liitännäisten suuri määrä hidastaa verkkosivun latausaikaa. Tämä voi olla yksi syy, miksi oman liitännäisen kirjoittaminen tietyissä tapauksissa kannattaa. Toinen syy on se, ettei itse kirjoitettua liitännäistä tarvitse päivittää jatkuvasti. Oman liitännäisen kohdalla suurin osa ajasta tuhlaantuu dokumentaation laatimiseen ja testaamiseen. Ilman kunnollista dokumentaatiota toisella kehittäjällä voi olla huomattavia ongelmia vian etsinnässä, kun liitännäisen sisäinen toiminta ei ole selkeää. Itse kirjoitetusta liitännäisestä tietoturva-aukot eivät tule esille niin selkeästi.

Ratkaisevin tekijä on verkkokehittäjän taidot, sillä vähemmänkin pätevä verkkokehittäjä osaa ylläpitää ja päivittää ladattavaa liitännäistä. Uusi liitännäinen ladataan ja määritellään asetukset. Tämän jälkeen vain huolehditaan liitännäisen sopivan visuaalisesti teemaan. Valmiiden liitännäisten käyttöönotto sujuu nopeasti, sillä liitännäisen kehittäjä on jo tehnyt kehitystyön etukäteen. Tällöin suurin osa ajasta kuluu asetusten määrittämiseen. Jos liitännäisestä löytyy virheitä, niistä raportoidaan eteenpäin ja toivotaan, että liitännäisen ylläpitäjä päivittää korjauksen seuraavaan liitännäisen versioon. Valmiiden liitännäisten käyttö säästää kehitysaikaa projektin alussa, mutta se aika siirtyy projektin ylläpitoon, kun syntyy tarve päivittää liitännäisiä. Omaa liitännäistä ei ole tarvetta päivittää, jos se on hyvin toteutettu, sillä sen toiminnallisuudet ovat huomattavasti kapeammat kuin valmiin liitännäisen. Itse koodaamalla turhia toiminnallisuuksia ei toteuteta rajallisen ajan takia.

Sisällönhallintajärjestelmille on myös tyypillistä, että ne käyttävät virhelokia, jonka perusteella voi etsiä mahdollisia koodausvirheitä teemoista ja liitännäisistä. Virheloki koostaa listaa PHP-virheistä ja ilmoittaa virheen sijainnista. Yleensä virheet näytetään vain verkkosivua ylläpitäville henkilöille. Näin huolehditaan siitä, ettei julkisen verkkosivun ulkoasu kärsi virheilmoituksista. Virheenkorjauslokin käyttö ei kuitenkaan estä virheiden syntymistä, jos itse koodattua liitännäistä ei testata laajasti. Lokia käytetään lähinnä vianetsintään ja virheiden pois sulkemiseen. Lokiin voidaan myös tallentaa itse koodatun liitännäisen sisäistä toimintaa, jolloin sitä voidaan hyödyntää lasilaatikkotestaamisessa.

## 2.4 Ulkoasun yhteensopivuus ja lähdekoodin kohdentaminen

Visuaalisella ulkoasulla tehdään verkkosivusta kutsuvan näköinen. Ulkoasun yksityiskohdat viimeistelevät kokonaisuuden. Sivulla vierailijan selain, laite ja yhteysnopeus määrittävät verkkosivun selailukokemuksen yksityiskohtaisesti. Esimerkiksi fonttien toistaminen vaihtelee selain- ja laitekohtaisesti huomattavasti. Mahdollisia laite- ja selainyhdistelmiä on olemassa paljon. Ulkoasun tulisi olla yhtenäinen laiteyhdistelmästä huolimatta. Näin viestitään vierailijalle, että verkkosivun edustama taho on luotettava ja jatkuva. Verkkosivustojen omistajat tahtovat tarjota käyttäjilleen mieleenpainuvia ja positiivisia kokemuksia mahdollisimman paljon. Tämän kokemuksen tarjoaminen voi vaatia uusimpien tekniikoiden käyttämistä, joiden tukea ei ole vielä lisätty yleisimpiin laitteisiin. Tällöin verkkokehittäjän on varmistettava, että tekninen lopputulos on käyttökelpoinen mahdollisimman laajalla käyttäjäpohjalla. Koodia voidaan kohdentaa vierailijan päätelaitteen ja selaimen mukaan. Päätelaitteiden poikkeavien selainominaisuuksien takia verkkokehittämistä mobiiliselaimille kuvaillaan vihamielisen ohjelmoinnin alueeksi. Vanhempien selainversioiden käyttäjille tehdään versio samasta informaatiosta, jota loppukäyttäjän selain tukee. Yleisemmät ongelmia aiheuttavia selaimet ovat vanhat Internet Explorer -selaimet. Vanhoja selaimia varten CSS-koodia kohdennetaan erityisesti ongelmallisiin selaimiin, jolloin huolehditaan ulkoasun pysyvän ehjänä mahdollisimman monella selaimella. Internet Explorer -selaimiin kohdennetaan yleensä CSS-koodia konditionaalisilla kommenteilla. [12; 13.]

Näin voidaan huolehtia, että tietty CSS- tai JavaScript-koodi ladataan vain Internet Exploreria käytettäessä. Koodiesimerkki 1:n CSS-koodin kohdennus toimii vain Internet Explorer -selaimien kanssa, joten muille yleisille selaimille tulee käyttää muita selaimen tunnistustapoja. Ratkaisuksi on olemassa JavaScript- ja PHP-pohjaisia ratkaisuja, jotka lisäävät body-elementtiin CSS-luokan käyttäjän selaimen mukaisesti. Suosituimmille sisällönhallintajärjestelmille on olemassa valmiita liitännäisiä tätä varten. Käyttämällä näitä liitännäisiä voidaan kohdentaa koodia vain tietyille selaimille.

```
<html>
  <head>
    <!--[if IE 6]>
      <style>
        body p {
          font-size:14px;
        }
      </style>
    <![endif]-->
  </head>
  <body>
    <p>Hello World!</p>
  </body>
</html>

```

Koodiesimerkki 1. Internet Explorerille kohdennettu CSS-tiedosto.

Liitännäinen lisää muita ominaisuuksia body-elementtiin, jolloin koodia voidaan myös kohdentaa näyttökoon mukaan mobiili- tai työpöytäversioon. Body-elementin CSS-luokan avulla koodia voidaan kohdentaa koodiesimerkki 2:n mukaisesti. [14.]

```
<html>
  <head>
    <script src="Modernizr.min.js" />
    <style>
        body.chrome.mac p {
            font-size: 11px;
        }
    </style>
  </head>
  <body class="chrome chrome11 mac desktop">
    <p>Hello World!</p>
  </body>
</html>
```

Koodiesimerkki 2. Selaimen nimen, version ja käyttöjärjestelmän lisääminen CSS-luokana HTML-dokumenttiin.

Liitännäiset perustuvat GET-pyynnön lukemiseen palvelimelta, jolloin pyynnön ylätunnisteessa lähetetään käyttäjäagentti. Se kertoo vierailijan selaimesta tarvittavat tiedot. Liitännäisissä on yleensä asetus, jolla voidaan tehdä sama selaimentunnistus JavaScriptillä. Suosittu selaimentunnistustapa on käyttää Modernizr-JavaScript-kirjastoa. Modernizr lukee käyttäjäagentin arvoja ja lisää oletetun selaimen nimen CSS-luokaksi verkkosivulle. Merkittävin ero tämän kirjaston käytössä on, että se lisää myös luokkina selaimen tukemat ominaisuudet. Koodiesimerkki 3:n avulla CSS-koodia voidaan kohdentaa selaimen tukemien ominaisuuksien mukaan. [15, s. 302.]

```
<style>
    .no-cssgradients .header {
      background: url("images/glossybutton.png");
    }
    .cssgradients .header {
      background-image: linear-gradient(cornflowerblue, re-beccapurple);
    }
</style>

```

Koodiesimerkki 3. Selaimen liukuvärituen havaitseva CSS-määritys.

Modernizr-JavaScript-kirjastoa voidaan hyödyntää myös JavaScriptillä tehtyjen ominaisuuksien kohdentamiseen. JavaScript-funktioita voidaan kohdentaa suoraan koodiesimerkin 4 mukaisesti.

```
<script>
      if (Modernizr.awesomeNewFeature) {
          showOffAwesomeNewFeature();
      } else {
          getTheOldLameExperience();
      }
</script>

```

Koodiesimerkki 4. Modernizr-kirjaston havaitsema JavaScript-tuen taso.

JavaScript-koodin avulla voidaan tehdä paljon ulkoasun ja ominaisuuksien viimeistelyä halutuille selaimille, mikä parantaa huomattavasti verkkosivuston käyttäjäkokemusta. Vanhojen selainversioiden tuki on yleensä rajoittava tekijä, kun halutaan lisätä uusia ominaisuuksia verkkosivustoille. Laajaa seilaintukea tehtäessä on erittäin tärkeää tehdä myös minimivaatimuksia tai vanhempia selaimia tukeva varaversio.

## 2.5 Älypuhelimien ja tablettien testattavat erikoisuudet

Nykyisiä älypuhelimia käytetään kosketusnäytöillä. Kosketusnäytön ja hiiren käyttö eroaa toistaan. Kosketusnäyttöjä käytettäessä elementtien päällä leijuminen on huomattavasti vaikeampaa, sillä se onnistuu vain painamalla pitkään elementin päällä. Yleensä leijumiseen perustuvat efektit otetaan pois käytöstä, kun selataan verkkosivun älypuhelimille tarkoitettuja versioita. JavaScriptin avulla funktioita voidaan sitoa eri käyttäjän interaktioihin näyttöikkunan kanssa.

Yhteen klikkaustapahtumaan voidaan sitoa tapahtumia seuraavassa järjestyksessä:

1. Touchstart
2. Touchmove
3. Touchend
4. Mouseover
5. Mousemove
6. Mousedown
7. Mouseup
8. Click.

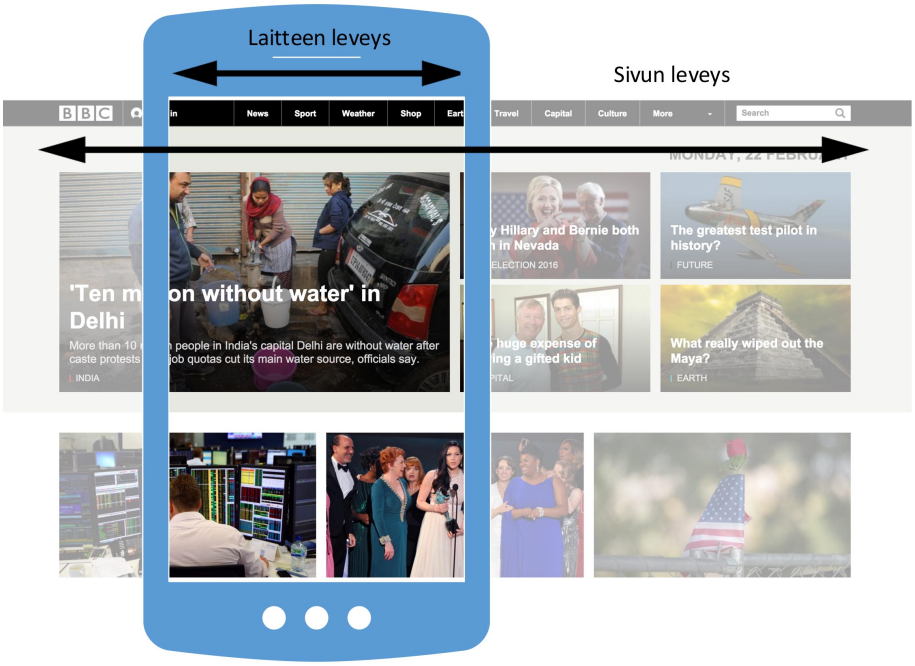
Tapahtumajärjestys koskee lähinnä JavaScriptin kirjoittamista. Testaamisvaiheessa väärään kosketustapahtumaan sidottu funktio tulee ilmi, sillä kosketustapahtumat eivät automaattisesti käynnisty ristiin keskenään. Esimerkiksi mousemove-tapahtuma kosketusnäytöllä ei käynnisty vain kosketuksesta. Samoin touchmove- ja mousemove-tapahtuma eivät vastaa samoja tapahtumia hiirellä ja kosketusnäytöllä. [16.]

Älypuhelimien korkeat näyttöresoluutiot

Korkearesoluutioisten näyttöjen kehitys älypuhelimiessa mahdollistaa terävämpien ja tiheämpien kuvien esittämisen pienemmillä näytöillä. Retina-resoluutio on Applen käyttämä markkinointitermi korkean pikselitiheyden näyttöteknologiasta. Retina-termillä puhekielessä viitataan yli 326 ppi:n pikselitiheyden omaaviin näyttöihin. Tarkemmin selitettynä Retina-näyttö tarkoittaa niin tarkkaa näyttöä, ettei ihmissilmä erota yksittäistä pikseliä näytöltä. Retina-näytön määritelmän täyttyvyys on aina katseluetäisyydestä riippuvainen. Älypuhelimien katseluetäisyys on huomattavasti tavallista tietokoneen näyttöä lyhyempi, joten älypuhelimien näyttöjen pikselitiheydet ovat erittäin korkeita. Muut valmistajat käyttävät eri termejä samasta Retina-tekniikasta. Yksi tällainen termi on QHD-näyttö, jonka resoluutio on 2560 x 1440 pikseliä. Retina-määritelmä täyttyy QHD-näytölle, kun näytön fyysinen koko on tarpeeksi pieni. [17; 18.]

Yleisiä näyttöjen pikselitiheysarvoja ovat 2–3,5. Tämä näkyy älypuhelimissa niin, että esimerkiksi Iphone 4 -älypuhelin ilmoittaa oman näyttöresoluutionsa olevan 320 pikseliä leveä, mutta todellinen näyttöresoluutio on kaksi kertaa suurempi, eli 640 pikseliä. Tällöin pikselitiheysarvo on 2. 320 pikseliä leveälle näytölle kohdennettu sisältö toistetaan näytöllä kaksi kertaa tiheämpänä. Keskivertoselaajan älypuhelimen näyttökoko on 4,0–4,49 tuumaa. Fyysisen näyttökoon kasvaessa voidaan olettaa, että käytettävissä laitteis-

sa on korkean pikselitiheyden näyttöjä, sillä tiheimmät näytöt ovat suurimman näyttökoon älypuhelimissa. [19; 20.] Kuva 1 näyttää verkkosivun toistamisen mobiiliselaimella, jos sivun sisältöä ei suhteuteta älypuhelimen näytölle.




Kuva 1. Verkkosivun oletusleveys, jos laitteen näytön leveyttä ei ole määritetty täsmäämään selainikkuna kokoon [21].

Näyttöresoluution kasvaessa ladatut valokuvat vaativat korkeampaa resoluutiota. Perinteisesti verkkosivuilla käytetyn kuvan DPI on ollut 72 kuvapistettä tuumaa kohden. Verkkosivulla käytettävän digitaalisen valokuvan DPI:n pitää olla korkeampi kuin aikaisemmin, sillä erittäin tiheät näytöt skaalaavat valokuvat tarkimman mahdollisen pikselitiheyden mukaan. Näyttöjen resoluutiotiheyden nouseminen vaikuttaa verkkosivujen ulkoasuihin toistamalla fontit ja pikseligrafiikan terävämmin, sillä ne toistetaan näytön alkuperäisen resoluution mukaisesti. Korkearesoluutio näytöt eivät vaikuta mukautuvien ulkoasujen murroskohtiin, jos metatiedon määritys on tehty oikein. Koodiesimerkki 5 ilmoittaa metatietona selaimelle HTML-dokumentin ylätunnuksessa, että laitteen ikkunan leveyden tulee olla sama kuin selainikkunan leveys.

```
<html>
  <head>
    <meta name="viewport" content="width=device-width">
  </head>
  <body>
    <p>Hello World!</p>
  </body>
</html>

```

Koodiesimerkki 5. Selaimen- ja näyttöikkunan koko saadaan täsmättyä, kun käytetään esimerkissä näkyvää metatagia HTML-dokumentissa [22].

Vaikka näyttöresoluutio älypuhelimessa on korkea, selaimen näyttöikkunaresoluutio ei kasva, sillä responsiivinen ulkoasu sopeutuu aina selainnäyttöikkunaresoluution mukaisesti eikä näytön alkuperäisen resoluution mukaan. CSS-koodissa määritetty pikselikoko on älypuhelimella tai tabletilla suurempi kuin 1. [22.]

HTML5

Melkein kaikki uusimmat mobiiliselaimet ovat HTML5-yhteensopivia. Virallinen W3C:n HTML5-standardi ja sen laajennukset vaihtelevat hieman, sillä HTML5-standardiin voidaan laskea mukaan myös kokeellisia selainten ominaisuuksia. HTML5-tukea voidaan mitata eri selaimien suhteen, sillä selaimet tukevat laajennuksia eri lailla. [23.]

Taulukko 1 listaa suosituimmat HTML5-selaimet, joita käytetään älypuhelimilla. Joistakin selaimista on olemassa useaa käyttöjärjestelmää tukevia versioita. Esimerkiksi Chrome-selaimesta on olemassa omat viralliset versiot Androidille ja Applen Iphonelle, muttei Windows Phonelle.

Taulukko 1. HTML5-mobiiliselaimet ja niiden selainmoottorit [24].

| **Selain** | **Tuetut mobiilikäyttöjärjestelmät** | **Selainmoottori** |
|---|---|---|
| Safari iOS | Apple Iphone, Apple Ipad | Webkit |
| Android Browser | Android | Webkit |
| Samsung Internet | Android | Webkit |

| | | |
|---|---|---|
| Google Chrome | Android, iOS | Blink |
| BlackBerry Browser | BlackBerry | Webkit |
| Nokia Browser | Nokia X, Symbian Belle S60 | Webkit |
| Internet Explorer | Window Phone, Windows 8.x | Trident |
| Opera Mobile | Android | Blink |
| Opera mini | Android, iOS, Window Phone | Presto |
| Firefox | Android, Meego, Firefox OS | Gecko |

Selaimia voidaan jakaa kategorioihin niiden käyttämän selainmoottorin perusteella, sillä selainmoottori määrittää suurimman osan selaimien perusominaisuuksista. Selainmoottorien erot huomataan yleensä virheinä tuetuissa standardeissa ja elementtien prosessointijärjestyksessä. [25.]

CSS3:n mediakyselyt

Selaimien ominaisuudet voivat erota toisistaan hieman, mutta ne huomataan viimeistään verkkosivuprojektien testausvaiheessa. Yleensä koodia kirjoittaessa käytettyjen CSS-valitsimien selaintuki tulee tarkistaa. Verkossa on olemassa moniakaan sellaisia työkaluja, joilla voidaan varmistaa miten selaimet tukevat erikoisempia tageja ja valitsimia. Yksi näistä työkaluista on Caniuse.com-verkkosivusto, joka listaa selaimien tukemia valitsimia ja tageja. Nykyaikaiset HTML5-selaimet tukevat CSS3-standardia, johon sisällytetty mediakyselyt. Uusin standardi mahdollistaa CSS-koodin kohdentamisen päätelaitteiden ominaisuuksien mukaisesti. [26.]

Mediakyselyjä voidaan määrittää eri ominaisuuksien mukaisesti seuraavasti:

- pikselitiheys
- kuvasuhde

- animointituki
- JavaScript-tuki
- näyttökorkeus
- näyttöleveys.

Mediakyselyssä selainikkunan leveys on keskeinen seikka, jotta mobiili- ja työpöytäselain voidaan erottaa toisistaan. Kaikki HTML5-työpöytäselaimet ymmärtävät mediakyselyitä. Tärkein huomioitava asia mediakyselyitä koodatessa on, että selainikkunan resoluutio eroaa laitteen näytön resoluutioista, sillä CSS-koodissa määritetty pikselin koko on älypuhelimessa tai tabletissa suurempi kuin 1. Päätelaitteelle kohdentaminen tehdään lisäämällä verkkosivun CSS-koodiin koodiesimerkki 6:n mukaisesti.

```
<style>
    @media (min-width: 769px) {
        body p {
         color:red
        }
    }
</style>
```

Koodiesimerkki 6. CSS-koodi määrittää verkkosivun oletusfonttivärin punaiseksi, kun selainikkunan leveys on suurempi kuin 769 pikseliä [27].

Tyylimääritykset periytyvät mediakyselyiden mukaisesti, joten on tärkeää käyttää mediakyselyitä ja niiden murroskohtia johdonmukaisesti. Mediakyselyitä voi kohdentaa tietylle leveysvälille tai hyödyntää niitä käyttämään johdonmukaisesti edellisiä määrityksiä, jolloin viimeisin tyylimääritys on voimassa. Nykyaikaiset verkkosivujen ulkoasujen tyylimääritykset kirjoitetaan *mobile first* eli mobiili ensin -tekniikan mukaisesti, jolloin oletuksena CSS-koodi kohdennetaan kapeimmille näyttökooille. Näyttökoon kasvaessa yli keskeytyskohdan CSS-määrityksiä täydennetään. [28.]

## 3 Responsiivisten verkkosivujen testaustyökaluja

Sivustolla vierailija selaa yksi alasivu kerrallaan valikkojen ja linkkien avulla. Myös verkkokehittäjä rakentaa verkkosivua yksi sivu kerrallaan. Tällöin on huolehdittava, etteivät uudet kehitettävät osiot häiritse muita verkkosivulla olevia ulkoasuja tai toiminnallisuuksia. Pienetkin koodimuutokset voivat rikkoa ulkoasuja tai aiheuttaa muita virheitä, joiden

näyttämistä vierailijalle halutaan välttää. Kun kyseessä on suuri verkkosivusto, virheiden huomaaminen voi olla haastavaa, sillä alasivujen läpi käyminen käsin on erittäin työlästä.

Yksittäisellä käyttäjällä voi olla useita laitteita, joilla verkkosivua selataan. Eri älypuhelimilla ja tableteilla selatun verkkosivun tulisi olla keskenään samanlainen. Selaamiseen käytettyjä laitteita voidaan hallita kosketusnäytön lisäksi nuolilla, rullalla, sauvaohjaimella ja tapeilla. Ulkoasun responsiivisuus tuo omat haasteensa verkkosivuston toteuttamiseen. Miten varmistetaan verkkosivun toimivan kunnollisesti jokaisella päätelaitteella, sillä päätelaitteiden näyttöresoluutiot ja selainmoottorit vaihtelevat paljon? Verkkosivujen teknisen toteutuksen on seurattava trendejä joita verkkosivun selaaja seuraa. Trendit voivat olla laitetrendejä tai niihin liittyvät navigointitavat. Uusimpia laitetrendejä seuraamalla varmistetaan toimiva informaation jakelu loppukäyttäjän ja verkkosivun omistajan välillä. [29; 12, s. 150.]

Älypuhelimien ja tablettien käyttämä tekniikka kehittyy jatkuvasti, mutta toistaiseksi toimistotyöskentelyn välineet eivät ole muuttuneet. Verkkosivuja kehitetään edelleen toimistoympäristössä suurilta näytöiltä, vaikka niitä selataan suurimmaksi osaksi älypuhelimilla. Koska verkkosivuja kehitetään suurilla näytöillä, voidaan olettaa, että niitä testataan myös eniten työpöytäkoon selainikkunalla. Verkkosivujen kehittämisessä älypuhelimella selaavat tulisi ottaa entistä enemmän huomioon, koska selaaminen tapahtuu suurimmaksi osaksi älypuhelimella. [12, s.152.]

## 3.1 Yhteensopivuustestaus emulaattorilla ja älypuhelimella

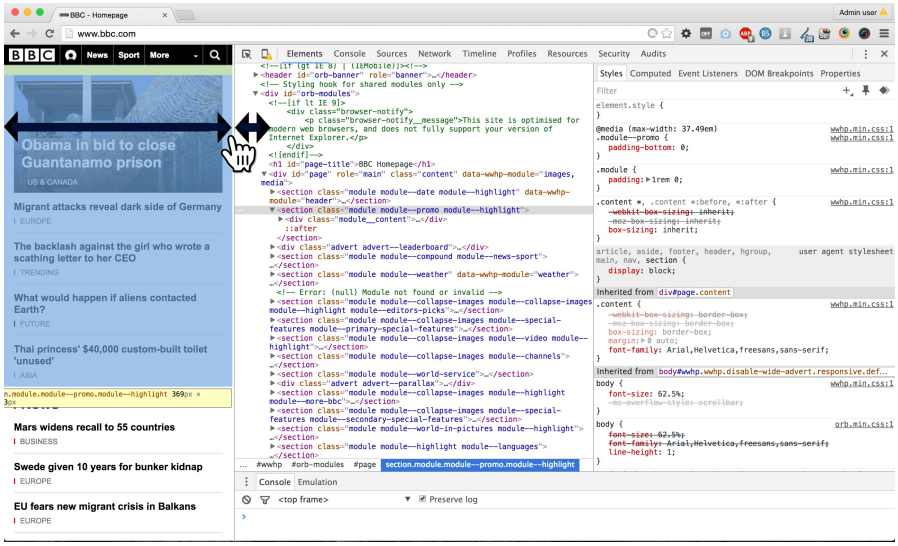
Yhteensopivuustestaaminen kaikilla päätelaitteilla on yleensä mahdotonta, koska uudet älypuhelimet ja alkuperäislaitteet ovat kalliita ja testausresurssit rajallisia. Testausvaiheessa varmistetaan, että lähdekoodin kohdennus on tehty oikein ja selain tukee samoja ominaisuuksia kuin kehittäjän käyttämä selain. Virheitä löydetään, jos selain jättää tiettyjä CSS-määrityksiä toistamatta. Kehittäjän tulisi tietää projektin alkuperäiset selaintukivaatimukset, jotta aikaa ei tuhlata laajan laitetuen toteuttamiseen.

Vanhojenkin puhelimien hankkimisessa testauskäyttöön on omat haasteensa, sillä käytettyjen puhelimien etsimiseen voi tuhlaantua paljon aikaa. Älypuhelimien ja selaimien valmistajat tarjoavat vaihtoehtoja, jotta verkkokehittäjien ei tarvitse hankkia suurta mää-

rää laitteita yhteensopivuuden varmistamiseksi. Nämä vaihtoehdot koostuvat lähinnä simulaattoreista, joita tarjotaan ilmaiseksi verkko- ja mobiilisovelluskehittäjille. Simulaattoreiden hyöty on, että niitä voidaan operoida tietokoneen kautta. URL-osoitteet voidaan liittää suoraan mobiiliselaimen osoitekenttään, mikä tekee niistä käteviä käyttää. Simulaattoreiden käyttöä puoltaa myös se, että laitevalmistajat tarjoavat tapoja, jolla voi tehdä viankorjausta simulaattorilla, joka voi olla oikean laitteen kanssa estetty. [30.]

Verkkokehittämisen parhaisiin käytäntöihin kuuluu, että kehittämisen yhteydessä tulisi käyttää simulaattoreita ja viimeinen vaatimusten hyväksymistestaus pitäisi hoitaa oikeilla laitteilla. Tämä käytäntö säästää aikaa huomattavasti, sillä käsintehty laitetestaus on hidasta. Oikealla laitteella pitää olla pääsy vähintään langattomaan verkkoon, mikä vaatii jonkun verran aikaa kirjautumistunnuksien syöttämisen. Suuri osa ajasta kuluu älypuhelimien asetusten määrittämiseen ja www-osoitteiden ja salasanojen kirjoittamiseen. Tämän kaiken tiedon syöttämisen jälkeen saadaan läpäisty testi vain yhdestä laitteesta, minkä jälkeen tämä kaikki pitää toistaa uudelleen eri laitteella. Käyttämällä oikeata laitetta testaamiseen pitää myös huolehtia akun varauksen tasosta, mikä ei ole ongelma simulaattorien käyttämissä. [31.]

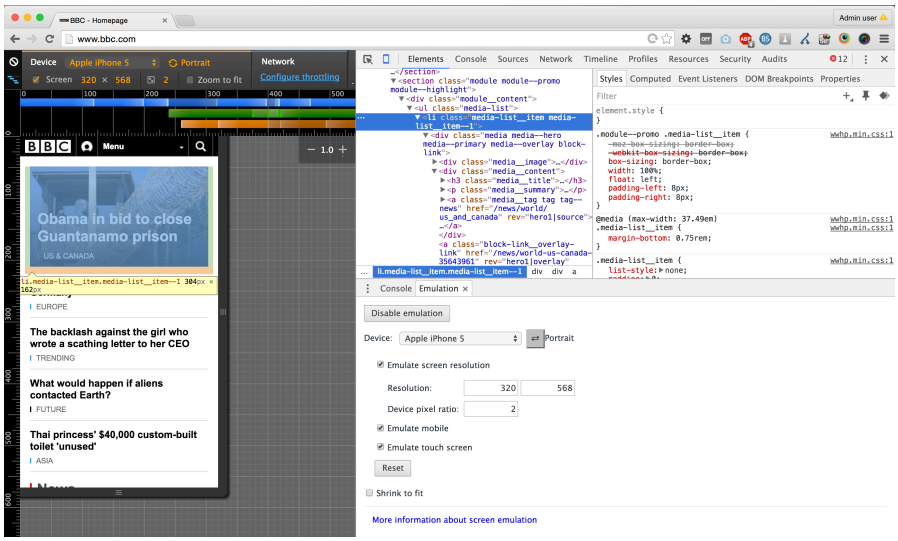
Kuva 2 näyttää, että yksinkertaisin tapa emuloida mobiiliselaimia on mukauttaa työpöytäselaimen ikkunaa kapeammaksi. Tällöin responsiivinen ulkoasu seuraa mediakyselyjen avulla asetettuja keskeytyskohtia. Selaimen kaventamista voidaan tehdä jokaisella HTML5-työpöytäselaimella, mutta todellista yhteensopivuudesta ei saada tuloksia, sillä itse laitteen käyttäytyminen ei liity selaimen kaventamiseen mitenkään. Laitetestaamisen tarkoituksena on saada selville yhteensopivuus mobiiliselaimien, käyttöjärjestelmien ja korkean resoluution kosketusnäyttöjen kanssa. Testaustavalla halutaan varmistaa, että responsiivinen verkkosivu on toimiva vaatimuksiin täsmäävillä laitteilla. Selaimen kaventamisella vain varmistetaan mediakyselyillä kohdennetun CSS-koodin ja JavaScriptin toimivuus.




Kuva 2. Google Chromen selainikkunan kavennus.

Verkkokehittäjien keskuudessa on suosittua käyttää Chromea. Sen Blink-selainmoottoria käyttämällä kehittämisessä päästään helposti perustoiminnallisuuksien yhteensopivuuteen muillakin selaimilla. Blink-selainmoottori on kehityshaara Webkit-selainmoottorista, joka on käytössä esimerkiksi Safari- ja Android Browser -selaimissa. Chromen verkkokehitystyökaluja käyttämällä voidaan myös emuloida älypuhelimien ominaisuuksia. Esimerkiksi laitteiden näyttöresoluutioita, pikselitiheyttä, verkkoyhteyden nopeutta, käyttäjäagentin havaitsemista ja GPS- ja kiihtyvyysanturin arvoja voidaan hallita. Tämä kehitystyökalujen ominaisuuden hallitseminen auttaa lähinnä vianetsinnässä ja toiminnallisuuksien koodaamisessa. [12, s.150; 32.]

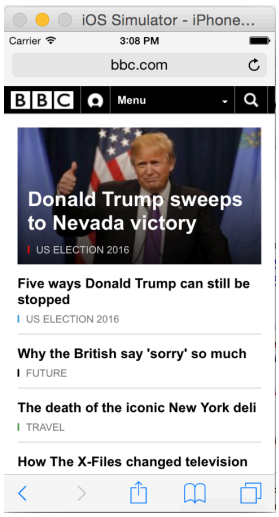
Emulointi tarkoittaa Chromen tapauksessa laitteiden imitointia, sillä selain vain kopioi muiden laitteiden käyttäjäagentin ja näyttöresoluutioiden arvoja. Imitoiminen ei anna luotettavia tuloksia laitetestaamisesta, sillä Chromen selainmoottori ei muutu [33]. Kuvassa 3 nähdään, että Chrome kaventaa selainnäyttöikkunan samaan resoluutioon ja pikselitiheyteen kuin Iphone 5 -älypuhelimessa. Chromen emulointivaihtoehdot sisältävät paljon eri laitevalmistajien näyttöresoluutioita.




Kuva 3. Chrome-selaimen emulaattorinäkymä Iphone 5 -älypuhelimelle.

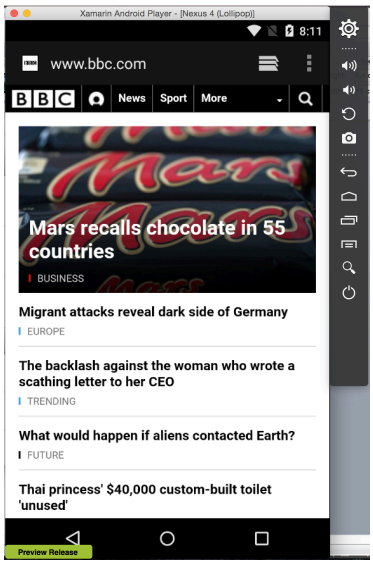
Myös Firefox-selaimella voidaan tarkastella samoin yleisimpiä selainikkunan resoluutioita. Firefoxilla ei voida kuitenkaan tarkastella sivujen käyttäytymistä eri selaimien tai laitteiden agentilla niin kuin Chromella. [34.]

Apple tarjoaa omaa simulaattoriaan Xcode-kehitystyökalupaketin yhteydessä. Sovelluksen nimi on iOS simulator. Simulaattori on alkuperäisesti tarkoitettu iOS-sovellusten kehittämiseen, mutta siinä on Mobile Safari -selain, jota voidaan käyttää verkkosivujen testaamiseen. Kuva 4 sisältää kuvankaappauksen simulaattorin mobiiliselaimesta. Simulaattorin hyvänä puolena on, että sen avulla voidaan simuloida Applen uusimpia laitteita, jolloin yksi simulaattori riittää testaamaan Ipad-tablettien ja Iphone-älypuhelimien eri versiot. Simulaattorin Mobile Safari -selaimelle voi tehdä vianmääritystä Safarin työpöytäversion avulla. iOS simulator -sovellus on saatavilla vain OS X -käyttöjärjestelmälle, joten Windows-käyttöjärjestelmän käyttäjä joutuu keksimään toisen tavan testata responsiivisia verkkosivuja Applen tuotteilla. [34.]




Kuva 4. Applen iOS-simulaattori.

Andoid-älypuhelinsimulaattoreita on olemassa Windowsille ja OS X:lle. Simulaattoreita voidaan käyttää testaamaan yhteensopivuutta ainakin Android Browser -selaimen kanssa. Simulaattorit luovat Androdille sopivan laiteympäristön, jonka avulla voidaan suorittaa Android-sovelluksia. Xamarin Android Player on virtuaalisointiin perustuva sovellus, jonka avulla voidaan ladata eri versioita Androidin käyttöjärjestelmästä. Simulaattorisovellus tarjoaa LG Nexus -älypuhelimista eri versioita. Ladattavissa olevia versioita voidaan kategorisoida Androidin version ja näyttöresoluution mukaan. Virtuaalisia laatikoita on olemassa 4-, 7- ja 10-tuumaisille puhelimille. Tarjolla olevat virtuaalilaitteiden näyttöresoluutiot ovat 480 x 800, 768 x 1280, 800 x 1280 ja 1080 x 1920. Kaksi viimeistä ja suurinta resoluutiota ovat tabletteja tai 5 tuuman teräväpiirtonäytön sisältävän älypuhelimia. [35.]

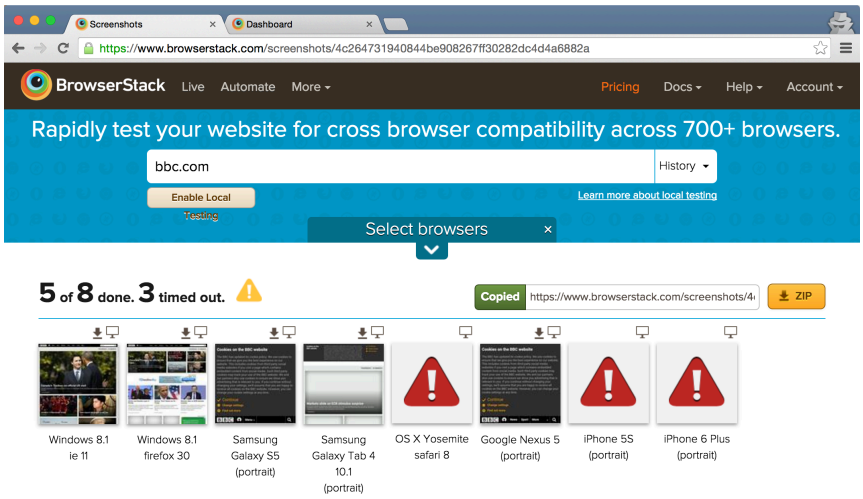



Kuva 5. Xamarin Android -simulaattori.

Androidille on olemassa myös virallinen simulaattori, joka on osa Android SDK -ohjelmistokehityspakettia. Simulaattorin käyttö vaatii koko kehitystyökalupaketin lataamista asennettavalle Windows- ja OS X -pohjaisille laitteille. Simulaattori on erittäin monipuolinen, ja sillä voidaan simuloida huomattava määrä eri laiteyhdistelmiä. Valittavia asetuksia simulaattorille ovat näytön fyysinen koko ja resoluutio, Android-versio, prosessoriarkkitehtuuri ja RAM-muistin koko. [34, s. 43.]

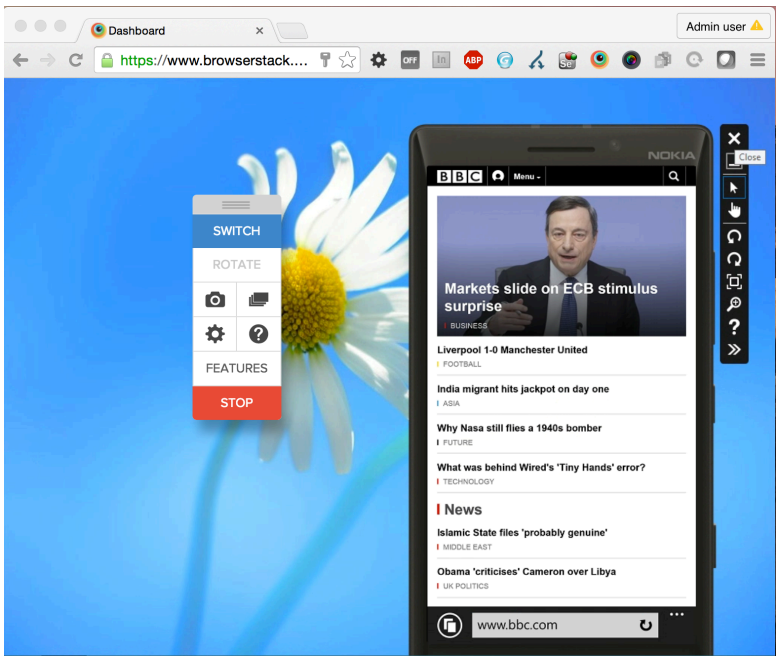
Microsoftin Windows Phonen eri versioille on olemassa Microsoftin emulaattoreita. Emulaattorin käyttö vaatii tarkkaa laitevaatimusten tukea ja Windows-käyttöjärjestelmän. Emulaattori on käytettävissä ilmaisen Visual Studio Community -tuotteen kanssa. Helpompi vaihtoehto Windows Phonen -selaimen testaamiseen on käyttää virtualisointipalveluita tai alkuperäistä laitetta. Virtualisointipalvelut tekevät pyydetystä laitteesta virtuaalisen version, jonka käyttö on mahdollista verkossa. [36.]

Yksi tällainen laitetestauspalvelu on Browserstack, joka ottaa kuvankaappauksia vaadituilla laitteilla halutusta sivusta. Kuvankaappaukset koostavat sivun koko pituuden yhteen kuvatiedostoon, jota voi tarkastella Browserstackin palvelussa. Esimerkki kuvankaappausgalleriasta kuvassa 6. Browserstackin kuvankaappaustyökaluun on olemassa rajapinta, jonka avulla laitepilven käyttöä voi automatisoida. [34, s. 56.]




Kuva 6. Browserstack.com-pilvipalvelun koostama kuvankaappausgalleria.

Testaamista voisi sanoa vain hyväksymis- ja yhteensopivuustestaamiseksi, sillä kuvatiedostot eivät kerro paljoa toiminnallisuuksien yhteensopivuudesta. Browsertackin pilvipalvelussa on olemassa myös reaaliaikainen käyttömahdollisuus, jolla voidaan varmistua toiminnallisuuksien yhteensopivuudesta.



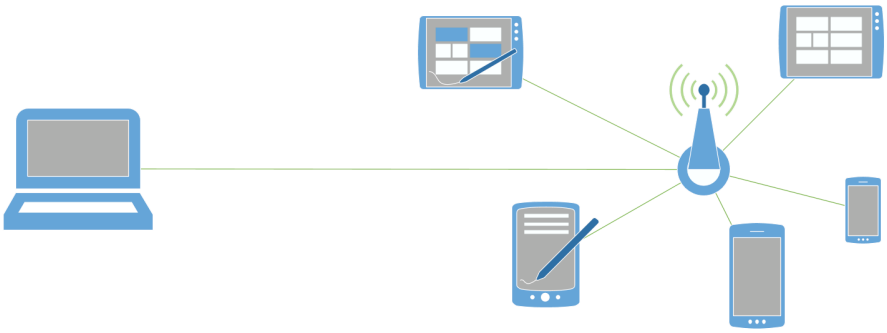

Kuva 7. Browserstack.com-pilvipalvelun reaaliaikainen käyttö selaimen avulla.

Laitteiden reaaliaikainen käyttö on epäkäytännöllistä, sillä testilaite on virtuaalilaatikko virtuaalilaatikon sisällä. Tätä yhdistelmää käytetään reaaliaikaisen verkkoyhteyden avul-

la, minkä takia laitteen näyttöä optimoidaan raskaasti. Verkkoyhteyden optimoinnin seurauksena kuvat ja fontit ovat pikselöityneitä.

## 3.2 Laitelaboratorio

Avoimien laitelaboratorioiden hyödyntäminen on vaihtoehto kalliiden laitteiden hankkimiselle. Ne ovat tiloja, jotka on sijoitettu teknologiayrityksien yhteyteen. Laitelaboratoriot mahdollistavat pääsyn alkuperäisiin laitteisiin ilmaiseksi. Yhteisön tarkoitus on saada mahdollisimman monipuolisia älypuhelimia ja tabletteja yhteen tilaan, jolloin tilasta syntyy avoin laitelaboratorio. Laitelaboratorioiden ideologia on tukea yksittäisiä paikallisia verkkokehittäjä, joilla ei ole resursseja hankkia laajaa eri laitteiden kirjoa. Laitelaboratorioille voi myös lahjoittaa omia laitteitaan, mutta usein paikalliset laitteiden maahantuojat lahjoittavat uusimpia puhelimiaan avoimille laitelaboratorioille. Laitelaboratorio sisältää yleensä suuren määrän älypuhelimia telineissä ja kytkettynä laturiin. Kuvio 6 havainnollistaa, miten kaikki laitteet ovat kytkettynä samaan langattomaan verkkoyhteyspisteeseen, mikä mahdollistaa paikallisen verkkoyhteyden muodostamisen laitteiden kesken. [37.]



Kuvio 6. Laitelaboratorion verkkokaavio.

Paikallisen verkkoyhteyden avulla laitteiden verkkoselaimia voidaan synkronoida keskenään. Mobiiliselaimien synkronoimisella vältytään URL-osoitteiden käsin syöttämiseltä. Itse synkronoimisen toteuttamiseen tarvitaan lähiverkkoon liitetty palvelin, joka lähettää omasta toiminnastaan käskyjä muiden laitteiden selaimille. Paikallisen selainsynkroinointipalvelimen muodostamiseen on olemassa kaupallisia ja ilmaisia vapaaseen lähdekoodiin perustuvia ratkaisuja. Yksi kaupallinen selaimen synkronointisovellus on Ghostlab, jota käsitellään luvussa 3.3. Ghostlabin kehittäjäyritys Vanamco myy myös pienempiä

laitelaboratorioita, joita on tarkoitus pitää oman työpisteen vieressä. Pienempi laitelaboratorio sisältää 2–3 laitetta ja telineen, joka pitää laitteet pystyssä. Laitelaboratorio on tällöin käytössä koodin kirjoitusvaiheessa, jolloin samalla huolehditaan testaamisesta kehityksen aikana. Tapaa kutsutaan kehitystestaamiseksi, jolloin yhteensopivuus- ja toiminnallisuustestaaminen hoidetaan kehitysvaiheessa eikä vasta kehittämisen jälkeen testausvaiheessa.

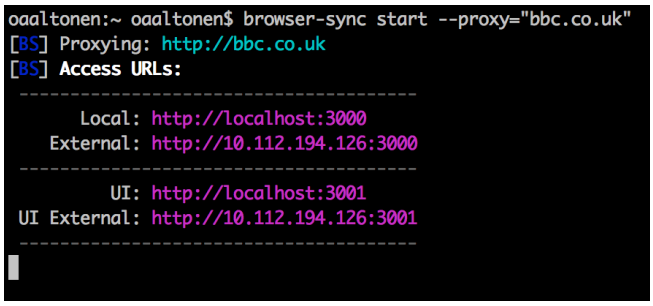
## 3.3 Synkronoidut selaimet

Synkronoituja selaimia käytetään laitelaboratoriossa ja kehittämisen yhteydessä. Ne nopeuttavat useiden selaimien käyttöä, kun niiden päätoimintoja voidaan synkronoida usealla selaimella. Selaimet seuraavat yhden pääselaimen toimintoja, tai ne synkronoivat kaikkia selaimia yhtäaikaisesti, jolloin minkä tahansa selaimen toiminnot peilataan toisiin selaimiin reaaliajassa. Synkronoiduilla selaimilla voidaan käydä kerralla läpi useita tapauksia. Näin säästyy aikaa verrattuna yksittäisten laitteiden käyttöön.

### Browsersync.io NPM-paketti

Browsersync on avoimen lähdekoodin NPM-paketti. Se on yhdysvaltalaisen JH-ohjelmistoyrityksen kehittämä ilmainen työkalu, joka peilaa selaimessa tapahtuvia interaktioita muihin päätelaitteisiin tai selaimiin. Tämän työkalun käyttö vaatii komentorivityöskentelyn hallitsemisen, sillä NPM-pakettia kontrolloidaan komentoriviltä. Komentorivityöskentely soveltuu hyvin verkkokehittäjille, sillä monia kehitystyökaluja käytetään komentoriviltä käsin. Toisin sanoen, muut kuin verkkokehittäjät eivät välttämättä osaa käyttää sitä.

Kuvassa 8 paikallinen palvelin käynnistetään komennolla ”browsersync start –proxy=’http://bbc.com’ ”. Tällöin Browsersync-työkalu luo paikallisen palvelimen porttiin 3000, joka peilaa proxy-valinnan sivua yhdistettyihin päätelaitteisiin.




Kuva 8. Browsersyncin komentorivinäkymä.

Kaikki selaimet, jotka ohjataan mainittuihin osoitteisiin, peilataan proxy-valinnalla määrättyyn osoitteeseen. Browsersync-työkalu kontrolloi päätelaitteiden selaimia JavaScriptillä; erityisesti se simuloi hiiren ja kosketusnäytön toimintoja muihin selaimiin. Esimerkiksi kun useita päätelaitteita tai selaimia on yhdistetty Browsersync-palvelimeen ja yhtä pääteselaimen sivua vieritetään alaspäin, kaikki muutkin päätelaitteiden selaimen reagoivat rullaamiseen samalla tavoin. Browsersync-työkalu peilaa elementtien klikkauksia ja lomakkeiden täyttöjä. Browsersync-työkaluun on liitetty Weinre-vianmääritystyökalu, jonka avulla päätelaitteiden toistamaa lähdekoodia voidaan tarkastella. Weinre on merkittävä apu toistamisvirheiden korjaamisessa, sillä kehittäjän ei tarvitse arvata, miksi päätelaite ei näytä elementtiä kuten muut selaimet. Weinren avulla voidaan tarkastella päätelaitteen toistamaa lähdekoodia kehittäjän omalta tietokoneelta. [38.]

Browsersync-työkalun käyttö monipuolistuu huomattavasti, jos sitä ei tarvitse käyttää komentoriviltä. URL-osoitteiden peilaus onnistuu vain lisäämällä proxy-valinta ohjelman käynnistyskomentoon. Proxy-valinta tulisi olla valittavissa graafisen käyttöliittymän kautta dynaamisesti. Tällöin myös käyttäjät, jotka eivät hallitse komentorivityöskentelyä, voisivat sujuvammin käyttää pakettia. Githubin yhteisön palautekeskusteluiden mukaan seuraavassa Browsersyncin versiossa 3 peilattu URL-osoite voidaan valita dynaamisesti graafisen käyttöliittymän avulla. Tällöin Browsersynciä voitaisiin käyttää myös laitelaboratorion yhteydessä. [39.]

Ghostlab-sovellus

Ghostlab on Browsersync-työkalun tapainen kaupallinen versio. GhostLab on maksullinen OS X -sovellus, joka tarjoaa samat toiminnallisuudet kuin Browsersync-työkalu. Ghostlab-sovelluksen käyttö ei tarvitse komentoriviä toimiakseen, vaan se voidaan ladata, asentaa graafisen käyttöliittymän avulla. Tällöin sitä voidaan myös hallita sovelluksen

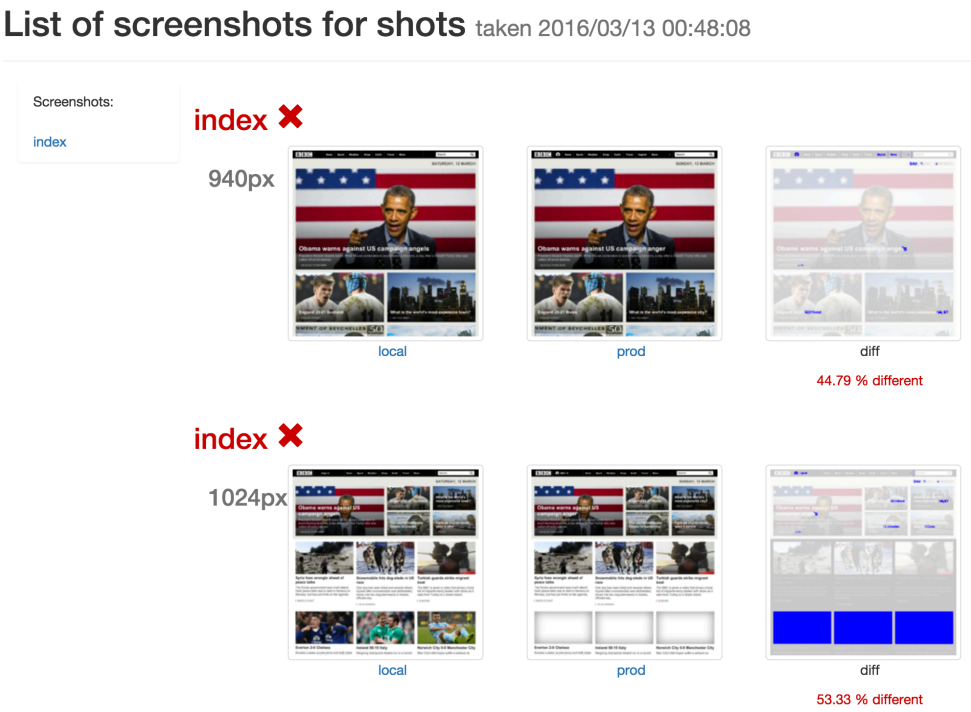
kautta. Ghostlab peilaa haluttua URL-osoitetta muihin samaan langattomaan verkkoon kytkettyihin laitteisiin samoin kuin Browsersync. Ghostlabissä on myös Weinre-vianmääritystyökalu, joka mahdollistaa minkä tahansa selaimen lähdekoodin tarkastelun.

## 3.4 Kuvavertailut

Kuvavertailuja käytetään regressiotestaamisessa kahden verkkosivuston version testaamiseen. Wraith on responsiivinen kuvankaappaustyökalu, jonka ajatus on verrata sivuston tuotanto- ja kehitysversiota keskenään. Vertailussa yhdistetään kaksi kuvankaappausta molemmista verkkosivun versioista yhteen kuvaan. Testi on läpäisty, jos kuvankaappauksista ei löydy eroavaisuuksia. Wraith keskittyy enemmän ulkoasun mukautumiseen määrätyille näyttöresoluutioille kuin yhteensopivuustestaamiseen. Ohjelmalle määritetään testattavien selainikkunoiden resoluutiot ja URL-osoitteet, joista ohjelma ottaa kuvankaappaukset valituista sivuista ja luo niistä vertailugallerian riippumatta testin onnistumisesta. [40.]

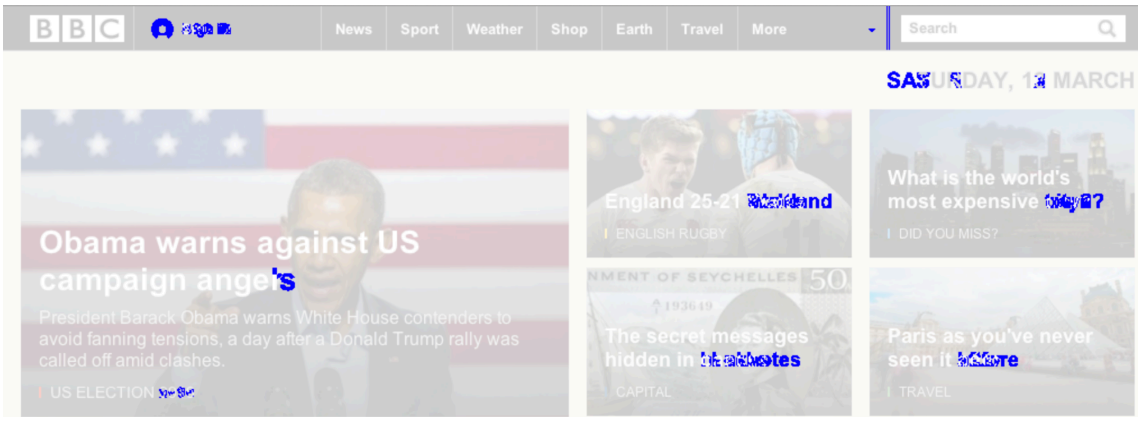
Wraith on alkujaan BBC:n kehittämä testityökalu, joka on jaettu avoimella lähdekoodin lisenssillä. Wraith asennetaan Ruby Gem –asennuspakettina, ja se vaatii toimiakseen vähintään PhantomJS-, CasperJS- tai SlimerJS-JavaScript-selaimen asennuksen. Nämä JavaScript-selaimet ovat selaimia ilman graafista käyttöliittymää, eli niitä käytetään komentorivin tai skriptien avulla. PhantomJS käyttää Safari- ja Chrome-selaimissa olevaa WebKit-selainmoottoria. Toinen vaihtoehto PhantomJS:lle on SlimmerJS, jonka käyttämä selainmoottori on Firefox-selaimessa käytössä oleva Gecko.

Kuvan 9 kuvavertailugalleria on luotu vertailemalla paikallisesti tallennettua verkkosivun kopiota BBC:n kotisivuun. Paikallisesta verkkosivun kopioista puuttuu myös muutama kuvake ja sivuston sisältöä on muutettu, mikä Wraithin tekemä kuvavertailutiedosto nostaa esille kuvassa 10.




Kuva 9. Wraithin luoma kuvankaappausgalleria.

Kuvavertailuissa on myös joitakin huonoja puolia. Wraith ottaa huomioon kaikki muutokset. Se ei osaa tunnistaa tekstiä taustasta, jos sisältö muuttuu kahden palvelimen välillä. Regressiotesti epäonnistuu tekstin pituuden ja sijainnin muuttuessa selainikkunassa [41].



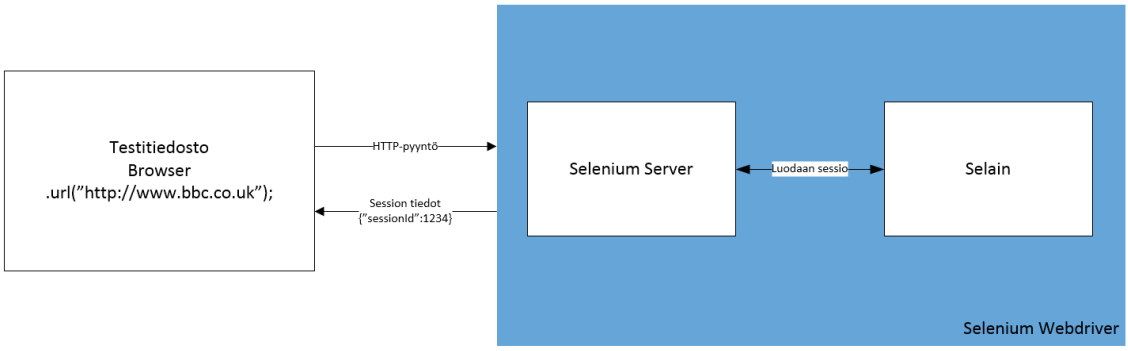

Kuva 10. Rajaus Wraithin tekemästä kuvavertailutiedostosta.

Käytännössä tämä tarkoittaa, että kahden vertailtavan sivun sisällöt on pidettävä synkronoituna aina testiä suorittaessa. Helpoiten tämä onnistuu verkkosivujen tietokantojen synkronoimisella ennen testin suorittamista. Sisällön muutoksien havaitsemista voi pitää

myös positiivisena ongelmana, sillä Wraitin avulla sivuston kaikki visuaaliset muutokset tulevat esille. Muutokset voisivat jäädä helposti huomaamatta, sillä verkkosivujen ulkoasun ja sisällön päivityksiä ei välttämättä dokumentoida tarkasti. Kuvavertailuilla ei myöskään voi korvata animaatioiden testaamista käsin, sillä vertailtaviin kuvagallerioihin ei voi tallentaa animaatioita tarkasti. Wraithilla ei voi voida myöskään tallentaa sivun lataamisen aikaisia tapahtumia. Parhaimmillaan Wraith on, kun halutaan varmistua, ettei julkiselle sivustolle aiheuteta tahattomia virheitä uuden koodiin käyttöönotossa palvelimella. Se soveltuu erityisesti laajoille verkkosivuille, joiden alasivujen läpikäyminen on hidasta ja tehotonta. [42.]

## 3.5 Automatisoidut testauskehikot

Selenium on selainautomaatiokirjasto, jonka avulla käyttäjän toimia selaimella voidaan kopioida ja toistaa koneellisesti. Selenium itsessään ei sisällä mitään testauskehikkoa, mutta sen avulla voidaan kirjoittaa testejä usealla ohjelmointikielellä. Se sisältää Java-pohjaisia työkaluja, jotka käyttävät selaimia suorittamaan testiin sisällytetyt toiminnallisuudet. Seleniumin mukana tulee ajuri Firefox-selaimelle, mutta sen tukea eri alustoille voidaan laajentaa asentamalla omat ajurit halutuille päätelaitteille. Testi tulkataan laiteajurin avulla alkuperäisen laitteen ymmärtämään muotoon, jolloin samoja testejä voidaan suorittaa useammalla eri laitteella. Yksi Selenium-pohjainen työkalu on Selenium Server, joka toimii Webdriverin ja selaimen välissä. Selenium Server tarjoaa ohjelmointirajapinnan, jonka avulla koneellisia komentoja voidaan välittää selaimelle. Se tulkkaa koodattuja komentoja selaimelle ja palauttaa pyydetyt arvot. [43; 44.] Kuvio 7 havainnollistaa miten Seleniumin komponentit keskustelevat keskenään.




Kuvio 7. Selenium Webdriver [44].

Selenium Webdriverin rajapinta mahdollistaa useiden ohjelmointikielien käytön. Se tukee Javaa, C#:a, Pythonia, Rubya, Perlia ja JavaScriptiä. Esimerkiksi JavaScript-versio on rakennettu NodeJS-paketiksi, joten se on helppo integroida osaksi muita NodeJS-pohjaisia testejä. Rajapinnan monipuolisuus on käynnistänyt useita omia testikehikko-projekteja, jotka on tarkoitettu eri ohjelmistotestien suorittamisen. Useiden testikehikkojen olemassaolo mahdollistaa samojen testien ajamisen testikehikkojen omilla synkseilla. [45.] Koodiesimerkki 7 on NightwatchJS-testikehikolla kirjoitettu Selenium-testi, joka syöttää Googlen hakukenttään sanat ”BBC UK”. Testi myös tarkistaa, sisältävätkö Googlen hakutulokset oikean sanayhdistelmän, jota odotetaan oikeaksi hakutulokseksi. NightwatchJS-testikehikko mahdollistaa CSS-valitsimien käytön HTML-elementtien kohdentamiseksi.

```
this.demoTestGoogle = function (browser) {
  browser
    .url('http://www.google.com')
    .waitForElementVisible('body', 1000)
    .setValue('input[type=text]', 'BBC UK')
    .waitForElementVisible('button[name=btnG]', 1000)
    .click('button[name=btnG]')
    .pause(1000)
    .assert.containsText('#main', 'UK - BBC News')
    .end();
};
```

Koodiesimerkki 7. NightwatchJS-testikehikon Selenium-syntaksi.

Koodiesimerkki 8 suorittaa saman Google-haun kuin koodiesimerkki 7, mutta se tarkistaa tulossivun otsikon. Vertailemalla koodiesimerkkejä 7 ja 8 voidaan havaita, että Selenium-testejä voidaan suorittaa monella eri tavalla ja monella eri testikehikolla.

```
var webdriver = require('selenium-webdriver'),
    By = webdriver.By,
    until = webdriver.until;

var driver = new webdriver.Builder()
    .forBrowser('firefox')
    .build();

driver.get('http://www.google.com/ncr');
driver.findElement(By.name('q')).sendKeys('BBC UK');
driver.findElement(By.name('btnG')).click();
driver.findElement(By.name('a')).click();
driver.wait(until.titleIs('BBC UK - Google Search'), 1000);
driver.quit();
```

Koodiesimerkki 8. Selenium Webdriverille kirjoitettu testi.

Seleniumia hyödyntäviä kaupallisia pilvipalveluita on paljon. Kaupalliset pilvipalvelut on pienille yrityksille ja yksittäisille kehittäjille edullinen vaihtoehto huolehtia verkkosivujen yhteensopivuudesta. Browserstack tarjoaa myös rajapintoja, joita voidaan käyttää yhteensopivuustestien ja toiminnallisuustestien suorittamiseen koneellisesti eri selain- ja käyttöjärjestelmäyhdistelmillä. Toinen tapa hyödyntää Browserstackin palveluita on tehdä toiminnallisuustestejä Selenium Webdriverin avulla. Automaattisia Selenium-testejä voidaan suorittaa ottamalla yhteys Browserstackin rajapintaan NodeJS-sovelluksella. Suoritetusta Selenium-testistä koostetaan video, johon testeihin koodatut toiminnot on tallennettu. Browserstackin rajapinta ymmärtää samanlaista koodisyntaksia kuin koodiesimerkit 7 ja 8, mutta selaimeksi määritetään vain Browserstackin laitepilvi. Selaimen parametreiksi annetaan haluttu laite- ja selainyhdistelmä. Testi lähetetään jonoon Browserstackin laitepilveen, ja sen tulokset ovat näkyvillä Browserstackin sivulla. [30; 46. ]

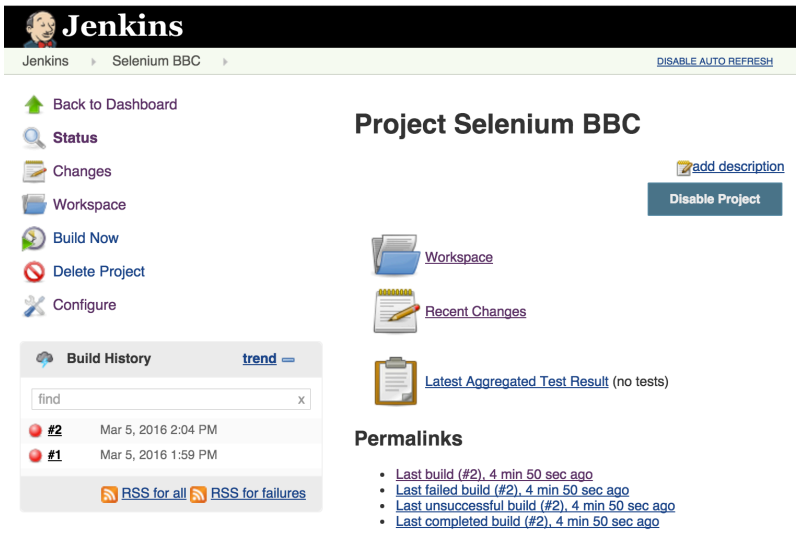
Jenkins

Jenkins on jatkuvan integraation sovellus, jonka avulla voidaan automatisoida testaustehtäviä ja työvaiheita. Se on ratkaisu verkkokehittämisen työnkulkuun eikä testaamistyökalu, mutta Jenkinsin avulla huolehditaan testien säännöllisestä suorittamisesta. Jenkinsille määritettyjä tehtäviä voidaan laukaista rajapintojen avulla tai suoraan sen omasta käyttöliittymästä käsin, jolloin voidaan välttää komentorivin käyttöä ja nopeuttaa käsin suoritettavien ylläpitotehtävien tekemistä. [47.]

Jatkuvan integraation käyttö testaamisessa pakottaa testien suorittamisen ennen koodin käyttöönottoa. Työstettäessä laajoja verkkosivuja koodimuutoksien vaikutuksia muihin osioihin on hankala ennustaa ilman huolellista regressiotestaamista. Sisällön tarkastelun avuksi voidaan käyttää regressiotestejä, joita käyttäessä muutosten tarkkailu tapahtuu koneellisesti. Koneellistesti tapahtuvan testien suorittamisen tarkkuus on parempi kuin ihmisen tekemä, sillä työntekijöiden suorittama testaus on aina subjektiivista. Regressiotestien koneellistaminen tuo itsevarmuutta myös kehittäjälle oman koodinsa suoriutumisesta, kun sen laajempia vaikutuksia voidaan arvioida Jenkinsillä suoritetuilla testeillä.

Jatkuvan integraation käyttöönottoon tarvitaan testipalvelin, jolla Jenkisiä ajetaan. Palvelimella on tarvittavat testitiedostot, joiden suorittaminen pitää määritellä omiksi Jenskinstehtävikseen. Suoritetut testit voivat olla mitä tahansa komentoriviltä suoritettavia tehtäviä. Hyviä esimerkkejä komentoriviltä suoritetuista testeistä ovat NodeJS:llä koodatut

Selenium-testit ja Wraith-vertailutestit. Kuvassa 11 on kuvankaappaus Jenkinsin käyttöliittymästä, josta voidaan havaitaan viimeisimpien tehtävien ajaminen on epäonnistunut.




Kuva 11. Jenkins-sovelluksen käyttöliittymä

Tehtäviä voidaan suorittaa tietyin väliajoin, käsin tai versionhallintatyökalujen yhteydessä. Tällöin kehittäjän itse ei tarvitse olla vastuussa testien ajamisesta vaan ne käynnistyvät automaattisesti. Jenkins luo lokitiedoston suoritetuista komennoista, joiden perusteella suoritetut testit tulkitaan onnistuneiksi tai epäonnistuneiksi.

Jenkins voidaan määrittää suorittamaan seuraavia testivaiheita:

- yhteensopivuustestaus
- suorituskykytestaus
- yksikkötestaus
- regressiotestaus
- käyttäjäpolkutestaus.

Suoritetuista testeistä koostettu raportti voidaan lähettää eteenpäin sähköpostitse tiketti-järjestelmään tai toiselle osapuolelle, joka huolehtii sivun ylläpidosta.

# 4 Testauskäytännöt Mirum Agency -mainostoimistossa

Mirum on maailmanlaajuinen mainostoimistoverkosto, joka tuottaa digitaalisia markkinointipalveluita, jotka perustuvat innovaatioihin, suunnitteluun, tietoon ja tekniikkaan. Mirumilla on yhteensä noin 2 200 työntekijää maailmanlaajuisesti. Toimistoja on yhteensä 46. Helsingissä sijaitsevassa toimistossa työskentelee noin 120 henkilöä. Toimiston henkilöstöön kuuluu projektipäälliköitä, asiakkuuksien hoitajia, copy writereita, graafisia suunnittelijoita ja verkkokehittäjiä. Helsingin toimisto tuottaa verkkokehityspalveluita, suunnittelupalveluita ja muita digitaalisia markkinointipalveluita. Mirum Helsingin sisäisesti tarjoamat palvelut koostuvat verkkosivujen konseptoinnista sekä palvelu- ja visuaalisesta suunnittelusta. Asiakkaisiin kuuluvat Finavia, Finnair, Microsoft, Nestlé Ice Cream, Nokia, RAY, Sako ja Sanoma. [48; 49; 50; 51]

Palveluiden teknisestä toteutuksesta ja testaamisesta vastaavat toimiston verkkokehittäjät, joita Helsingin toimistossa on noin 17 henkilöä. Heitä ei ole jaettu erikseen front-end- ja backend-osaajiin, vaan jokainen verkkokehittäjä työskentelee ulkoasujen ja backendiä hallitsevien sisällönhallintajärjestelmien kanssa. Verkkokehittäjät on jaoteltu työkokemuksensa mukaan projekteihin. Helsingin toimiston käyttämät teknologiat perustuvat avoimen lähdekoodin ratkaisuihin, joissa erityisesti Wordpress- ja Drupal-sisällönhallintajärjestelmät ovat yrityksen erityisosaamista. Näitä sisällönhallintajärjestelmiä käyttämällä saadaan kustannustehokkaasti ja nopeasti aikaan laadukkaita sivuja, joita on helppo kehittää edelleen. [52.]

Mirumin tuottama tekninen laatu heijastuu asiakkuuksien imagoon välittömästi, siksi on tärkeää huolehtia verkkosivujen huolellisesta testaamisesta ennen niiden julkaisemista. Mirumilla on jo käytössä tapoja verkkosivujen testaamiseksi kehitystyön aikana, mutta yhtenäinen testaustapa verkkokehittäjien keskuudessa puuttuu. Tavoitteena on kartoittaa testaamista tällä hetkellä ja sitä, minkälaisissa tilanteissa tarvitaan enemmän käytäntöjä, jotta tuotettu tekninen laatu pysyy yhtenäisenä. Verkkokehittäjät ovat toistaiseksi hoitaneet testauksen. Tämän takia he tietävät parhaiten, minkälaista laatua Mirum tuottaa verkkokehitysprojekteissa. Mielestäni heidän mielipiteensä kertoo, mihin laadunhallinnassa tulisi kiinnittää enemmän huomiota. Parhaat ideat testaamiseen löytyvät niiltä henkilöiltä, jotka ovat tekemisissä vikojen ja virheiden kanssa jatkuvasti. He ovat tietoisia mahdollisista virhetyypeistä, ja siksi heillä on paras tietotaito siihen, miten testausta tulisia suorittaa.

Mirumin verkkokehittäjät eivät kuitenkaan ole ohjelmistokehityksen laadunhallinnan ammattilaisia, vaikka he hoitavat projektien testausta. Koska he hoitavat Mirumin laadunvalvontaa, he ovat myös halukkaita kokeilemaan ja tutkimaan testausmetodeja, jotka voivat auttaa heidän vastuunsa hoitamisessa. Tästä syystä heillä on taipumus tutkia uusia testaustyökaluja, jotka voisivat nopeuttaa työtä. Mirumilla uusien kehikoiden tai testaustyökalujen käyttöönotto on kuitenkaan erityisen hidasta. Vaikka verkkokehittäjä olisi käyttänyt jotain omaa testaustapaansa projektin aikana, tieto parhaista käytännöistä ei leviä muille verkkokehittäjille. Tämän takia Mirumin tuottama laatu vaihtelee huomattavasti. Mirumin tekemät projektit ovat lähes aina asiakasprojekteja, jolloin uusien työkalujen käyttöönotto on aina riippuvainen asiakkaiden tarpeista. Toistaiseksi Mirumin asiakkaat eivät ole vaatineet laadultaan moitteettomia ja laajasti testattuja projekteja, joten Mirumin testaustapoja ei ole yritetty tehostaa.

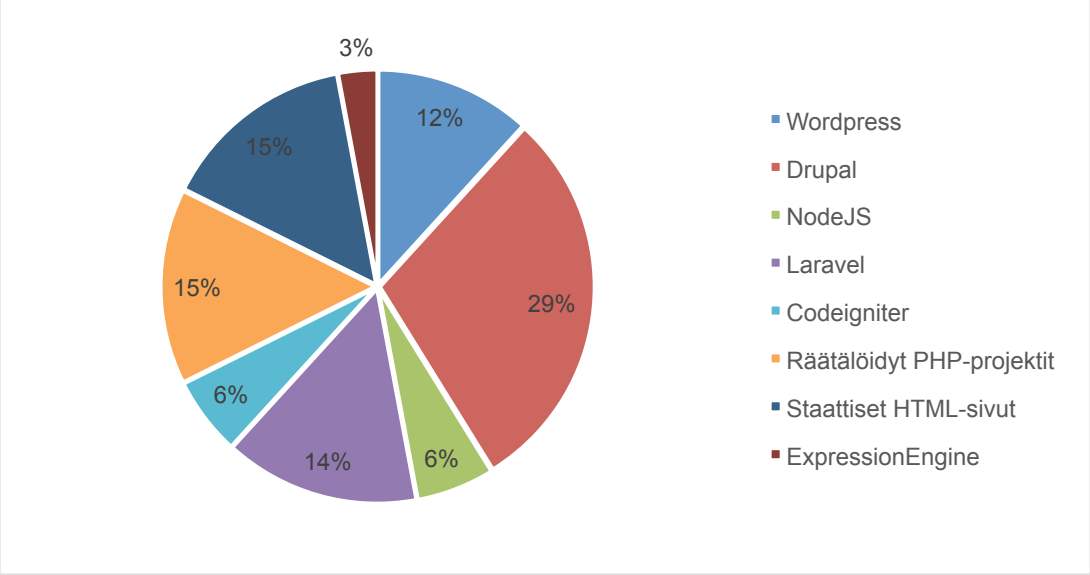
Insinöörintyön osana tehtiin kysely, jonka tarkoituksena oli saada selville vanhat testauskäytännöt, virhealtteimmat laitteet, epäkohdat projektin hallinnassa sekä suunta, johon verkkokehittäjät itse tahtoisivat viedä testausvaihetta projekteissa. Kysely luotiin englanniksi Google Forms -työkalun avulla verkkoon, ja se lähetettiin verkkokehittäjille sähköpostitse vuoden 2015 joulukuun lopulla. Viimeiset vastaukset kyselyyn saatiin tammikuun 2016 alussa, johon mennessä kyselyyn oli tullut vastauksia 13. Silloin yrityksessä oli 17 verkkokehittäjää, joten vastausprosentti oli kohtuullinen. Seuraavissa luvuissa käydään läpi tietoja kehittäjien käyttämistä työkalusta ja koodauskehikoista. Kysely oli jaettu 6 osa-alueeseen: kehitysympäristöt, resursointi, mobiililaitteet, laitelaboratorion käyttöaste ja testaustyökalujen jakaminen. Tarkemmat kysymykset ja niiden asettelut ovat liitteessä 2. Tarkat vastaukset kyselyyn ovat liitteessä 1.

## 4.1 Käytössä olevat verkkokehitys-ympäristöt

Suurimassa osassa Mirumin tekemistä verkkokehitysprojekteista käytetään Git-versionhallintatyökaluja. Drupal-projektit hyödyntävät Drupal-verkkohotellipalvelu Acquian tarjoamia versionhallintyökaluja, joissa on paljon Drupal-keskeisiä työkaluja helpottamaan sivujen ylläpitoa. Esimerkiksi tuotannossa olevien verkkosivustojen ylläpitotehtäviä voidaan suorittaa komentorivin kautta. Muut kuin Drupal-pohjaisien projektien lähdekoodi säilytetään omassa Git-isännöintipalvelussa.

Mirumin tuottamat verkkoprojektit on pääosin kirjoitettu PHP-ohjelmointikielellä. Verkkokehittäjien keskuudessa suostuin koodieditori on PhpStorm-editori, johon on saatavilla liitännäisiä monipuolisesti Drupal-, WordPress- ja Laravel-projekteihin. Drupal-pohjaisissa projekteissa käytetään Drupalin omia koodauskäytäntöjä. Tyylitiedostojen koodaamiseen käytetään yleensä CSS-koodin esikäsittelijöitä. Yleisin esikäsittelijä on Compass, joka on asennettu Drupalin räätälöitävään Omega-teemaan oletuksena. Compassin avulla tyylitiedostoja voidaan jakaa omiin alikansioihin, jolloin tyylimäärittelyiden rakenteen ymmärtäminen on helpompaa.

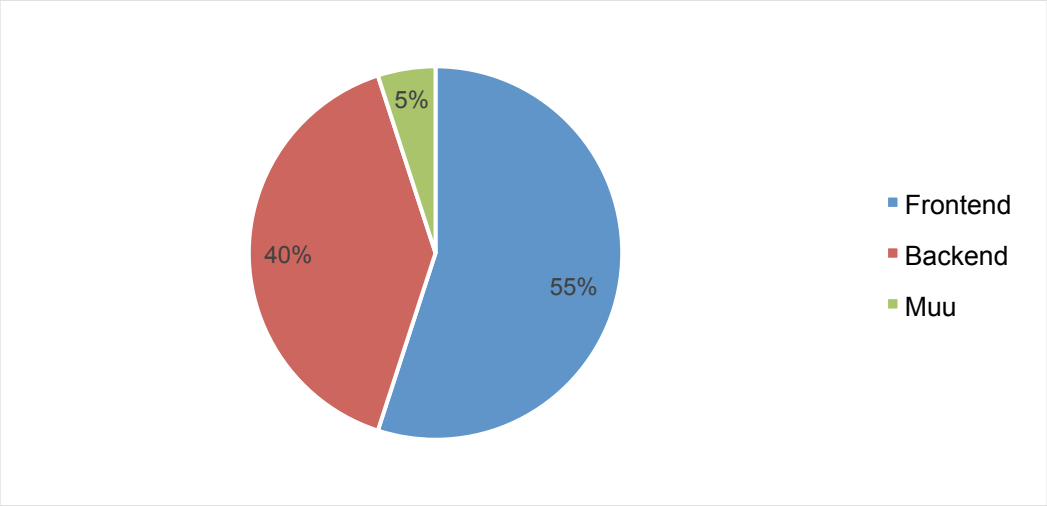
Kuviossa 8 näkyvät verkkokehittäjien käyttämät koodauskehikot ja sisällönhallintajärjestelmät. Kysymys oli muotoiltu: Mitä alustoja tai kehikkoja kehität? Vastauslomakkeessa oli mahdollista valita useampia alustoja kuin yksi, mikä selittää suuren vastausten määrän. Saatuja vastauksia kysymykseen oli yhteensä 13, ja niistä voidaan päätellä, että kolme yrityksen verkkokehittäjää ei käytä aktiivisesti Drupal-pohjaisia verkkosivustoja. Ylivoimaisesti käytetyin alusta on Drupal-sisällönhallintajärjestelmä. Seuraavaksi yleisimmät vastaukset ovat Laravel-kehikko, räätälöidyt PHP-projektit ja staattiset HTML-sivut. Kaikki mainitut saivat viisi vastausta. Kolmannella sijalla oli Wordpress-sisällönhallintajärjestelmä.




Kuvio 8. Mirumin verkkokehittäjien käyttämät alustat.

Kyselyssä saatiin myös selville, mihin teknologiaan verkkokehittäjät ovat enemmän orientoituneet. Front-end-vastausvaihtoehdon oli valinnut 11 vastaajaa ja backend-vastausvaihtoehtoon 8 vastaajaa. Kysymykseen oli mahdollista valita useampi kuin yksi

vaihtoehto, mikä selittää suuren valintojen määrän. Vastaajat ovat hieman enemmän orientoituneita front-end-kehittämiseen.




Kuvio 9. Kehittäjien osaamisen orientoituminen.

Mielestäni painotus johtuu sisällönhallintajärjestelmien käytöstä. Sisällönhallintajärjestelmää käytettäessä kaikkiin projekteihin ei tarvita backend-osaamista, sillä sisällönhallintajärjestelmä tarjoaa toiminnallisuuden toteuttamiseen valmiita ratkaisuita. Valmista liitännäistä käytettäessä räätälöintiä vaatii vain sivuston ulkoasu.

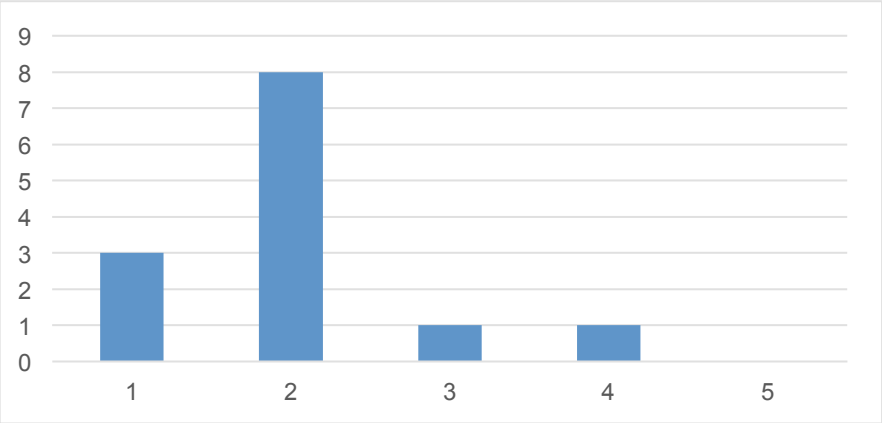
## 4.2 Nykyiset verkkosivujen testauskäytännöt

Kyselyn avulla saatiin myös selville, mitä testaustyökaluja verkkokehittäjät käyttävät tällä hetkellä. Saatujen vastauksien laajuus vaihtelee suuresti. Verkkokehittäjät käyttävät paljon Applen iOS-simulaattoria, sillä ainakin neljän vastaajaa ilmoitti käyttävänsä kyseistä simulaattoria yhteensopivuustestaamiseen. Yhteensä neljä verkkokehittäjää vastaa käyttävänsä Browserstack-pilvipalvelua projektien testaamisessa.

Testattujen laitteiden määrä suurimmillaan on 10 laitetta, mutta vastauksien joukosta löytyy myös 2–3 testattua laitetta projektia kohden. Vastaukset ovat erittäin monipuolisia, sillä niissä on eritelty Browserstack-pilvipalvelussa, alkuperäisenä ja simulaattorin avulla testatut laitteet. Muita käytössä olevia työkaluja ovat Xip.io, Wraith, laitelaboratorio, selaimen emulointityökalut. Käytettyjen työkalujen monipuolisuus ei ole suuri. Älypuhelimien pääasialliseksi testaustavaksi sanoisin Browserstack-pilvipalvelu ja iOS-simulaattorin.

Verkkosivujen Internet Explorerin -yhteensopivuustestaamisen pääasiallinen työkalu on Microsoftin tarjoamat virtuaalilaatikot, joita käyttämällä voidaan saada Internet Explorer -yhteensopivuus varmistettua.

Helsingin toimistossa olevassa laitelaboratoriossa on yhteensä 15 eri älypuhelinta ja tablettia. Kyselyssä haluttiin myös selvittää kyseisen laitelaboratorion käyttöastetta. Laitelaboratorion älypuhelimet ja tabletit ovat yhdistettynä omaan langattomaan verkkoyhteyspisteeseen. Laitelaboratorion yhteydessä on myös oma tietokone, johon on asennettu selaimen synkronointiohjelma GhostLab. Kuvioiden 10–14 pylväsdiagrammeissa vastaajien määrä esitetään aina Y-akselilla. Kuviosta 10 näkee, että kolme vastaajaa ei ole käyttänyt laitelaboratoriota koskaan. Enemmistö vastaajista on valinnut kohdan 2, mikä viestii laitelaboratorion huonosta käyttöasteesta.




Kuvio 10. Laitelaboratorion käyttöaste.

Vastaajat ovat kommentoineet kysymyksen yhteyteen, että laitelaboratoriota ylläpidetään huonosti. Kenttään on myös kommentoitu, että laboratoriossa olevia laitteita käytetään enemmän erikseen kuin GhostLab-sovelluksen avulla, sillä laitelaboratorion yhteydessä oleva tietokone ei ole käyttökunnossa. Laitteista puuttuu myös latureita, joilla saataisiin kaikkien laitteiden akut pysymään täytenä. Testaamista hidastaa huomattavasti, jos tarvittavan laitteen akku on tyhjä.

## 4.3 Yleisesti virheitä aiheuttavat älypuhelimet ja tabletit

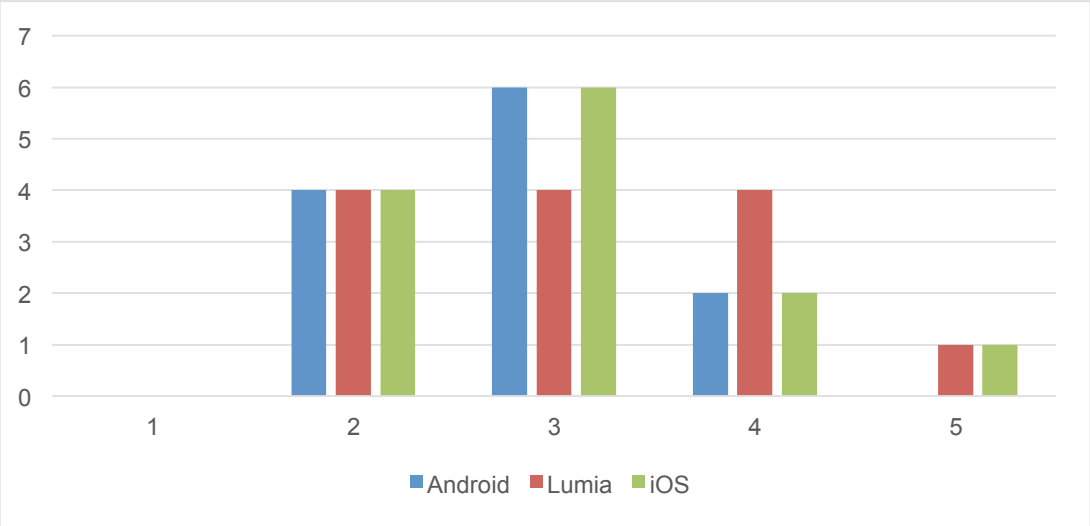
Mirumin selainkäytäntönä on, että se tukee tavallisempien selaimien viimeisintä versioita kahdella viimeisimmällä käyttöjärjestelmäversiolla. Päätuetut selaimet ovat Chrome, Firefox, Safari ja Internet Explorer.

Esimerkiksi Chromea tuetaan seuraavien käyttöjärjestelmien versiossa:

- OS X 10.9
- OS X 10.10
- Windows 7
- Windows 8
- iOS 7
- iOS 8
- Android 4.3
- Android 4.4.

Käyttöjärjestelmien kahta viimeisintä versiota voidaan vertailla vain Browserstack-pilvipalvelussa, jossa käyttöjärjestelmän versiota voi vaihtaa. Laitelaboratoriossa on myös olemassa vanhoja älypuhelimia ja tabletteja, joiden selaimia tai ohjelmistoja ei automaattisesti päivitetä. Tällöin on mahdollista testata verkkosivua myös alkuperäisellä, vanhemmalla ohjelmistoversiolla. Käytännössä laitteiden ohjelmisto- tai selainversioista ei ole olemassa mitään dokumenttia, josta voisi tarvittaessa tarkistaa olemassa olevien laitteiden versioita.

Kyselyssä kartoitettiin myös yleisempiä laitteita, jotka aiheuttavat yhteensopivuusvirheitä. Laitteissa painotettiin juuri älypuhelimien ja tablettien selaimia. Kysymykset aseteltiin älypuhelimien käyttöjärjestelmän mukaan. Vastausvaihtoehdot olivat asteikolla 0–5, jossa 0 tarkoittaa, ettei laite aiheuta koskaan virheitä ja 5 tarkoittaa jatkuvia löydettyjä virheitä. Kuvio 11 esittää, kuinka yleistä on löytää yhteensopivuusvirheitä eri laitteiden kesken. iOS- ja Lumia-laitteet ovat eniten virheitä aiheuttavia käyttöjärjestelmiä.




Kuvio 11. Android-, Lumia- ja iOS-laitteiden aiheuttamat yhteensopivuusvirheet.

Yleiset vastaajilta saadut kommentit koskevat Safari Mobile -selainta. Kommenttien mukaan selaimelle on hankala toteuttaa tiettyjä toiminnallisuuksia, jotka aiheuttavat vaikeasti korjattavia virheitä muissa selaimissa. Kommenteissa on myös mainittu Lumia-puhelimet, joiden käyttämän Internet Explorerin versio aiheuttaa virheitä.

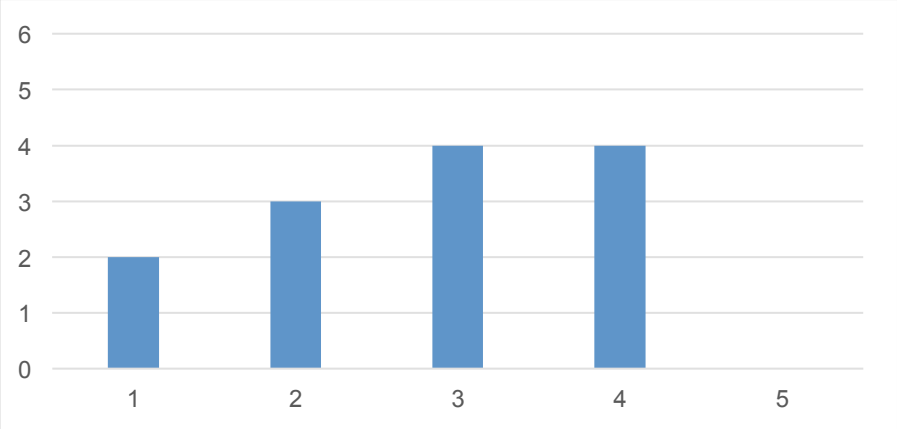
## 4.4 Projektinhallinnasta saatu palaute

Mirumin verkkokehittäjät ovat viime kädessä se taho, jotka ovat vastuussa kirjoittamansa koodin laadusta. Verkkokehittäjät ovat omissa projekteissa ne henkilöt, jotka testaavat omaa työtään jatkuvasti. Useissa projekteissa on vain yksi verkkokehittäjä tekemässä koodausta ja testausta, joten he ovat myös se taho, joka saa palautteen vioista ja virheistä. Mielestäni verkkokehittäjien itsensä antama palaute Mirumin tuottamasta laadusta ja työtavoista antaa realistisen kuvan siitä, kokevatko he nykyiset testaustyökalut tehokkaina.

Verkkokehittäjiltä kysyttiin, kuinka paljon aikaa he käyttävät testaamiseen projektia kohti. Vastauksien mittakaava oli noin 2–15 % projektiin käytetystä ajasta. Vastaustauksista kerätty keskiarvo on noin 6,3 %. Perinteisessä ohjelmistokehityksessä varataan testaamiseen 20 % kokonaiskehitysajasta. Roy Straussin ja Patrick Hoganin mukaan verkkokehitysprojekteissa testaamiseen varattu aika olisi jopa 33 % projektin kokonaisaikaan verrattuna [53, s. 35]. Vastauksien matala keskiarvo kertoo projektien laadusta sen, ettei

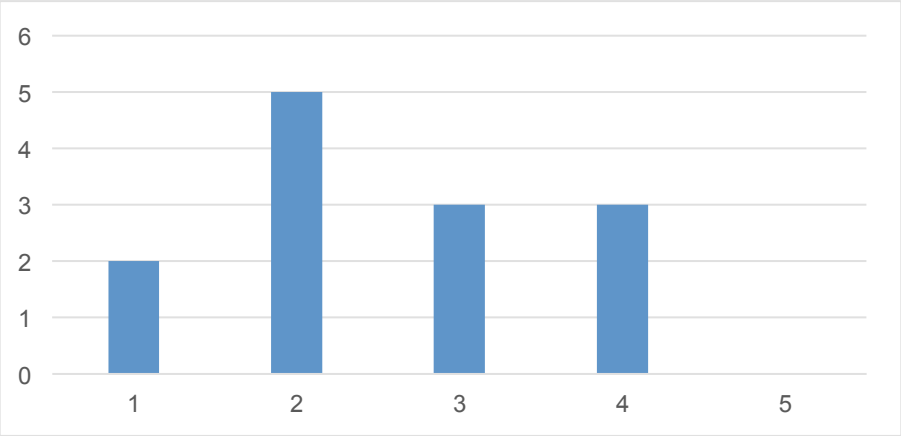
testaamista priorisoida tarpeeksi tai projektit eivät vaadi laajaa testaamisesta, niiden koska ulkoasupainotteisia.

Tavoitteena oli saada selville projektinhallinnan epäkohdista, jossa verkkokehittäjät kokevat projektin vaatimuksien olevan epäselvät. Kuvio 12 kertoo, että vastaajat tietävät keskivertoa paremmin, minkä tyyppistä testaamista asiakkaalle on luvattu. Vastausten jakauma painottunut vastausvaihtoehdoille 3 ja 4.



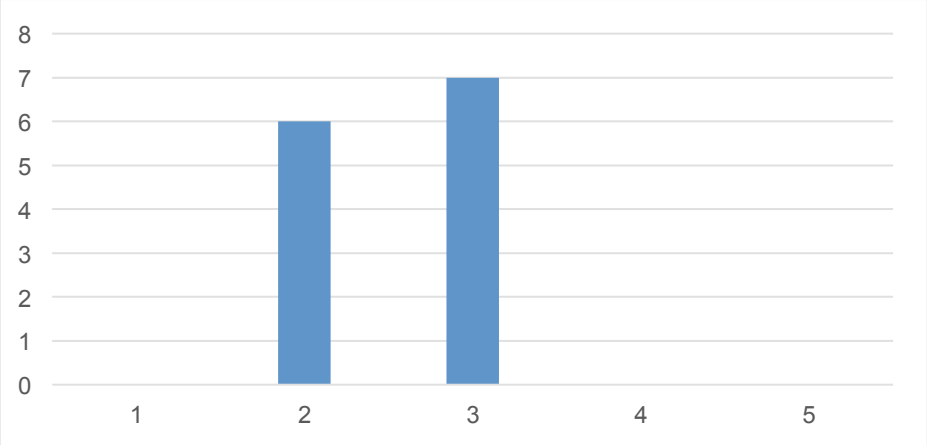

Kuvio 12. Kuinka tietoinen olet asiakkaalle luvatusta testaustavasta?

Kysymysvalinnat oli esitetty vastaajille seuraavasti: missä vastausvaihtoehto 1 tarkoittaa ”En ole tietoinen” ja vastausvaihtoehto 5 ”Olen täysin tietoinen”. Kuvion 13 kysymys oli muotoiltu seuraavasti: Kuinka tietoinen olet asiakkaalle luvatusta testausajasta? Kuvion 12 tai kuvion 13 kysymyksen tuloksissa ei ole yhtään vastausta siitä, että verkkokehittäjää olisi informoitu täysin luvatusta testaus ajasta tai tavasta, mikä on luvattu asiakkaalle. Asiakaspalautteen kannalta olisi tärkeätä tiedottaa verkkokehittäjää asiakkaan odotuksista, jottei pääse syntymään näkemyseroja kehitettävän verkkosivun valmiusasteesta. Projektin valmiusaste on aina subjektiivista varsinkin, jos verkkokehittäjän itsensä pitää arvioida työnsä laatua.




Kuvio 13. Kuinka tietoinen olet asiakkaalle luvatusta testausajasta?

Kyselyssä haluttiin myös saada selville, onko testaamiseen panostettu tarpeeksi resursseja. Näin saadaan realistinen kuva kehittäjille annetusta ajasta huolehtia testaamisesta. Kuviosta 14 näkyy, että suurin osa vastaajista on valinnut vastausvaihtoehdoksi kohdan 3. Toiseksi eniten vastauksia on kohdassa 2. Tämä kertoo mielestäni siitä, että vastaajat ovat hieman tyytymättömiä resursseihin, jotka testaamiseen on alkuaan varattu.




Kuvio 14. Onko testaukseen resursoitu tarpeeksi aikaa?

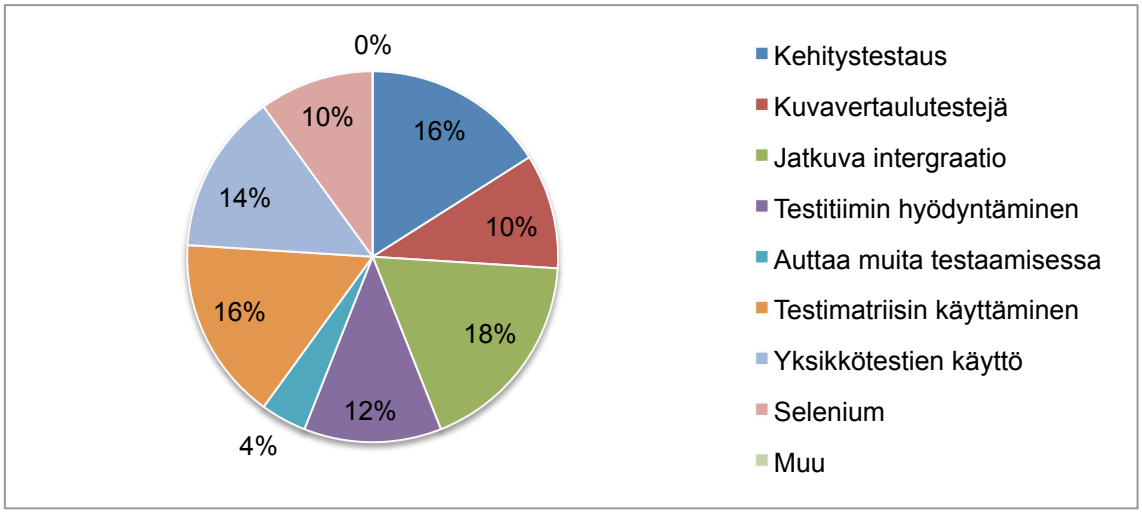
Muu yleinen kyselystä kerätty palaute koskee myös projektinhallintaa. Kahdessa kommentissa kritisoidaan, ettei testaamiselle jätetä aikaa projektien loppuvaiheessa. Toinen kommentti mainitsee myös, että testausajan pituuden säilyttämistä sovittuna on pahimmillaan puolustettava hanakasti. Samassa kommentissa on mainitaan myös viime hetken muutokset, jotka häiritsevät useasti testaamisen loppuun viemistä. Seuraavassa on ote yhdestä kyselyyn jätetystä kommentista.

> ”Monissa projekteissa on vain lista tuetuista selaimista ja älypuhelimista. Sen lisäksi on olemassa vain yleinen tehtävä testaamiselle, jota ei ole tarkennettu sen enempää.”

Kommentin perusteella testaustapojen määrittämiselle on tarvetta ennen projektin alkua. Testaustapojen puuttuminen ei ainakaan lisää kehittäjän halua testata omaa työtään huolellisemmin, sillä kehittäjän tulee itse olla innovatiivinen sovellettavien testaustapojen hyödyntämisessä. Testaamista työvaiheena pidetään aikataulun venyttäjänä, sillä virheiden löytyminen tarkoittaa yleensä verkkokehittäjän työmäärän kasvua. Mirumissa verkkokehittäjien kaksoisrooli koodaajana ja testaajana on huono yhdistelmä, sillä yleensä testaajia ei voi pitää vastuussa virheiden löytämisestä [60, s. 36]. Mirumin testauskäytäntöjen takia testaajaa voidaan syyttää tehdyistä virheistä koodaajana.

Vastauksissa on myös muutamia myönteisiä kommentteja. Kommenteissa mainitaan viimeaikaisista parannuksista, jotka koskevat juuri parempien resurssien varaamista testaamiselle. Yksi ehdotus kommenteissa on, että muut kehittäjät osallistuisivat testaamiseen arvioimalla toisten kehittäjien kirjoittamaa koodia. Koodia ei voisi käyttää ilman toisen kehittäjän hyväksyntää. Tämä ehdotus parantaisi laadunvalvontaa huomattavasti, sillä koodaajan tekemiä muutoksia arvioitaisiin myös toisen verkkokehittäjän näkökulmasta. Koodin luettavuus ja laatu paranisi, kun koodista tehdään parannusehdotuksia. Hyvä parannusehdotus voisi olla esimerkiksi kommenttien lisääminen koodiin.

Kyselyssä käsiteltiin sitä, minkälaisia testausmetodeja verkkokehittäjät olisivat kiinnostuneita oppimaan tulevaisuudessa. Kuvio 15 kertoo testaustavoista, joita verkkokehittäjät ovat kiinnostuneita hyödyntämään tai oppimaan. Kysymyksen vastausvaihtoehdot olivat valmiiksi valitut edustamaan ohjelmistotestaamisen eri vaihteita tai metodeja. Vastaajilla oli mahdollisuus valita useampia vastausvaihtoehtoja. Verkkokehittäjien vastaukset edustavat mielestäni heidän omia mielipiteitään siitä, mitkä kehitystavat olisivat tehokkaita testaustapoja heidän omassa työssään. Verkkokehittäjät tietävät parhaiten, mitkä kohdat heidän tekemässään teknisessä ratkaisussa ovat puutteellisia, joten uskon heidän tunnistavan oikeantyyppisen testaamistavan tarpeen. Toisaalta he eivät kuitenkaan ole ohjelmistotestaamisen ammattilaisia, joten heidän valitsemansa vaihtoehdot on valittu omien mielikuvien perusteella, paitsi silloin, kun heillä on omakohtaista kokemusta aiheesta.




Kuvio 15. Oletko kiinnostunut oppimaan seuraavien testausmetodien käyttöä?

Eniten vastauksia on saanut vaihtoehto jatkuva integraatio: yhteensä yhdeksän vastaajaa on valinnut sen. Vastauksista voi päätellä, että verkkokehittäjät tahtoisivat eniten oppia jatkuvan integraation työtapoja. Jos verkkokehittäjille annettaisiin mahdollisuus käyttää jatkuvan integraation työtapaa, he todennäköisesti ymmärtäisivät sen hyödyt ja olisivat motivoituneita sopeutumaan erilaiseen työskentelytapaan.

Toiseksi suosituimmat vastaukset olivat kehitystestaus ja testimatriisin täyttäminen testaamisen varmistamiseksi. Molempiin vaihtoehtoihin oli siis vastattu kahdeksan kertaa. Kehitystestaamisen käyttö lyhentäisi testausaikaa, kun testaamista suoritetaan hajautetusti kehitystyön aikana. Tällöin mahdollisiin virhetilainteisiin voidaan valmistautua paremmin, kun potentiaalisten virhetapauksien määrä on tiedossa. Mielestäni vastaajat haluaisivat kyseenalaistaa testaamisen suorittamista projektin aikana ja kokeilla, miten testaaminen voidaan suorittaa hajautetusti. Myös vastausvalinnan testimatriisin käyttämisen sijoittuminen kertoo, että verkkokehittäjillä ei ole tarpeeksi ohjeita, joita seurata testaamisen suorittamisessa. Perusohjeiden seuraaminen testaamisen suorittamisessa toisi varmuutta oman tai toisen koodin laadun arvioimisessa.

## 5 Uusien testaus- ja kehitystapojen suositukset

Kyselyyn annetusta yleisestä palautteesta voi tulkita kehittäjien tarvitsevan yleisen ohjeen testitapauksien suorittamiseen. Ohje voisi sisältää painotukset testaamisalueille, jolloin verkkokehittäjä voisi saada ideoita, minkälaisia testejä hänen tulisi vähintään suo-

rittaa omalleen tai toisen koodaamalle projektille. Vaihtoehtoina olisi myös resursoida toinen kehittäjä ideoimaan, minkälaisia testejä koodaajan tulisi vähintään suorittaa, jotta tuotettu laatu olisi hyväksyttävällä tasolla.

Ohjeen voisi lisätä myös laitelaboratorion yhteyteen, jolloin muut kuin pelkät verkkokehittäjät voisivat kiinnittää huomiota verkkosivun laatuun. Tämä myös vaatisi, että laitelaboratorio olisi jatkuvasti käyttökunnossa. Laitelaboratorion ylläpitoon tulisi panostaa niin, että siihen kuuluvat laitteet pysyisivät ladattuna ja niitä voisi ohjailla synkronoidusti pääkoneen kautta. Ilman selaimien synkronointia laitelaboratorion käyttö on tehotonta. Laitelaboratorion käyttökynnys kasvaa, jos testaamisympäristön asettamiseen kuluu liian kauan aikaa. Testausvastaisuus madaltuisi huomattavasti, jos testaamisen suorittamiseen olisi kunnon testiympäristö. Testaamisohjeesta voidaan tehdä yksi yleisempi versio, joka koskisi kaikkia Mirumin tuottamia verkkosivujen vaatimuksia.

## 5.1 Testaamisen automatisointi jatkuvan integraation avulla

Palautekyselyn perustella testaaminen tehdään tällä hetkellä suurimmaksi osaksi käsin Browserstack-pilvipalvelussa ja iOS-simulaattorilla. Testaustyövaihe tulisi automatisoida mahdollisimman pitkälle samantyyppisissä projekteissa. Testaamisen tehokkuus kasvaisi huomattavasti, kun verkkosivulle tehtäisiin kaikki perustestit automaattisesti. Testaamisen taso ei olisi enää subjektiivista, vaan Mirumilla olisi oma testausalusta, joka suorittaisi testit automaattisesti Mirumin omia vaatimuksia seuraten. Potentiaalisia testitapauksia, joita voisi automatisoida testipalvelimella olisivat

- suosituskykytestaus
- käyttöliittymätestaus
- regressiotestaus
- HTML-dokumentin validointi
- yhteensopivuustestaus.

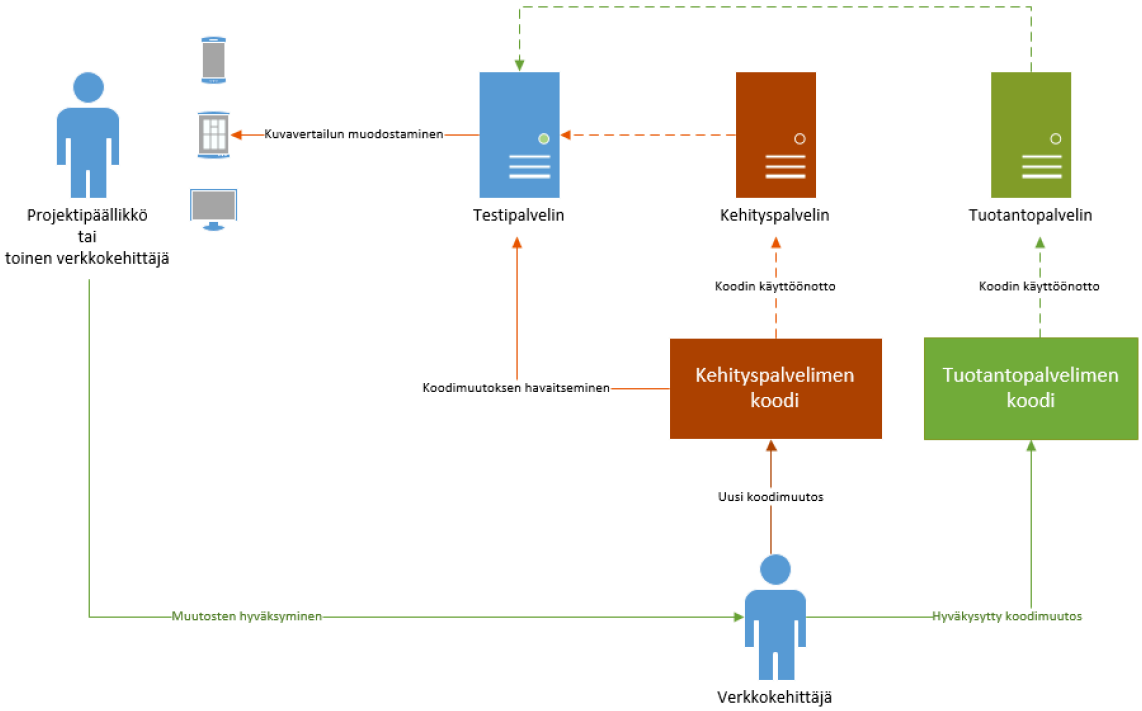
Näitä testitapauksia sovellettaisiin ohjelmistotestauksen B-mallin mukaisesti, jolloin suorituskykytestauksen tuloksia analysoidaan ja suunnitellaan kattava testi, joka pätee kaikkiin Mirum verkkokehitysprojekteihin. Mirumille testipalvelin olisi erittäin tehokas työkalu, sillä sen voisi kytkeä suorittamaan testejä mille tahansa yrityksen ylläpitämälle verk-

kosivulle. Kirjoittamalla testejä edes yhdelle verkkosivulle päästään testien automaatiossa pitkälle, sillä samoja testejä voidaan kopioida muihin projekteihin.

Jatkuvan integraation käyttö testaamisessa pakottaisi testien suorittamisen ennen koodin käyttöönottoa. Laajojen verkkosivuja työstettäessä koodimuutoksien vaikutuksia muihin alasivuihin on hankala ennustaa ilman huolellista regressiotestaamista. Sisällön tarkastelun avuksi voidaan käyttää regressiotestejä, joita käyttäessä muutosten tarkkailu tapahtuu automaattisesti. Regressiotestien käyttö toisi varmuutta myös kehittäjälle oman koodin suoriutumisesta, kun sen laajempia vaikutuksia voidaan arvioida. Regressiotestien laajamittaiseen käyttöönottoon tarvittaisiin testipalvelin, joka olisi Jenkinsin toimintaympäristö. Jenkins-tehtävien suorittaminen voidaan määrittää aloitettavaksi tietyin väliajoin, käsin tai esimerkiksi versionhallintatyökalujen yhteyteen.

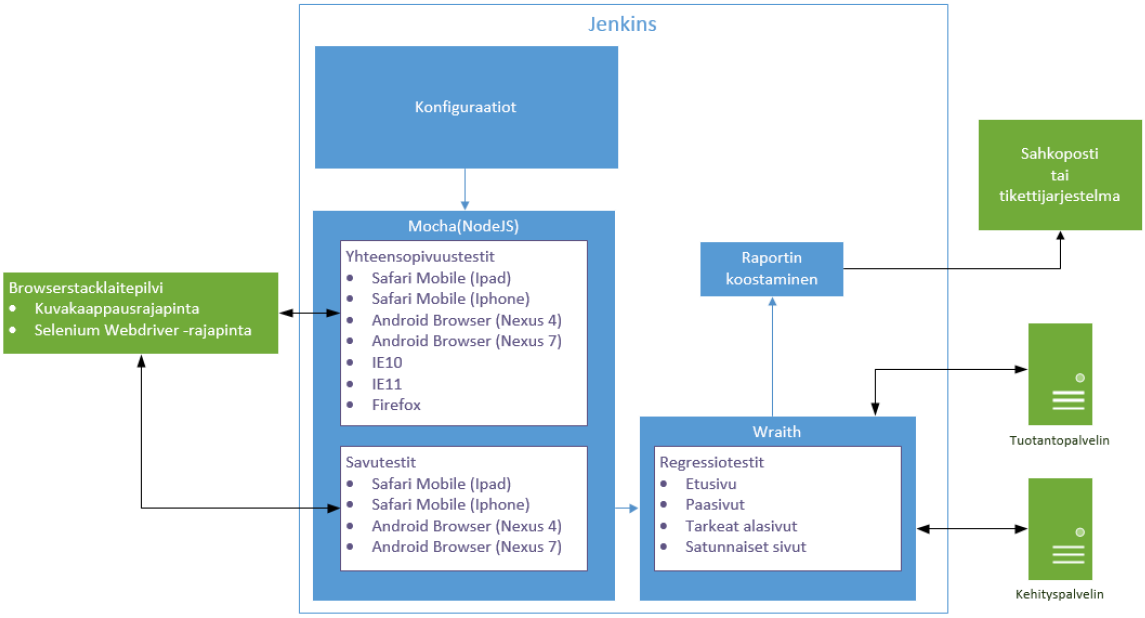
Regressiotestien suorittamiseen tarvitaan Wraith-kuvavertailutyökalu, joka olisi asennettu palvelimelle. Wraith vaatii sivukohtaisen asetustiedoston, johon määritellään testatut näyttökoot, sallittu virhemarginaali ja läpikäydyt URL-osoitteet. Tuotantopalvelimen ja kehityspalvelimen sisältöjen tulisi olla mahdollisimman samat, ettei Wraith epäonnistu muuttuneen sisällön takia. Wraith käynnistettäisiin räätälöidyllä skriptillä, joka ajettaisiin aina uuden koodin käyttöönoton yhteydessä. Kun Wraith-testi olisi suoritettu, Jenkins lähettäisi sähköpostitse ilmoituksen sivustosta vastaavalle henkilölle. Sähköposti sisältäisi linkin Wraithin muodostamaan verkkogalleriaan. Sähköpostin vastaanottanut henkilö voisi olla projektipäällikkö tai kehittäjä itse, joka arvioisi Wraithin löytämät muutokset. Wraith-kuvavertailutesti epäonnistuu, jos sisältö on muuttunut tuotantoympäristössä. Tämä voi häiritä testien läpäisyä. Vertailtavan kehityspalvelimen tietokanta tulisi synkronoida ennen Wraithin suorittamisesti. Näin varmistetaan, että saadut testitulokset ovat luotettavat. Jenkins voidaan kytkeä synkronoimaan tietokanta ennen kuvavertailutestien suorittamista, jolloin prosessi olisi täysin automaattinen.

Kuvio 16 visualisoi, miten työnkulku etenee koodimuutoksia tehtäessä. Merkittävin etu Jenkinsiä hyödyntävässä testausympäristössä olisi, että sen käyttöönotto muille sivustoille olisi nopeaa. Uuden sivuston testaamisen vaaditaan Wraithille asetustiedosto ja uusi Jenkins-tehtävä, joka laukaistaan halutulla tavalla. Sivuston liitämisen testaamispalvelimeen tulisi olla helppoa, jotta kynnys käyttöönottoon myös yksinkertaisissa projekteissa olisi mahdollisimman matala.




Kuvio 16. Jatkuvan integraation työjärjestys.

Kuvio 17 näyttää tarkemmin kuvion 16 testipalvelimen sisältämän Jenkinsin suorittamat toiminnot. Jenkinsin koostaman raportin tulisi olla mahdollisimman helppolukuinen, jolloin muut kuin verkkokehittäjät voisivat käyttää ja arvioida verkkosivun laatua. Kun testaamisen tuloksien arvioiminen voidaan ulkoistaa verkkokehittäjän vastuualueelta muille, seurataan hyvän testaamisen periaatteita.




Kuvio 17. Jatkuvan integraation järjestelmäkaavio.

Samaa testipalvelinta voidaan käyttää usean verkkosivun kuvavertailun muodostamiseen, joten kuvan mukaista ympäristöä ei räätälöidä jokaiselle ylläpidettävälle verkkosivulle. Kuviossa 17 sivukohtaiset tiedot olisivat vain projektikohtaisissa konfiguraatio tiedostossa. Kuvio 17 havainnollistaa testityökalujen suoritusjärjestyksen, jos eri testimetodeja halutaan suorittaa automaattisesti ketjussa. Järjestelmäkaavioon on myös merkitty yhteensopivuus- ja savutestit, jotka tehdään käyttäen Browserstackin laitepilven rajapintaa, joten niitä ei tarvitse suorittaa paikallisesti asennetuilla selaimilla. Rajapintaa käyttämällä vältytään paikallisesti asennettujen selaimien ylläpidosta ja asentamisesta. Vasta kun yhteensopivuus- ja savutesti on ajettu, käynnistetään Wraith-kuvavertailutyökalu, jonka avulla voidaan tehdä visuaalisia regressiotestejä.

## 5.2 Kehitystestauksen käyttö synkronoiduilla selaimilla

Verkkosivuja selataan ja tullaan tulevaisuudessa selaamaan yhä enemmän älypuhelimilla. Tämän takia ulkoasun kehittämisessä pitäisi keskittyä enemmän älypuhelimella selailevan käyttäjän näkökulmaan käyttämällä simulaattoreita kehitystyöaikana. Virheet korjattaisiin jo koodia kirjoittaessa, kun ulkoasua tarkasteltaisiin jatkuvasti usealla älypuhelimen selaimella. Testaaminen jo koodia kirjoittaessa estäisi tilanteen, jossa ennen verkkosivun tai sen osan julkaisua korjattavien virheiden määrä olisi vielä tuntematon. Kun

testaamista suoritetaan jatkuvasti, yhteensopivuusvirheet huomattaisiin mahdollisimman aikaisessa vaiheessa.

Synkronoituja selaimia käyttämällä säästetään aikaa, kun jokaiselle murroskohdalle on oma selainikkuna koko valmiina esillä. Mukautuvaa ulkoasua ei tällöin testata enää yksi selainikkunaresoluutio kerrallaan, vaan selaimien synkronoinnilla on mahdollista tarkastaa yhdellä sivulatauksella useampi eri keskeytyskohdan versio. Merkittävimpiä etuja on, että verkkosivun eri keskeytyskohtien tarkistamiseen ei tarvita muita toimia kuin verkkosivun päivittäminen selaimessa. Jokaista keskeytyskohtaa ei tällöin tarvitse tarkistaa läpi yhdellä selaimella, vaan jokaiselle keskeytyskohdalle on olemassa oma simulaattori tai selain.

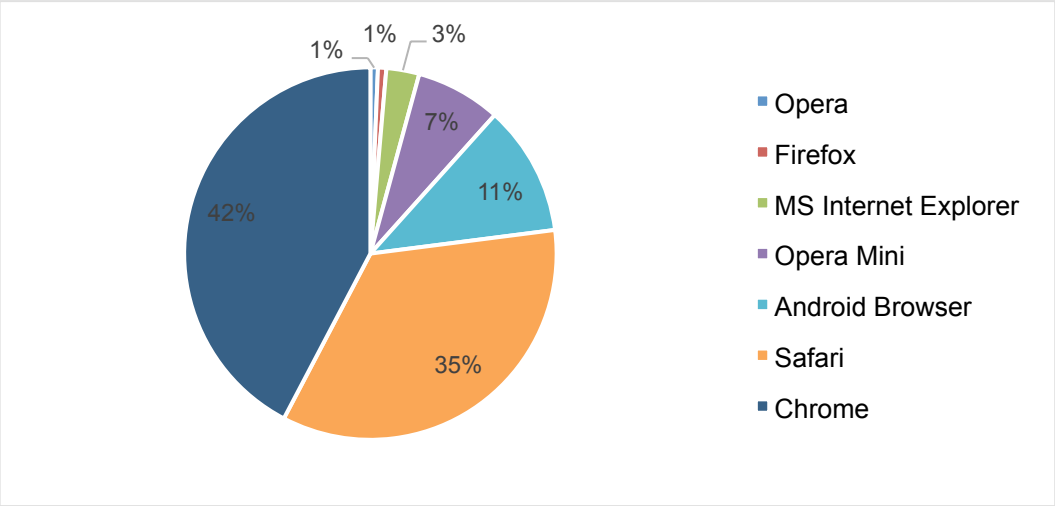
Kun testaaminen tehdään kehitysvaiheen aikana, yksittäisiä virheitä on huomattavasti nopeampaa korjata kehitystyön ollessa vielä kesken. Vaikka virhettä ei heti korjattaisikaan, verkkokehittäjä on ainakin tietoinen sen olemassaolosta. Pelkästään sillä, että älypuhelin tai tabletti on mukana kehittämisessä, pieneltä näytöltä selaavan käyttäjän näkökulmaa otettaisiin paremmin huomioon. Kuviossa 18 havainnollistetaan, miten verkkokehittäjän tulisi käyttää synkronoituja selaimia ja mobiiliselaimia. Verkkokehittäjän tulisi käyttää mahdolliset lisänäytöt älypuhelinsimulaattoreihin, jotta kehitystestaaminen olisi mahdollisimman vaivatonta. Synkronoitujen selainikkunoiden määrä riippuu paljon siitä, kuinka paljon kehittäjällä on näyttöpinta-alaa käytettävissään. Jos kehittäjällä on käytössä vain yksi näyttö, synkronoituja selaimia ei voi käyttää montaa kerrallaan.


Kuvio 18. Kehitystestaamisen aikana käytössä olevat simulaattorit.

Jos verkkokehittäjille ei ole mahdollista hankia lisänäyttöä tilan puutteen takia, myös alkuperäisiä laitteita voidaan käyttää yhteensopivuuden varmistamiseen. Tällöin älypuhelinta voisi pitää esillä kehittäjän työpöydällä, eikä lisänäyttöpinta-alaa tarvittaisi älypuhelinsimulaattorien käyttämiseen. Älypuhelimen tulisi tällöin olla samassa langattomassa verkossa kuin verkkokehittäjän tietokoneen, jotta Browsersyncin selaimen synkronointi toimii. Alkuperäisten laitteiden käyttäminen ulkoasun koodaamisen aikana vähentäisi testausaikaa huomattavasti ainakin yhden laitteen kohdalla. Kehityksessä mukana oleva älypuhelin tarvitsisi ainakin virtalähteen ja telineen, jotta verkkokehittäjän työergonomia pysyisi samalla tasoilla kuin aikaisemmin. Älypuhelintelineen tulisi olla samalla korkeudella kuin muut lisänäytöt, jotta verkkokehittäjän työasento pysyisi edelleen ergonomisena. Vertaisin älypuhelimia ja niiden telineitä pienikokoiseen laitelaboratorioon, jolloin esillä olevia älypuhelimia voisi käyttää hyväksi.

## 5.3 Testaukseen käytetyt selaimet ja selainikkunakoot

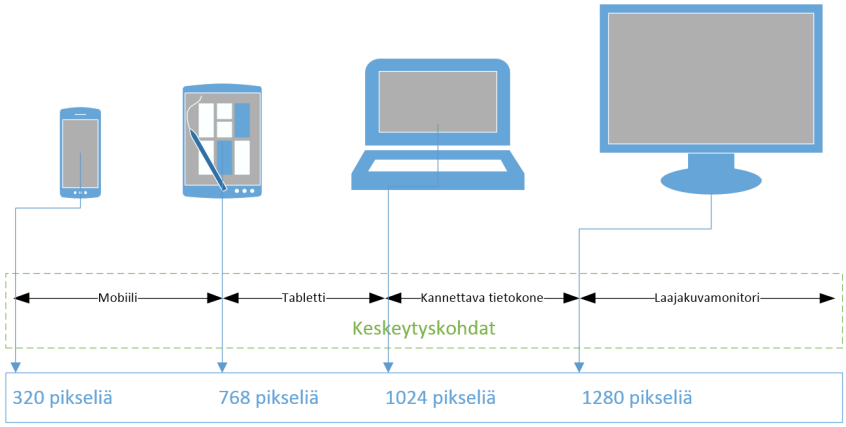
Kehitystestaamisessa mukana olevat älypuhelimet tulisi valita analytiikan ja virhealttiuden mukaan. Kyselystä saatujen tuloksien mukaan virheherkimpiä älypuhelimia ovat Microsoftin Lumiat, joiden Internet Explorer -selaimet aiheuttavat paljon ongelmia. Seuraavaksi eniten virheitä aiheuttivat iOS-laitteet. Helpoin tapa karsia ulkoasun yhteensopivuusongelmat olisi hankkia muutama Lumia-älypuhelin, joita voisi käyttää ulkoasun yhteensopivuuden varmistamisessa kehitystyön aikana. Applen tuotteiden yhteensopivuus voidaan varmistaa käyttämällä iOS Simulator -ohjelmistoa, joka on jo laajasti verkkokehittäjien käytössä. Ainoa muutos simulaattorin käyttöön olisi käyttää sitä jatkuvasti kehitystyön aikana synkronoituna Browsersyncin kautta eikä vain testaamisvaiheessa. Verkkokehittäjille tehdystä kyselystä puuttuu tarkemmat tiedot siitä, mitkä selaimet erityisesti aiheuttavat yhteensopivuusvirheitä. Tarkastelemalla tarkemmin maailmanlaajuisia selain- ja älypuhelintilastoja voidaan päätellä, mitä muita laitteita kannattaisi testata.




Kuvio 19. Yleisimmät mobiiliselaimet ja niiden markkinaosuudet [54].

Kuvio 19 esittää suosituimpia mobiiliselaimia maailmanlaajuisesti. Tietyistä selaimista voidaan päätellä käytössä oleva käyttöjärjestelmä. Esimerkiksi Android Browser -selain on käytössä vain Android-käyttöjärjestelmillä ja Internet Explorer on käytössä Windows Phonella. Mobile Report 2015:n mukaan yleisin älypuhelin- ja tablettikäyttöjärjestelmä on Android. Jos verrataan tilastoa yleisimpiin selaimiin, voidaan päätellä, että yleisin selain Androidilla on Chrome. [55.] Käyttäisin ainakin yhtenä testiselaimena Android Browseria, sillä Chrome on jo yleisesti verkkokehittäjien käytössä. Testaamista ei enää kannata painottaa lisää jo laajasti käytössä olevalla Chromella. Painottamalla Android Browseria vältytään päällekkäiseltä testaamiselta Chromella.

Testattavien laitteiden käyttämä selainikkunakoko on tärkeässä roolissa, sillä sen leveys määrittää toistettavan ulkoasun version. Yleisempiä murroskohtia verkkosivuilla ovat 320, 769 ja 1024 pikseliä. Kuvan 12 perusteella testaukseen käytettyjä selaimia pitäisi valita ainakin jokaiselle murroskohdalle, mieluiten juuri ennen murroskohdan muutosta. Esimerkiksi Iphone 4s -älypuhelimen selainikkunan leveys on 320 pikseliä. Sen käyttämä selain on Mobile Safari. Tämän älypuhelimen simulaattori saadaan käyttöön iOS Simulator -sovelluksesta. Nexus 4 -älypuhelimen selainikkunan leveys on 768 pikseliä, jolloin sen Android Browser -selain toistaa tablettiversioita. Nexus 4 -älypuhelimen simulaattori on saatavilla aikaisemmin esitellyssä Xamarin Android Player -sovelluksessa.




Kuva 12. Selainikkunoiden leveysjana, jonka perusteella testilaitteet tulisi valita.

Jos halutaan vielä varmistaa, ettei Lumia-älypuhelimen selain aiheuta virheitä, puhelin voidaan liittää kehitysympäristöön langattoman verkkoyhteyden avulla. Käyttämällä näitä kahta simulaattoria ja yhtä älypuhelinta Browsersyncin avulla kehitystyön aikana verkkosivujen responsiivinen yhteensopivuustestaus on saatettu hyvälle tasolla. Valittuja laitteita tulisi myös hyödyntää automaattisissa kuvankaappaustesteissä ja Selenium-testeissä.

## 5.4 Palaute esitetyistä testaustavoista

Kyselyn tuloksia esitettiin 18.3.2016 pidetyssä kokouksessa, jossa oli paikalla suurin osa Mirumin verkkokehittäjistä. Esityksessä käytiin läpi myös potentiaalisia parannusehdotuksia, joilla epäkohtia testauksen suorittamisessa voitaisiin korjata, esimerkiksi kehitystestaustyötapa, johon sisältyi synkronoitujen selaimien käyttö useiden simulaattoreiden tai alkuperäislaitteen avulla. Esityksen jälkeen aiheesta keskusteltiin noin 50 minuuttia, ja tavoitteena oli saada palautetta esitetyistä ratkaisuista ja päättää yksityiskohdista, mihin suuntaan testipalvelimen kehittämistä pitäisi viedä. Kokouksessa oli tarkoitus luoda mahdollisimman neutraali ympäristö, jossa mielipiteitä testaustavan muutoksesta tuodaan esille. Tarkoitus oli myös kysyä testaajilta mielipiteitä siihen, minkälaisia tilanteita automaattisella testausvaiheella halutaan välttää. Pääkeskusteluaiheiksi muodostuivat testikehikot, Jenskinsin yksityiskohdat ja järjestelmän ylläpito. Välittömästi kokouksen jälkeen diaesitys lähetettiin osallistujille sähköpostitse ja heitä pyydettiin kommentoimaan aihetta kirjallisesti lyhyessä kyselyssä. Kyselyssä oli kolme avointa vastauskenttää ja kolme kysymystä, jossa vastausvaihtoehdot olivat kyllä tai ei. Koko kysely on liitteessä

4, ja kaikki kyselystä kerätyt kolme vastausta ovat kokonaisuudessaan liitteestä 5. Seuraavassa on lyhyt kooste kokouksessa esiin nostetuista aiheista ja kyselyn tuloksista.

Kehitystestaus

Kehitystestaustavan kommentoitiin vaativan huomattavasti lisänäyttöpinta-alaa. Jos verkkokehittäjät käyttäisivät selaimien synkronoimista, kaikille halukkaille tulisi tarjota lisänäyttöjä testaustavan käyttöön. Samaan aiheeseen kommentoitiin, ettei tila työpöydällä riitä usean lisänäytön pitämiseen. Myös oikeita testilaitteita pitäisi olla jatkuvasti verkkokehittäjien käytettävissä, jotta testaustapaa voisi hyödyntää. Yksi kokoukseen osallistunut verkkokehittäjä ilmoitti jo käyttävänsä Browsersync-työkalua työssään. Keskustelun jälkeen lähetetyssä kyselyssä kaksi kolmesta vastaajasta ilmoitti, että he voisivat käyttää kyseistä testaustapaa jatkuvasti. Kriittisin kommentti koski työtapojen yksilöllisyyttä:

> ”En usko testaustapojen yhdistämiseen, sillä lopputuloksen tulisi aina olla testattu tuote riippumatta meidän työtavastamme.”

Kommentin kirjoittaja oli myös maininnut, että testauskehitys on erittäin hidasta. Testaustavan käyttö vaatisi usean testauslaitteiden ja lisänäyttöjen hankkimisen, jos testaustapa halutaan ottaa laajemmin käyttöön.

Selenium-testit

Kokouksen keskusteluissa tuotiin esille Selenium-testien haitat, sillä niillä on taipumusta antaa vääriä positiivisia tuloksia. Eräs kyselyyn vastannut näki potentiaalisena riskinä testituloksien kasaantumisen liian suureksi taakaksi. Testiraporttien kasaantuminen voi johtua esimerkiksi projektityyppien yhtyeensopimattomuudesta testausympäristöön. Vastaaja ehdotti, että tämä tulisi ehkäistä tekemällä testituloksista mahdollisimman helppolukuisia. Keskustelussa esitettiin, että liian kattavan testikirjaston tekeminen on tehotonta, sillä jokaisen projektin HTML-sivun rakenne on erilainen, jolloin testit on joka tapauksessa räätälöitävä. Vaihtoehdoksi tähän esitettiin, että toiminnallisuustestille rakennettaisiin datataso, joka ymmärtäisi erityyppisiä sivupohjia. Datatasolla olisi oma logiikkansa, joka tarkistaisi saatavilla olevat elementit ennen niiden testausta. Datatason käyttö tekisi testitiedostoista yleisesti päteviä, sillä yhtä testiä voitaisiin käyttää kaikissa Mirumin ylläpitämissä verkkosivuissa, koska datataso ymmärtäisi testattavan verkkosivun rakenteen. Esimerkiksi hakukentän testitapauksessa datatasolle lisättäisiin jokaisen sivun hakuken-

tän HTML-rakenne, jolloin datataso ilmoittaa testitiedostolle, mitä valitsimia testin tulisi käyttää. Datatason koodaaminen vaatisi C#-koodin ymmärtämistä, josta Mirumin verkkokehittäjillä ei ole osaamista. Tämän hankaloittaisi testien ylläpitoa huomattavasti, kun datatason ylläpitoon tarvittaisiin päteviä henkilöitä.

JavaScript-testikehikkojen käyttöä puoltaa se, että kaikki verkkokehittäjät osaavat koodata JavaScriptiä. Keskustelussa tuotiin esille myös, että nuoria JavaScript-testikehikkoja on olemassa paljon. Kaikki potentiaaliset vaihtoehdot eivät ole vielä täysin valmiita, sillä suurin osa JavaScript-testikehikoista on olemassa vasta kehitysvaiheessa. Tämän takia on olemassa riski, että Mirumin testikirjastoa aletaan koodaata väärällä JavaScript-testikehikolla, jota ei kehitetä enää tulevaisuudessa.

Muutama verkkokehittäjä oli halukas oppimaan lisää Selenium-testien kirjoittamisesta, joten tehtiin aloite työpajan pitämisestä. Työpajan tavoitteena olisi opettaa kaikille verkkokehittäjille, miten automaattisia testejä tehdään käytännössä. Näin verkkokehittäjille voitaisiin antaa vinkkejä, miten aiemmin mainittuja vääriä positiivisia tuloksia voitaisiin välttää. Ennen työpajan pitämistä pitäisi myös valita testikehikko, jonka oppiminen pitkällä tähtäimellä kannattaisi.

Jenkins

Kokouksessa tuotiin esille, millä tavoin automaattiset testit tulisi käynnistää. Osallistujat toivat esille huolen, ettei testejä tulisi suoritettaisi liian useasti, sillä liika testitulosten koostaminen voi johtaa testitulosten laiminlyöntiin. Tärkein aiheeseen liittyvä kysymys oli, voidaanko laukaisimet kytkeä Drupal-isännöintipalvelu Acquian versionhallintatyökaluihin, joita melkein kaikki Mirumin Drupal-pohjaisista projekteista käyttävät. Ehdottamaani testien käsin suorittamista ei kannatettu, sillä käsin laukaisu mahdollistaisi testien suorittamisen unohtamisen. Tällöin testit eivät enää palvelisi tarkoitustaan, kun niitä ei käytettäisi jatkuvasti. Jos testien suorittamisen kytkettäisiin versionhallintatyökaluihin, tällöin pitää huolehtia, etteivät testiraportit täytä tikettijärjestelmää tai testien arvioijan sähköpostia, kun testejä suoritetaan jatkuvasti. Paras ratkaisu tällöin olisi automaattisten testien suorittaminen säännöllisin aikavälein.

Miltei kaikki olivat samaa mieltä siitä, että automaattisien testien säännöllinen ajaminen toisi varmuutta oman työn arvioimiseen. Tuloksien luotettavuutta pitäisi myös osata arvioida, sillä yksi palautekyselyyn vastannut oli huolissaan, että automaattisten testien yllä-

pito kasvattaa verkkosivujen huoltotaakkaa. Huoltotaakan kasvu pitäisi pystyä myös laskuttamaan asiakkailta, mikä voi olla hankalaa, sillä asiakkaat olettavat testauksen kuuluvaan jo palveluihin. Sama vastaaja kommentoi, että hän on kuitenkaan sataprosenttisesti testaamisen takana, kunhan testauksen automatisointi toteutetaan vain niiltä osin, kuin se on järkevää.

Jatkuvan integraation testitapaukset

Esittelyn suuri yleisö mahdollisti ajettavien testien ideoimisen. Keskustelussa tuli idea, jossa verkkosivujen vasteaikaa tarkistettaisiin jatkuvasti. Jos verkkosivu on hidas tai ei vastaa Jenkinsin suorittamaan testiin, palvelimen hitaudesta lähetetään ilmoitus asianomaisille henkilöille. Tällöin Mirumilla olisi oma varmistusprosessinsa sille, että asiakkaiden verkkosivut pysyvät käyttökunnossa. Keskusteluissa tuotiin myös esille, että yksi testattava ominaisuus olisi sivustojen tärkeimmät käyttäjäpolut ja linkkiketjut. Jokaisen verkkosivu tärkeimpiä sisäisiä linkkejä seurattaisiin automaattisella testillä. Testin ajaminen olisi automaattista. Jos linkkiketju häiriintyisi, testi epäonnistuisi. Testin rakentaminen vaatisi pääsyn verkkosivun vierailijoiden analytiikkaan, josta vierailluimmat sivut saataisiin selville. Samasta analytiikasta saataisiin myös suosituimmat laitteet, joiden avulla voitaisiin tehdä yhteensopivuustestausta. Esityksessä tuotiin myös esille eri selaimilla otettujen kuvankaappausten vertailu keskenään. Tällöin Browserstackin rajapinnasta saatuja päätelaitteiden kuvatiedostoja vertaisiin keskenään samoin kuin Wraithin koostamassa kuvavertailugalleriassa. Metodin toimivuutta epäiltiin, sillä selaimet toistavat fontteja ja grafiikkaa eri tavoin, jolloin testistä voisi tulla epäluotettava.

Kolmannen osapuolen palvelut

Keskustelussa tuotiin esille, että Browserstackin rajapinnalle on olemassa vaihtoehtoja. Yksi verkkokehittäjä oli käyttänyt Crossbrowsertesting-testauspilven rajapintaa ja hän oli sen toimivuuteen tyytyväinen. Crossbrowsertesting olisi varteen otettava vaihtoehto Browserstackille. Toinen vaihto ehtotestaamiseen olisi myös ostaa testaamista kokonaisvaltaisena palveluna. Markkinoilla on olemassa verkkokehitysprojekteihin ratkaisuja, joissa projektin edistymisestä pidetään automaattista lokia. Yksi tällainen palvelu on Speedcurve, joka seuraa kehitysprojektin edistystä sen edetessä. Se mahdollistaa virheiden tarkan seuraamisen kehitysprojektin eri vaiheissa. Speedcurven käyttö tarjoaisi paljon työkaluja laadun valvontaan, jolloin testipalvelimelle ei olisi enää tarvetta.

Muut aiheet

Esitys avasi keskustelun testaustapojen yhtenäistämisestä ja siitä, miten koko projektitiimin tulisi orientoitua paremmin testaamisesta huolehtimiseen omilla osa-alueillaan. Kokouksessa keskusteltiin myös siitä, miten projekteissa tulisi painottaa enemmän testaamisen tärkeyttä. Kun projektien hallinta orientoituu testauspainotteisemmaksi, virheiden etsiminen on tehokkaampaa. Testaaminen on Mirumin sisäinen asia, joka pitää opettaa kaikille projekteihin osallistuville tahoille. Testipalvelimen käyttö pitäisi perustella asiakkaille ja projektitiimille niin, että se varmistaa verkkosivun olevan yhteensopiva sovittujen vaatimusten kanssa.

Keskustelussa myös ihmeteltiin, mikseivät nykyiset asiakkaat ole tyytymättömiä Mirumin tuottamaan laatuun, jos tällä hetkellä projekteja ei testata minkään prosessin mukaisesti. Nykyinen käytäntö on, että asiakkaan löytämät virheet korjataan aina hänen pyynnöstään. Asiakkaat luottavat siihen, että Mirumilla on testausprosessi olemassa. Todellisuudessa asiakkaat raportoivat suurimman osan virheistä.

Samalla viikolla, kun esitys pidettiin, Mirumissa aloitti täysipäiväinen järjestelmätestaaja. Toistaiseksi hänen tehtäviinsä kuuluu verkkokehitysprojektien testaaminen käsin, mutta hänen on tarkoitus keskittyä konsultoimaan projektitiimejä ja osallistua testaustauksen suorittamiseen tarvittaessa. Testaajalla oli paljon asiantuntemusta siitä, mihin suuntaan automaattista testausta pitäisi viedä. Hänen tietotaidoillaan voidaan varmistaa testipalvelimen hyödyllisyys, kun sille löytyy ainakin yksi aktiivinen käyttäjä.

# 6 Yhteenveto

Insinöörityössä oli tarkoituksena löytää tehokas tapa testata responsiivisia verkkosivuja mainostoimisto Mirum Agencyssa. Mainostoimistolla oli jo olemassa muutamia testaustyökaluja, joilla verkkosivujen yhteensopivuus testataan eri selaimien kanssa. Kehitysprojekteihin osallistuvat verkkokehittäjät testasivat omaan työtään ilman ohjeistusta, jolloin testaustavat eivät olleet johdonmukaisia. Kaikki verkkokehittäjät tekivät testauksen omalla tavallaan, jolloin testausvaihe jäi helposti pintapuoliseksi eikä tieto parhaista testaustavoista välittynyt muille verkkokehittäjille.

Tavoitteena oli parantaa testaustyökalujen saatavuutta verkkokehittäjille, jotta testaaminen tehtäisiin huolellisemmin ja yhtenäisesti projektityypin mukaan. Osana insinöörityötä verkkokehittäjille tehtiin kysely, jonka tarkoituksena oli saada selville vanhat testauskäytännöt, virhealteimmat laitteet, epäkohdat projektin hallinnassa ja suunta, johon verkkokehittäjät itse tahtoisivat viedä testausta. Kerättyjä tuloksia analysoitiin ja niiden perusteella tehtiin suunnitelma, miten Mirumin testaustapoja voitaisiin syventää. Yksi ratkaisuehdotus oli lisätä verkkokehitysprojekteihin testipalvelin, jonka avulla kaikille Mirumin tuottamille verkkosivuille suoritettaisiin yhteensopivuus-, regressio- ja toiminnallisuustestit.

Insinöörityön tuloksena Mirumilla on suunnitelma, minkälaisia tapauksia tulisi testata automaattisesti testipalvelimella. Tuloksena syntyi myös paljon testitapauksia, joita hyödynnetään testipalvelimen prototyypin rakentamisessa. Insinöörityön osana syntyi testaamisen suunnitelma, jolla testaaminen voidaan automatisoida käyttäen jatkuvan integraatioita. Kyselyn tulosten esittelyä seuraavalla viikolla testipalvelimen toteutusta alettiin suunnitella kolmen hengen tiimin voimin. Tiimiin kuului teknologiavetäjä, järjestelmätestaaja ja yksi verkkokehittäjä. Testipalvelimen käyttämät testikehikot päätettiin, ja sen toiminnallisuuksille asetettiin välietappeja, jotta testaustasoa aluksi porrastettaisiin. Testipalvelimelle sijoitettaisiin yleistason testit, jotka varmistaisivat kerran vuorokaudessa verkkosivujen olevan pintapuolisesti kunnossa. Ensimmäinen askel olisi verkkosivun etusivun lataaminen, ja kaikki etusivulla olevat linkit käydään läpi mekaanisesti. Tällöin varmistuttaisiin verkkosivun etusivulla olevien linkkien toiminnasta. Muut askeleet ovat verkkosivun HTML-dokumentin validoiminen, Google PageSpeed -palvelun raportin koostaminen, eri selainkuvankaappauksien tekeminen käyttäen Browserstackin rajapintaa ja viimeinen askel olisi luoda kuvanvertailuja eri selaimilla otetuista kuvankaappauksista. Yhteensopivuustesti epäonnistuisi, jos eri selainversiot poikkeaisivat liikaa toisistaan. Viimeiseen askeleeseen kuvankaappausvertailu tehtäisiin samaa tekniikkaa käyttäen kuin Wraith-kuvavertailutyökalu.

Tarkoitus on jakaa testausympäristö kahteen pääleiriin: testipalvelimeen, jossa on yleistason varmistustestit, ja projektikohtaisiin testeihin. Se varmistaa automaattisesti, että Mirumin tuottamat palvelut ovat hyvien käytäntöjen mukaisia. Testipalvelimen ylläpidosta vastaa järjestelmätestaaja, jonka päätyökaluksi testipalvelin olisi tarkoitettu. Toinen ympäristö koostuu projektikohtaisista testeistä. Siinä varmistetaan verkkosivujen toiminnallisuudet. Verkkokehittäjien on tarkoitus huolehtia verkkosivukohtaisista testien ylläpidosta. Heille pidetään työpaja Selenium-testien koodaamisesta Mocha-testikehikolla, joka

on JavaScriptillä kirjoitettu testikehikko. Mocha-testikehikko on myös huomattavasti yleisempi kuin esimerkiksi Nightwatch-testikehikko. Verkkokehittäjät ovat varmasti innokkaampia opiskelemaan testikehikkoa, jonka käyttö on keskivertoa suositumpaa. Työpajassa kirjoitettuja automaattisia testejä koodattaisiin valmiiksi oikeisiin projekteihin, jotta kaikki verkkokehittäjät kohtaisivat käytännössä ongelmatyyppejä, joita Selenium-testien koodaamisessa voi tulla vastaan. Testipalvelimelle on tarkoitus tallentaa erityyppisten toiminnallisuuksien testitiedostoja, jolloin verkkokehittäjät voivat tarvittaessa kopioida testejä yhteisestä säilöstä omiin projekteihinsa.

Insinöörityö nosti esiin mahdollisuuden, että Mirumin kaikille projekteille tehtäisiin vähintään tietyt testit. Organisaation orientoituminen testaamisen painottamiseen tapahtuu hiljalleen. Muutamia merkkejä tästä on jo olemassa. Yksi esimerkki on, että palautekyselyn tuloksia on pyydetty esitettävän myös koko Mirumin Helsingin toimiston henkilökunnalle. Tällöin koko organisaatiota voidaan sivistää testauksen periaatteista ja huonoista käytännöistä, jotka kasvattavat testivastaisuutta. Muita esimerkkejä organisaation suhtautumisen muutoksesta on suunniteltu työpaja verkkokehittäjille ja suunnitelma rakentaa testipalvelimen prototyyppi. Kolmas kehitysaskel on, että Mirumilla on nyt yksi testaaja, jonka tehtäviin kuuluu verkkokehittäjien testaustaakan keventäminen. Testauspalvelimen prototyypin valmistuttua laajennettua testausta ruvetaan todennäköisesti myymään lisäpalveluna jatkuvan kehityksen projekteille. Testipalvelimen käyttöönoton jälkeen ruvetaan myös tutkimaan mahdollisuutta, että jatkuva integraatio otetaan käyttöön Mirumin ylläpitämille verkkosivuille. Haasteena on, miten lisäpalvelua markkinoidaan asiakkaille, jotka kokevat jo maksavansa testaamisesta.

## Lähteet


1 Tolvanen, Perttu. 2015. Läski ylläpitosopimus. Verkkodokumentti. Vierityspalkki. <http://vierityspalkki.fi/2015/10/29/laski-yllapitosopimus/>. Luettu 13.12.2015.

2 Mustafa, K & Khan, R. 2007. Software Testing Consepts and Practices. Oxford: Alpha Science International.

3 Ficher, Darlene & Wisniewski, Jeff. 2015. Web Development Best Practices. Online Magazine 1/2015, s. 57–60.

4 Christie, James. The seductive and dangerous V Model. Testing experience magazine 12/2008.

5 Kohan, Bernard. 2010. What is a Content Management System (CMS)? Verkkodokumentti. Comentum. <http://www.comentum.com/what-is-cms-content-management-system.html>. Luettu 19.12.2015.

6 Plugin Directory. 2015. Verkkodokumentti. Wordpress-yhteisö. Wordpress.org. <https://wordpress.org/plugins/>. Luettu 12.12.2015.

7 Drupal build something amazing. 2015.Verkkodokumentti. Drupal.org <https://www.drupal.org/>. Luettu 12.12.2015.

8 SimpleTest Overview. 2014. Verkkodokumentti. Drupal.org. <https://www.drupal.org/node/394888>. Luettu 15.12.2015.

9 Testing (D7 and D8) / SimpleTest (D6). 2015. Verkkodokumentti. Drupal.org.<https://www.drupal.org/simpletest>. Luettu 29.12.2015.

10 Testing (simpletest) Tutorial (Drupal 7). 2015. Verkkodokumentti. Drupal.org. <https://www.drupal.org/simpletest-tutorial-drupal7>. Luettu 19.12.2015.

11 Novothy, Michael. 2015. The best tests for Wordpress. Verkkodokumetti. WP test. <http://wptest.io/>. Luettu 16.1.2016.

12 Lewis, Joseph & Moscovitz, Meitar. 2009. AdvancED CSS. New York: Springer-Verlag.

13 Rocheleau, J. 2015. 10 Useful Fallback Methods For CSS and Javascript. Verkkodokumentti. Honkiat. <http://www.hongkiat.com/blog/css-javascript-fallback-methods/>. Luettu 13.12.2015.

14 Hosszu, Kalman. 2010. Browser class. Verkkodokumetti. Drupal.org. <https://www.drupal.org/project/browserclass>. Luettu 21.12.2015.

15 Flirtman, Maximiliano. 2013. Programming the Mobile Web. Sebastopol: O'Reilly Media.

16 Wilson, Chris & Kinlan Paul. 2013. Touch And Mouse - Together Again For The First Time. Verkkodokumentti. HTML5 Rocks. <http://www.html5rocks.com/en/mobile/touchandmouse/>. Luettu 14.12.2015.

17 Hemphill, Kenny. 2015. What's a Retina display? What's a Retina HD display? Which Apple devices have Retina displays? And are they worth the money? Verkkodokumetti. MacWorld. <http://www.macworld.co.uk/feature/apple/what-retina-hd-display-are-they-worth-money-apple-3466732/>. Luettu 29.12.2015.

18 Rouse, Margaret. 2014. QHD (quad high definition). Verkkodokumentti. Whatis. <http://whatis.techtarget.com/definition/QHD-quad-high-definition>. Luettu 29.12.2015.

19 Reese, Ryan. 2015. Media Queries: Width vs. Device Width. Verkkodokumentti. Sitepoint. <http://www.sitepoint.com/media-queries-width-vs-device-width/>. Luettu 12.12.2015.

20 Nanji, Ayaz. 2014. Mobile Trends: Most Popular Phones, screen sizes, and Resolution. Verkkodokumentti. Marketin Profs. <http://www.marketingprofs.com/charts/2014/25740/mobile-trends-most-popular-phones-screen-sizes-and-resolutions/>. Luettu 21.12.2015.

21 Koch, Peter-Paul. 2010. Combining meta viewport and media queries. Verkkodokumentti. Quirksmode. <http://www.quirksmode.org/blog/archives/2010/09/combining_meta.html>. Luettu 05.01.2016.

22 CSS Pixel Widths. Verkkodokumentti. Canbike.org. <http://www.canbike.org/CSSpixels/>. Luettu 12.12.2015.

23 Leenheer, Niels. 2015. Specification. Verkkodokumentti. HTML5 Test. <https://html5test.com/about.html>. Luettu 16.12.2015.

24 Firtman, Maximiliano. 2016. HTML5 compatibility on mobile and tablet browsers with testing on real devices. Verkkodokumentti. Mobile HTML5. <http://mobilehtml5.org/>. Luettu 29.2.2016.

25 Garsiel, Tali & Irish, Paul. 2011. How Browsers Work: Behind the scenes of modern web browsers. Verkkodokumentti. HTLM5 Rocks. <http://www.html5rocks.com/en/tutorials/internals/howbrowserswork/#The_rendering_engines_threads>. Luettu 10.1.2016.

26 Deveria, Alexis & Scroors, Lennart. 2015. About. Verkkodokumentti. CanIuse. <http://caniuse.com/#info_about>. Luettu 5.1.2016.

27 Perrier, Jean-Yves. 2015. @media. Verkkodokumentti. Mozilla Developers Network. <https://developer.mozilla.org/en-US/docs/Web/CSS/@media>. Luettu 9.1.2016.

28 Kuisma, Kimmo. 2012. Vieraskynä: Päätelaitteiden monimuotoisuus hallintaan Mobile first -strategialla. Verkkodokumentti. Vierityspalkki. <http://vierityspalkki.fi/2012/04/12/vieraskyna-paatelaitteiden-monimuotoisuus-hallintaan-mobile-first-strategialla/>. Luettu 8.1.2016.

29 Lepage, Pete. 2015. Responsive web design basics. Verkkodokumentti. Google Developers. <https://developers.google.com/web/fundamentals/design-and-ui/responsive/fundamentals/?hl=en>. Luettu 20.12.2015.

30 Hicks, William. 2015. Building a Mobile Device Testing and Development Lab @ the UNT Libraries. Library Technology Reports. 10/2015, s. 37–43.

31 .Mobile Application Testing - Emulators v/s Simulators & Real Mobile Device - Comparison & Usage. 2014. Verkkodokumentti. EVideotuition. Slideshare. <http://www.slideshare.net/videotuition/mobile-application-testing-emulators-vs-simulators-real-mobile-device-comparison-usage>. Luettu 11.12.2015.

32 Testing Your Responsive Web Design with Chrome Developer Mobile Emulator Tool. Verkkodokumentti. Joostrap. <http://www.joostrap.com/support/tutorials-videos/204-testing-your-responsive-web-design-with-chrome-developer-mobile-emulator-tool>. Luettu 12.12.2015.

33 Mobile Application Testing - Emulators v/s Simulators & Real Mobile Device - Comparison & Usage. 2014. Verkkodokumentti. EVideotuition. Slideshare. <http://www.slideshare.net/videotuition/mobile-application-testing-emulators-vs-simulators-real-mobile-device-comparison-usage>. Luettu 11.12.2015.

34 Fielding, Jonathan. 2014. Beginning responsive web design with HTML5 and CSS3. New York: Heinz Weinheimer.

35 3 Best Android Emulators for Mac OS / Macbook |Run and install Android apps on your Mac OS X, Macbook Air/Pro. Verkkodokumentti. Techapple. <http://techapple.net/2014/05/3-best-android-emulators-for-mac-os-macbook-run-and-install-android-app-on-your-mac-os-x-macbook-airpro/>. Luettu 3.1.2016.

36 Herken, Daniel. 2015. Testing For And With Windows Phone. Verkkodokumentti. Smashing Magazine. <https://www.smashingmagazine.com/2015/05/testing-for-windows-phone/>. Luettu 20.2.2016.

37 About the project. 2014. Verkkodokumentti. Helsinki Open devicelab. <http://devicelab.fi/>. Luettu 12.2.2016.

38 Browsersync Documentation. 2016. Verkkodokumentti. Browserssync. <https://www.browsersync.io/docs/>. Luettu 20.2.2016.

39 Osbourne, Shane. 2015. Dynamic proxy setting. Verkkodokumentti. GitHub.<https://github.com/BrowserSync/UI/issues/6>. Luettu 5.3.2016.

40 Requirements. Verkkodokumentti. Wraith. <http://bbc-news.github.io/wraith/os-install.html >. Luettu 24.3.2016.

41 Kligman, Kate. 2015. Using Wraith for Visual Regression Testing. Verkkodokumentti. Patheon. <https://pantheon.io/docs/guides/visual-diff-with-wraith/>. Luettu 20.2.2016.

42 Carmi, Adam. 2014. Selenium Based Visual Test Automation. Verkkodokumentti. Slideshare. <http://www.slideshare.net/adamcarmi/selenium-based-visual-test-automation?next_slideshow=1>. Luettu 1.2.2016.

43 Platforms Supported by Selenium. 2016 Verkkodokumentti. SeleniumHQ. <http://www.seleniumhq.org/about/platforms.jsp#operating-systems>. Luettu 22.2.2016.

44 . Developer Guide - Overview. 2015. Verkkodokumentti. NightwatchJS. <http://nightwatchjs.org/guide>. Luettu 20.2.2016.

45 Leyba, Jason. 2016. selenium-webdriver. Verkkodokumentti. NPM. <https://www.npmjs.com/package/selenium-webdriver>. Luettu 20.2.2016.

46 Node.js. Verkkodokumentti. Browserstack. <https://www.browserstack.com/automate/node>. Luettu 2.2.2016.

47 Uhlinger, Jay. 2014. Is Implementing Continuous Integration Worth It? Verkkodokumentti. Acquia. <https://www.acquia.com/blog/implementing-continuous-integration-worth-it>. Luettu 15.2.2016.

48 Introducting Mirum: A Modern global company. Verkkodokumentti. Mirum Agency. <https://www.mirumagency.com/blog/introducing-mirum-modern-global-company>. Luettu 25.2.2016.

49 About. Verkkodokumentti. Mirum Agency. <https://www.mirumagency.com/about>. Luettu 1.1.2016.

50 Company.nokia.com. Verkkodokumentti. Vierityspalkki. <http://vierityspalkki.fi/julkaisut/company-nokia-com/>. Luettu 24.2.2016.

51 Mirum Agency. Verkkodokumentti. Mainostajan hakemisto. <http://mainostajanhakemisto.fi/organisaatio/mirum-agency/>. Luettu 21.2.2016.

52 Tolvanen, Perttu. 2015. Avoin lähdekoodi ei ole avointa, jos sitä ei ole jaettu. Verkkodokumentti. Vierityspalkki. <http://vierityspalkki.fi/2015/09/29/avoin-lahdekoodi-ei-ole-avointa-jos-sita-ei-ole-jaettu/>. Luettu 22.2.2016.

53 Strauss, Roy. 2010. Developing Effective Website: A Project Managers Guide. Burlington: Focal Press.

54 Mobile/Tablet Top Browser Share Trend. Verkkodokumentti. Netmarketshare. <https://www.netmarketshare.com/browser-market-share.aspx?qprid=1&qpcustomb=1>. Luettu 1.3.2016.

55 Global Internet Report 2015. Verkkodokumentti. Internet Society. <http://www.internetsociety.org/globalinternetreport/assets/download/IS_web.pdf> Luettu 3.12.2015.

# Front-end testing methods -kyselyn tulokset

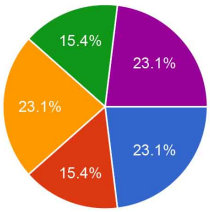
onni.aaltonen@mirumagency.com
Edit this form

# 13 responses

View all responses  Publish analytics

## Summary

### How are you hosting your local development environment?

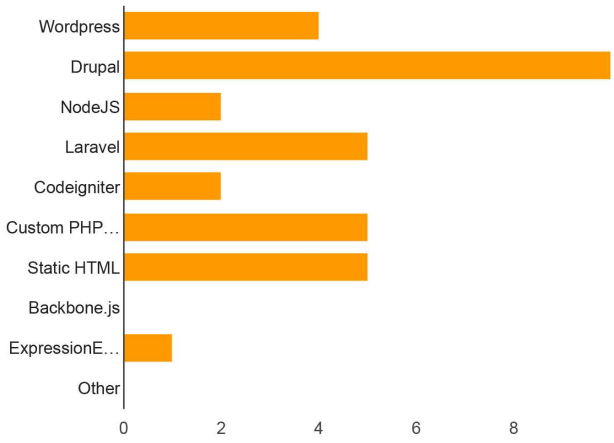



| | | |
|---|---|---|
| Debian VM | 3 | 23.1% |
| Vagrant VM | 2 | 15.4% |
| Native os X | 3 | 23.1% |
| Mamp | 2 | 15.4% |
| Other | 3 | 23.1% |

### How do you see yourself more as a front end or backend developer?


| | | |
|---|---|---|
| Frontend | 11 | 84.6% |
| Backend | 8 | 61.5% |
| Other | 1 | 7.7% |

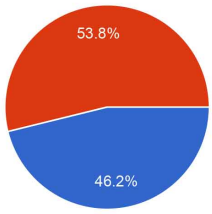
### What platforms or frameworks you develop mostly?




| | | |
|---|---|---|
| Laravel | **5** | 41.7% |
| Codeigniter | **2** | 16.7% |
| Custom PHP projects | **5** | 41.7% |
| Static HTML | **5** | 41.7% |
| Backbone.js | **0** | 0% |
| ExpressionEngine | **1** | 8.3% |
| Other | **0** | 0% |

## Resourcing

**What is the average length of the projects that you participate?**

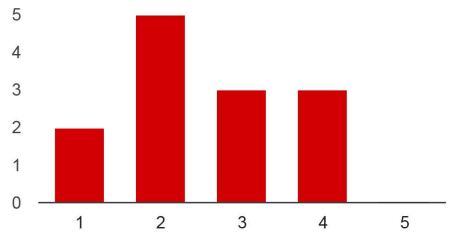



| | | |
|---|---|---|
| less than 1 month | **6** | 46.2% |
| 1 to 6 months | **7** | 53.8% |
| 6 to 12 months | **0** | 0% |
| Other | **0** | 0% |

**How much time do you spend on testing per project?**

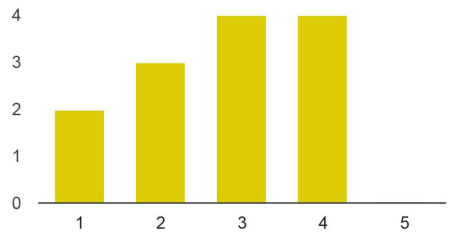
10
2
2%
3%
4%
15%
5%
10%

**Have you been informed about the amount of testing promised to the client?**




| | | |
|---|---|---|
| None: 1 | **2** | 15.4% |
| 2 | **5** | 38.5% |
| 3 | **3** | 23.1% |
| 4 | **3** | 23.1% |
| Fully: 5 | **0** | 0% |

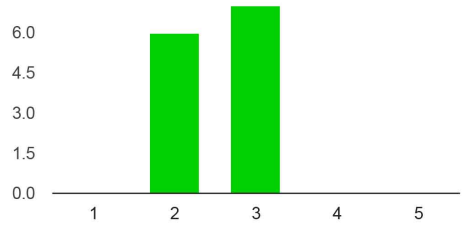
**Have you been informed about the testing methods promised to the client?**




| | | |
|---|---|---|
| None: 1 | **2** | 15.4% |
| 2 | **3** | 23.1% |

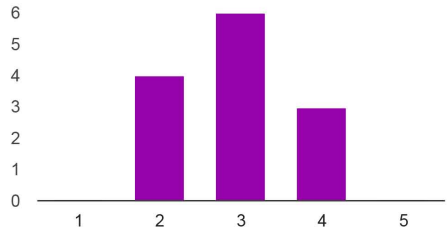
| | | |
|---|---|---|
| 3 | **4** | 30.8% |
| 4 | **4** | 30.8% |
| Fully: 5 | **0** | 0% |

**Is time allocated enough to cover testing?**




| | | |
|---|---|---|
| None: 1 | **0** | 0% |
| 2 | **6** | 46.2% |
| 3 | **7** | 53.8% |
| 4 | **0** | 0% |
| Fully: 5 | **0** | 0% |

**In your opinion, is the project team aware of their roles when testing in their own field of expertise?**




| | | |
|---|---|---|
| None: 1 | **0** | 0% |
| 2 | **4** | 30.8% |
| 3 | **6** | 46.2% |
| 4 | **3** | 23.1% |
| Fully: 5 | **0** | 0% |

**General comments about project managing and Quality Assurance**

**methods?**

We should be doing more testing, but this requires some strong project management to protect the allocated testing time. Although, unfortunately this usually ends up being me fighting to keep this time for testing and not last minute changes.

We most certainly do not test enough on all projects. Recently thugh more and more projects have proper testing allocated - but when it comes to overall quality we could do much better. For example peer reviews during development.

Not usually thorough

Scrum method helps testing, because it happens during the development process.

No.

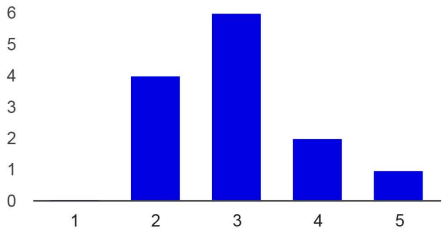
A lot of projects just have a list of supported browser versions and mobile devices. Other than that, there's just a generic requirement for "testing", which isn't specified in more detail. However, most projects don't have special features or integrations that would require testing, so basic UI, front-end and mobile device testing covers most of the bugs.

Generally yes, although more testing can never be bad.

Is usually not enough time to test it thoroughly. Can always use more time.

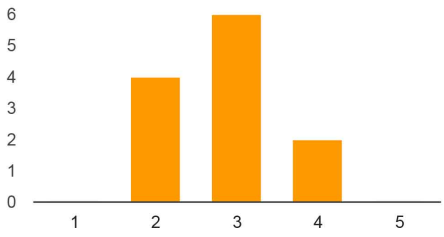
## Mobile devices

### Apple iOS devices (Safari, Chrome)



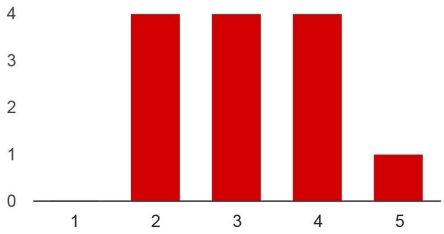

| | | |
|---|---|---|
| Never: 1 | **0** | 0% |
| 2 | **4** | 30.8% |
| 3 | **6** | 46.2% |
| 4 | **2** | 15.4% |
| All the time: 5 | **1** | 7.7% |

### Android devices (Chrome, Android Browser, Opera Mini)




| | | |
|---|---|---|
| Never: 1 | **0** | 0% |
| 2 | **4** | 33.3% |
| 3 | **6** | 50% |
| 4 | **2** | 16.7% |
| All the time: 5 | **0** | 0% |

**Lumia devices (Internet Explorer)**




| | | |
|---|---|---|
| Never: 1 | **0** | 0% |
| 2 | **4** | 30.8% |
| 3 | **4** | 30.8% |
| 4 | **4** | 30.8% |
| All the time: 5 | **1** | 7.7% |

**Have you noticed any specific device/simulator and browser combination that causes bugs?**

Apple iOS devices implement certain features in a weird way, which causes bugs/missing features that can be hard to fix.

Apple iPad (v4 maybe and OS version 8) with native Safari caused a lot of headache

iOS can be buggy or lack implementation. Internet explorer is usually a headache.

Generally Windows with IE10 or whatever browser lumias use.

iOS Safari has its own quirks, it seems.

Android browser nearly always has some issues. Also mobile safari on IOS 8 has some odd 'features'.

No

### How many mobile devices do you usually test a site or project with?

ios simulator android phones windows phones phones

I usually test with 2-4 different mobile devices.

2-3

5-10 - as many with real devices as possible, although the IOS emulator is excellent.

Natively 4 mobile phones and around 3 tablets.

about 10, native machine and simulator

Depengins on project and allocated testing time 1-6 different devices. Using xcode simulator, browserstack, physical devices

1-4 (native)

I usually test with BrowserStack or similar. I also use native devices (at least an iPhone and an iPad, and an Android phone and one Android tablet. Sometimes a Lumia is also included) and virtual machines for internet explorer.

Around 3-5, using browserstack. 1-3 using physical devices.

XCode simulator, native iPad and Android phones.

IOS simulator on mac, browserstack, and maybe one or two lumias, android devices and few apple mobile devices.

### Which testing tools do you use, especially with mobile devices?

simulators.

xip.io same local network browser stack simulators

Browserstack, do we have accounts to other serveces?

Device Lab VirtualBox Wraith Native browsers

BrowserStack, device lab. I do not use mobile SDK emulators for testing.

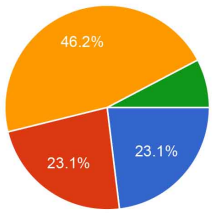
browserstack, device lab

Browserstack, "device lab", browser development tools.

browserstack, wraith, virtualbox (Modern.ie)

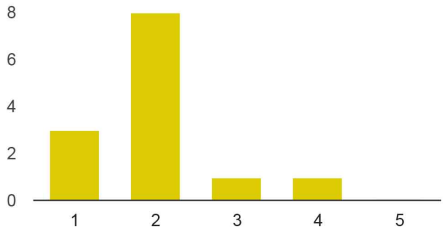
Browserstack, VirtualBox

### What is the best channel to share frontend bugs to devices?




## Usage of the device lab

### How often do you use the device lab?




| | | |
|---|---|---|
| Never: 1 | **3** | 23.1% |
| 2 | **8** | 61.5% |
| 3 | **1** | 7.7% |
| 4 | **1** | 7.7% |
| All the time: 5 | **0** | 0% |

### Do you have any comments about the device lab?

Is it working? Chargers?

I usually bring the devices to my desk..... It's just easier that way.

We need to get usb hubs to get all the devices powered.

Idea is good but I have use devices separately, since the computer didn't work for some reason. Also not all devices are connected with charger or data cable. And the testing session usually starts by searching some devices in the office area.

it's never up and running

Sounds like a great idea. Would love to try it.

Always best to test on a physical device!

Devices missing or not charged, chargers missing... Device lab covered in unrelated junk.

QA is a very different department and developers shouldn't be testign their own code after preliminary testing during development.

I use it always when testing. I haven't been involved in so many projects yet so I haven´t done so much testing

## Sharing information of potential testing tools

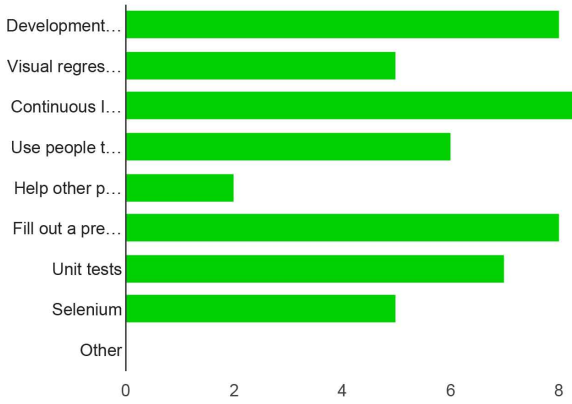
### Do you use or know any testing tools that should be introduced to our developers?

PHPUnit, CasperJS

Litmus for email/newsletter testing.

End-to-end testing could be automated with PhantomJS, Selenium and similar tools.

### Are you interested in learning to use these testing methods:




| | | |
|---|---|---|
| Development testing (Test while coding) | **8** | 72.7% |
| Visual regression tests | **5** | 45.5% |
| Continuous Integration (Automatic test protocols when code is being deployed) | **9** | 81.8% |
| Use people to cover testing | **6** | 54.5% |
| Help other people to cover testing | **2** | 18.2% |
| Fill out a pre launch check list of tested browsers | **8** | 72.7% |
| Unit tests | **7** | 63.6% |
| Selenium | **5** | 45.5% |
| Other | **0** | 0% |

### Any other comments about the topic? What would you like to improve?

**How would you make sure that testing is done effectively?**

I would like to do more automated tests, like behavioural specs and unit tests.

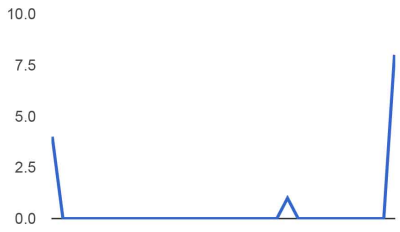
As well as technical testing, we should be testing other team members work. Not just testing from a technical perspective, but also usability and concept.

We have a lot of good tools and testing methods available, but we're not really integrating these into our projects and development processes unless the client actually demands it. Other than that, it only happens when the developer has extra time to play around with a specific testing tool. It might help if the testing requirements and methods were clearly spelled out in the project specs or contracts with the client. We should also inform project managers more about testing and how to sell it to clients.

Have a QA team.

We need a dedicated testing team.

## Number of daily responses



# Front-end testing methods -kyselylomake



Edit this form

## Frontend testing methods

This is a short survey on device testing. Data from the survey will be used in a bachelor's thesis to evaluate testing methods utilized in the company. The premise of this study is that it's possible to streamline the company's testing tools and methods even with developers using different technologies and frameworks. Please answer this survey based on your own experience and opinions. You may answer in English or Finnish.

* Required

**How are you hosting your local development environment? ***

- ○ Debian VM
- ○ Vagrant VM
- ○ Native os X
- ○ Mamp
- ○ Other:

**How do you see yourself more as a front end or backend developer?**

Select both if you develop both technologies equally

- ☐ Frontend
- ☐ Backend
- ☐ Other:

**What platforms or frameworks you develop mostly?**

- ☐ Wordpress
- ☐ Drupal
- ☐ NodeJS
- ☐ Laravel
- ☐ Codeigniter
- ☐ Custom PHP projects
- ☐ Static HTML
- ☐ Backbone.js
- ☐ ExpressionEngine
- ☐ Other:

## Resourcing

Are you aware of the requirements for the end-product?

**What is the average length of the projects that you participate?**

- ◯ less than 1 month
- ◯ 1 to 6 months
- ◯ 6 to 12 months
- ◯ Other:

**How much time do you spend on testing per project?**
Give an estimation of time spent on frontend testing. Give your answer in percentages (%).

**Have you been informed about the amount of testing promised to the client?**

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| None | ◯ | ◯ | ◯ | ◯ | ◯ | Fully |

**Have you been informed about the testing methods promised to the client?**
Is load or security testing needed?

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| None | ◯ | ◯ | ◯ | ◯ | ◯ | Fully |

**Is time allocated enough to cover testing?**

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| None | ◯ | ◯ | ◯ | ◯ | ◯ | Fully |

**In your opinion, is the project team aware of their roles when testing in their own field of expertise?**
Including usability testing, browser testing, functional testing, user paths

| | 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|
| None | ◯ | ◯ | ◯ | ◯ | ◯ | Fully |

**General comments about project managing and Quality Assurance methods?**
Are confident that your work has been tested thorough?

## Mobile devices

How often do you discover bugs with mobile device browsers?

**Apple iOS devices (Safari, Chrome)**

1 2 3 4 5

Never ◯ ◯ ◯ ◯ ◯ All the time

**Android devices (Chrome, Android Browser, Opera Mini)**

1 2 3 4 5

Never ◯ ◯ ◯ ◯ ◯ All the time

**Lumia devices (Internet Explorer)**

1 2 3 4 5

Never ◯ ◯ ◯ ◯ ◯ All the time

**Have you noticed any specific device/simulator and browser combination that causes bugs?**

**How many mobile devices do you usually test a site or project with?**
Please mention if you used simulator, virtual machine or native machine.

**Which testing tools do you use, especially with mobile devices?**
E.g. xip.io, browserstack, device lab, simulators, Wraith

**What is the best channel to share frontend bugs to devices?**
Slack, Tech meetings, Other?

◯ Slack

- ◯ Tech meetings
- ◯ Spreadsheet
- ◯ Other:

## Usage of the device lab

**How often do you use the device lab?**

1 2 3 4 5

Never ◯ ◯ ◯ ◯ ◯ All the time

**Do you have any comments about the device lab?**
What are the reasons to use it or not using it?

## Sharing information of potential testing tools

**Do you use or know any testing tools that should be introduced to our developers?**
I.e. potential methods or tools that everybody could use for larger or smaller projects

**Are you interested in learning to use these testing methods:**
Choose all that apply

- ☐ Development testing (Test while coding)
- ☐ Visual regression tests
- ☐ Continuous Integration (Automatic test protocols when code is being deployed)
- ☐ Use people to cover testing
- ☐ Help other people to cover testing
- ☐ Fill out a pre launch check list of tested browsers
- ☐ Unit tests
- ☐ Selenium
- ☐ Other:

**Any other comments about the topic? What would you like to improve? How would you make sure that testing is done effectively?**

Please give some general feedback about the topic.

Submit

*Never submit passwords through Google Forms.*

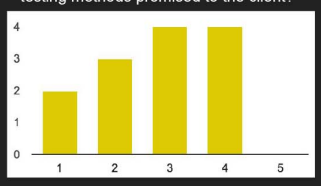
**Testing survey results -diaesitys**

# Testing survey results

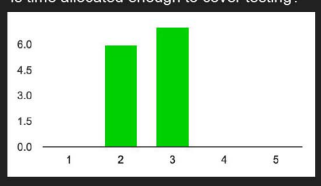
## Average testing time per project 6.3%

Scale of answers from 2–15%

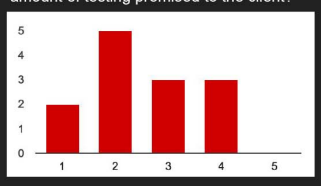
Have you been informed about the testing methods promised to the client?




Is time allocated enough to cover testing?

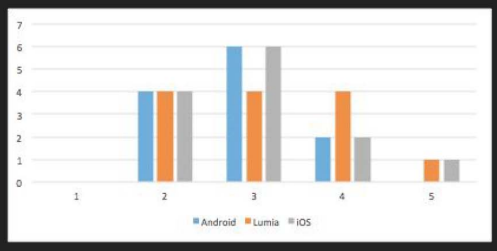



Have you been informed about the amount of testing promised to the client?



## Most common non compatible devices

Generally watch out for Safari Mobile and IE for Lumia




## Current testing tools

- Browserstack
- iOS Simulator
- Modern.ie
- Wraith

## General comments

- “We should have a testing team”
- “Testing methods are not specified
- “ We are not selling the testing tools to clients”
- “I want more automated tests”
- All results

## Solutions

- Development testing
- Continuous integration

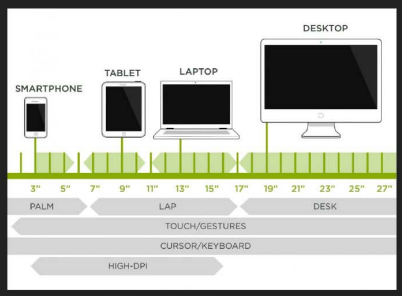
## Development testing with browsersync.io

- For front-end coding phase
- No more constant browser resizing
- Xamarin Android player
- iOS Simulator
- Native other phones (Lumia's)

- Demo

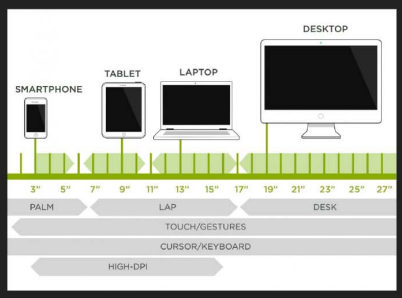
## Simulator device per breakpoint

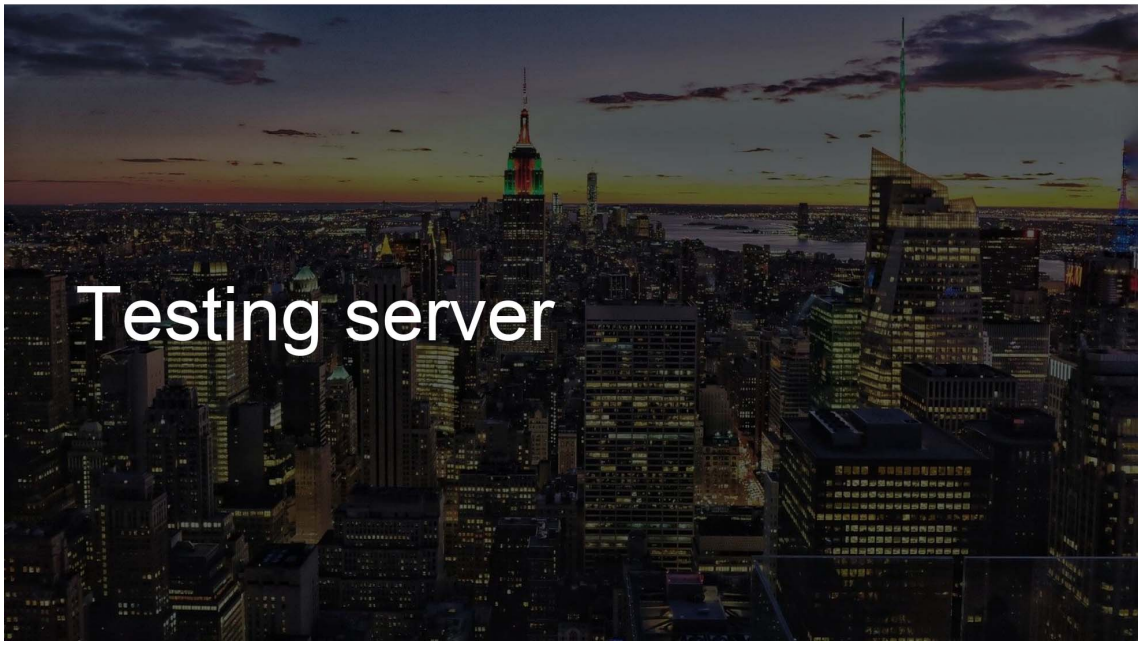
- Iphone 4s for Mobile (Viewport width 320)
- Nexus 7 for Tablet (Viewport width 768)
- IE 10 for Desktop (Viewport width 1024)



## Reloading with Browsersync from

- UI (Global installation)
- Grunt plugin
- Gulp plugin



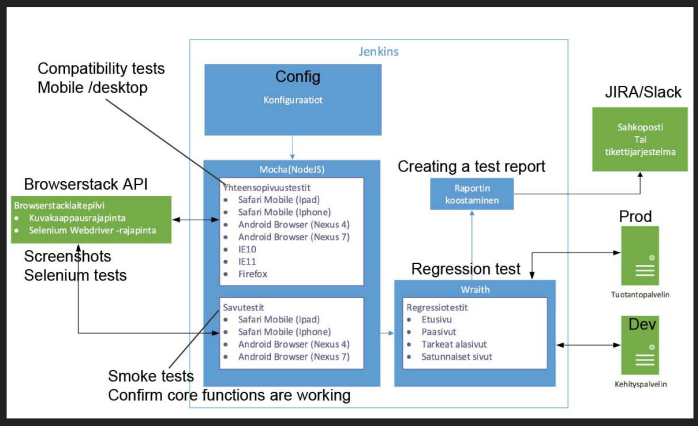
## Automate

- Compatibility testing (Browserstack Screenshots API, Wraith-Selenium)
- Functionality testing (Browserstack Automate API)
- Regression testing (Wraith, Wraith-Selenium)
- Performance reviews (Gtmetrix API)

## Goals

- Automate our testing
- Have a minimum standard of testing
- To make testing easy and fast with our own test library
- To have documentation of our testing
- Make sure that core functionalities of sites are working
- Delicate the test review away from developers

# Continuous Integration




# Continuous Integration

Results of tests would would be:

- Issue ticket
- Compatibility galleries
- Compatibility diffs
- Regression galleries
- Functionality videos
- Passed tests

## Decisions

For Testing server

- Frameworks? (Javascript?, NightwatchJS, Selenium-webdriver, Mocha, other)?
- API's (Crossbrowsertesting or Browserstack)
- Hosting of test files (Project repo's or in one general repo for all tests?)
- Version control build triggers? From beanstalk?
- Manual test build?
- What should we test?
- What stages would you like to automate types?
- Submit your opinions

## Presentation debriefing -kyselylomake

# Presentation debriefing

Write your honest option's about the presented solutions.

1. **Could you see yourself using the presented front-end development method with Browsersync?**
   *Mark only one oval.*
   - ◯ Yes
   - ◯ No

2. **Do you have any concerns about the testing method?**
   Did you understand the benefits of using the method?

   .....

3. **Are you interested of writing Selenium tests?**
   *Mark only one oval.*
   - ◯ Yes
   - ◯ No
   - ◯ Maybe
   - ◯ Other: .....

4. **What would you like to automate with Jenkins?**
   Test phases, db syncing, Maintenance tasks, Regression tests? Compatibility tests?

   .....

5. **Are you interested to help setup Jenkins?**
   *Mark only one oval.*
   - ◯ Yes
   - ◯ No

6. **General comments**
   What would you like to avoid with the automatic tests? What would you like to do with Jenkins?

   .....

   .....

   .....

   .....

   .....

**Presentation debriefing -kyselyn tulokset**

Presentation debriefing

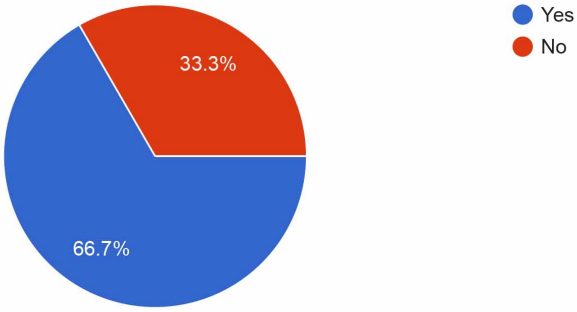
QUESTIONS RESPONSES 3

3 responses

SUMMARY INDIVIDUAL

Accepting responses

Could you see yourself using the presented front-end development method with Browsersync?
(3 responses)



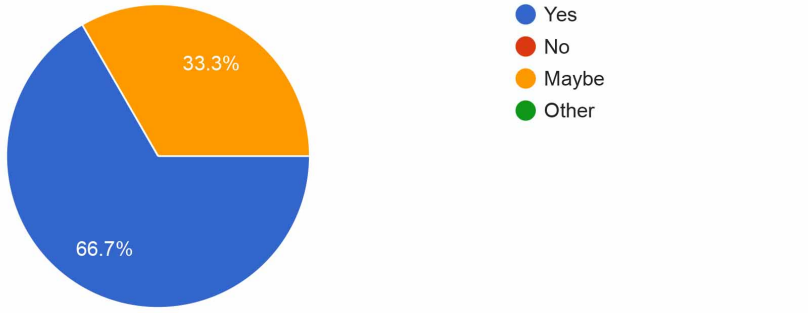

Do you have any concerns about the testing method? (3 responses)

Availability of screen space for all the device simulators, and availability / status of actual testing devices.

I understood the benefits. Developing this way is super slow, if after all changes you test "all" devices. Rather do quick test on the browser and more exrtencive testing on different devices later. I guess it's more like "what fits your way of working". I don't think we should all have same way of working, but offcource same result (=having tests in our project and tested product)

Nope, Browsersync is great!

## Are you interested of writing Seleni



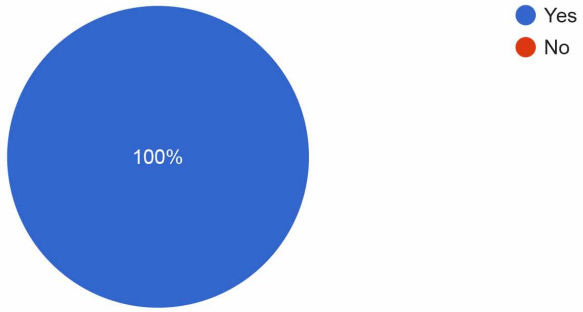

## What would you like to automate with Jenkins? (3 responses)

Maintenance tasks, regression and compatibility tests

Depending on the project. Basically everything that can be automated; unit tests, e2e tests, nightly db dump from production to dev etc. etc.

All written tests for a project

## Are you interested to help setup Jenkins? (3 responses)




## General comments (3 responses)

It should be easy to set up and go through the results. A possible risk is that if the tests end up being a huge pain (confusing to set up, incompatible with some types of projects, too many false negatives or flooding the developers with too many emails, etc), people will just ignore the whole process and come up with their own. Which would mean our unified testing process would just stagnate and go unused.

Tests are always code that needs maintenance just like the code itself. Adding tests just because we want to be able to say "yes we test" is not a valid reason, it will add the development time and it will add the maintenance costs (because on every change we need to also update tests). So I'm 100% behind testing, but we should automate the parts of testing that makes sense.

Not sure, let's see what jenkins can do!