Haaga-Helia
ammattikorkeakoulu Oy

Nykyaikaisen työkalupakin kokoaminen front end -puolella aloitteleville ohjelmoijille

Kristina Wiik

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2018

Tekijä(t)
Kristina Wiik

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön otsikko
Nykyaikaisen työkalupakin kokoaminen front end -puolella aloitteleville ohjelmoijille

Sivu- ja liitesivumäärä
109 + 6

Tutkimuksen tavoitteena on selvittää mitä front end -kehityksessä käytettäviä työkaluja kannattaa opetella tulevaisuuden työtä ajatellen. Tutkimuksen kohteena on Suomessa toimivat kaiken kokoiset ohjelmistoyritykset. Tutkimuksessa haastatellaan ohjelmistoyritysten front end -kehityksestä vastaavia henkilöitä. Tutkimustulosten pohjalta kootaan nykyaikainen työkalupakki aloitteleville ohjelmoijille. Tarkoituksena on, että ohjelmoija voi opetella niitä työkaluja, joita hänen tulevaisuuden työssään mahdollisesti käytetään.

Työssä käsitellään front end -kehitykseen liittyviä työkaluja, ja erityisesti sellaisia, joiden avulla voidaan rakentaa toimiva työkalupakki. Opinnäytetyön alkaessa työstä rajattiin pois palvelinpuolen eli back end -työkalut, mutta haastattelujen jälkeen työkaluja rajattiin lisää suuren määrän vuoksi. Työstä rajattiin pois muun muassa mobiiliohjelmointiin liittyviä työkaluja, selaintyökaluja, pienempiä kirjastoja sekä ohjelmointia osittain sivuuttavia työkaluja. Lisäksi työstä rajattiin pois sellaisia koodin kirjoitusvälineitä, joita haastateltavat eivät maininneet itse käyttävänsä.

Opinnäytetyön tietoperusta koostuu kahdesta luvusta. Niistä ensimmäisessä käydään front end -kehitystä yleisesti läpi sekä tarkastellaan front end -kehityksen tulevaisuuden näkymiä. Toisessa luvussa esitellään ohjelmistoyrityksissä käytössä olevia työkaluja, jonka lisäksi jokaisesta työkalukategoriasta kerrotaan yleisellä tasolla. Työkalut on pyritty esittelemään niin, että niihin on helppo tutustua ja ymmärtää niiden tarkoitus.

Tutkimuksessa käytettiin kvalitatiivista eli laadullista tutkimusmenetelmää. Tutkimukseen osallistui kahdeksan (n = 8) henkilöä kahdeksasta eri ohjelmistoyrityksestä. Tutkimusaineistoa kerättiin teemahaastatteluilla, jotka pidettiin 4.6 – 2.7.2017 välisenä aikana. Aineistot analysoitiin aineistolähtöisellä sisällönanalyysillä.

Tuloksista ilmeni, että front end -ohjelmointia aloittaessa on hyvä opetella ensin JavaScriptin, HTML:n ja CSS:n perusteet. Perusteiden osaaminen koskee myös tulevaisuutta, johtuen työkalujen tiheästä vaihtelusta. Vasta perusteiden jälkeen kannattaa siirtyä JavaScript-kehyksiin. Lisäksi taustalla tapahtuva siirtymä komponenttipohjaiseen kehitykseen kannustaa valitsemaan kehyksen, joka tukee kyseenomaista kehitystapaa. Haastateltavien yrityksissä käytetään samoja työkaluja kuin muualla maailmassa, joten tärkeimmiksi työkalun valintakriteereiksi nousevat suosio ja hyvät kokemukset.

Asiasanat
Web-ohjelmointi, ohjelmointiympäristö, työkalut, ohjelmistokehitys, ohjelmistoyritykset

Sisällys

1 Johdanto 1
1.1 Työn tavoite ja rajaus 1
1.2 Työn rakenne 2
1.3 Käsitteet 3
2 Front end -kehitys 5
3 Front end -työkalut 7
3.1 JavaScript-työkalut 7
3.1.1 Sovelluskehykset ja kirjastot 7
3.1.2 Kääntäjät ja käännettävät kielet 14
3.1.3 Tarkistusohjelmat 16
3.1.4 Testaustyökalut 17
3.2 CSS-työkalut 20
3.2.1 CSS-sovelluskehykset ja käyttöliittymän muotoilu 20
3.2.2 CSS-esikäsittelijät ja CSS:n tyylittely 22
3.3 Hallinnointityökalut 24
3.3.1 Paketinhallintajärjestelmät 24
3.3.2 Tehtävänsuorittajat ja moduulien niputtajat 26
3.4 Muita työkaluja 28
3.4.1 Koodieditorit ja kehitysympäristöt 28
3.4.2 Versionhallintajärjestelmät 31
4 Tutkimuksen toteutus 34
4.1 Laadullinen tutkimusote 34
4.2 Aineistonkeruumenetelmät 34
4.3 Tutkimuksen aineisto 36
4.4 Aineiston analysointi 39
5 Tutkimustulokset 43
5.1 Käytössä olevat front end -työkalut 43
5.1.1 Koodin kirjoitusvälineet 43
5.1.2 JavaScript-sovelluskehykset ja -kirjastot 49
5.1.3 CSS-sovelluskehykset ja käyttöliittymän muotoilu 55
5.1.4 Kääntäjät ja käännettävät kielet 58
5.1.5 CSS:n tyylittely 61
5.1.6 Koodin tarkistaminen 64
5.1.7 Paketinhallinta 65
5.1.8 Tehtävänsuoritus ja moduulien niputus 69
5.1.9 Versionhallinta 71
5.1.10 Testaustyökalut 73

5.2 Työkalujen käyttö ja tulevaisuus 74
5.2.1 Työkalujen arvioitu käyttöikä 74
5.2.2 Työkalut tulevaisuudessa 76
5.3 Mitä haastateltavat suosittelevat opettelemaan? 80
6 Pohdinta 83
6.1 Tulosten tarkastelu 83
6.2 Tutkimuksen luotettavuus 90
6.3 Tutkimuksen eettisyys 91
6.4 Johtopäätökset, työkalupakin muodostaminen ja jatkotutkimusehdotukset 94
6.5 Opinnäytetyöprosessin ja oman oppimisen arviointi 95
Lähteet 97
Liitteet 110
Liite 1. Lista pois rajatuista työkaluista 110
Liite 2. Saatekirje 111
Liite 3. Suostumuslomake 112
Liite 4. Teemahaastattelurunko 113
Liite 5. Työkalujen käytön syyt kvantifioituna 114
Liite 6. Työkalupakki ja työkalujen väliset suhteet 115

1 Johdanto

Web-kehitys on muuttunut viime vuosina merkittävästi. Älypuhelimien ja tablettien käyttö on kasvanut huomattavasti, jonka seurauksena käyttäjien odotukset ovat muuttuneet. Tarvitaan entistä nopeampia ja responsiivisempia sovelluksia, jotka vastaavat nykypäivän tarpeisiin. Odotusten muuttuessa myös ohjelmoijien tapa ratkaista asioita on muuttunut. Aikaisemmin palvelin hoiti suurimman osan verkkosovelluksen toiminnasta, mutta muutoksen myötä kehitystä on siirretty entistä enemmän selainpuolen, eli front end -kehitykseen. (Stangarone 1.9.2015.)

Front end -kehityksen kasvun myötä web-kehitykseen tarkoitetun JavaScript-kielen ympärille on syntynyt rikkaita yhteisöjä, jotka ovat parantaneet sovelluksia entisestään rakentamalla erilaisia työkaluja (Haviv 2016, luku 1). Tänä päivänä kehityksestä tekeekin haasteellista oikeiden työkalujen valinta kehityksen tueksi, sillä erilaisia työkaluja on tarjolla lukuisia ja niitä syntyy jatkuvasti lisää (Aquino & Gandee 2016, luku 1). Kehittäjien pitää olla tietoisia eri HTML, CSS ja JavaScript-kielten versioista sekä jatkuvasti vaihtuvista alustoista. Ja vaikka mobiililaitteiden suosion kasvu on muuttamassa front end -kehityksen maailmaa, verkkosivujen tulee toimia edelleen myös isoilla ruuduilla. (Taylor & Smith 2015, luku 4.)

Erilaisten front end -työkalujen nopean kasvun ja vaihtuvuuden kuvailemiseen on muutaman vuoden sisään kehitetty termi, joka on nimeltään JavaScript-väsymys. Termi kuvailee, kuinka väsyttävää ja hankalaa kehittäjien on pysyä ajan tasalla uusista trendeistä. JavaScript-väsymys saattaa koskea ketä tahansa ohjelmoijaa, mutta etenkin aloittelevia ohjelmoijia, sillä pienenkin sovelluksen tekemiseen tarvitaan erilaisten työkalujen osaamista. (Jones 2017.)

Mistä siis tietää, mitä kannattaa opetella ja mitä ei? Tämä ajatus syntyi tutkijan omasta kokemuksesta opiskellessa alaa. Aiemmissa selvityksissä on tutkittu määrällisillä tavoilla, mitä työkaluja ohjelmoijat käyttävät maailmalla (katso esimerkiksi Nolan 29.11.2016; The State of JavaScript 2017a; Stack Overflow 2017a). Työkaluista ei ole kuitenkaan riittävästi syvällistä tietoa, kuten esimerkiksi miksi työkaluja valitaan ja mitä asioita kannattaa opetella tulevaisuuden työtä ajatellen.

1.1 Työn tavoite ja rajaus

Tämän tutkimuksen tavoitteena on selvittää laadullisin menetelmin, mitä front end -kehityksessä käytettäviä työkaluja kannattaa opetella tulevaisuuden työtä ajatellen. Tutki-

muksen kohteena on Suomessa toimivat ohjelmistoyritykset, joilla saattaa olla toimipisteitä myös ulkomailla. Tutkimuksessa haastatellaan ohjelmistoyritysten front end -kehityksestä vastaavia. Tavoitteena on saada vastaus seuraaviin kysymyksiin:

1. Mitä front end -työkaluja ohjelmistoyrityksissä käytetään ja miksi?
2. Mitkä ovat suosituimpien käytössä olevien työkalujen hyvät ja huonot puolet?
3. Mikä on front end -työkalujen käyttöikä?
4. Miltä ohjelmistoyritysten lähitulevaisuus näyttää työkalujen suhteen?
5. Mitä työkaluja, kieliä ja muita ohjelmointiin liittyviä asioita kannattaa opetella alan työtä varten?

Tutkimustulosten pohjalta kootaan nykyaikainen työkalupakki aloitteleville ohjelmoijille. Tarkoituksena on, että ohjelmoija voi opetella niitä työkaluja, joita hänen tulevaisuuden työssään mahdollisesti käytetään. Työkalupakkia voi esimerkiksi hyödyntää omassa portfoliossaan hakiessaan työpaikkaa. Työkalupakki on tarkoitettu ensisijaisesti aloitteleville ohjelmoijille, mutta siitä voivat hyötyä myös kokeneemmat ohjelmoijat, jotka haluavat päivittää työkalupakkiaan. Tutkimuksen tuloksia voivat hyödyntää myös opetusta suunnittelevat oppilaitokset sekä työkaluja projekteihin valitsevat ohjelmistoyritykset.

Tässä opinnäytetyössä käsitellään front end -kehitykseen liittyviä työkaluja, ja erityisesti sellaisia, joiden avulla voidaan rakentaa toimiva työkalupakki. Opinnäytetyön alkaessa työstä rajattiin pois pelkästään palvelinpuolen eli back end -työkalut, mutta haastattelujen jälkeen työkaluja rajattiin lisää suuren määrän vuoksi. Työstä rajattiin pois muun muassa mobiiliohjelmointiin liittyviä työkaluja, selaintyökaluja, pienempiä kirjastoja sekä ohjelmointia osittain sivuuttavia työkaluja. Lisäksi työstä rajattiin pois sellaisia koodin kirjoitusvälineitä, joita haastateltavat eivät maininneet itse käyttävänsä. Rajauksessa on pyritty ottamaan selvää parhaan mukaan, mikä työkalu liittyy mihinkin kategoriaan. Liitteessä 1 on lueteltuna haastateltavien mainitsemia työkaluja, jotka rajattiin pois jälkikäteen ja joita ei siten käsitellä tässä opinnäytetyössä.

1.2 Työn rakenne

Tämä opinnäytetyö noudattaa Haaga-Helian perinteisen raportin rakennetta tutkimustyyppiselle opinnäytetyölle, joka jakaantuu johdantoon, tietoperustaan, empiiriseen osaan ja pohdintaan. Opinnäytetyön johdannossa, eli ensimmäisessä luvussa esitellään yleisjohdanto sekä opinnäytetyön aihe, tavoite, rajaukset ja tarpeelliset käsitteet.

Opinnäytetyön tietoperusta koostuu luvuista kaksi ja kolme. Toisessa luvussa käydään front end -kehitystä yleisesti läpi sekä tarkastellaan front end -kehityksen tulevaisuuden näkymiä. Kolmannessa luvussa esitellään ohjelmistoyrityksissä käytössä olevia työkaluja,

jonka lisäksi jokaisesta työkalukategoriasta kerrotaan yleisellä tasolla. Työkalut on pyritty esittelemään niin, että työkaluihin on helppo tutustua ja ymmärtää niiden tarkoitus.

Opinnäytetyön empiirinen osa koostuu luvuista neljä ja viisi. Luvussa neljä kerrotaan tutkimuksen toteutuksesta, kuten tutkimusotteesta, aineistonkeruumenetelmistä, tutkimuksen aineistosta ja aineiston analysoinnista. Luvussa viisi esitellään tutkimustulokset, jotka käydään läpi teemahaastattelurungon mukaisessa järjestyksessä.

Työn viimeisessä vaiheessa, eli pohdinnassa tarkastellaan aluksi tutkimuksen tuloksia ja verrataan niitä empiiriseen osioon. Sen jälkeen tarkastellaan tutkimuksen luotettavuutta ja eettisyyttä. Tämän jälkeen esitellään johtopäätökset ja jatkotutkimusehdotukset. Tutkimuksen johtopäätöksissä esitellään myös tutkimuksen pohjalta muodostettu työkalupakki. Lopuksi arvioidaan opinnäytetyöprosessia ja omaa oppimista.

1.3 Käsitteet

Ajax — Lyhenne sanoista Asynchronous JavaScript and XML. Tekniikka, joka mahdollistaa verkkosivuston päivittämisen ilman, että sivustoa tarvitsee ladata kokonaisuudessaan uudelleen.

Alustariippumaton — Esimerkiksi ohjelma, joka ei ole riippuvainen tietystä alustasta tai käyttöympäristöstä toimiakseen.

Avoin lähdekoodi — Ohjelmiston koodi, joka on vapaasti tarkasteltavissa ja muokattavissa.

CLI, komentokehote — Lyhenne sanoista Command Line Interface. Suomeksi komentokehote eli liittymä, jonka avulla henkilö pystyy kommunikoimaan tietokoneen kanssa antamalla komentoja tekstin muodossa.

CSS — Lyhenne sanoista Cascading Style Sheets. Tyylikieli, jolla määritellään usein verkkosivujen ulkoasua.

DRY-periaate — Lyhenne sanoista Don’t Repeat Yourself. Ohjelmistokehityksessä käytettävä periaate, jonka tarkoituksena on vähentää toistettavan koodin määrää.

HTML	Lyhenne sanoista Hypertext Markup Language. Ohjelmointikieli sivuston rungon rakentamiseen.
IoT	Lyhenne sanoista Internet of Things. Suomeksi esineiden internet, jossa erilaiset laitteet ovat yhteydessä internettiin ja muun muassa kommunikoivat sen avulla.
Kehittäjä	Henkilö, joka kehittää ohjelmistoja.
Koodi	Ryhmä ohjeita, kuten esimerkiksi kirjaimia ja numeroita, jotka muodostavat ajettavan ohjelman.
Käyttöliittymä	Esimerkiksi käyttöjärjestelmän tai laitteen osa, jonka avulla käyttäjä pystyy olemaan vuorovaikutuksessa tuotteeseen.
Moduuli	Tietokoneohjelman yksittäinen osa.
Natiivisovellus	Sovellus, joka on kehitetty tietylle alustalle.
Node.js	Ympäristö, joka mahdollistaa JavaScriptin hyödyntämisen palvelinpuolella.
Responsiivinen	Internet-sivujen ulkoasu, joka mukautuu näytölle sopivaksi laitteesta riippumatta.
Tietomalli	Malli, joka kuvaa tiedon rakennetta ja rakenneosien välisiä suhteita.

2 Front end -kehitys

Front end -kehityksellä tarkoitetaan kaiken selaimessa näkyvän ohjelmoimista, jossa luodaan sisältöä verkkosivuihin tai -sovelluksiin HTML-, CSS- ja JavaScript-kielillä (Ajzele 2015, luku 8; Atkinson 2015, luku 2). HTML:ää käytetään elementtien kirjoittamiseen, jotka koostuvat tekstistä sekä muista sisäkkäisistä elementeistä. Kyseiset elementit muodostavat sivuston sisällön, joka koostuu teksteistä, linkeistä ja kuvista. CSS on visuaalinen suunnittelutyökalu, jolla muokataan miltä HTML:n sisältö näyttää. CSS:lla voi esimerkiksi muokata HTML-elementtiä tekemällä mitä tahansa visuaalisia muokkauksia, kuten taustavärin vaihdon, fontin koon tai sijoittamalla elementin vaaka- tai pystysuoraan. JavaScriptiä käytetään lisäämään interaktiivisuutta. Kun käyttäjä esimerkiksi klikkaa, liikuttaa hiirtä tai kirjoittaa, JavaScript muuttaa verkkosivun HTML-rakennetta, elementin tyyliä tai lisää kokonaan uuden elementin. (Taylor & Smith 2015, luku 9; Turner & Leonard 2017, luku 1.)

Front end -kehityksessä pyritään saavuttamaan miellyttävä käyttäjäkokemus käsittelemällä käytettävyyttä, saavutettavuutta ja suorituskykyä (Ajzele 2015, luku 8). Käytettävyydellä tarkoitetaan laadun määrettä, jolla arvioidaan käyttöliittymän käytön helppoutta (Nielsen 2012). Saavutettavuus on sitä, että ymmärretään ketkä verkkosivua käyttävät sekä ennakoidaan kaikenlaisten käyttäjien tarpeita ja erilaisia tapoja käyttää tietoa (Smashing Magazine 2015). Suorituskyvyllä tarkoitetaan verkkosivujen latausnopeutta (Wagner 2016, luku 1). Käyttäjäkokemuksella tarkoitetaan yleisesti sitä, miltä tuotteen käyttäminen tuntuu (Vuokola 2014, 7).

Käyttäjäkokemuksen lisäksi front end -kehityksessä kannattaa ennakoida myös tulevaisuutta. Aterin (2017) mukaan kerran muutamassa vuodessa teknologioiden kehityksessä tapahtuu jotain mullistavaa, jolloin kehitys harppaa askeleen eteenpäin. Tällä kertaa on kyse Progressive Web Appeista (PWA), jotka ovat uudenlaisia verkkosovelluksia. PWA on yhdistelmä natiivisovelluksen ja verkkosivuston ominaisuuksia, joka saa uusia valtuuksia vähitellen. Aluksi on tavallinen verkkosivusto, joka käyttäjän vieraillessa ehdottaa sivuston pikakuvakkeen asentamista älypuhelimen etusivulle. Kun käyttäjä hyväksyy asentamisen ja käynnistää sivuston pikakuvakkeen kautta, sovellus ehdottaa tulevien ilmoituksien saamista. Tällaisella tavalla sovellus saa valtuuksia vähitellen ja muuttuu verkkosivustosta natiivisovellusta muistuttavaksi. (Ater 2017, luku 1.)

Googlen (2018) mukaan PWA:t perustuvat erinomaisen käyttäjäkokemuksen tarjoamiseen. Verkkosovelluksiin kuuluu saatavuus ilman yhteyttä, eli ne eivät ole riippuvaisia käyttäjän verkkoyhteydestä. Käyttäjä voi esimerkiksi lähettää viestin, sammuttaa sovelluksen ja jatkaa myöhemmin, jolloin viesti saapuu verkkoon palatessa. PWA:han kuuluvat

myös nopeat latausajat sekä ilmoitukset, jotka pitävät käyttäjää ajan tasalla. Lisäksi PWA tarjoaa natiivisovelluksen ulkoasun ja pikakuvakkeen, jotka parantavat käyttäjäkokemusta. (Ater 2017, luku 1.)

Myös monissa yritysten blogeissa puhutaan Progressive Web Appeista. Esimerkiksi Uotila (13.4.2017) kirjoittaa Qvikin blogissa, että PWA:t haastavat jo tänä päivänä natiivisovellukset, ja että ne yleistyvät muutaman vuoden kuluessa. Lamian (30.11.2017) blogissa mainitaan, että PWA:t eivät ole pelkkä trendi, vaan ne ratkaisevat moniakin asioita. Esimerkiksi PWA-sovelluksia ei tarvitse tehdä jokaiselle eri alustalle eikä niitä tarvitse päivittää samalla tavalla kuin natiivisovelluksia. Natiivisovellusten ongelmana on ollut erottuminen sovelluskaupoissa, mutta PWA-sovelluksia voidaan jakaa kaikille verkkosivustolla käynneille. (Lamia 30.11.2017.)

Toinen tulevaisuuteen liittyvä asia, josta monet yritykset kirjoittavat nettisivuillaan, on komponenttipohjainen kehitys. Jakobus ja Marah (2016, luku 9) kuvailevat komponenttien olevan käyttöliittymän toiminnallisuuden uudelleenkäytettäviä paloja. Azaustre (2016) kirjoittaa, että komponenttipohjaisuus näkyy tänä päivänä, ja että se tulee määrittämään web-kehityksen tulevaisuutta. Myös Vänttinen (7.3.2017) kirjoittaa Nord Softwaren blogissa, että komponenttipohjaiset ratkaisut ovat tulevaisuuden tapa rakentaa sovelluksia. Wilcox (16.5.2017) kirjoittaa Developer Economics ohjelmistokehittäjien tutkimusohjelman blogissa, että JavaScript-sovelluskehys Angular ja JavaScript-kirjasto React, jotka molemmat perustuvat komponenttipohjaiseen kehitykseen, taistelevat kehityksen tulevaisuudesta. Wilcoxin (16.5.2017) mukaan on hyvin todennäköistä, että sekä React että Angular kasvavat entisestään seuraavina vuosina.

3 Front end -työkalut

Tässä luvussa käsitellään front end -työkaluja, jotka ovat jaettu JavaScript-, CSS-, hallinnointityökaluihin sekä muihin työkaluihin. Jokaisen eri työkalutyypin kohdalla kerrotaan yleisellä tasolla työkaluista, jonka jälkeen esitellään ohjelmistoyrityksissä käytössä olevia työkaluja.

3.1 JavaScript-työkalut

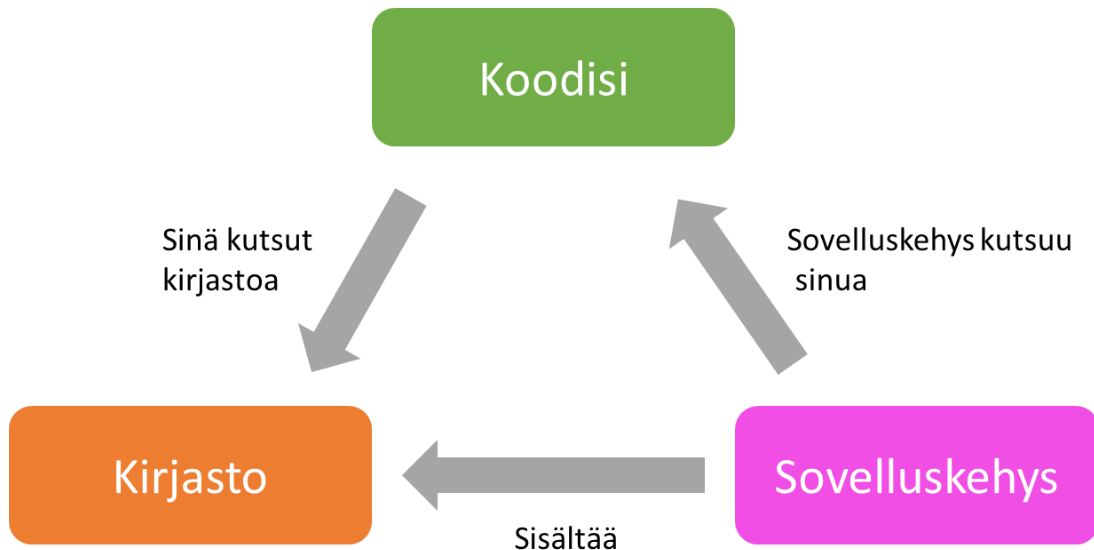
Tässä alaluvussa käydään läpi JavaScript-kieleen liittyviä työkaluja. Näitä ovat sovelluskehykset ja kirjastot, kääntäjät, tarkistusohjelmat ja koodin testaustyökalut.

3.1.1 Sovelluskehykset ja kirjastot

JavaScript-kirjastot ovat uudelleenkäytettäviksi kirjoitettuja kokoelmia ennalta määritetyistä toiminnoista. Kirjastot koostuvat yleensä ennalta ohjelmoiduista luokista ja metodeista, joita hyödynnetään kutsumalla kirjaston mukana tulevia funktioita. Kirjaston sisältämiä funktioita yhdistämällä voi rakentaa isoja ja monimutkaisia kokonaisuuksia tehokkaasti, sillä kirjastojen sisältämää koodia ei tarvitse kirjoittaa uudelleen. (Ballard 2015, luku 5; Kumar 2015.)

JavaScriptin sovelluskehys on sen sijaan ryhmä apuvälineitä, jotka helpottavat ja nopeuttavat JavaScriptin kirjoittamista (Gonzalez 2016). Kehysten käyttö kehitettävässä sovelluksessa lisää laatua, luotettavuutta, johdonmukaisuutta ja kestävyyttä. Kehysten käyttäminen helpottaa ohjelman kirjoittamista monelle eri alustalle ja selaimelle. Lisäksi, sovelluskehys usein tarjoaa ennalta määritettyjä kirjastojen asetuksia, jolloin asetusten määrittämiseen ei tarvitse käyttää ylimääräistä aikaa. Sopivan sovelluskehyksen valitseminen voi jättää enemmän aikaa keskittyä sovelluksen tarkempiin vaatimuksiin, jolloin ei tarvitse keskittää arvokasta aikaa sovelluksen infrastruktuuriin. (Ballard 2015, luku 6; Gonzalez 2016.)

Kirjastot ja sovelluskehykset sekoitetaan usein keskenään, mutta niillä on kuitenkin oleelliset erot (Ballard 2015, luku 6). Sovelluskehys on kuin sovelluksen runko, joka sanelee, miten sovelluksen tulisi toimia. Kirjasto sen sijaan hallinnoi, miten koodia kirjoitetaan. (Thomas 2015, luku 9.) Kumarin (2015) mukaan sovelluskehyksen ja kirjaston keskeisen eron voidaan selittää termillä "inversion of control", eli vapaasti suomennettuna käänteinen hallinta. Kun kutsut metodia kirjastosta, hallinnoit. Sen sijaan sovelluskehyksen kanssa hallinta on käänteinen: sovelluskehys kutsuu sinua (ks. kuvio 1). (Kumar 2015.)



Kuvio 1. Kirjasto, sovelluskehys ja koodisi kuvailtuna (Kumar 2015)

Kirjaston tärkein ominaisuus on joustavuus, sillä pelkkää kirjastoa käyttäessä sovelluksen voi jäsentää omiin vaatimuksiin. Kirjastot ovat myös helpommin opittavissa sovelluskehyksiin verrattuna. Tästä huolimatta sovelluksen voi suunnitella huonosti, kun ohjelmoijalla on vapaat kädet. Sovelluksen suunnitteluun voi vaikuttaa myös ohjelmoinnin puuttuva yhdenmukaisuus, sillä ohjelmoijilla saattaa olla erilainen tapa kirjoittaa koodia. Sovelluskehys voi tällöin suojella huonolta sovelluksen suunnittelulta, sillä se ottaa isomman vastuun sovelluksen rakenteesta. (Thomas 2015, luku 9; Waikar 2015, luku 1.)

Kumarin (2015) mukaan sovelluskehystä ei kuitenkaan voi helposti vaihtaa toiseen, mutta kirjaston voi. Myös Vänskä (2016, 40) kuvailee sovelluskehyksien ja kirjastojen eron olevan se, miten paljon ne vaativat tai sallivat eri osien muuttamista myöhemmin.

Seuraavaksi käydään läpi JavaScript-sovelluskehyksiä ja -kirjastoja. Näihin lukeutuvat AngularJS, Angular, React, Aurelia, Backbone, Bacon, Electron, Ember, jQuery, Knockout, Polymer ja Vue.

AngularJS on vuonna 2009 Misko Heverin ja Adam Abronsin kehittämä avoimen lähdekoodin JavaScript-sovelluskehys, joka on tarkoitettu selainpuolen kehitykselle (Branas, Chardemani, Frisbie & Haviv 2016, luku 1). AngularJS:n avulla voi rakentaa hyvin jäsenneltyjä, dynaamisia ja yhden sivun web-sovelluksia, jotka ovat luotettavia, tehokkaita ja vaivattomasti ylläpidettävissä. (Ballard 2015, luku 6; Chaudhary & Kumar 2015, luku 11.) AngularJS:stä on lyhyen ajan sisällä tullut yksi tunnetuimmista ja käytetyimmistä selainpuolen JavaScript-sovelluskehyksistä (Williamson 2015, luku 1).

Alkuvuosina AngularJS perustui MVC (Model-View-Controller) -arkkitehtuuriin (Minar 19.7.2012). Yleisesti arkkitehtuurissa Model edustaa liiketoimintalogiikkaa ja tietosisältöä, ja View käyttöliittymää, eli kaikkea selaimessa näytettävää. MVC-arkkitehtuurissa Controller vastaa pyyntöjen käsittelemisestä, eli tiedon kulusta Modelin ja Viewin välillä. (Chauhan 2014.) Ajan saatossa sovelluskehyksen arkkitehtuuri vähitellen muuttui, jonka seurauksena sitä kutsutaan tänä päivänä MVW:ksi (Model-View-Whatever). Whatever tarkoittaa "whatever works for you", eli vapaasti suomennettuna "mikä ikinä sinulle toimii", sillä AngularJS on joustava arkkitehtuurien suhteen. (Minar 19.7.2012.)

Yksi huomattavimmista AngularJS:n erityispiirteistä on direktiivit (engl. directives), jotka erottavat AngularJS:n muista sovelluskehyksistä (Sheppard, Miller & Liptak 2015, luku 9; Williamson 2015, luku 9). AngularJS:ssä sovelluksen mallipohjat (engl. templates) yhdistetään siten, että HTML-tageihin ja -attribuutteihin lisätään direktiivejä, jotka laajentavat HTML:n toimintoja käyttäen taustalla toimivaa JavaScript-koodia. Direktiivien etu on se, että HTML-pohjaa on helpompi seurata, kun sitä ei ole täytetty JavaScript-koodilla. AngularJS:n sisäänrakennettujen direktiivien lisäksi ohjelmoija voi tehdä omia direktiivejä tarvittaville toiminnoille. (Dayley & Dayley 2015, luku 5.)

Angular on vuonna 2016 Googlen julkaisema JavaScript-sovelluskehys, joka on AngularJS:stä kokonaan uudelleen tehty versio. Angular on kirjoitettu TypeScript-kielellä, ja sen arkkitehtuuri ja toiminnallisuudet eroavat selvästi sen suositusta edeltäjästä. (Mikkonen 2016a.) Tietoviikon toimittajan Juuso Mikkosen (2016a) mukaan uusi Angular tavoittelee suositun React-kirjaston haastamista.

Angularin verkkosivujen (2017) mukaan sovelluskehyksellä on helppo rakentaa sovelluksia verkkoon, jotka toimivat sekä mobiililaitteissa että työpöydillä. Angularilla rakennetaan sovelluksia HTML:llä sekä JavaScriptillä tai esimerkiksi TypeScript-kielellä, joka kääntyy lopulta JavaScriptiksi. Sovelluksia kehitetään kirjoittamalla HTML-mallipohjia Angularin merkintäkielellä, joita hallinnoidaan komponenttiluokilla. Edellisten ohella palveluihin lisätään sovelluslogiikkaa sekä niputetaan komponentteja ja palveluita moduuleihin. (Angular 2017a; Angular 2017b.)

React on vuonna 2013 Facebookin julkaisema, avoimen lähdekoodin JavaScript-kirjasto, joka on tarkoitettu käyttöliittymien rakentamiseen. React käsittää MVC-arkkitehtuurin View-osaa. Facebook kehitti Reactin, jotta ohjelmistokehittäjät voisivat rakentaa suuria, Facebookin ja Instagramin kaltaisia sovelluksia, joissa tietosisältö muuttuu jatkuvasti. Viime vuosina Reactin käyttö on kasvanut räjähdysmäisesti, ja React on haastanut suosiossa AngularJS:ää. (Krill 2014; Vänskä 2016, 40.)

Reactin lähestymistapa käyttöliittymään on modulaarinen, eli sovellus kootaan yksittäisistä rakennuspalikoista, joita kutsutaan komponenteiksi (engl. components). Komponentit ovat käyttöliittymän toiminnallisuuden uudelleenkäytettäviä paloja, ja ne ovat verrattavissa AngularJS:n direktiiveihin. (Jakobus & Marah 2016, luku 9.) Yhdistelemällä komponentteja, sovelluksesta voidaan rakentaa isoja ja vaikuttavia kokonaisuuksia (Cassio de Sousa 2015, luku 1).

React on tunnettu siitä, että se on nopea käsittelemään DOM:a (document object model). React käyttää virtuaalista DOM:a, joka selvittää mitä käyttöliittymän osia pitää päivittää. Virtuaalinen DOM toimii siten, että se pitää todellisen DOM:n muistissaan, ja päivittää DOM:n kopiota tarvittavilla muutoksilla. (Jakobus & Marah 2016, luku 9.) Kun sivustolla tehdään muutoksia, koko sivua ei päivitetä joka kerta. Sen sijaan React käyttää älykästä algoritmia, joka laskee etukäteen pienet muutokset ja päivittää ainoastaan ne. Kyseinen tapa on erittäin tehokas, ja yksi syy Reactin suosioon. (Casciaro & Mammino 2016, luku 8.)

Reactilla on vaihtoehtoinen laajennusosa JavaScriptille, joka yhdistää JavaScriptin ja HTML:n. Sitä kutsutaan nimellä JSX, ja sitä käytetään kuvailemaan, miltä käyttöliittymän tulisi näyttää. (Cassio de Sousa 2015, luku 2; Facebook 2017a; Facebook 2017b.) JSX muistuttaa rakenteeltaan HTML:ää, ja sitä kirjoitetaan HTML:n tavoin. JSX on kuitenkin puhdasta JavaScriptiä, sillä se käännetään lopuksi JSX-kääntäjällä JavaScriptiksi. (Gackenheimer 2015, luku 3.) Cassio de Sousan (2015, luku 2) mukaan JSX:n käytön etuna on muun muassa se, että sillä on helpompi visualisoita sovelluksen rakennetta.

Aurelia on ilmainen, avoimen lähdekoodin JavaScript-sovelluskehys web-, mobiili- ja työpöytä -kehitykselle, joka perustuu avoimiin web-standardeihin. Sovelluskehys tarjoaa kattavan valikoiman erilaisia työkaluja ja ominaisuuksia, joilla voi rakentaa monipuolisia ratkaisuja. Aurelia perustuu JavaScript-moduulien kokoelmaan, joihin kuuluvat muun muassa riippuvuusinjektio, reititin ja metatiedot. Sovelluskehyksen moduuleja voi käyttää moniin tarkoituksiin, mutta ne ovat parhaimmillaan yhdessä käytettynä. Kyseenomaiset moduulit on kirjoitettu JavaScriptillä ja TypeScriptillä. (Aurelia 2017.)

Backbone on Jeremy Ashkenasin kehittämä vuonna 2010 julkaistu kirjasto, joka antaa nimensä mukaisesti rungon web-sovellukselle. Backbone-kirjasto on kevyt ja nopea, sekä se antaa ohjelmoijille paljon liikkumavaraa sovellusten kehittämisessä. (Mikkonen & Nummi 2016a, 45-46.) Sivuston rakenne kootaan kirjaston tarjoamilla malleilla (engl. models), kokoelmilla ja näkymillä (engl. views) (Backbone 2017).

Bacon on pieni JavaScript-kirjasto, joka on funktionaalinen ja reaktiivinen (Bacon.js 2017). Kirjaston funktionaalisuus ja reaktiivisuus tarkoittavat sitä, että funktionaalisia suunnittelumalleja (engl. design patterns) käytetään edustamaan jatkuvasti muuttuvia arvoja (Antani, Timms & Mantyla 2016, luku 3.3).

Electron on avoimen lähdekoodin sovelluskehys, jolla voi rakentaa alustariippumattomia sovelluksia käyttäen JavaScriptiä, CSS:ää ja HTML:ää. Sovelluskehyksen kehitti GitHub vuonna 2013, alun perin Atom-koodieditoria varten. Vuotta myöhemmin kehyksen lähdekoodi avattiin muiden käytettäväksi, jolloin siitä tuli suosittu työkalu kehittäjien keskuudessa. (Electron 2017.)

Ember on ilmainen ja avoimen lähdekoodin JavaScript-sovelluskehys, jolla voi rakentaa monipuolisia verkkosivustoja. Sovelluskehys sisältää sisäänrakennetun kehitystyökalupakin ja kattavasti ominaisuuksia, jotka ovat hyödyksi nykyaikaisten verkkosovellusten parissa. (Ember 2017.) Emberin GitHubin esittelyn (2017a) mukaan sovelluskehys auttaa kehittäjiä ohjelmoimaan tehokkaasti sovelluksesta riippumatta.

Koska Ember on kokonaisvaltainen sovelluskehys, joka noudattaa palvelinpuolesta tuttua MVC-arkkitehtuuria, se sopii hyvin kehittäjille, jotka omaavat kokemusta Java- tai Ruby-ohjelmointikielistä. Lisäksi kaikenkattavuutensa johdosta, sovelluskehykseen ei tarvitse välttämättä lisätä kirjastoja. Kokonaisvaltaisen yhtenäisyys auttaa myös kirjoittamaan samantyyppistä koodia projektien välillä. (Mikkonen & Nummi 2016a, 44.)

jQuery on suosittu, John Resigin vuonna 2006 kehittämä JavaScript-kirjasto, joka yksinkertaistaa HTML-dokumentin käsittelemistä (Bibeault, Katz & De Rosa 2015, luku 1). jQueryn verkkosivujen (2017) mukaan kirjasto on pieni, nopea ja monipuolinen, sekä se helpottaa muun muassa HTML-dokumentin muokkaamista, animointia ja Ajax-kutsujen käyttöä. Kirjaston hyöty näkyy muun muassa siinä, että muutama rivi jQuery-koodia vastaa useampaa perinteisen JavaScriptin-koodiriviä. jQuerin motto onkin "Write less, do more", eli "kirjoita vähemmän, tee enemmän". (Bibeault ym. 2015, luku 1.1.)

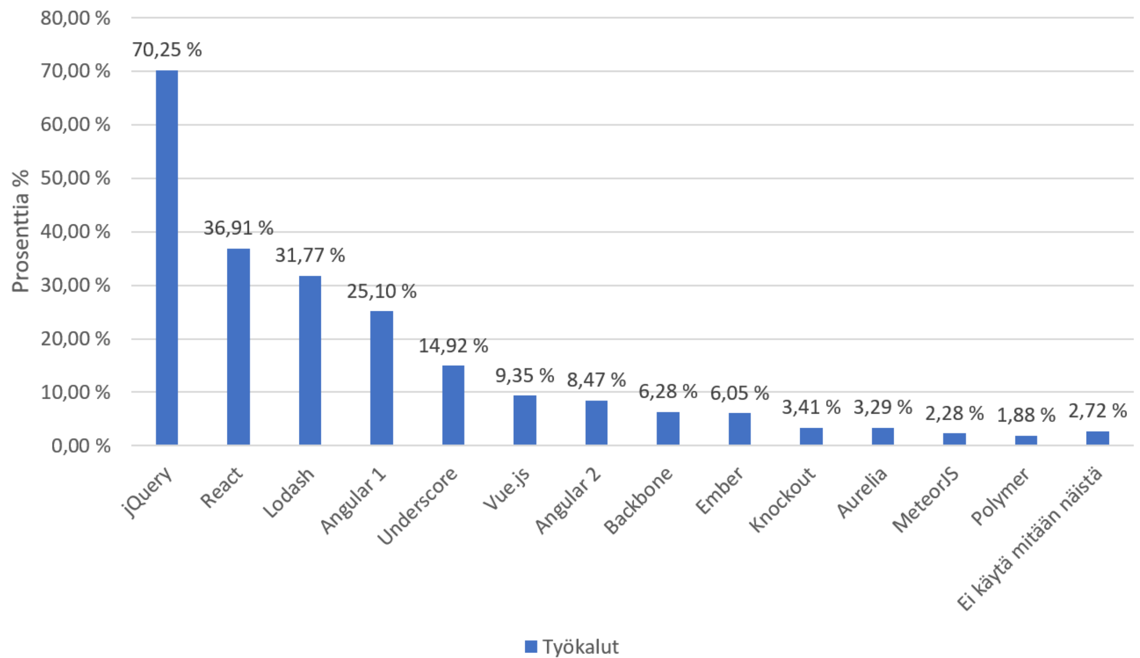
Knockout on avoimen lähdekoodin ja MVVM (Model-View-ViewModel) arkkitehtuurin JavaScript-kirjasto, jolla voidaan kehittää monipuolisia käyttöliittymiä (Knockout 2017a; Knockout 2017b). Kirjaston MVVM-arkkitehtuurissa on kaksisuuntainen tiedonsiirto Viewin ja ViewModelin välillä, joka mahdollistaa automaattisen tietojen muutoksen. ViewModel-osa toimii Modelin ja Viewin välissä vastaten logiikan käsittelystä. (Chauhan 2014.)

Knockoutin tärkeimpiin ominaisuuksiin kuuluvat tyylikäs riippuvuuksien seuranta (engl. elegant dependency tracking), deklaratiiviset sidonnat (engl. declarative bindings) ja triviaalisesti laajennettavuus (engl. trivially extensible). Tyylikäs riippuvuuksien seuranta päivittää automaattisesti tarvittavat osat tietomallin muuttuessa. Deklaratiivinen sidonta liittää käyttöliittymän osat tietomalliin. Lopuksi triviaalinen laajennettavuus -ominaisuus luo räätälöityä sovelluksen käyttäytymistä helppokäyttöisillä deklaratiivisilla sidonnoilla vain muutamalla koodirivillä. (Knockout 2017b.)

Polymer on kevyt ja avoimen lähdekoodin JavaScript-kirjasto, jonka kehityksen takana on Googlen front end -ohjelmoijien tiimi Chrome-organisaatiosta. Polymerilla rakennetaan sovelluksia hyödyntämällä web-komponentteja, joita voidaan luoda ja käyttää uudelleen. Komponentteja käyttäessä voi esimerkiksi luoda elementtejä, jotka toimivat selaimen sisäänrakennettujen elementtien kanssa. Komponentteja voi hyödyntää myös jakamalla sovelluksen erikokoisiin komponentteihin. (Polymer Project 2017a; Polymer Project 2017b.)

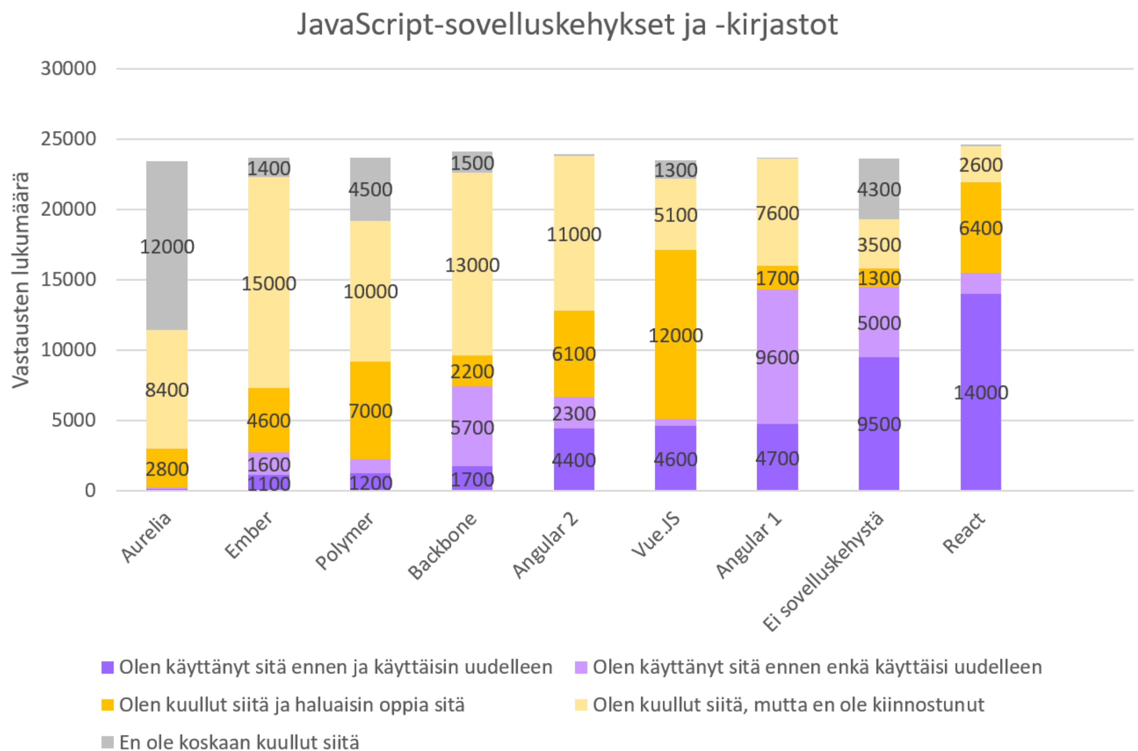
Vue on Evan Youn vuonna 2014 kehittämä JavaScript-kirjasto, joka on tarkoitettu käyttöliittymien kehittämiseen (You 11.2.2014). Vue on suunniteltu muokattavaksi vähitellen, jonka vuoksi sitä kutsutaan progressiiviseksi kehykseksi. Vuen ydin keskittyy pääosin View-osaan, ja sitä on vaivatonta yhdistää muihin kirjastoihin tai projekteihin. Muokattavuutensa ansiosta Vue voi olla sekä kirjasto että kokonainen sovelluskehys. (Vue.js 2017a; Vue.js 2017b.)

Seuraavaksi tarkastellaan Ashley Nolanin selvitystä front end -työkalujen käytöstä, joka suoritettiin vuoden 2016 lopussa ja johon vastasi yli 4700 ohjelmoijaa. Osallistumispyyntöjä jaettiin muun muassa seuraavilla palveluilla: Twitter, Reddit ja LinkedIn. (Nolan 29.11.2016.) Selvityksen tuloksia on esitelty monilla ohjelmointiin liittyvillä verkkosivuilla. Kuviossa 2 esitellään ohjelmoijien vastauksia koskien JavaScript-kirjastojen ja -sovelluksien käyttöä. Selvityksestä ilmenee, että suosituimmat työkalut olivat jQuery, React ja Angular 1 eli AngularJS. Nolanin (29.11.2016) mukaan huomattavaa on se, että Vue-kirjasto on ollut esillä viime aikoina.



Kuvio 2. Ohjelmoijien käyttämät JavaScript-kirjastot ja/tai -sovelluskehykset (Nolan 29.11.2016)

Seuraavaksi tarkastellaan toista, vuonna 2017 tehtyä selvitystä nimeltä The State of JavaScript, johon vastasi yli 20 000 ohjelmoijaa ympäri maailmaa (The State of JavaScript 2017a). Tämänkin selvityksen tuloksia on jaettu erilaisilla ohjelmointiin liittyvillä palveluilla. Liittyen JavaScript-sovelluskehyksiin ja -kirjastoihin, selvityksestä käy ilmi, että suurin osa ohjelmoijista käyttäisi Reactia uudelleen (ks. kuvio 3). Sen sijaan Angular 1:n, eli AngularJS:n kohdalla suurin osa sovelluskehystä käyttäneistä ohjelmoijista ei käyttäisi sitä uudelleen. Myös moni AngularJS:stä kuullut ei ollut kiinnostunut siitä. Lisäksi tämänkin selvityksen kohdalla Vue-kirjasto tulee esiin siinä merkityksessä, että siitä ollaan kuultu ja sitä ollaan kiinnostuttu oppimaan. (The State of JavaScript 2017b.)



Kuvio 3. Ohjelmoijien vastauksia liittyen JavaScript-sovelluskehyksiin ja -kirjastoihin (The State of JavaScript 2017b)

Edellisten selvitysten lisäksi Stack Overflow teki vuonna 2017 selvityksen, jossa kysyttiin erilaisia ohjelmointiin liittyviä asioita kehittäjiltä. Selvitykseen osallistui 64 000 ohjelmoijaa 213 eri maasta. (Stack Overflow 2017a.) Kyseisestä selvityksestä ilmeni, että sovelluskehyksistä, kirjastoista ja muista teknologioista React oli tykätyin työkalu. Vastaajat, jotka ilmoittivat käyttäneensä Reactia, ilmaisivat kiinnostuksensa käyttöön jatkossakin. Sen sijaan AngularJS oli järjestyksessä neljäs, jota ei toivottu. AngularJS:ää käyttävät ohjelmoijat eivät olleet kiinnostuneet käytön jatkamisesta. (Stack Overflow 2017b.) Selvityksen mukaan AngularJS oli käytetyin sovelluskehyksistä ja kirjastoista, kun taas React sijoittui selvityksessä toiseksi (Stack Overflow 2017c).

3.1.2 Kääntäjät ja käännettävät kielet

Lähdekoodin kääntäjä (engl. transpiler) on kääntäjä, joka kääntää ohjelmointikielen toiseen, saman tyyppiseen ohjelmointikieleen (Horton & Vice 2016, luku 4; Pop 2015, luku 6). Lähdekoodin kääntäjä mahdollistaa koodin kirjoittamisen yksinkertaisemmassa muodossa sekä tarjoaa uusien toiminnallisuuksien hyödyntämisen. Näin ollen kääntäjän käyttäminen parantaa koodin luettavuutta ja tekee koodista ylläpidettävämpää. (Simpson 2015, luku 2.)

Kääntäjää voidaan käyttää esimerkiksi uusimpien JavaScript-versioiden muuttamiseen selaimen ymmärtämäksi JavaScriptiksi. Tästä esimerkkinä on ECMAScript 6, lyhyemmin ES6, joka on tämän päivän JavaScript-version, ES5:den, seuraava versio. (Sheppard ym. 2015, luku 24.) ES6 ei käänny suoraan selaimelle ymmärrettäväksi, joten sen kääntämiseen tarvitaan kääntäjää (Eisenman 2016).

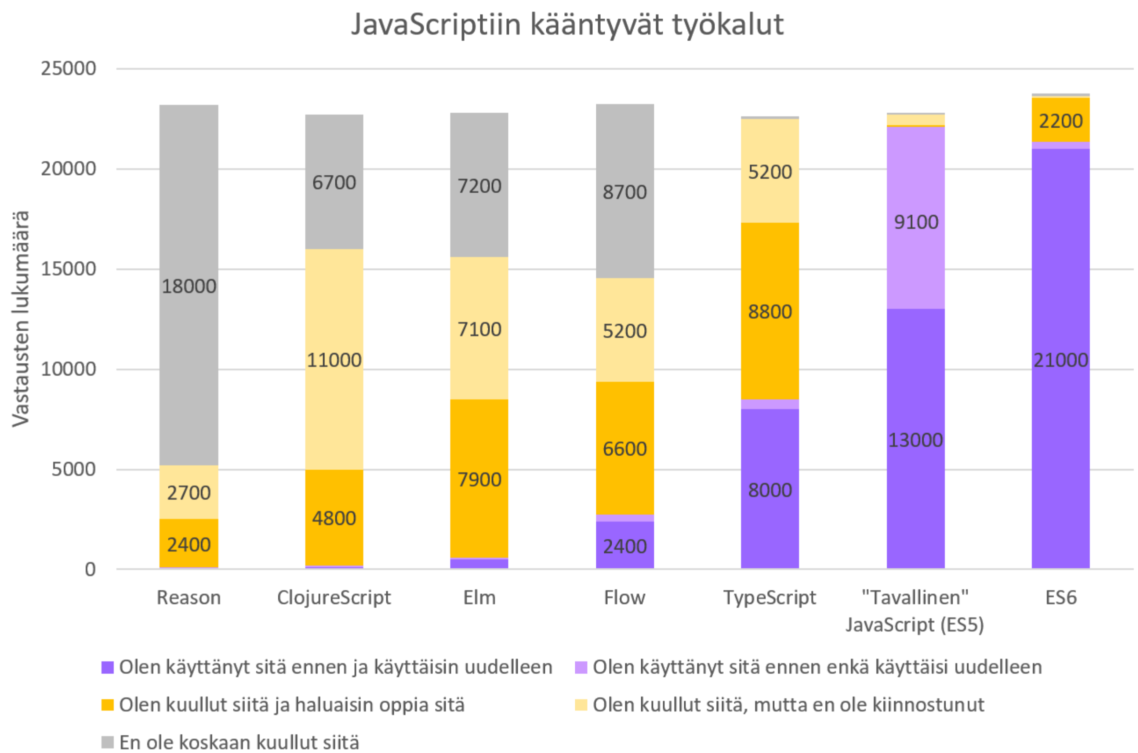
Seuraavaksi esitellään Babel-niminen kääntäjä. *Babel* on lähdekoodin kääntäjä, joka kääntää JavaScriptin versioita selaimelle yhteensopivaan muotoon, ES5:lle (Eisenman 2016). ES6:n lisäksi Babelilla on tuki JSX:lle (Stefanov 2016, luku 4). Chaun (2017, luku 3) mukaan Babel on hyvin suosittu kääntäjä.

Eisenmanin (2016) mukaan lähdekoodin kääntäjiä voidaan käyttää myös JavaScript-tyyppisiin kieliin, joiden syntaksi on helppokäyttöisempi ja yksinkertaistettu tavalliseen JavaScriptiin verrattuna. Tällaisia kieliä ovat muun muassa CoffeeScript ja TypeScript (Eisenman 2016).

CoffeeScript on sivujensa (2017) mukaan pieni kieli, joka kääntyy JavaScriptiksi. CoffeeScriptin tarkoituksena on tuoda esiin JavaScriptin parhaat puolet yksinkertaisesti (CoffeeScript 2017). Tavalliseen JavaScriptiin verrattuna, kielen syntaksi on luettavampaa, ja se on saanut vaikutteita muun muassa Ruby- ja Python-kielistä. CoffeeScriptissä ei käytetä puolipisteitä, pilkkuja eikä aaltosulkeita. (Reyna 2015, luku 1.)

TypeScript-kieli on Eisenmanin (2016) mukaan pohjimmiltaan JavaScriptiä, johon on lisätty tyyppimerkintöjä. Kasagoni (2017) ja Järvinen (2014) ovat kuvailleet TypeScriptin olevan JavaScriptin "ylijoukko", jonka voi kääntää takaisin JavaScriptiksi. Se, että TypeScript on JavaScriptin ylijoukko, tarkoittaa myös sitä, että kaikki JavaScript-koodi on pätevää TypeScript-koodia (Kasagoni 2017; Järvinen 2014). TypeScriptin voi kääntää selaimelle sopivaksi TypeScript-kääntäjällä, joka lyhyemmin tunnetaan nimellä TSC (Järvinen 2014).

Seuraavaksi tarkastellaan aiemmin mainitun The State of JavaScript -selvityksen tuloksia liittyen JavaScripiksi kääntyviin työkaluihin. Kuvion 4 mukaan JavaScriptiin kääntyvistä työkaluista ES6:sta oli käytetty eniten 21 000 äänellä ja sitä oltiin valmiita käyttämään jatkossakin. Myös TypeScript näyttää olevan käytetty ja kiinnostava työkalu, sillä 8 000 ohjelmoijaa oli käyttänyt sitä ja käyttäisi uudelleen. Lisäksi 8 800 ohjelmoijaa oli kuullut TypeScriptistä ja haluaisi oppia sitä. (The State of JavaScript 2017c.)



Kuvio 4. Ohjelmoijien vastauksia liittyen JavaScriptiksi kääntyviin työkaluihin (The State of JavaScript 2017c)

3.1.3 Tarkistusohjelmat

Tarkistusohjelma (engl. linting tool) on ohjelma, joka tarkistaa, että ohjelmoijan koodi vastaa ohjelmointikielen määrittelemiä muotoilusääntöjä. Tarkistusohjelmat toimivat siten, että ne jäsentävät ohjelmoijan tuottamaa koodia, etsivät siitä virheitä ja ilmoittavat niistä. Ohjelman ilmoittamat virheet voivat esimerkiksi olla puolipisteiden puuttuminen koodirivien lopusta, aaltosulkeiden puuttuminen if-lauseesta tai jokin muu saman tyyppinen huolimattomuusvirhe koodissa. Tarkistusohjelma tulee usein kehitysympäristön mukana, mutta sen voi myös ladata manuaalisesti tekstieditoriin lisäosana. Ohjelmakoodia voi tarkistaa front end -kehityksessä muun muassa HTML-, CSS- ja JavaScript-kielistä. (Dahlstrom 2015; Haney 2015.)

Dahlstromin (2015) mukaan tarkistusohjelmia käyttäessä ohjelmoija säästää runsaasti aikaa, sillä koodia kirjoittaessa ohjelma ilmoittaa virheistä reaaliaikaisesti. Tarkistusohjelma myös ylläpitää koodin laatua, joten ohjelmoijan tarvitsee keskittyä vain olennaiseen, eli ohjelmoimiseen. Lisäksi koko ohjelmointitiimi pystyy seuraamaan samaa ohjelmointistandardia. Vastaavasti ilman tarkistusohjelmaa ohjelmoija tallentaisi kirjoittamansa koodin ja ajaisi sen, jonka jälkeen konsolissa näkyisi ohjelmointivirhe tietyllä rivillä. Ohjelmoija tarkistaisi kyseisen rivin, etsisi ja korjaisi virheen ja ajaisi ohjelman uudelleen. (Dahlstrom 2015.)

Seuraavaksi esitellään kolme tarkistusohjelmaa JavaScript-kielelle, jotka ovat ESLint, TSLint ja JSHint. *ESLint* on avoimen lähdekoodin, vuonna 2013 Nicholas C. Zakasin luoma tarkistusohjelma JavaScriptille ja JSX:lle. Tyypiltään ESLint on staattinen analyysiohjelma, eli sillä voi tarkistaa virheet ajamatta itse koodia. ESLint-tarkistusohjelmaa käytetään löytämään sellaisia koodin rakenteita, jotka eivät noudata asetettuja muotoilusääntöjä. ESLint kehitettiin alun perin sen takia, että ohjelmoijat voisivat asettaa omia tarkistussääntöjä JavaScript-koodille. (ESLint 2017.)

JSHint on avoimen lähdekoodin tarkistusohjelma JavaScript-kielelle, joka on helposti säädettävissä ohjelmointiympäristössä. Ohjelma on staattinen analysointityökalu, joka ilmoittaa yleisistä virheistä ja vioista koodissa. Näitä virheitä voivat olla esimerkiksi syntaksivirheet tai muuttujiin liittyvät ongelmat. (JSHint 2017.)

TSLint on staattista analyysia käyttävä koodin tarkistusohjelma TypeScript-kielelle. TSLint tarkistaa muun muassa koodin luettavuutta ja toimintavirheitä, ja sen voi räätälöidä käyttämään omia muotoilusääntöjä. TSLint on saatavilla monissa nykyaikaisissa editoreissa. (TSLint 2017.)

Seuraavassa taulukossa (ks. taulukko 1) on aiemmin mainitun Nolanin (29.11.2016) selvitys vuoden 2016 lopusta liittyen front end -työkalujen käyttöön. Taulukon vastaukset koskevat JavaScript-linttereiden käyttöä. Vastausten perusteella seuraavista suosituin työkalu ohjelmoijien keskuudessa oli ESLint.

Taulukko 1. Ohjelmoijien vastaukset liittyen JavaScript-linttereiden käyttöön (Nolan 29.11.2016)

Työkalu	Äänten määrä	Prosenttia %
En käytä JavaScript-lintteriä	1229	23,39%
JSLint	992	18,88%
JSHint	727	13,84%
ESLint	2123	40,41%
xo	24	0,46%
Muu (tarkenna)	159	3,03%

3.1.4 Testaustyökalut

Testaustyökaluilla testataan ohjelmistojen toimivuutta. Erilaisia testaustapoja ja niihin tarkoitettuja työkaluja on monia. Tämän opinnäytetyön kannalta käydään läpi yksikkötestaukseen ja funktionaaliseen testaukseen liittyviä työkaluja.

Yksikkötestaus (engl. unit testing) on menetelmä, jossa testataan koodin yksittäisten osien toimivuutta. Tavallisesti yksikkötesti on koodinpätkä, joka testaa toisen, pienen koodinpätkän käyttökelpoisuutta. (Dasa 2016, luku 2.) JavaScript-kielessä testattavia koodinpätkiä ovat muun muassa objektit, funktiot, luokat ja moduulit (Burchard 2017, luku 3). Yksikkötestaukseen on olemassa testauskehyksiä, joilla helpotetaan testien kirjoittamista ja ajamista (Gupta, Prajapati & Singh 2015).

Funktionaalisella testauksella taas testataan ohjelmiston toimivuutta käyttäjän näkökulmasta (Hauser 2016). Funktionaalista testausta voidaan automatisoida sille tarkoitetuilla työkaluilla mallintamaan käyttäjän toimintoja ja sitä kautta testaamaan sovelluksen toimivuutta. Nämä toiminnot voivat olla esimerkiksi lomakekentän täyttö tai linkkien klikkaus. (Pajunen 27.12.2013.)

Sekä funktionaalisella että yksikkötestauksella on hyötyjä. Pajusen (27.12.2013) mukaan funktionaalisen testauksen automatisoitu testausprosessi nopeuttaa testausta ja sitä myötä tulee säästettyä rahaa. Yksikkötesteihin liittyen Dasa (2016) kirjoittaa, että yksikkötestaus on nopeampaa kuin manuaalinen testaus, jonka seurauksena yksikkötestit säästävät aikaa ja sitä myötä rahaa. Näiden lisäksi yksikkötesteihin liittyen, testaus kannustaa ohjelmoijaa kirjoittamaan parempaa koodia, jolloin kirjoitettavan koodin virheet vähenevät. Koodi pysyy myös lyhyempänä ja helppolukuisempana. (Dasa 2016, luku 2.)

Seuraavaksi esitellään yksikkötestaukseen liittyviä sovelluskehyksiä, jotka ovat: Mocha, Jest ja Enzyme. *Mocha* on JavaScript-testauskehys, joka on verkkosivujensa (2017) mukaan monipuolinen, helppokäyttöinen ja joustava. Mocha toimii Node.js:ssä ja selaimessa, jonka lisäksi sillä on paljon erilaisia ominaisuuksia. Näihin ominaisuuksiin kuuluu muun muassa: uudelleentestaus-toiminto, testausten raportointi, hitaiden testien korostus, JavaScript API testejä varten ja paljon muuta. (Mocha 2017.)

Jest on JavaScript-testausalusta, joka on Facebookin kehittämä ja jota yritys itse käyttää kaikissa JavaScript-projekteissaan. Ominaisuuksiltaan Jest on nopea ja helppokäyttöinen, joiden lisäksi se ei vaadi konfiguroimista React ja React Native -projekteissa. Työkalua voi myös käyttää JavaScriptiin kääntyvien kielien kanssa, kuten muun muassa TypeScriptin. (Jest 2017a.) Lisäksi Jestiä voi Reactin ohella käyttää minkä tahansa JavaScript-kirjaston tai -sovelluskehyksen kanssa (Jest 2017b).

Enzyme on JavaScript-aputyökalu Reactin testaamiseen, jonka on suunnitellut Airbnb. Työkalu helpottaa React-komponenttien testaamista tarjoamalla tarvittavia työkaluja.

(Porcello & Banks 2017, luku 10.) Työkalu on myös yhteensopiva kaikkien tärkeimpien testaustyökalujen kanssa (Airbnb 2017).

Seuraavaksi käydään läpi funktionaaliseen testaukseen tarkoitettuja työkaluja, jotka ovat Selenium ja PhantomJS. *Selenium* on alun perin vuonna 2004 Jason Hugginsin kehittämä työkalu selaimen automatisointiin. Työkalu on tarkoitettu testaamaan web-sovelluksien toimivuutta automatisoimalla testausprosesseja, mutta sillä voi automatisoida myös web-pohjaisia hallintatehtäviä. Selenium sisältää joukon erilaisia työkaluja testien automatisointiin, jotka soveltuvat monipuolisesti erilaisien web-sivujen testaamiseen. Työkalun tärkeimpiin ominaisuuksiin kuuluu se, että sillä pystyy testaamaan monella eri selaimella. (Selenium 2017.)

PhantomJS on Arya Hidayatin vuonna 2010 kehittämä ”päätön selain” (engl. headless browser), jonka tärkeimpiin käyttötarkoituksiin kuuluu web-sovellusten testaaminen. (PhantomJS 2017a; PhantomJS 2017b). Päättömällä selaimella tarkoitetaan sitä, että työkalu toimii ilman käyttöliittymää. Selaimen päättömyydellä on muutamia etuja. Päätön selain on nopea, koska sillä ei ole käyttöliittymää hidastamassa. Lisäksi työkalu toimii hiljaa taustalla eikä häiritse, sillä sitä ei näy työpöydällä. (Rossel 2017.)

Nolanin (29.11.2016) selvityksen mukaan käytetyin testaustyökalu vuonna 2016 oli Mocha (ks. taulukko 2). Vuoteen 2015 verrattuna Mochan käyttö oli kasvanut seitsemällä prosentilla. Myös The State Of JavaScriptin (2017d) selvityksessä Mocha oli suosituin testaustyökalu vuonna 2017, jossa 11 000 työkalua käyttäneistä käyttäisi sitä uudelleen.

Taulukko 2. Ohjelmoijien käyttämät testaustyökalut JavaScriptille (Nolan 29.11.2016)

Työkalu	Äänten määrä	Prosenttia %	Eroavuus (vuoteen 2015)
En käytä työkalua JavaScriptin testaamiseen	2 519	47,94%	-11,72%
Jasmine	889	16,92%	+0,55%
Mocha	1 174	22,34%	+7,30%
Tape	77	1,47%	-0.01%
Ava	88	1,67%	N/A
QUnit	221	4,21%	+0,36%
Jest	180	3,43%	+2,64%
Muu (tarkenna)	106	2,02%	+0,34%

Taulukon 2 mukaan Mochalla on ollut suhteellisesti eniten kasvua verrattuna muihin taulukon ilmoittamiin työkaluihin. Selvää kasvua taulukon työkaluista on tehnyt myös Jest. Huomattavaa on myös vastausten lukumäärän lasku testityökalujen käyttämättömyyteen noin -11 prosentilla.

3.2 CSS-työkalut

Tässä alaluvussa käydään läpi CSS-kieleen liittyviä työkaluja. Näitä ovat CSS-sovelluskehykset, käyttöliittymän muotoiluun tarkoitetut työkalut, CSS-esikäsittelijät ja CSS:n tyylittelyyn tarkoitetut työkalut.

3.2.1 CSS-sovelluskehykset ja käyttöliittymän muotoilu

CSS-sovelluskehys, joka tunnetaan myös front end -sovelluskehyksenä, on ryhmä hyviä käytänteitä verkkosivujen tekemiseen. CSS-sovelluskehyksiä voi hyödyntää monipuolisesti: joko käyttäen yksinkertaista tai monimutkaisempaa sovelluskehystä. Yksinkertaisessa sovelluskehyksessä määritetään vain yhden sivun pohjia (engl. template), jotka alustetaan jokaiselle verkkosivuston sivulle. Sen sijaan monimutkainen CSS-sovelluskehys on yhdistelmä HTML:n, CSS:n ja JavaScriptin tiedostoja ja kansioita, jotka määrittävät koko sivuston rakenteen. (Kyrnin 2016.)

Jain (2014, 5) on kuvaillut CSS-sovelluskehyksen koostuvan yleensä:

- CSS-ruudukosta (engl. grid), joka määrittää eri elementtien paikan verkkosivuilla.
- Sivuston toimivuuden ratkaisuista, joissa on huomioitu sivuston näkyminen oikein eri selaimilla.
- HTML-elementtien typografisista tyylin määritelmistä.
- CSS-luokista, joilla tyylitellään käyttöliittymän komponentteja.

CSS-sovelluskehyksillä on monia hyötyjä. Kehykset tekevät yleisiä asioita ohjelmoijien puolesta, mikä lisää ohjelmoinnin nopeutta ja sitä myötä tehokkuutta. Kehyksiä on helppo käyttää, sillä ne sisältävät typografisia tyylin määritelmiä. Ohjelmoijan ei siis tarvitse huolehtia sivuston yhdenmukaisuudesta. Kehykset ovat myös erittäin hyödyllisiä responsiivisten verkkosivujen tekoon, sillä ne mukauttavat sisällön erilaisille näytöille. Lisäksi CSS-sovelluskehykset ovat vakaita ja turvallisia, sillä ne ovat hyvin testattuja ja ylläpidettävissä. (Kyrnin 2015, luku 1; Zea 2015, luku 4.)

CSS-sovelluskehyksien huonoihin puoliin voidaan lukea sen, että ohjelmoijalla on vähemmän kontrollia verkkosivustosta kehyksiä käyttäessä. Kehyksien opettelussa saattaa myös mennä aikaa, sillä kehykset ovat laajoja ja niillä on jyrkkä oppimiskäyrä. Lisäksi ohjelmoija ei välttämättä käytä kaikkia kehyksen antimia, mikä saattaa tuoda ylimääräistä raskautta sovellukseen. (Kyrnin 2015, luku 1.)

Seuraavaksi käydään läpi CSS-sovelluskehyksiä, jotka ovat Bootstrap ja ZURB Foundation. *Bootstrap* on maailman suosituin CSS-sovelluskehys HTML:n, CSS:n ja JavaScriptin kanssa kehittämiseen. Bootstrap on avoimen lähdekoodin kehys, ja sillä pystyy rakenta-

maan responsiivisia toteutuksia, jotka toimivat kaikilla päätelaitteilla. Bootstrap sisältää muun muassa erilaisia komponentteja, responsiivisen CSS-ruudukon ja tehokkaita lisäosia. (Bootstrap 2017a.)

Bootstrapin ovat alun perin kehittäneet Mark Otto ja Jacob Thornton Twitterillä vuonna 2010, jolloin Bootstrapin nimi oli Twitter Blueprint. Työkalua käytettiin ensin yhtiön sisällä, jonka jälkeen se julkaistiin virallisesti 19.8.2011. Julkaisun jälkeen Boostrapia on päivitetty säännöllisesti sekä uudelleenkirjoitettu muutaman kerran. (Bootstrap 2017b; Bootstrap 2017c.)

ZURB Foundation on vuonna 2011 julkaistu CSS-sovelluskehys, joka on responsiivinen ja saavutettava. Foundationin ominaisuuksiin kuuluu muun muassa semanttisuus, modulaarisuus, joustavuus ja kustomoitavuus. Sovelluskehys sisältää erilaisia työkaluja ja apuvälineitä, kuten esimerkiksi navigointikomponentteja, elementtejä ja typografiaa. Foundation on myös helposti opittava ja käytettävä. Monipuolinen kokonaisuus tekee Foundationista maailman kehittyneimmän CSS-sovelluskehyksen. (Foundation 2017a; Foundation 2017b; Foundation 2017c.)

Seuraavaksi esitellään käyttöliittymän muotoiluun tarkoitettuja työkaluja, joita ovat Material Design ja Flexbox. *Material Design* on muotokieli, joka pyrkii yhdenmukaistamaan käyttäjäkokemuksia laitteista tai niiden käyttämistä alustoista riippumatta. Muotokielessä innovaatio, teknologia ja tiede yhdistyvät hyvän muotoilun periaatteiden kanssa. Kieli sisältää valoa, materiaalia ja varjostuksia, muodostaen kolmiulotteisen ympäristön. (Material Design 2017a; Material Design 2017b.) Kolmiulotteisuus antaa syvyyttä ja rakennetta pinnoille, jonka seurauksena elementtien päällekkäisyydet selkeytyvät (Mew 2015, luku 1).

CSS Flexible Module Level 1, joka tunnetaan lyhyemmin *Flexbox*-nimellä, on asettelumalli CSS:lle. Flexboxin avulla voidaan helpottaa erilaisten osien asettelua niin, että ne asettuvat käyttöliittymissä sujuvasti. (Weyl 2017, luku 1.) Asettelumallissa on säiliö (engl. container), jonka sisällä olevat osat (engl. items) voi laittaa mukautumaan haluamallaan tavalla (W3C 2017). Flexboxia käyttämällä ohjelmoija voi sanella esimerkiksi tilan jakoa, järjestystä, linjakkuutta tai osien venytystä. Lisäksi sisällön voi sijoittaa pysty- tai vaakasuuntaan ja järjestää uudelleen. (Weyl 2017, luku 1.)

Flexboxilla on monia hyötyjä. Asettelumallin elementit voi asettaa toimimaan erilaisten päätelaitteiden mukaan. Lisäksi Flexbox toimii sujuvasti responsiivisilla sivuilla, joissa sisältöä voi venyttää. Flexboxin käyttö korvaa sujuvuutensa ansiosta CSS-sovelluskehykset. (Weyl 2017, luku 1.)

3.2.2 CSS-esikäsittelijät ja CSS:n tyylittely

CSS-esikäsittelijä on tyylikieli, joka tehostaa ohjelmointia laajentamalla tavallisen CSS:n ominaisuuksia, kuten esimerkiksi lisäämällä tavanomaisia ohjelmoinnin toimintoja. CSS-esikäsittelijät toimivat siten, että ne optimoivat tietosisältöä ensin esikäsittelijän formaatissa, ja lopulta muuttavat kyseisen formaatin selaimen ymmärtämäksi CSS:ksi. Käyttämällä CSS-esikäsittelijöitä ohjelmointi on vaivattomampaa ja tehokkaampaa. (Kyrnin 2015, luku 4; Prabhu 2015, luku 1.)

Prabhun (2015, luku 1) mukaan CSS-esikäsittelijöiden tavallisimpiin laajentamiseen liittyviin toimintoihin kuuluu:

- Muuttujat (engl. variables), joilla voi tallentaa uudelleenkäytettävää tietoa, kuten esimerkiksi värejä ja fonttityylejä.
- Sisäkkäiset määritykset (engl. nesting), joilla voi sisentää koodia, jolloin kokonaisuus on selkeämpi.
- Perintä, jonka avulla voi jakaa ominaisuuksia eri kohdissa, jolloin se auttaa ohjelmoimaan DRY-periaatteen mukaan.
- Operaattorit, joita käytetään tekemään tavanomaisia matemaattisia laskuja.
- Mixinit, joilla voi lisätä CSS-määrityksiä useammalle elementille, mikä vähentää koodin uudelleenkirjoittamista.

Seuraavaksi esitellään CSS-esikäsittelijöitä, jotka ovat Sass, Less ja Stylus. *Sass* on CSS-esikäsittelijä, joka mahdollistaa CSS:n laajennuksen monipuolisilla ominaisuuksilla. Esikäsittelijää käyttäessä voi hyödyntää laajennuksia, kuten muuttujia, sisäkkäisiä määrityksiä ja mixineitä. Sass sisältää myös värien ja muiden arvojen muuttamiseen tarkoitettuja funktioita sekä muita hyödyllisiä lisätoimintoja. (Sass 2017a.)

Sass-esikäsittelijällä on kaksi syntaksia, joista ensimmäinen on Sass ja toinen SCSS. Sass-syntaksi on näistä vanhempi ja suppeampi kirjoitustavaltaan. Sass-syntaksissa ominaisuudet eritellään eri riveille puolipisteiden sijaan. Lisäksi syntaksissa käytetään sisennyksiä eikä sulkeita. Sen sijaan SCSS-syntaksi on CSS:n laajennus, eli kaikki CSS-tiedostot ovat päteviä SCSS-tiedostoja. (Sass 2017b.)

Less on Alexis Sellierin vuonna 2009 kehittämä CSS-esikäsittelijä, joka laajentaa CSS-kieltä. Less lisää muun muassa muuttujia, funktioita ja mixineitä. (Less 2017a; Less 2017b.) Kuten Sassin SCSS-syntaksissa, kaikki CSS-koodi on myös kelvollista Less-koodia (Prabhu 2015, luku 4). Alun perin Less oli kirjoitettu Ruby-ohjelmointikielellä, mutta myöhemmin se uudelleenkirjoitettiin JavaScriptillä (Less 2017b).
Stylus on CSS-esikäsittelijä, joka kääntyy puhtaaksi CSS:ksi. Styluksella on paljon erilaisia ominaisuuksia, kuten muuttujat, sisäkkäiset määritykset ja mixinit. Lisäksi Styluksen

syntaksi on joustava: aaltosulkeiden sekä puoli- ja kaksoispisteiden käyttö on vapaaehtoista. (Stylus 2017.)

CSS-esikäsittelijöiden lisäksi CSS:ää voi muotoilla myös muilla työkaluilla. Seuraavaksi esitellään työkaluja, jotka ovat PostCSS, CSS Modules ja Styled-Components. *PostCSS* on lähivuosina paljon suosiota saanut työkalu, jolla voi muokata CSS:ää JavaScriptin lisäosilla (engl. plugins). Nämä pluginit voivat olla laajennuksia, eli tavallisten CSS-esikäsittelijöiden ominaisuuksia, kuten muuttujia tai mixineitä. Mutta ne voivat olla paljon muutakin, kuten esimerkiksi CSS-koodin tarkistaminen tai eri selainten tuen lisääminen. Näitä lisättyjä ominaisuuksia muutetaan lopuksi puhtaaksi CSS:ksi. (Alabes & Tarkus 2017.)

CSS Modules on työkalu, joka mahdollistaa CSS:n kirjoittamisen niin, että kirjoitettavat tyylit ovat oletuksellisesti paikallisia. Toisin sanoen, CSS Modulesissa käytetään CSS-luokkien nimiä, jotka lopulta käännetään yksilöllisiksi nimiksi. Työkalun avulla vältetään globaalit nimi-muuttujat, jonka seurauksena ei tarvitse huolehtia siitä, että tyylit sekoittuvat ja aiheuttavat konflikteja. (Alabes & Tarkus 2017.)

Styled-Components on työkalu, jonka avulla CSS:ää voi kirjaimellisesti kirjoittaa JavaScript-komponenttien sisällä. Työkalua käyttäessä ei siis tarvitse yhdistää eri tiedostoja yhteen, vaan tyylit kirjoitetaan suoraan komponentin sisään. (Styled-Components 2017a.) Styled-Components on suunniteltu erityisesti komponenttipohjaiselle aikakaudelle (Styled-Components 2017b).

Bertolin (2017, luku 7) mukaan Styled-Components on lupaava kirjasto, sillä se huomioi aiemmat ongelmat liittyen komponenttien tyylittelyyn. Myös Pääkkö (21.3.2017) kirjoittaa Symbion blogissa, että Styled-Components on kiertänyt aiempien tyylikirjastojen hankaluudet, ja että se on yksi tämän hetken parhaimmista työkaluista tyylittelyyn.

The State of JavaScript vuoden 2017 selvityksen mukaan Sass-esikäsittelijä oli ensimmäisenä vertailussa, jossa työkalua ennen käyttäneet käyttäisivät sitä uudelleen. Toisena tuli tavallinen CSS ja kolmantena Less-esikäsittelijä. Myös uusi CSS-JavaScriptissä-tyylittelytapa näkyi ohjelmoijien käytössä, joka tuli neljäntenä. (The State of JavaScript 2017e.) Myös Nolanin (29.11.2016) selvityksen mukaan Sass oli käytetyin työkalu peräti 63:lla prosentilla ja Less tuli toisena 10:llä prosentilla.

3.3 Hallinnointityökalut

Tässä alaluvussa käsitellään paketinhallintajärjestelmiä sekä tehtävänsuorittajia ja moduulien niputtajia.

3.3.1 Paketinhallintajärjestelmät

Paketinhallintajärjestelmä tarkoittaa nimensä mukaisesti järjestelmää, jolla hallinnoidaan paketteja, toisin sanoen kirjastoja. Paketinhallintajärjestelmä tunnetaan myös nimellä riippuvuuksienhallintajärjestelmä (engl. dependency manager), ja sillä voidaan ladata sekä ylläpitää erilaisia paketteja ja niiden riippuvuuksia. (Ambler & Cloud 2015, luku 1.) Järjestelmän paketit voivat sisältää esimerkiksi sovelluskehyksiä tai pienempiä komponentteja (Baumgartner 2016, luku 4).

Kirjastoja päivitetään jatkuvasti, ja ne ovat riippuvaisia toisista kirjastoista, jotka ovat taas toisista kirjastoista riippuvaisia. Modernissa front end -kehityksessä käytetään paljon erilaisia kirjastoja, ja siksi niiden riippuvuudet voivat vaikeuttaa ohjelmointia ilman kunnollista hallintaa. (Sheppard ym. 2015, luku 3.) Esimerkiksi, jos useampi ladattu paketti vaatii saman kirjaston ladattavaksi eri versioissa, paketinhallintajärjestelmä vastaa kyseisestä ristiriidasta. Se voi esimerkiksi asentaa molemmat tai antaa kehittäjän päättää kumman version se haluaa asentavan. (Baumgartner 2016, luku 1.4.) Käyttämällä paketinhallintajärjestelmää, ohjelmoijan ei siis tarvitse manuaalisesti ladata, päivittää ja yleisesti käsitellä sovelluksien riippuvuuksia, jolloin kehittämistyö helpottuu. (Sheppard ym. 2015, luku 3.)

Seuraavaksi käydään läpi seuraavia paketinhallintajärjestelmiä: npm, Bower, Yarn ja jspm. *Npm* (node package manager) on maailman suurin paketinhallintajärjestelmä JavaScriptille, joka tarjoaa lähes puoli miljoonaa ilmaista pakettia. Npm on ylivoimaisesti käytetyin työkalu, ja sitä käyttää yli 6,5 miljoonaa ohjelmoijaa kuukausittain. Ohjelmoijat voivat npm-paketinhallintajärjestelmää käyttäen hallita riippuvuuksia sekä löytää, uudelleenkäyttää ja jakaa koodia. Lisäksi ohjelmoijat voivat koota lataamastaan koodista uusia ratkaisuja haluamallaan tavalla. (npm 2017; Sheppard ym. 2015, luku 3.)

Npm-paketinhallintajärjestelmää voi käyttää myös tehtävien suoritukseen ja automatisointiin. Npm:ssä on package.json-tiedosto, jossa on ajettavat komennot. Näitä komentoja kutsutaan npm-skripteiksi, ja niitä voi lisätä tarpeen mukaan. (Haviv 2016, luku 11.) Yleisesti tehtävien suorituksesta kerrotaan tarkemmin luvussa 3.3.2.

Bower on Twitterin kehittämä paketinhallintajärjestelmä, joka on suunniteltu erityisesti front end -kehitykselle (Baumgartner 2016, luku 1.4.1). Bower asentaa kirjastoja, sovel-

luskehyksiä ja muita apuohjelmia, jotka sisältävät HTML:ää, CSS:ää, JavaScriptia sekä kuvia ja fontteja. Boweria käytetään komentokehotteen kautta, ja sen asennus vaatii npm-paketinhallintajärjestelmän asennuksen, jonka avulla Bower ladataan. Kyseisellä paketinhallintajärjestelmällä voidaan ainoastaan asentaa pakettien oikeat versiot sekä niiden riippuvuudet. (Bower 2017.)

Koska Bower on suunniteltu front end -kehitykselle, se käyttää tasaista riippuvuuskaavaa (engl. flat dependency graph), jonka seurauksena paketin riippuvuus ladataan vain kerran. Esimerkiksi, jos monet paketit ovat riippuvaisia jQuery-kirjastosta, Bower lataa kyseenomaisen kirjaston ainoastaan yhden kerran. Tämä mahdollistaa sen, että sivuston loppukäyttäjä lataa vain yhden version kirjastosta, mikä tekee sivuston ladattavuudesta nopean. (Bower 2017; Sheppard ym. 2015, luku 3.)

Yarn on vuonna 2016 julkaistu pakettihallintajärjestelmä, jonka Facebook kehitti Exponentin, Googlen ja Tilden avustamana korvaamaan npm:n puutteita. Koska Facebookin omat projektit, kuten muun muassa React, ovat riippuvaisia npm:n rekisteristä, kohtasi Facebook ongelmia npm:n suosion ja pakettien kasvun myötä. (McKenzie, Nakazawa & Kyle 2016.) Ongelmat koskivat McKenzien ym. (2016) mukaan:

- Johdonmukaisuutta riippuvuuksia asennettaessa eri käyttäjille ja koneille.
- Hitautta pakettien asentamisessa.
- Turvallisuusongelmia, koska npm sallii pakettien ajaa koodia asennuksen aikana.

Yarn korvaa npm:n tai muiden paketinhallintajärjestelmien työnkulkua, ja on yhteensopiva npm:n pakettirekisterin kanssa. Luotettavuus ja johdonmukaisuus taataan lukitsemalla riippuvuudet tiettyihin versioihin ja varmistamalla, että tiedostojen rakenne on joka kerta sama asennuksen jälkeen. Yarnilla pakettien asentaminen on nopeaa, koska se ajaa useita asennuksia rinnakkain ja käyttää välimuistia hyvin. Yarn ratkaisee turvallisuusongelman pakettien asennusten yhteydessä estämällä oletuksena riippuvuuden sisältämän koodin ajamisen. (McKenzie ym. 2016; Mikkonen 2016b.)

Jspm on paketinhallintajärjestelmä, jolla voi ladata JavaScript-moduuleja useasta eri rekisteristä, kuten esimerkiksi npm:stä ja GitHubista. Jspm pohjautuu universaaliin SystemJS-moduulinlataajaan, joka tukee useaa eri moduuliformaattia. Tuettuja moduuliformaatteja ovat muun muassa CommonJS, ES6, AMD ja globals. (Jspm 2018.) Paketinhallintajärjestelmän hyötyihin lukeutuu esimerkiksi se, että työkalu ratkaisee riippuvuuksiin liittyviä ongelmia tukemalla useita versioita samasta paketista (Angular University 25.9.2015).

3.3.2 Tehtävänsuorittajat ja moduulien niputtajat

Buildaus-työkalut (engl. build tools) auttavat ohjelmistojen rakentamisessa helpottamalla erilaisten tehtävien suoritusta. Ne auttavat tehostamaan työnkulkua siten, että lopputulos on käyttökelpoista. (Seitz ym. 2017, luku 1.) Buildaus-työkalujen tehtäviä voivat olla esimerkiksi kääntäjien ja esikäsittelijöiden suoritus, riippuvuuksien hallinta, käytettävien työkalujen optimoiminen ja kehityspalvelimen ajaminen (A M & Sonpatki 2016, luku 11).

Buildaus-työkaluja on paljon erilaisia omine vahvuuksineen (A M & Sonpatki 2016, luku 11). Näitä työkaluja ovat muun muassa tehtävänsuorittajat (engl. task managers) ja moduulien niputtajat (engl. module bundlers). Tehtävänsuorittajat suorittavat erilaisia tehtäviä automatisoidusti. Vastaavasti moduulien niputtajat esimerkiksi paketoivat JavaScript-moduuleja ja niiden riippuvuuksia yhteen tai muutamaan tiedostoon. (Seitz ym. 2017, luku 1.)

Seuraavaksi esitellään tehtävänsuorittajat, jotka ovat Grunt ja Gulp. *Grunt* on työkalu, joka on tarkoitettu tehtävien automatisointiin. Gruntilla voi automatisoida monia tehtäviä, kuten esimerkiksi koodin tarkistamisen, testien ajamisen, kääntämisen ja minifioinnin (engl. minification), jossa poistetaan tarpeettomat merkit koodista. (Grunt 2017.) Työkalulle on olemassa runsaasti lisäosia, jotka automatisoivat tehtäviä. Lisäosia voi myös tehdä itse, mikäli tarvittavaan tehtävään ei löydy sopivaa lisäosaa. (Reynolds 2016, luku 1.)

Gulp on tehtävien automatisointiin tarkoitettu työkalu. Gulp käyttää pieniä lisäosia tiedostojen muokkaamiseen ja käsittelemiseen. Tiedostojen hallinnointi onnistuu ketjuttamalla (engl. pipe) lisäosien toiminnallisuuksia yhteen halutussa järjestyksessä. (Maynard 2017.)

Aiemmin mainittu Nolanin (29.11.2016) selvitys osoittaa, että tehtävien suorittamiseen käytettiin mielellään eniten Gulp-työkalua (ks. taulukko 3). Toiseksi eniten oli käytössä npm-skriptejä, ja kolmanneksi Grunt-työkalua. Gruntin suosio oli kuitenkin laskenut 15% vuoteen 2015 verrattuna ja npm-skriptien suosio noussut jopa 22%. Samalta näyttää The State of JavaScriptin vuoden 2017 selvityksen mukaan, jossa 11 000 Grunt-työkalua käyttänyttä ohjelmoijaa eivät käyttäisi sitä uudelleen. Saman selvityksen mukaan Gulp oli suosituin tehtävänsuorittajista. (The State of JavaScript 2017f.)

Taulukko 3. Ohjelmoijien vastaukset liittyen tehtävänsuorittajien käyttöön (Nolan 29.11.2016)

Tehtävänsuorittajat	Äänten määrä	Prosenttia %	Eroavuus (vuoteen 2015)
Gulp	2290	43,69%	-0,2%
NPM-skriptit	1356	25,81%	+22,65%
Grunt	626	11,91%	-15,65%
Make	62	1,18%	N/A
GUI Application (i.e Codekit)	100	1,90%	N/A
Muu (tarkenna)	238	4,53%	-0,35%
En käytä tehtävän-suorittajaa	582	11,08%	-8,45%

Seuraavaksi käydään läpi moduulien niputtajat Browserify ja Webpack. *Browserify* on työkalu, jolla voi niputtaa JavaScript-moduuleja ja hyödyntää Node.js:n tyylistä syntaksia 'require'. Moduulien niputuksen lisäksi työkalu mahdollistaa npm-ekosysteemin käytön, josta voi ladata projektiin tarvittavia paketteja. Koska työkalu käyttää samaa syntaksia kuin Node, sen avulla voi käyttää npm:stä Nodelle tarkoitettuja paketteja, jotka eivät muuten toimisi selaimessa. (GitHub 2017b.) Browserify paketoi tiedostot ja niiden riippuvuudet bundle.js-nimiseen tiedostoon (Browserify 2017).

Webpack on moduulien niputtaja JavaScript-sovelluksille, jonka päätehtävä on niputtaa JavaScript-tiedostoja. Niputuksen lisäksi sillä on monia muitakin ominaisuuksia. (GitHub 2017c.) Feldmanin, Bagnardin, Hojbergin ja Hallin (2016, luku 13) mukaan Webpack voi myös muun muassa niputtaa CSS:ää ja kuvia, esikäsitellä ja kääntää tiedostoja, tukea samanaikaista latausta sekä suorittaa Hot Module Replacementia (HMR), eli päivittää reaaliaikaisesti moduuleja.

Nolanin (29.11.2016) selvitys loppuvuodesta 2016 osoittaa (ks. taulukko 4), että moduulien niputtamiseen käytettiin eniten Webpack-työkalua. Webpackin käyttö oli myös noussut peräti 31:llä prosentilla verrattuna vuoteen 2015. Myös The State of JavaScriptin (2017f) vuoden 2017 selvityksestä ilmenee, että Webpack oli suosituin moduulien niputtaja.

Taulukko 4. Ohjelmoijien vastaukset liittyen moduulien niputtajien käyttöön (Nolan 29.11.2016)

Moduulien niputtajat	Äänten määrä	Prosenttia %	Eroavuus (vuoteen 2015)
En käytä moduulien niputtajaa	1716	32,66%	-21,24%
RequireJS	407	7,75%	-5,71%
Browserify	560	10,66%	-5,81%
Webpack	2166	41,23%	+30,74%
Rollup	89	1,69%	N/A
JSPM	119	2,26%	+0,04%
Muut (tarkenna)	197	3,75%	+0,28%

3.4 Muita työkaluja

Tässä luvussa käydään läpi front end -työkaluja, jotka jäivät irrallisiksi muista kategorioista. Näihin työkaluihin kuuluvat koodieditorit ja kehitysympäristöt sekä versionhallintajärjestelmät.

3.4.1 Koodieditorit ja kehitysympäristöt

Ohjelmoimiseen tarkoitettuja työkaluja on pääsääntöisesti kahden tyyppisiä: editoreja ja ide-kehitysympäristöjä (engl. integrated development enviroment). Ohjelmoimiseen voi käyttää ihan tavallista tekstieditoria, mutta ohjelmoimiseen suunniteltu koodieditori tarjoaa enemmän vaihtoehtoja koodin hallinaan. Kauan ennen koodieditoreita on ollut myös komentorivipohjaisia editoreja, joita käytetään yhä. Komentorivipohjaisista editoreista muun muassa Emacs, Vim ja Nano ovat edelleen suosiossa. (Kotilainen 2016, 49; Mikkonen & Nummi 2016b, 50-51.)

Kehitysympäristöt ovat editoreja laajempia ja kokonaisvaltaisempia työkaluja. Kehitysympäristöihin sisältyy muun muassa kehittyneempi editori, jossa on tavalliseen koodieditoriin verrattuna enemmän ohjelmointia helpottavia ominaisuuksia. Näihin kuuluvat esimerkiksi ohjelmointivirheiden etsintä, koodirakenteen parannusehdotukset ja koodin automaattinen täydennys. Kehitysympäristöistä löytyy myös erilaisia lisäosia ja muita työkaluja helpottamaan ohjelmointia. (Kotilainen 2016, 49; Mikkonen & Nummi 2016b, 51.) Kehitysympäristöjen huono puoli on se, että ne ovat suunniteltu jatkuvasti tarkistamaan kirjoitettavaa koodia. Tästä johtuen kehitysympäristön editori saattaa ajoittain olla hidas, jonka vuoksi sitä voi olla tuskallista käyttää. (Adams 2015, luku 1.)

Verrattuna kehitysympäristöihin, koodieditoreissa on vähemmän toimintoja, jonka vuoksi on helpompi keskittyä olennaiseen – eli ohjelmointiin. Kehitysympäristöissä olevat työkalut paikataan editoreissa komentorivityökalujen kautta. Koodieditorien hyvä puoli on se, että

ne ovat nopeita ja kevyitä, joten ne sopivat hyvin esimerkiksi pienempien muutosten tekemiseen. Koodieditorit eivät kuitenkaan yllä ominaisuuksiltaan vielä kehitysympäristöjen tasolle. Koodieditoreihin kehitetään kuitenkin kovaa tahtia uusia lisäosia ja ominaisuuksia, jonka seurauksena koodieditorien ja kehitysympäristöjen erot hämärtyvät vähitellen. (Mikkonen & Nummi 2016b, 51.)

Rozeman (2015) mukaan sellaista editoria ei kuitenkaan ole, joka sopisi jokaiseen tilanteeseen ja ratkaisuun. Koodieditorien ja kehitysympäristöjen valintaan vaikuttaa suuresti kirjoitettava ohjelmointikieli, koska jotkut kielet tarvitsevat enemmän tukitoimintoja. Esimerkiksi Java-projektiin sopii paremmin kehitysympäristö, sillä se vaatii runsaasti tiedostoja. Lisäksi Java-ohjelmointikieli hyötyy kehitysympäristön tarjoamista analyysiominaisuuksista. Dynaamisen JavaScriptin kirjoittajille taas on mielekkäämpää kirjoittaa koodieditorilla, koska raskaat kehitysympäristöt saattavat häiritä ohjelmointia. (Mikkonen & Nummi 2016b, 51; Nummi & Mikkonen 2016.)

Seuraavaksi käsitellään koodieditoreja, jotka ovat Atom, Sublime Text ja Visual Studio Code. *Atom* on ilmainen, moderni ja avoimen lähdekoodin tekstieditori. Editori on monialustainen ja se toimii OS X-, Windows- ja Linux-käyttöjärjestelmissä. Atom on rakennettu JavaScript-, HTML-, CSS- ja Node.js-yhdistelmällä, joiden lisäksi se toimii Electron-sovelluskehyksellä. (Atom 2017.)

Mikkosen ja Nummen (2016) mukaan Atomin vahvuus on muokattavuus, sillä editori tarjoaa monipuolisesti erilaisia lisäosia. Uusia ominaisuuksia ja toimintoja voi esimerkiksi valita tuhansista avoimen lähdekoodin paketeista. Vastaavasti editorin heikkouksiin kuuluu teknologiat, jolla Atom on rakennettu. JavaScriptin takia editorin suorituskyky ei ole kaikkein paras, mutta kehitystä on tapahtunut huomattavasti entisiin versioihin verrattuna. (Mikkonen & Nummi 2016b, 52.)

Sublime Text on hyvin suosittu, alustariippumaton tekstieditori. Sublime Text on tunnettu nopeasta käynnistymisestä, käyttönopeudesta ja useiden ohjelmointikielten tuesta. Näiden lisäksi Sublime Text tunnetaan siitä, että se värjää koodin elementit eri väreillä. (Adams 2015.) Sublime Textin (2017) verkkosivujen mukaan, editorin lataaminen ja käyttö on ilmaista, mutta jatkuvaan käyttöön kannattaa ostaa lisenssi. 70 dollaria maksava lisenssi on käyttäjäkohtainen, ja editoria voi käyttää monella tietokoneella ja käyttöjärjestelmällä. (Sublime Text 2017.)

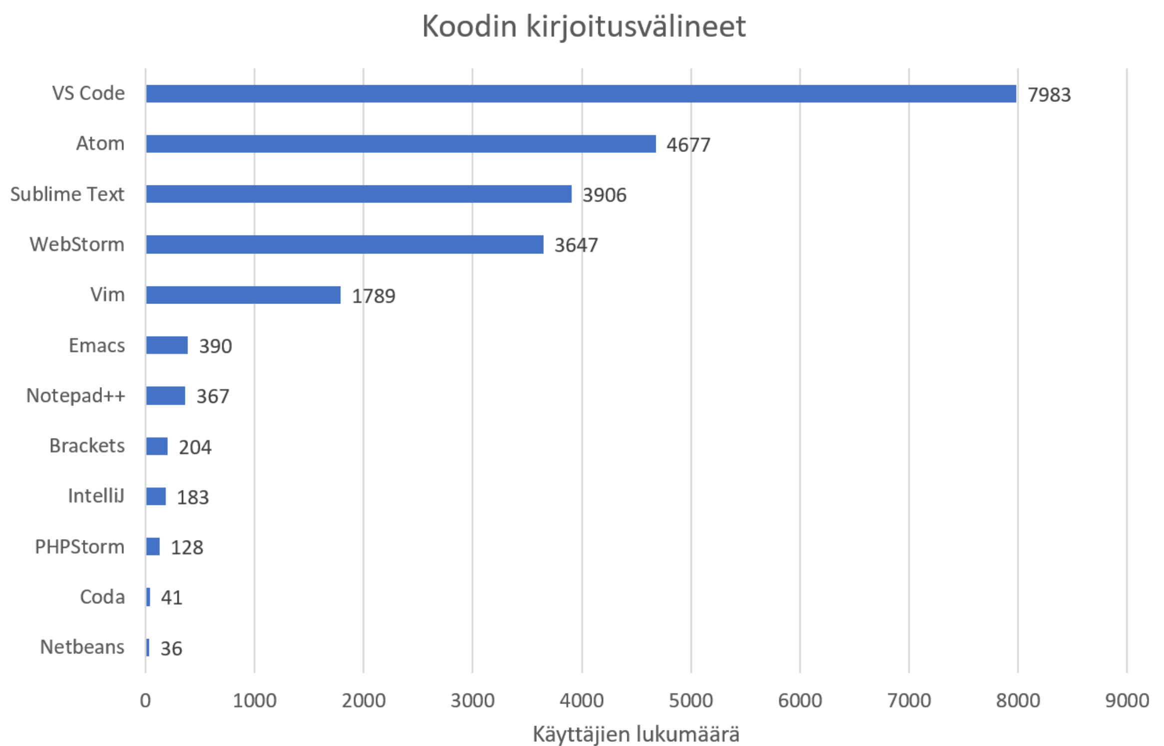
Sublime Textin vahvuuksiin kuuluu Mikkosen ja Nummen (2016) mukaan editorin nopeus, laajennettavuus ja yksinkertaisuus. C++:lla ja Pythonilla rakennettu editori käynnistyy no-

peasti ja avaa helposti tekstitiedostoja, joissa on kymmeniä tuhansia rivejä koodia. Laajennettavuus tehdään lisäämällä lisäosia käyttämällä editorin ohjelmointirajapintaa. Lisäksi Sublime Text on loppujen lopuksi todella yksinkertainen työkalu käyttää. Vastaavasti editorin heikkouksiin kuuluu sulkeutuneisuus. Editorin kehitystahti on ollut hidasta ja epätasaista viime vuosina. Lisäksi editorilla ei ole omaa paketinhallintaa, joka pitäisi huolta päivityksistä ja lisäosista, vaan se pitää ladata erillisenä osana itse. (Mikkonen & Nummi 2016b, 52.)

Visual Studio Code tai lyhyemmin VS Code on avoimen lähdekoodin monialustainen koodieditori, joka on saatavilla Windowsille, MacOS:lle ja Linuxille. VS Code on ilmainen, Microsoftin rakentama koodieditori. Editoriin kuuluu sisäänrakennettu tuki JavaScriptille, TypeScriptille ja Node.js:lle. Editori tukee myös monia muita kieliä, kuten Pythonia, C++:aa, C#:a, PHP:tä ja Go:ta. Editori perustuu Electron-ohjelmistokehykseen, joka tunnettiin aiemmin Atom Shellinä. (Visual Studio Code 2017; Westhuizen 2016, luku 8.)

VS Codeen kuuluu hyviä ominaisuuksia, kuten muun muassa älykkäät syntaksien korostus ja ennakoiva tekstinsyöttö, jotka perustuvat muuttujatyyppeihin, funktioiden määritelmiin ja ladattuihin moduuleihin. Editorista löytyy lisäksi sisäänrakennettu terminaali, jota voi halutessaan käyttää. Yksi VS Coden tärkeimmistä ominaisuuksista on sisäänrakennettu virheenkorjaus-tuki, joka nopeuttaa ohjelmoimista. Tekstieditoriin on sisäänrakennettuna myös Git-versionhallinnan yleisimpiä komentoja, mikä auttaa hallitsemaan koodia kehittämisen ohella. Editoria voi myös halutessaan laajentaa ja muokata, asentamalla esimerkiksi uusia kieliä, teemoja tai muita palveluita. (Mikkonen & Nummi 2016b, 52; Visual Studio Code 2017.)

Aiemmin mainitussa The State of JavaScriptin vuoden 2017 selvityksessä kysyttiin ohjelmoijien käyttämiä koodin kirjoitusvälineitä. Kyseisen selvityksen mukaan reilusti käytetyin koodieditori oli VS Code lähes 8 000:lla äänellä (ks. kuvio 5). Toiseksi käytetyin oli Atom ja kolmanneksi Sublime Text. (The State of JavaScript 2017g.) Sen sijaan aiemmin mainitun Stack Overflown vuoden 2017 selvityksen mukaan web-ohjelmoijien keskuudessa Sublime Text oli kolmanneksi suosituin, VS Code viidenneksi ja Atom tuli vasta seitsemäntenä (Stack Overflow 2017d).



Kuvio 5. Ohjelmoijien käyttämät koodin kirjoitusvälineet (The State of JavaScript 2017g)

3.4.2 Versionhallintajärjestelmät

Versionhallintajärjestelmä (engl. version control system) tarkoittaa järjestelmää, jolla hallinnoidaan tiedostojen ja hakemistojen eri versioita. Järjestelmä tallentaa tiedostoihin tehtyjä muutoksia, joka antaa mahdollisuuden siihen, että tiedostojen eri versioihin voi palata aina tarpeen tullen. Järjestelmällä voi seurata muutokseen liittyviä tietoja, kuten esimerkiksi kuka on tehnyt muutoksen ja milloin. Lisäksi se mahdollistaa useiden kehittäjien työskentelyn samassa projektissa. (Daityari 2015, luku 1.)

Daityarin (2015) mukaan versionhallintajärjestelmät voidaan jakaa kahteen ryhmään: keskitettyihin (engl. centralized) ja hajautettuihin (engl. distributed). Keskitetyissä järjestelmissä projektissa on yksi palvelimella sijaitseva pääarkisto, johon kehittäjät tekevät muutoksia. Sen sijaan hajautetussa järjestelmässä kehittäjät tekevät muutoksia omissa arkistoissaan (engl. repository), jotka ovat kopioita pääarkistosta. Tämän jälkeen muutokset tarkistetaan ja lopuksi yhdistetään pääarkistoon. Yhdistäminen pääarkistoon vaatii useimmiten kirjoitusoikeudet, jotka myönnetään kehittäjille erikseen. Vastaavasti keskitetyissä järjestelmissä kaikki muutoksen tehneet tarvitsevat pääsyn pääarkistoon. (Daityari 2015, luku 1.)

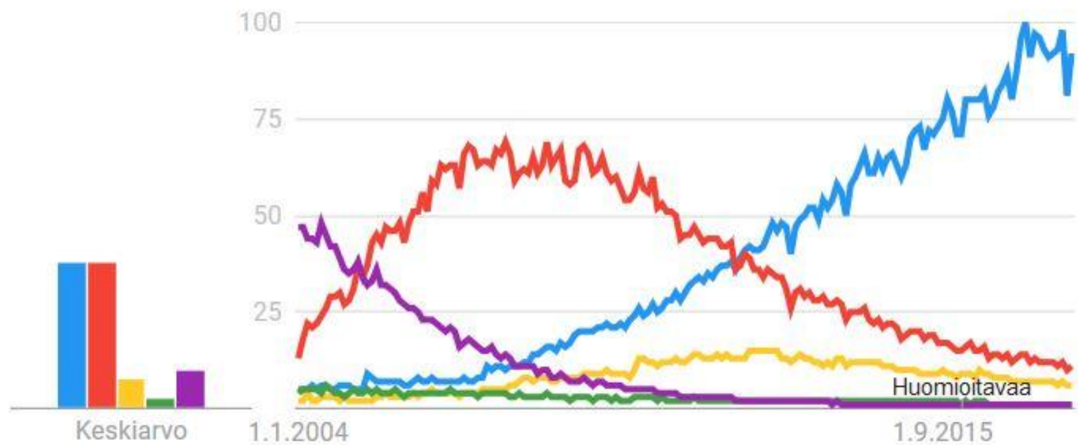
Daityarin (2015) mukaan hajautetulla versionhallintajärjestelmällä on monia hyötyjä verrattuna keskitettyyn järjestelmään. Käyttöoikeuksien käsittelemisen helppouden lisäksi ne

mahdollistavat esimerkiksi arkistojen jakamisen kehittäjien kesken ja työskentelyn jatkumisen, mikäli pääpalvelin kaatuu. Hajautetut versionhallintajärjestelmät ovatkin saaneet paljon kannatusta ensimmäisten julkaisujen jälkeen vuonna 2005. (Daityari 2015, luku 1.)

Seuraavaksi esitellään kaksi versionhallintajärjestelmää, jotka ovat Git ja Mercurial. *Git* on suosittu, vuonna 2005 julkaistu avoimen lähdekoodin versionhallintajärjestelmä, joka on tyypiltään hajautettu (Laster 2016, luku 1). Järjestelmä on nopea, sillä suurin osa toiminnoista suoritetaan paikallisesti ilman yhteyttä palvelimeen (Git 2017a). Gitillä on myös ainutlaatuinen haarautumismalli (engl. branching model), joka mahdollistaa mutkattoman haarojen käsittelyn. Haaroja voi helposti esimerkiksi luoda tietyille tarkoituksille, sekä vaihtaa ja poistaa. (Git 2017b.)

Mercurial on avoimen lähdekoodin versionhallintajärjestelmä, joka on Gitin tapaan hajautettu tyypiltään. Hajautuneisuutensa vuoksi järjestelmän toiminnot ovat nopeita ja sillä pystyy ylläpitämään isojakin projekteja. Mercurialin toimintoja voi myös laajentaa lisäosilla monin tavoin. Toimintoja voi lisätä esimerkiksi ottamalla käyttöön ohjelman mukana tulleita lisäosia. Kehittäjä voi halutessaan luoda lisäosia myös itse. Lisäksi Mercurial käyttää kehittäjille tuttuja komentoja muista versionhallintajärjestelmistä, mikä tekee ohjelmasta vaivattomasti opittavan ja käytettävän. (Mercurial 2017.)

RhodeCoden (2016) selvityksen mukaan Git oli vuonna 2016 selvästi suosituin versionhallintajärjestelmä. RhodeCode vertaili viittä versionhallintajärjestelmää käyttämällä vertailussaan muun muassa Google Trends ja Stack Overflow -palveluja. Kyseiset viisi versionhallintajärjestelmää tulivat Stack Overflowin vuoden 2015 kehittäjäselvityksestä. (RhodeCode 2016.) Saman Google Trends -haun perusteella vuonna 2018 tammikuussa, Git on edelleen suosituin versionhallintajärjestelmä (ks. kuva 1).



Kuva 1. Versionhallintajärjestelmien vertailua Google Trends -palvelussa (26.1.2018)

Kuvan 1 diagrammin mukaan suosituin versionhallintajärjestelmä on vaihtunut kolme kertaa vuosien 2004 ja 2018 välillä. 2004 vuoden alussa suosituin versionhallintajärjestelmä oli Concurrent Versions System, jonka syrjäytti noin vuoden myöhemmin Subversion. Diagrammin mukaan Git nousi suosituimmaksi versionhallintajärjestelmäksi noin vuonna 2011.

4 Tutkimuksen toteutus

Tutkimuksen kohteena on Suomessa toimivat kaiken kokoiset ohjelmistoyritykset. Tutkimuksen tavoitteena on selvittää mitä front end -työkaluja ohjelmistoyritykset käyttävät ja miksi. Näiden vastausten pohjalta opinnäytetyössä kootaan suositeltava front end -työkalupakki alalle suuntaaville ohjelmoijille. Tarkoituksena on, että ohjelmoija voi harjoitella työkalupakin työkaluja etukäteen ja mahdollisesti tehdä oman portfolion työnhakua varten työkalupakkia hyödyntäen.

4.1 Laadullinen tutkimusote

Tämän opinnäytetyön tutkimusotteeksi valittiin kvalitatiivinen, eli laadullinen tutkimus. Kanasen (2014) mukaan laadullinen tutkimus tarkoittaa tutkimusotetta, jonka tavoitteena on saada syvällinen ymmärrys ilmiöstä käyttämättä lukuihin perustuvia menetelmiä. Laadullisessa tutkimuksessa ilmiötä kuvaillaan siis sanoin ja lausein. Vastaavasti kvantitatiivisessa eli määrällisessä tutkimuksessa käytetään tilastollisia, eli määriin pohjautuvia menetelmiä. (Kananen 2014, 18.) Laadullisessa tutkimuksessa mitataan siis laatua, ei määrää (Kananen 2014, 95).

Laadullinen tutkimus on sopiva menetelmäksi yleensä silloin, kun ilmiöstä ei tiedetä tarpeeksi ja kun sitä pyritään ymmärtämään. Jos aiheesta ei ole tehty tutkimusta tai siitä ei ole malleja, teorioita tai tietoa, käytetään silloin ensin laadullista tutkimusta. Laadullisessa tutkimuksessa on tavoitteena saada syvällinen tulkinta aiheesta. Tutkimusotteessa ei voida yleistää, vaan siinä voidaan tutkia kattavasti muutamia havaintoyksikköjä. Ilmiön kattava tutkiminen antaa mahdollisuuden perusteelliseen kuvaukseen. (Kananen 2014, 16-17.)

Tämän opinnäytetyön tutkimuksessa haluttiin saada syvällinen ymmärrys siitä, miten ohjelmistoyritykset ovat päätyneet käyttämiinsä työkaluihin, kauanko niitä käytetään ja mitä työkaluja haastateltavat suosittelevat opettelemaan tulevaa työtä ajatellen. Lisäksi tämän opinnäytetyön kannalta kattava ilmiön kuvaus auttaa suositeltavan työkalupakin rakentamisessa, sillä valinnoille on helpompi antaa perustelut.

4.2 Aineistonkeruumenetelmät

Yksi yleisimmistä aineistonkeruumenetelmistä laadullisessa tutkimuksessa on haastattelu (Tuomi & Sarajärvi 2009, 71; Kananen 2014, 70). Kysymyksiin pohjautuva aineistonkeruumenetelmä sopii parhaiten, kun tutkitaan aikomuksiin perustuvaa käyttäytymistä. Haastattelua käytetään silloin, kun on tarve selvittää toiminnan syy. Tällöin on hyvä kysyä

asiaa suoraan. (Tuomi & Sarajärvi 2009, 71-72.) Opinnäytetyön tutkimukseen valittiin haastattelu, koska kysymällä suoraan koettiin saavan tietoa parhaiten.

Tutkimuksen haastattelutyypiksi valittiin teemahaastattelu eli puolistrukturoitu haastattelu. Kanasen (2014, 70) mukaan teemahaastattelu on haastattelun muoto, jossa keskustellaan eri teemoista eli aihealueista. Keskustelu alkaa ja etenee aihe aiheelta, jossa tutkija kysyy avoimia kysymyksiä. Avoimiin kysymyksiin vastatessa tutkija voi kysyä lisää tarkentavia kysymyksiä, jotka voivat auttaa tutkijaa ymmärtämään ilmiötä paremmin. (Kananen 2014, 79-80.) Vertailun vuoksi lomakehaastattelu on täysin strukturoitu, jossa kysymykset on kirjoitettu etukäteen. Toisessa päässä taas on avoin haastattelu, jossa keskustellaan ilman ennalta määrättyjä teemoja. (Kananen 2014, 70.)

Teemahaastattelu tulee kysymykseen silloin, kun asiaa ei tunneta kovin hyvin (Kananen 2014, 76). Haastattelumenetelmässä haastattelija pyrkii löytämään käsityksen tutkittavasta asiasta (Kananen 2014, 72). Tässä opinnäytetyössä teemahaastattelu koettiin sopivimmaksi vaihtoehdoksi, sillä esimerkiksi työkalujen käytön syistä ei tiettävästi ole tehty samankaltaisia tutkimuksia. Tutkittavaan asiaan haluttiin myös saada syvällinen ymmärrys, jotta työkalupakin valitut työkalut olisivat hyvin perusteltavissa. Lisäksi tutkittavia aiheita oli muutama, jonka vuoksi haastattelu teemoiksi jaettuna nähtiin sopivana. Front end -työkaluja on myös paljon erilaisia, eikä niitä kaikkia voi tietää. Tämän vuoksi haluttiin saada mahdollisuus siihen, että voidaan kysyä tarkentavia kysymyksiä. Teemahaastattelua varten muodostettiin teemahaastattelurunko (ks. liite 4), joka pohjautui tietoperustaan.

Haastattelutyypin valinnan jälkeen tutkimuksen aineistoa etsittiin pääasiassa tarkastelemalla Suomessa toimivien ohjelmistoyritysten verkkosivuja ja niiden yhteystietoja. Yritykset saattoivat olla myös kansainvälisiä, mutta sellaisia, jotka toimivat myös Suomessa. Aineiston etsimisessä hyödynnettiin myös LinkedIn-sivuja. Lisäksi Haaga-Helian epäviralliseen Facebook-ryhmään jätettiin ilmoitus aineiston keräämisestä.

Tuomen ja Sarajärven (2009, 85) mukaan laadullisessa tutkimuksessa on hyvä kerätä tietoa sellaisilta henkilöiltä, joilla on tietoa tai kokemusta aiheesta. Aineiston kerääminen aloitettiin lähettämällä haastattelukutsu, eli saatekirje sähköpostiviestillä. Kutsu kohdennettiin sellaisille henkilöille, jotka yritysten verkkosivujen yhteystietojen perusteella vaikuttivat omaavan tietämystä yrityksen käyttämistä front end -työkaluista. Lisäksi kolme haastattelukutsua lähetettiin LinkedIn-verkkopalvelun välityksellä. Muutamien lähetysten jälkeen kirjeen alkua muutettiin muutamalla sanalla sopivammaksi, joten kaikki eivät ole saaneet täsmälleen samaa kutsua. Liitteestä 2 löytyy viimeisin versio.

Kanasen (2014, 73) mukaan tutkimusmateriaali tulisi pitää luottamuksellisena. Kutsussa kerrottiin, että haastattelu nauhoitetaan analyysia varten, ja että tutkimus on anonyymi sekä yrityksen että haastateltavan osalta. Kutsussa kerrottiin myös, että tutkimustuloksia käytetään ainoastaan tutkimustarkoituksiin. Lisäksi kutsussa ilmoitettiin, että haastattelu tulisi kestämään noin 1 – 1,5 tuntia.

Kutsussa ilmoitettiin myös, että osallistumisesta saa elokuvalipun Finnkinon elokuvateatteriin. Elokuvalipulla yritettiin houkutella mahdollisimman montaa henkilöä haastatteluun. Tutkija kustansi elokuvaliput itse. Kutsuun liitettiin myös teemahaastattelurunko. Tuomen ja Sarajärven (2009) mukaan on suotavaa, että haastateltava tutustuu haastattelukysymyksiin etukäteen. Kysymykset valmistavat haastateltavaa vastaamaan paremmin kysymyksiin, jolloin haastattelusta saa enemmän tietoa. (Tuomi & Sarajärvi 2009, 73.)

Kutsu lähetettiin kaiken kaikkiaan 52:lle eri yrityksen työntekijälle. Muutama viestin saaneista ilmoitti, että he lähtevät lomalle eivätkä siksi kerkeä osallistumaan. Alkukesä ei siis luultavasti ollut kaikkein otollisin aika aineiston etsimiselle. Lopulta näistä seitsemän henkilöä ilmoitti osallistuvansa haastatteluun. Lisäksi yksi henkilö otti yhteyttä nähtyään Facebook-ryhmän ilmoituksen. Henkilöltä varmistettiin, että hän on sopiva haastatteluun. Kaiken kaikkiaan haastatteluun saatiin kahdeksan (n = 8) henkilöä.

Haastattelut sovittiin pääasiassa sähköpostiviestien välityksellä, ja ne pidettiin aikavälillä 4.6 - 2.7.2017. Seitsemän haastatteluista pidettiin Skype-pikaviestipalvelun kautta ja yksi kasvotusten rauhallisessa tilassa. Kanasen (2014, 85) mukaan haastattelut tulee nauhoittaa, sillä muistiinpanojen kirjoittaminen häiritsee keskustelua. Ennen haastattelutilannetta osallistujille lähetettiin tai annettiin suostumuslomake allekirjoitettavaksi nauhoituksen sopimisesta (ks. liite 3). Tämän jälkeen haastattelut pidettiin ja nauhoitettiin Amolto Call Recorder -nimisellä ohjelmalla tai älypuhelimen tallentimella. Ajallisesti lyhyin haastattelu kesti 40 minuuttia ja pisin tunnin ja 47 minuuttia. Haastattelujen aikana ei tullut tilanteita, jotka olisivat häirinneet haastattelua merkittävällä tavalla. Yhden haastateltavan kohdalla kuului hetken pesukoneen ääntä ja toisen kohdalla ajoneuvo ajoi kovaa ikkunan takaa, mutta keskusteluun palattiin normaalisti.

4.3 Tutkimuksen aineisto

Kanasen (2014) mukaan laadullisessa tutkimuksessa ei ole tiettyä havaintoyksiköiden vähimmäismäärää, ja tutkimuksen voi tehdä myös yhdellä havaintoyksiköllä. Laadullisessa tutkimuksessa laatu on tärkeämpää kuin havaintoyksiköiden määrä. (Kananen 2014, 95.) Tuomen ja Sarajärven (2009, 86) mukaan aineiston määrää voidaan perustella esi-

merkiksi siten, että valitaan kyseisestä aiheesta parhaiten tietävät henkilöt. Tutkimukseen osallistui kahdeksan (n = 8) henkilöä, joista jokainen oli töissä eri ohjelmistoyrityksessä. Tässä tutkimuksessa kyseinen määrä nähtiin riittävänä, sillä henkilöillä oli laaja ja kattava tietämys aihealueesta. Tietoa kerääntyi haastatteluista niin paljon, että työkalupakin kokoaminen onnistui.

Tutkimuksen osallistujilla oli vahva ymmärrys front end -työkaluista, ja he pystyivät puhumaan niistä kattavasti. Osallistujien mukaan heidän front end -osaaminen oli laajaa, monipuolista ja syvällistä. Moni ilmoitti sen olevan heidän ykkösosaamistaan. Osallistujista yksi kertoi tehneensä aiemmin paljon front end -kehitystä, mutta nykyisin varsinainen ohjelmointi on jäänyt vähemmälle. Osallistuja koki, ettei enää ole paras osaaja, mutta omaa hyvin vahvan ymmärryksen nykypäivän uusista teknologioista. Lisäksi muutama osasi kertoa parhaiten omissa projekteissa käytetyistä työkaluista eikä koko työpaikan laajuudelta. Tämä oli kuitenkin ymmärrettävää, sillä projekteja voi olla talon sisällä paljon erilaisia, eikä yksi henkilö voi tietää kaikkea varsinkin suurikokoisissa yrityksissä.

Osallistujien työnkuvat olivat monenlaisia: ohjelmoijasta yrityksen perustajaan (ks. taulukko 5). Osallistujista seitsemän ilmoitti ohjelmoivansa työssään. Haastateltavista eniten työnkuvallisesti oli ohjelmoijia ja toiseksi eniten yrityksen perustajia.

Taulukko 5. Osallistujien työnkuvat

Osallistuja (kpl)	Työnkuva
3	Ohjelmoija/front end -ohjelmoija
1	Vanhempi ohjelmistosuunnittelija
1	Ohjelmistoarkkitehti
1	Kehityspäällikkö
2	Yrityksen perustaja

Osallistujilla oli kokemusta alalta vajaan vuoden ja kahdenkymmenen vuoden välillä (ks. taulukko 6). Kokemukseksi laskettiin vuodet, jonka aikana henkilö on saanut elantoa alan työstä. Kuten taulukosta 6 ilmenee, suurimmalla osalla haastateltavista oli kokemusta alalta vähintään kuusi vuotta, ja usealla jopa 11 vuotta ja enemmän.

Taulukko 6. Osallistujien kokemus alalta vuosissa

Osallistuja (kpl)	Kokemus vuosissa
2	0 – 5
3	6 – 10
2	11 – 15
1	16 – 20

Henkilöiden koulutukset vaihtelivat toisen asteen koulutuksista korkeamman asteen koulutuksiin (ks. taulukko 7). Eniten oli yliopiston suorittaneita. Kaiken kaikkiaan kuudella kahdeksasta haastateltavasta oli tutkinto korkeakoulusta.

Taulukko 7. Osallistujien koulutukset

Osallistuja (kpl)	Koulutus
1	Ammattikoulu
1	Lukio
2	Ammattikorkeakoulu
4	Yliopisto

Ohjelmistoyritysten toimialat, joissa henkilöt olivat töissä, olivat IT-palvelut, ohjelmistojen suunnittelu ja valmistus sekä viestintälaitteiden valmistus (ks. taulukko 8).

Taulukko 8. Ohjelmistoyritysten toimialat

Ohjelmistoyritys (kpl)	Toimiala
1	IT-palvelut
6	Ohjelmistojen suunnittelu ja valmistus
1	Viestintälaitteiden valmistus

Ohjelmistoyritysten perustamisvuodet vaihtelivat vuosien 2000 ja 2014 välillä (ks. taulukko 9). Yrityksistä suurin osa perustettiin vuosien 2005 ja 2009 välillä.

Taulukko 9. Ohjelmistoyritysten perustamisvuodet

Ohjelmistoyritys (kpl)	Perustamisvuosi
1	2000 – 2004
5	2005 – 2009
2	2010 – 2014

Ohjelmistoyritysten liikevaihdoissa oli jonkin verran eroa (ks. taulukko 10). Vuonna 2016 yritysten liikevaihdosta suurin osa oli alle 10 miljoonaa.

Taulukko 10. Ohjelmistoyritysten liikevaihdot

Ohjelmistoyritys (kpl)	Liikevaihto v. 2016 (milj. €)
6	0 – 9
0	10 – 19
0	20 – 29
0	30 – 39
1	40 – 49
1	50 – 59

Ohjelmistoyritysten henkilöstön lukumäärä vaihteli reippaasti, kuten taulukko 11 osoittaa. Eniten oli yrityksiä, joissa on alle 100 työntekijää.

Taulukko 11. Ohjelmistoyritysten henkilöstön lukumäärä

Ohjelmistoyritys (kpl)	Henkilöstön lukumäärä
4	0 – 99
2	100 – 199
0	200 – 299
0	300 – 399
1	400 – 499
1	500 – 599

4.4 Aineiston analysointi

Tuomi ja Sarajärvi (2009, 91) kirjoittavat, että laadullista aineistoa voi analysoida sisällönanalyysilla, joka on tavanomainen analyysimenetelmä kvalitatiiviselle tutkimukselle. Sisällönanalyysissa asiakirjoja voidaan analysoida järjestelmällisesti. Asiakirjat ovat tekstimuotoon saatavat aineistot, kuten esimerkiksi haastattelu, kirjat, raportit tai puhe. (Tuomi & Sarajärvi 2009, 103.) Tässä tutkimuksessa käytetään sisällönanalyysia, sillä se on laadulliselle tutkimukselle perinteinen materiaalien analyysimenetelmä.

Sisällönanalyysi voidaan jakaa kolmeen eri analyysimuotoon: aineistolähtöiseen, teoriaohjaavaan ja teorialähtöiseen (Tuomi & Sarajärvi 2009, 95-97). Tässä tutkimuksessa käytettiin aineistolähtöistä sisällönanalyysiä. Tuomen ja Sarajärven (2009, 95) mukaan aineistolähtöisessä sisällönanalyysissä teoria muodostuu aineistosta, ja aineistoa tarkastellaan tutkimuksen tarkoituksen kautta. Teoriaan pohjautuvissa analyyseissä taas tarkastellaan

aineistoa jonkin teorian näkökulmasta (Tuomi & Sarajärvi 2009, 119-120). Opinnäytetyössä haluttiin saada vastaus tutkimuskysymyksiin eikä esimerkiksi kokeilla tietyn teorian paikkansapitävyyttä, jonka vuoksi aineistolähtöinen sisällönanalyysi nähtiin sopivampana vaihtoehtona.

Aineistolähtöinen sisällönanalyysi aloitetaan haastattelujen kuuntelemisella, jonka jälkeen litteroidaan aineisto (Tuomi & Sarajärvi 2009, 109). Litterointi tarkoittaa erilaisten aineistojen kirjoittamista tekstimuotoon, jotta sitä voidaan käsitellä analysointimenetelmillä (Kananen 2014, 101). Litterointia on eri tasoista, kuten esimerkiksi referoiva litterointi, peruslitterointi, sanatarkka litterointi ja keskusteluanalyyttinen litterointi (Yhteiskuntatieteellinen tietoarkisto 2017).

Tämän tutkimuksen litterointitasoksi valittiin peruslitterointi. Peruslitteroinnissa aineisto litteroidaan sanatarkasti, mutta ilman täytesanoja, toistoja, äännähdyksiä tai keskenjääviä tavuja. Lisäksi litteroinnissa huomioidaan tunneilmaisut, kuten esimerkiksi nauru. Peruslitteroinnista voidaan jättää litteroimatta sisältöä, joka ei kuulu asiayhteyteen. Kyseistä litterointitasoa käytetään analysoidessa asiasisältöä. (Yhteiskuntatieteellinen tietoarkisto 2017.) Opinnäytetyön kannalta peruslitterointi nähtiin parhaaksi tasoksi, sillä tarkoituksena oli analysoida ainoastaan asiasisältöä eikä esimerkiksi merkittäviä tunnetiloja tai käyttäytymistä. Aineisto litterointiin, jonka kokonaispituudeksi tuli 66:den sivun mittainen doc-asiakirja. Asiakirjassa käytetty fontti oli Arial, fonttikoko 11 ja riviväli 1,0.

Litteroinnin jälkeen aineisto redusoidaan eli pelkistetään, sitten klusteroidaan eli ryhmitellään, ja lopuksi abstrahoidaan eli luodaan teoreettisia käsitteitä. Redusoinnissa aineistoon perehdytään, ja mukaan otetaan sellaiset asiat, jotka vastaavat tutkimustehtävään. Tämän jälkeen mukaan otettavat asiat tiivistetään. Klusteroinnissa tiivistettyjä ilmauksia ryhmitellään siten, että ilmaukset kerätään luokittain yhteen esimerkiksi sisällöltään samankaltaisten ilmaisujen mukaan. Lopuksi abstrahoinnissa ilmauksista muodostetaan teoreettisia käsitteitä pienemmistä suurempiin, niin kauan kuin mahdollista. (Tuomi & Sarajärvi 2009, 108-113.)

Tutkittavaan aineistoon perehdyttiin ja sille esitettiin tutkimuskysymyksiä. Aineistosta rajattiin pois sellaiset vastaukset, jotka eivät vastanneet tutkimuskysymyksiin ja jotka jäivät opinnäytetyön rajauksen ulkopuolelle. Lisäksi aineistoista rajattiin pois työkalujen oppimiskäyrään liittyvät vastaukset tutkittavan aineiston suuruuden vuoksi. Tiettyihin kysymyksiin vastanneet ajatuskokonaisuudet maalattiin doc-asiakirjassa saman värisiksi. Esimerkiksi kaikki vastaukset, jotka koskivat kysymystä ”Miksi käytätte kyseistä työkalua?”, värjättiin sinisellä. Tämän jälkeen asiayhteyksiä tiivistettiin hyödyntäen Microsoft Word -ohjelman

kommentointi-toiminnallisuutta. Tiivistetyt ilmaukset ryhmiteltiin siten, että esimerkiksi kaikki Bootstrap-työkaluun liittyvät miksi-kysymystä koskevat vastaukset jaettiin omaan doc-asiakirjaan. Taulukossa 12 on esimerkki redusoinnista eli pelkistämisestä, jossa on alkuperäisilmaukset ja niitä vastaavat pelkistetyt ilmaukset.

Taulukko 12. Esimerkki redusoinnista eli pelkistämisestä

Alkuperäisilmoitukset	Pelkistetyt ilmaukset
”…Bootstrappi nyt on aika lailla melkee suosituin tommone… kirjasto ni…”	- Bootstrap suosituin luokassaan
”No seki tuli asiakkaalta, että se oli aikaisemmin käytössä siinä järjestelmässä mitä me tehtiin tai me tehtiin semmoinen… vähän niin kuin oma sovellus, mutta siinä oli koodipohja jo valmiina. Ni se oli jo yleisesti käytössä siinä, ni sieltä se tuli. Ja sehän on aika… Bootstrappi on suosittu ja hyväksi havaittu tämmöinen CSS frameworkki, jolla on sitten helppo tehdä sitä UI:ta, Gridiä.”	- Asiakas päätti, että käytetään - Oli järjestelmässä jo ennestään käytössä - Suosittu - Hyväksi havaittu - Helppo tehdä UI:ta
”Varmaan aika aikataulullisista syistä, että sillä aika äkkiä saa tehtyä… ihan nättiäkin leiskaa, ja aika monellehan se on tuttu. Et varmaan aika samoilla perusteilla kuin tota Reactiakin käytetään, just et se on monelle tuttu ja sillä äkkiä päästään liikkeelle ja nopeeta tehtyy.”	- Bootstrappia käytetään aikataulullisista syistä, eli sillä saa nopeasti tehtyä - Bootstrappin layout on ihan nättiä - Bootstrap on monelle tuttu - Bootstrapilla pääsee nopeasti liikkeelle
”No se on ehkä ihan tullut sitten ihmisten, eri ihmisten kokemuksen myötä, ja toki sitten jotkut asiakaskeissitkin sanelee joissain tapauksissa, että mitä toivotaan käytettävän.”	- Eri ihmisten kokemusten myötä valittu - Osa asiakastapauksista sanelee mitä toivovat käytettävän

Sen jälkeen, kun esimerkiksi kaikki Bootstrap-työkaluun liittyvät miksi-kysymykseen vastaukset oltiin ryhmitelty omaan doc-asiakirjaan, ryhmien sisällä olevat pelkistetyt ilmaukset luokiteltiin samankaltaisuuksien mukaan (ks. taulukko 13).

Taulukko 13. Esimerkki klusteroinnista eli ryhmittelystä

Pelkistetyt ilmaukset	Alaluokat
Bootstrap suosituin luokassaan Suosittu Bootstrap on monelle tuttu	Suosio
Bootstrappia käytetään aikataulullisista syistä, eli sillä saa nopeasti tehtyä Bootstrapilla pääsee nopeasti liikkeelle	Nopeus
Oli järjestelmässä jo ennestään käytössä	Ennestään käytössä
Asiakas päätti, että käytetään Osa asiakastapauksista sanelee mitä toivovat käytettävän	Asiakkaan päätös
Eri ihmisten kokemusten myötä valittu Hyväksi havaittu	Kokemus
Helppo tehdä UI:ta	Helppous
Bootstrappin layout on ihan nättiä	Viehättävyys

Klusteroinnin jälkeen aineisto abstrahoitiin, eli muodostettiin teoreettisia käsitteitä niin kauan kuin mahdollista (ks. taulukko 14). Toisin sanoin samaa asiaa tarkoittavat alaluokat yhdistettiin ja niistä muodostettiin oma, ylempi teoreettinen käsite. Lopuksi kaikki ylemmät käsitteet eli yläluokat yhdistettiin viimeiseen, ylimpään eli yhdistävään luokkaan.

Taulukko 14. Esimerkki abstrahoinnista

Alaluokat	Yläluokat	Yhdistävä luokka
Kokemus Viehättävyys	Tunnepohjaiset kokemukset	Bootstrapin käytön syyt ohjelmistoyrityksissä
Suosio	Verkosto	
Nopeus Helppous	Ominaispiirteet	
Ennestään käytössä Asiakkaan päätös	Projektin lähtökohdat	

Tutkimusaineistoa voidaan analyysin jälkeen kvantifioida, eli jatkaa määrällisin menetelmin. Tuolloin voidaan esimerkiksi laskea, kuinka monta kertaa tietty vastaus esiintyy aineistossa. (Tuomi & Sarajärvi 2009, 120.) Tutkimuksen aineisto kvantifioitiin muutamien vastausten osalta, jotta tuloksia olisi helpompaa tarkastella. Aineistot, joita kvantifioitiin, olivat käytössä olevat työkalut, työkalujen käytön syyt sekä työkalut ja asiat, joita haastateltavat suosittelivat opettelemaan.

5 Tutkimustulokset

Opinnäytetyön tutkimustulokset esitetään teemahaastattelurungon mukaisesti. Osaan suosituimmista työkaluista haastateltavilta kysyttiin käytettävän työkalun hyviä ja huonoja puolia. Näiden kysymysten tarkoitus oli auttaa tutkijaa tekemään päätökset liittyen suositeltavan työkalupakin muodostamiseen. Haastateltavat ovat joidenkin työkalujen kohdalla maininneet vain joko hyvät tai huonot puolet riippuen siitä, mitä mielipiteitä heillä on ollut. Tuloksissa on esitetty myös aiheeseen sopivaksi koettuja haastateltavien lainauksia sisennettynä ja kursivoituna. Yritysten nimet ovat sensuroitu viidellä tähdellä (*), mikäli nimi on esiintynyt haastateltavien lainauksissa.

5.1 Käytössä olevat front end -työkalut

Tässä alaluvussa käsitellään front end -työkaluihin liittyviä vastauksia. Työkaluista käsitellään seuraavat: koodin kirjoitusvälineet, JavaScript-sovelluskehykset ja -kirjastot, CSS-sovelluskehykset ja käyttöliittymän muotoilu sekä kääntäjät ja käännettävät kielet. Lisäksi käsitellään CSS:n tyylittelyyn, koodin tarkistamiseen, paketinhallintaan, tehtävänsuoritukseen, moduulien niputukseen ja versionhallintaan liittyvät työkalut sekä lopuksi testaustyökalut.

5.1.1 Koodin kirjoitusvälineet

Tutkimuksesta ilmeni, että yrityksissä käytetään front end -ohjelmoimiseen koodieditoreita, kehitysympäristöjä ja terminaalipohjaisia editoreja. Eniten oli käytössä koodieditoreita, joista yrityksissä käytetyimmät ja suosituimmat olivat Atom, Sublime Text ja Visual Studio Code.

Materiaalista nousi esiin ohjelmistoyrityksissä *koodin kirjoitusvälineiden valintaan* liittyviä asioita, jotka liittyivät henkilökohtaiseen tukemiseen ja optimoimiseen. Henkilökohtainen tukeminen koski valinnanvapautta, suosittelua ja tukemista. Liittyen valinnanvapauteen, työntekijät saivat valita vapaasti ohjelmointiin käyttämänsä editorin. Yrityksissä ei sanota mitä tulisi käyttää, vaan valinta tehdään sen mukaan, mikä ohjelmoijalle toimii parhaiten. Erään vastaajan mukaan editorin saa valita vapaasti, koska ulospäin ei näy millä koodi on kirjoitettu. Toisin sanoen, ohjelmoinnin tulos on samannäköinen.

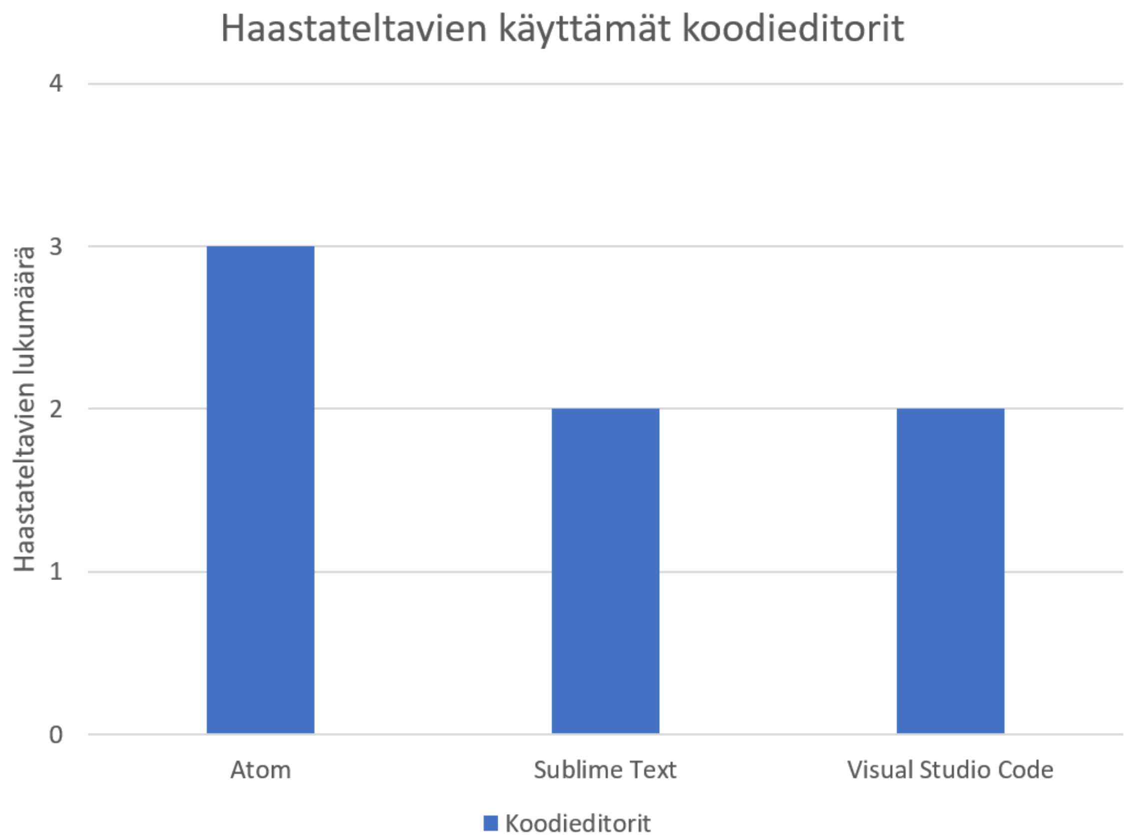
Liittyen suositteluun, eräässä yrityksessä on yleinen käytäntö, että ohjelmoimiseen käytetään Atomia. Toisessa yrityksessä jokaiselle kehittäjälle asennetaan työsuhteen alkaessa Visual Studio 2017. Tukemiseen liittyen, erään vastaajan mukaan yrityksessä kysytään mitä ohjelmoijat haluavat käyttää, jonka pohjalta valitaan yhteinen koodin kirjoitustyökalu.

Toisessa yrityksessä tuetaan lisenssin hankinnassa. Jos ohjelmoija haluaa koodieditoriin lisenssin, se hankitaan. Optimoiminen koski selvittämistä. Yhdessä yrityksessä otetaan selvää mikä on mihinkin tilanteeseen paras ratkaisu.

Materiaalista nousi esiin myös *haastateltavien toteamuksia* liittyen kehitysympäristöjen ominaispiirteisiin. Vastaajat kokivat kehitysympäristöjen olevan tehottomia. Erään haastateltavan mukaan heidän yrityksessä front end -puolella ei ole käytössä kehitysympäristöjä, koska ne ovat raskaita. Muutama haastateltavista kertoi kokeilleensa kehitysympäristöjen käyttöä ja todenneensa, että ne ovat raskaita. Eräs kokeilleista koki, että kehitysympäristöissä ei ole tarpeeksi potkua pitämään montaa tiedostoa auki yhtä aikaa.

> *”Mä en oo oikeen IDE:ä… mä käytin itekkin, mä oon käyttäny Web Stormia, mä oon käyttäny Eclipseä, mä oon käyttäny PHP Stormia, mutta… ne on niin raskaita… että mullakin on parhaillaan neljää, viittä, kymmentä tiedostoa yhtä aikaa auki ja mä parhaimillaan teen kymmeneen, viitentoistaan tiedostoon yhtä aikaa hommia… niin se vaatii aika paljon potkua koneelta.”*

Koska lähtökohtaisesti haastateltavat saivat itse päättää mitä koodin kirjoitustyökalua käyttävät ohjelmoimiseen, kysyttiin heiltä syitä työkalujen valintaan. Kahdeksasta haastateltavasta seitsemän ilmoitti ohjelmoivansa työssään. Kuvion 6 mukaan haastateltavat käyttivät eri koodieditoreita hyvin tasaisesti.



Kuvio 6. Haastateltavien käyttämät koodieditorit (n = 7)

Atomin valintaan vaikutti ominaispiirteet, tunnepohjaiset kokemukset ja hinta. Liittyen ominaispiirteisiin, Atom koettiin tehokkaaksi, monipuoliseksi ja kattavaksi. Tehokkuuteen liittyen haastateltavat kokivat Atomin todella kevyeksi. Atomia voi pyörittää myös huonommallakin koneella. Erään vastaajan mukaan JavaScript-pohjaisen Atomin kehitys pääsi ikään kuin kynnyksen yli, ettei se ollut enää hidas, jonka vuoksi sitä on miellyttävä käyttää.

Liittyen monipuolisuuteen, Atomiin on saatavilla paljon erilaisia koodausta helpottavia lisäosia. Erään vastaajan mukaan muun muassa TypeScript ja Angular -lisäosat sekä automaattinen täydennys ovat hyvä lisä. Liittyen kattavuuteen, eräs vastaajista koki, että Atomissa on kaikki mitä hän tarvitsee. Tunnepohjaiset kokemukset koskivat häiritsemättömyyttä. Yksi haastateltavista oli mielissään, ettei Atomissa ole huomautuksia liittyen maksullisen version ostoon, joita taas Sublime Textissä on. Lopuksi hintaan liittyen, muutama haastateltavista mainitsi myös hinnan työkalun valintaan. Atomin valintaan liittyi myös se, että se on ilmainen.

Atomin hyvät puolet liittyivät aikaan, tunnepohjaisiin kokemuksiin ja ominaispiirteisiin. Aika liittyi nykyaikaisuuteen. Käyttäjät pitivät hyvänä puolena sitä, että Atom tuntuu modernilta ja tekee paljon uusia asioita. Tunnepohjaiset kokemukset koskivat miellyttävyyttä

ja häiritsemättömyyttä. Miellyttävyyteen liittyen Atomia on miellyttävä käyttää. Häiritsemättömyyteen liittyen Atom antaa tehdä työn rauhassa eikä häiritse. Ominaispiirteet koskivat lisäosien monipuolisuutta ja muokattavuutta. Hyviksi puoliksi luettiin sen, että Atomilla on paljon erilaisia lisäosia. Erikseen mainittiin Reactiin sekä eri kieliä ja niiden versioita varten liittyvät osat. Muokattavuuteen liittyi lisäosien helppo asentaminen. Myös selkeät väriteemat olivat puolia, joista käyttäjät pitivät.

Atomin huonot puolet sen sijaan liittyivät ominaispiirteisiin, ja ominaispiirteistä nousi esiin tehottomuus. Haastateltavien mukaan Atom on tehoton, joka käytännössä ilmenee muun muassa hidasteluna sekä koneen tehojen suurella kulutuksella. Erään haastateltavan mukaan editori ilmoittaa virheestä, jos lisäosia on paljon, vaikka koodiin ei olisi koskettu. Tästä seuraa se, että editori on käynnistettävä uudelleen. Taulukkoon 15 on erikseen kerättynä haastattelujen pohjalta hyvät ja huonot puolet. Taulukosta ilmenee, että hyviä puolia on huomattavasti huonoja puolia enemmän.

Taulukko 15. Atomin hyvät ja huonot puolet

Hyvät puolet	Huonot puolet
• Kustomoitava • Monipuolisesti lisäosia • Reactiin liittyvät lisäosat • Eri kieliä ja niiden versioita varten liittyvät osat • Selkeät väriteemat • Lisäosien asennus helppoa • Modernin tuntuinen • Tekee paljon uusia asioita • Miellyttävä käyttää • Ei häiritse	• Hidastelee välillä • Syö koneen tehoja • Heikko kaatumisen ehkäisy

Visual Studio Coden valinta liittyi ominaispiirteisiin ja tunnepohjaisiin kokemuksiin. Ominaispiirteet koskivat yhteensopivuutta, nopeutta ja monipuolisuutta. Liittyen yhteensopivuuteen erään vastaajan mukaan TypeScript toimii parhaiten Visual Studio Coden kanssa. Haastateltavan mukaan TypeScript ja VS Code ovat molemmat Microsoftin kehittämiä, jonka vuoksi ne toimivat hyvin yhdessä. Nopeuteen liittyen editorin koettiin olevan tarpeeksi nopea. Monipuolisuutta koskien editorissa koettiin olevan tarpeeksi hyviä ominaisuuksia. Tunnepohjaiset kokemukset liittyivät paremmuuteen. Eräs vastaaja oli sitä mieltä, että Visual Studio Codessa on Atomin ja Sublime Textin parhaat puolet, ja että editorin täydennys-ominaisuus toimii paremmin kuin muissa editoreissa.

> *”No Sublime Text on aika nopee, tykkäsin siinä siitä. Ja sitten Atom nyt on tietysti muuten vaan parempi, tykkäsin siitä sen takia, ja sit tossa Visual Studiossa on vähän molempien ehkä parhaat puolet. Et se on aika nopee ja sit siinä on tarpeeks*

kaikkia herkkuja. Ja sit siinä on tota intellisenseäki vähä, eli toi täydennys toimii aika paljon paremmin kuin missään muussa.”

Haastateltavat kokivat *Visual Studio Coden hyviksi puoliksi* ominaispiirteisiin ja tunnepohjaisiin kokemuksiin liittyvät asiat. Ominaispiirteet koskivat toimivuutta ja tehokkuutta. Liittyen toimivuuteen, erään vastaajan mukaan editorissa toimii hyvin linttaus, eli koodin tarkistaminen. Kun linttaus on asetettu hyvin, myös automaattinen korjaus toimii tosi hyvin. Lisäksi koodin formatointi ja koodin korostus toimivat mainiosti. Editori on toiminut vastaajien mukaan kaikin puolin hyvin. Tehokkuuteen liittyen eräs vastaajista koki, että Atomiin verratessa Visual Studio Codessa ei ole tullut suorituskyvyllisiä ongelmia. Tunnepohjaiset kokemukset koskivat paremmuutta. Editorissa ei koettu olevan vain yhtä hyvää asiaa, vaan parhaat puolet kaikista editoreista.

Visual Studio Coden huonoiksi puoliksi haastateltavat kokivat ominaispiirteisiin liittyvät asiat, jotka koskivat huonoa käytettävyyttä. Erään vastaajan mukaan editorin pikanäppäimet ovat erilaiset kuin mihin hän oli tottunut Atomissa ja Sublime Textissä. Haastateltava mainitsi kuitenkin, että ne saa säädettyä itselle sopiviksi ja ne ovat nopeasti opeteltavissa. Lisäksi tiedoston joutuu tallentamaan monta kertaa, jotta editori korjaa kaikki virheet. Erään vastaajan mukaan on ikävää, että oletusasetuksiin ei pääse käyttöliittymän kautta käsiksi. Vastaaja koki, että editorissa on ristiriitaa asetusten määrittämistiedoston ja käyttöliittymän välillä.

”…mikä on vähän itestä ikävää, että kun sen asetuksia säätää, niin perusasetuksista mikä on puhtaasti konffifilu, et ei oo tehty niille asetuksille oikee käyttöliittymää ilmeisesti, tai ainakaan ite en kokenu että se ois ollu riittävän hyvä. Niin sitten kun sieltä säädetään se, et kuinka monta speissiä yks täbi on, niin siellä pystyy säätää vaan yhden arvon, mut sitten taas kaikille eri tiedostotyypeille on kuitenki olemassa oma arvonsa minkä voi sit säätää footerista. Nii vähän semmosta ristiriitaa, et voit säätää yleisen defaultin konffiin, mut et pysty säätämään niitä eri tiedostotyyppejä.”

Taulukossa 16 ilmenee tiivistetyssä muodossa Visual Studio Coden hyvät ja huonot puolet. Kuten taulukosta näkyy, usea VS Coden hyvä ominaisuus on liittynyt koodin kirjoitusvaiheessa esiintyviin toiminnallisuuksiin. Huonoissa puolissa on sen sijaan korostunut käyttöliittymän käytettävyyden ongelmat.

Taulukko 16. Visual Studio Coden hyvät ja huonot puolet

Hyvät puolet	Huonot puolet
• Koodin tarkistus toimii tosi hyvin • Automaattinen tekstinkorjaus toimii hyvin • Koodin formatointi toimii hyvin • Koodin korostus toimii hyvin • Ei suorituskyvyllisiä ongelmia • Kaikin puolin toimii hyvin • Parhaat puolet kaikista editoreista	• Tiedoston joutuu tallentamaan monesti, jotta kaikki virheet korjaantuvat • Oletusasetuksiin ei pääse käyttöliittymän kautta • Ristiriitaa asetusten määrittämistiedoston ja käyttöliittymän välillä • Eri pikanäppäimet kuin Atomissa ja Sublime Textissä

Syyt Sublime Textin käyttöön liittyivät ominaispiirteisiin ja tunnepohjaisiin kokemuksiin. Ominaispiirteisiin liittyviä asioita olivat tehokkuus, kustomoitavuus ja monipuolisuus. Tehokkuuteen liittyen erään käyttäjän mukaan editori on kevyt ja toimii sujuvasti Macintosh-tietokoneella. Kustomoitavuuteen liittyi se, että Sublime Text on kustomoitava ja että sen pystyy kustomoimaan minimaaliseksi. Monipuolisuuteen liittyen, editorilla on hyvä tuki eri kielten syntaksin korostukselle. Tunnepohjaiset kokemukset koskivat viehättävyyttä. Sublime Text koettiin nätin näköiseksi.

Sublime Textin hyvät puolet liittyivät ominaispiirteisiin, jotka koskivat tehokkuutta, monipuolisuutta ja toimivuutta. Tehokkuuteen liittyen editori mainittiin nopeaksi, ja että se toimii todella nopeasti myös linttereiden kanssa. Erään vastaajan mukaan hän pystyy pitämään puoli vuotta projektia auki, jolloin konetta ei tarvitse käynnistää uudelleen. Monipuolisuuteen liittyen, Sublime Textistä löytyy paljon pakettimanagerin kautta asennettavia lisäosia. Toimivuuteen liittyen editorissa toimii hyvin erilaiset väriteemat. Lisäksi tiimille pystyy asettamaan samat koodin kirjoitusmääräykset. Myös editorin hakutoiminnon koettiin olevan todella hyvä.

Editorin huonot puolet liittyivät ominaispiirteisiin, jotka koskivat tehottomuutta, huonoja lisäosia ja puutteellisuutta. Liittyen tehottomuuteen, erään vastaajan mukaan editori ei osaa palauttaa ikkunoita Macintoshissa samoille virtuaalipöydille, missä ne ovat aiemmin olleet ennen koneen uudelleenkäynnistämistä. Lisäksi Sublime Text kaatuu Windowsilla noin kolme kertaa viikossa. Liittyen huonoihin lisäosiin, yhden käyttäjän mukaan editoriin saa lisäosia, mutta ne eivät ole kovin hyviä. Lopuksi Sublime Text koettiin jokseenkin puutteelliseksi. Vastaaja toivoi editoriin hyvin tehtyä Git-integraatiota, ja olisi valmis jopa maksamaan sellaisesta.

Taulukossa 17 on kerättynä Sublime Textin hyvät ja huonot puolet. Hyvissä ja huonoissa puolissa on huomattavissa hieman päällekkäisyyttä esimerkiksi lisäosien osalta. Lisäksi ristiriitaisuutta esiintyy myös liittyen siihen, että Sublime Text pystyy hyvien puolien mu-

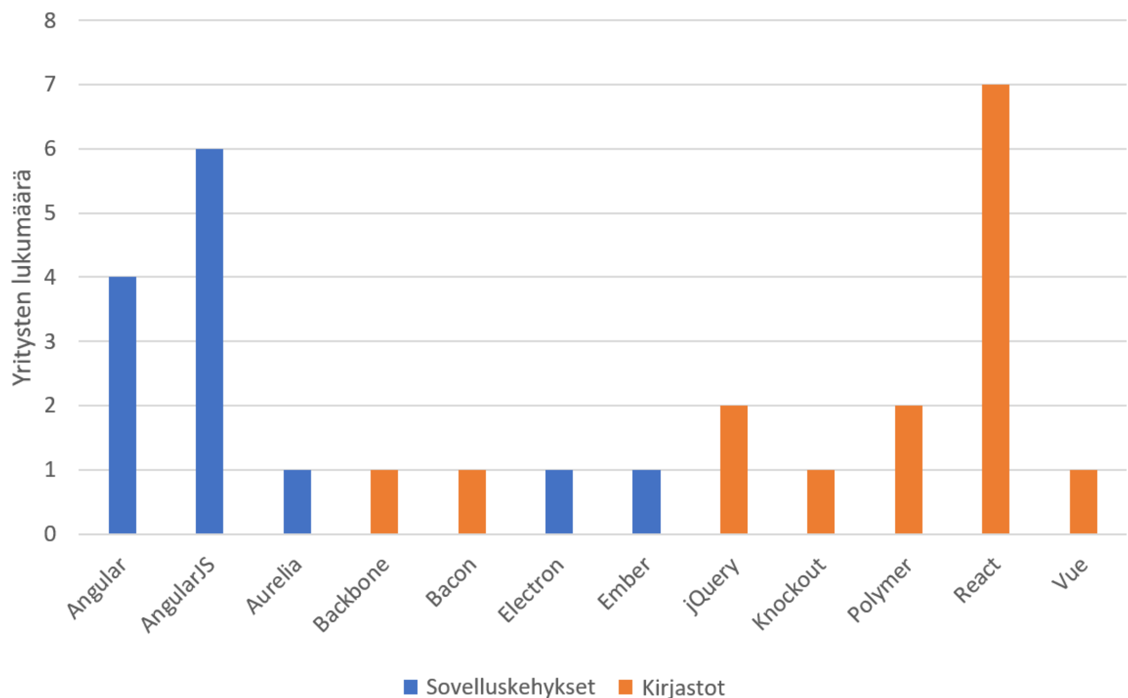
kaan pitämään ikkunoita auki pitkään ja toisaalta huonoissa puolissa sovelluksen on mainittu kaatuvan usein.

Taulukko 17. Sublime Textin hyviä ja huonoja puolia

Hyvät puolet	Huonot puolet
• Nopea • Toimii nopeasti linttereiden kanssa • Pystyy pitämään auki kuukausia käynnistämättä konetta • Paljon lisäosia • Toimivat väriteemat • Pystyy asettamaan tiimille samat koodin kirjoitusmääräykset • Hakutoiminto on tosi hyvä	• Ei osaa palauttaa ikkunoita samoille virtuaalipöydille Macintoshissa • Kaatuu Windowsilla monesti • Ei hyviä lisäosia • Hyvin tehty Git-integraatio puuttuu

5.1.2 JavaScript-sovelluskehykset ja -kirjastot

Haastatteluista ilmeni, että yrityksissä käytetään sekä JavaScript-sovelluskehyksiä että -kirjastoja ohjelmoimisen apuna (ks. kuvio 7). Luvusta jätettiin tarkastelun ulkopuolelle seuraavat työkalut Aurelia, Ember, Knockout, Vue ja Backbone, sillä yksi haastateltavista ei ottanut kantaa miksi kyseisiä työkaluja käytetään. Lisäksi kolmen haastateltavan kohdalla jäi epäselväksi, puhuttiinko ”Angulariin” viitattaessa AngularJS:stä vai kokonaan uudesta versiosta Angularista. Kyseisille haastateltaville lähetettiin tarkentava pyyntö, johon vastasi yksi. Kahden muun kohdalla ”Angulariin” viittaaminen merkittiin AngularJS:ksi, koska sitä pidettiin todennäköisempänä vaihtoehtona ikänsä vuoksi.



Kuvio 7. Ohjelmistoyrityksissä käytössä olevat JavaScript-sovelluskehykset ja -kirjastot (n = 8)

Suosituin työkalu oli React, jota mainittiin käytettävän jopa seitsemässä yrityksessä. Lisäksi haastatteluista kävi ilmi, että Reactia käytetään yritysten sisällä eniten verrattuna muihin JavaScript-sovelluskehyksiin ja -kirjastoihin. Muutaman vastaajan mukaan Reactilla tehdään kaikki uudet projektit. Erään haastateltavan mukaan Reactilla tehdään lähes 80% asiakasprojekteista, ja toisen mukaan React on selvästi käytetyin työkalu yrityksessä.

Toiseksi suosituin työkalu oli vastaajien mukaan AngularJS. Muutama haastateltava mainitsi, että AngularJS:ää käytetään vanhoissa projekteissa. Myös Angularia oli käytössä jo neljässä yrityksessä.

Vastaajien mukaan *syyt Reactin käyttöön* yrityksissä liittyivät verkostoon, tunnepohjaisiin kokemuksiin, ominaispiirteisiin ja projektin lähtökohtiin. Verkostoon liittyvät asiat koskivat avointa lähdekoodia, ylläpidettävyyttä ja suosiota. Se, että React perustuu avoimeen lähdekoodiin, koettiin yhtenä syynä valintaan. Liittyen ylläpidettävyyteen, haastateltavien mukaan työkalun valintaan on vaikuttanut yleinen tuki, sillä React on hyvin ylläpidetty. Erään vastaajan mukaan asiakasta saattaa pelottaa, että liian eksoottisen sovelluskehyksen

valinnan johdosta tulevaisuudessa ei löydy enää osaajia. Liittyen suosioon, vastaajien mukaan suurin tekijä työkalun valintaan on suosio, jonka myötä löytyy myös osaajia.

Tunnepohjaiset kokemukset koskivat miellyttävyyttä. Haastateltavien mukaan ohjelmoijat ovat todenneet kokeilemalla työkalun olevan hyvä ja että sitä on miellyttävä käyttää. Valintaan on vaikuttanut myös ohjelmoijien matala kynnys lähteä ohjelmoimaan Reactilla. Erään haastateltavan mukaan parasta on ollut se, että kukaan ei ole valittanut Reactista, mikä on ollut hyvä merkki työkalun kokonaisvaltaisesta toimivuudesta.

Ominaispiirteet koskivat helppokäyttöisyyttä ja yhteyttä mobiilikehitykseen. Helppokäyttöisyyteen liittyen, haastateltavien mukaan Reactin valintaan vaikutti tekemisen tehokkuus. React on helppo ja sillä on yksinkertainen konsepti. Moni myös osaa Reactia, ja uudet työntekijät oppivat sen helposti. Mobiilikehitykseen liittyen erään haastateltavan mukaan Reactilla on silta mobiilikehitykseen, jolloin on luontevaa käyttää samaa kirjastoa pohjalla. React Nativella voi tehdä helposti mobiilikehitystä Reactilla.

Projektin lähtökohtiin liittyvät syyt koskivat asiakkaan valintaa, kehittäjien valintaa ja asiakkaan menettämistä. Vastaajien mukaan Reactin oli valinnut joko asiakas käytettäväksi tai työkalu oltiin kehittäjien kanssa valittu. Lisäksi eräs haastateltavista mainitsi, että yksi syy Reactin käyttöön liittyi asiakkaan menetykseen. Yritys oli menettänyt asiakkaan johtuen siitä, että joku toinen yritys osasi myydä React-osaamistaan paremmin. Silloin haastateltava totesi, että asiaan tehdään muutos.

Reactin hyvät puolet liittyivät ominaispiirteisiin, tunnepohjaisiin kokemuksiin ja verkostoon. Ominaispiirteet koskivat joustavuutta, dynaamisuutta, tehokkuutta, skaalautuvuutta, vaivattomuutta, helppokäyttöisyyttä, selkeyttä ja yhdenmukaisuutta. Joustavuuteen liittyen Reactilla pystyy tekemään helposti komponentteja asioista, joita pystyy käyttämään joustavasti ja haluamallaan tavalla. Lisäksi erään vastaajan mukaan Reactin tilaaja-kuuntelija-ajattelumalli antaa paljon vapautta tekemiseen. Lisäksi hyvänä asiana pidettiin Reactin modulaarisuutta, että se ei sido liikaa tiettyyn kaavaan tai kehykseen, vaan sitä voi käyttää sen verran kuin on tarvetta. Liittyen dynaamisuuteen, vastaajien mukaan React on dynaaminen verrattuna vanhempiin Java-pohjaisiin web-teknologioihin.

Liittyen tehokkuuteen, React nähtiin tehokkaana työkaluna ja erityisesti DOM:n manipuloinnissa. Erään haastateltavan mukaan on hämmentävää, että Reactilla pystyy tekemään sovelluksen todella lyhyessä ajassa, mikä on suuri muutos vanhaan. Lisäksi React vähentää tehokkaasti virheitä, ja vaikka niitä tulee, ovat ne helposti löydettävissä Reactin työkalupakin avulla. Skaalautuvuuteen liittyen kirjasto nähtiin skaalautuvan hyvin.

Koskien vaivattomuutta, erään haastateltavan mukaan Reactin modulaarisuuden takia koodia tarvitsee kirjoittaa vähemmän. Liittyen helppokäyttöisyyteen, haastateltavat kokivat, että Reactilla on todella helppo ohjelmoida, ja että React tekee perinteisesti vaikeista asioista helppoja. Koskien selkeyttä, vastaajien mukaan kirjasto on selkeä oppia ja kirjaston arkkitehtuurin saa pidettyä selkeänä. Liittyen yhdenmukaisuuteen, erään vastaajan mukaan on todella helppo aloittaa tekemään natiivia sovellusta React Nativella, jos osaa Reactia.

Tunnepohjaiset kokemukset liittyivät kokemukseen Reactista. Erään vastaajan mukaan Reactilla tekeminen on kuin taikuutta. Verkosto taas liittyi laajaan ekosysteemiin. Vastaajien mukaan Reactissa on suuri ekosysteemi johtuen suuresta suosiosta.

> *”Se tekee mun mielest semmosist asioista mitkä on ollu perinteisesti tosi vaikeita webbikehitykseen, ni se tekee niist tosi helppoja. Et kun se prinsiippi on se, että piirrä käyttöliittymä jostain sovelluksen tilasta tällä nimenomaisella hetkellä, ni se on vaan niin lapsellisen helppoa tehdä se.”*

Reactin huonot puolet liittyivät ominaispiirteisiin ja rajoitteisiin. Ominaispiirteet koskivat liiallista joustavuutta ja huonoa käytettävyyttä. Liialliseen joustavuuteen liittyen eri ihmisten tai tiimien tekemät projektit voivat olla todella eri näköisiä. Reactin modulaarisuuden takia se ei sido tiettyyn ohjelmointitapaan. Koska React on kirjasto eikä sovelluskehys, sen ympärille joutuu kokoamaan itse erilaisia vaihtoehtoja. Huonoon käytettävyyteen liittyen, erään vastaajan mukaan ominaisuuksien (engl. property) ja tilojen hallinnan käyttö (engl. state) voi olla sekavaa.

> *”…todella helposti mennään siihen et käytetään liikaa tilojen hallintaa. Elikkä komponentti itessään voi muuttaa, aatellaan komponentilla voi vaikka arvoa että onko se auki vai ei… niin monesti liian helposti lähetään siihen, että sen pystyy kontrolloimaan sitä omaa tilaansa… kun sen ei välttämättä tarvii, et pelkkä property riittäis siinä.”*

Rajoitteet koskivat lisenssin kyseenalaisuutta. Haastateltavat kokivat epävarmuutta lisenssin käyttöön liittyen ja kokivat sen vaikeaksi. Erään haastateltavan mukaan Facebook voi evätä lisenssin käytön niin halutessaan. Esimerkiksi yksi syy lisenssin epäämiseen voi olla riita toisen yrityksen kanssa, joka käyttää Reactia.

Seuraavassa taulukossa on listattuna Reactin hyvät ja huonot puolet (ks. taulukko 18). Kirjastolla on vastausten perusteella enemmän hyviä kuin huonoja puolia. Hyvät puolet liittyivät suurilta osin kirjaston ominaisuuksiin.

Taulukko 18. Reactin hyvät ja huonot puolet

Hyvät puolet	Huonot puolet
• Komponenttien tekeminen helppoa • Komponenttien käyttö joustavaa • Yhteys mobiilikehitykseen • Dynaaminen • Tilaaja-kuuntelija-ajattelumalli antaa vapautta • Ei sido tiettyyn tapaan ohjelmoida • Tehokas • Tehokas DOM:n manipuloinnissa • Vähentää virheitä • Skaalautuu hyvin • Pienempi koodin kirjoitusmäärä • Helppokäyttöinen • Selkeä oppia • Arkkitehtuurin saa pidettyä selkeä-nä • Laaja ekosysteemi	• Kyseenalainen lisenssi • Ei sido tiettyyn tapaan ohjelmoida • Eri ihmisten projektit voivat olla eri näköisiä • Ympärille joutuu kokoamaan itse erilaisia vaihtoehtoja • Staten ja propertyn käyttö voi olla sekavaa

Sovelluskehyksistä käytetyin oli AngularJS. Haastateltavien mukaan *syyt AngularJS:n käyttöön* yrityksissä liittyivät projektin lähtökohtiin ja verkostoon. Projektin lähtökohdat liittyivät kehittäjien kesken päättämiseen sekä asiakkaan valintaan. Haastateltavien mukaan kehittäjien kanssa on mietitty, millä sovelluskehyksellä tai kirjastolla kannattaisi sovellusta lähteä tekemään. Erään haastateltavan mukaan työkalun valinta on yleensä asiakkaan, koska asiakkaalle tehdään töitä. Koskien verkostoa, syyt työkalun valintaan liittyivät haastateltavien mukaan osaamiseen. Vastaajien mukaan suurin vaikuttaja työkalun valintaan on suosio, jonka myötä löytyy osaajia. Erään vastaajan mukaan asiakasta saattaa pelottaa, että tulevaisuudessa ei löydy enää osaajia, jos työkalu ei ole tunnettu.

AngularJS:n hyvä puoli koski tunnepohjaista kokemusta, joka liittyi miellyttävään tulokseen. Erään vastaajan mielestä sillä saa hienoja juttuja aikaan. Vastaavasti *AngularJS:n huonot puolet* liittyivät ominaispiirteisiin ja tunnepohjaisiin kokemuksiin. Ominaispiirteet liittyivät heikkoon suunnitteluun ja joustamattomuuteen. Liittyen suunnitteluun, haastateltavien mukaan sovelluskehys on huonosti suunniteltu, sillä se lähestyy ratkaistavia ongelmia väärästä suunnasta. Lisäksi sovelluskehyksen abstraktiot koettiin huonoiksi. Joustamattomuuteen liittyen sovelluskehyksen huono puoli on MVC-arkkitehtuuri, koska silloin ei ole täyttä kontrollia siitä mitä tekee. Vastaajat kokivat, että vahva lainalaisuus siitä, miten asiat pitää tehdä ei aina ole paras ratkaisu. Lopuksi tunnepohjaiset kokemukset liittyivät huonoihin kokemuksiin. Eräs vastaaja koki AngularJS:n tolkuttomaksi, huonoksi ja järjettömäksi.

Syyt Angularin käyttöön liittyivät projektin lähtökohtiin ja tunnepohjaisiin kokemuksiin. Liittyen projektin lähtökohtiin Angularin käyttöönotto oli päätetty kehittäjien kesken. Eräässä yrityksessä vertailtiin suosituimpia JavaScript-sovelluskehyksiä ja -kirjastoja, kuten Angularia ja Reactia, jonka jälkeen päädyttiin Angularin valintaan. Haastateltavan mukaan yrityksessä koettiin, että Angular sopii parhaiten projektiin. Lisäksi Angularin käytön syy johtui asiakkaan valinnasta. Tunnepohjaisetkokemukset koskivat aikaisempia kokemuksia. Viimeisimmästä Angularista on hyviä kokemuksia, sillä sovelluksen ideologia on lähellä Reactia.

Angularin hyvät puolet koskivat verkostoa, ominaispiirteitä ja tunnepohjaisia kokemuksia. Verkosto liittyi hyvään tukeen. Erään vastaajan mukaan Angularissa on laaja tuki ja käyttäjäkunta. Ominaispiirteet koskivat infrastruktuuria, helpottamista ja nopeutta. Liittyen infrastruktuuriin, erään haastateltavan mukaan Angular jakaantuu useisiin komponentteihin, joissa HTML, TypeScript ja CSS ovat erikseen. Helpottamiseen liittyen Angularissa on käytössä Angular CLI, jolla luodaan yksinkertaisilla komennoilla automaattisesti komponentteja. Angular CLI luo muun muassa rungot valmiiksi, jolloin on helppo tehdä uusia toiminnallisuuksia. Nopeuteen liittyen Angular koettiin latautuvan nopeasti. Lopuksi tunnepohjaiset kokemukset koskivat miellyttävää tulosta. Angular koettiin hienoksi siinä mielessä, että sillä saa hienoja juttuja aikaan.

Angularin huonot puolet liittyivät ominaispiirteisiin ja tunnepohjaisiin kokemuksiin. Ominaispiirteet liittivät jyrkkään oppimiskäyrään, heikkoon suunnitteluun ja joustamattomuuteen. Koskien oppimiskäyrää, erään haastateltavan mukaan Angularissa on jyrkkä oppimiskäyrä. Liittyen heikkoon suunnitteluun, erään vastaajan mukaan Angular on huonosti suunniteltu, sen abstraktiot ovat huonoja ja se lähestyy ratkaistavia ongelmia väärästä suunnasta. Liittyen joustamattomuuteen, Angular nähtiin joustamattomana MVC-arkkitehtuurin takia, koska silloin ei ole täyttä kontrollia siitä mitä tekee. Tunnepohjaiset kokemukset koskivat huonoja kokemuksia. Haastateltava koki, ettei Angularissa ole mitään järkeä, se on tolkuton ja huono.

Seuraavassa taulukossa on listattuna Angularin hyvät ja huonot puolet (ks. taulukko 19). Angularilla oli vastausten perusteella enemmän huonoja kuin hyviä puolia, ja ne liittyvät sovelluskehyksen arkkitehtuuriin.

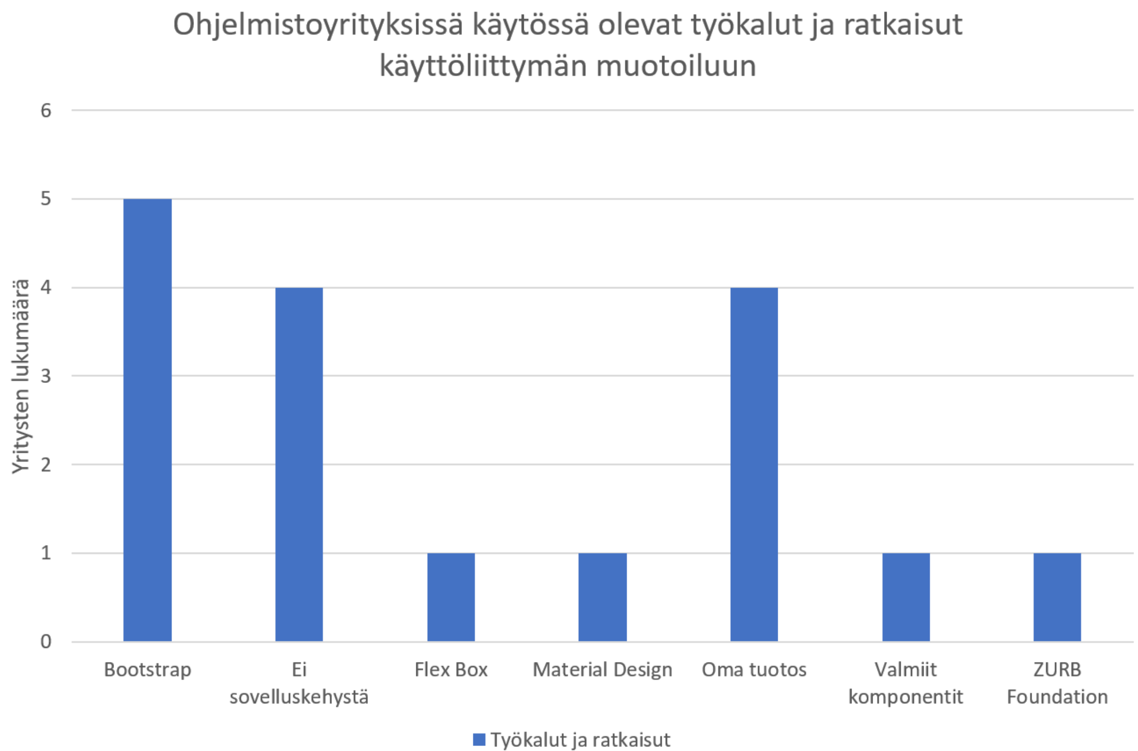
Taulukko 19. Angularin hyvät ja huonot puolet

Hyvät puolet	Huonot puolet
• Jakaantuu useisiin komponentteihin • Angular CLI • Latautuu nopeasti • Iso tuki ja käyttäjäkunta • Saa hienoja juttuja aikaan	• Jyrkkä oppimiskäyrä • Huonosti suunniteltu • Abstraktiot huonoja • Järjetön • Tolkuton • Huono • Joustamaton arkkitehtuuri

Muiden JavaScript-sovelluskehysten ja -kirjastojen käytön syyt liittyivät projektin lähtökohtiin ja toteutukseen liittyviin ratkaisuihin. Projektin lähtökohdat koskivat asiakkaan valintaa ja vuosikausia sitten valitsemista. Liittyen asiakkaan valintaan, Polymerin valinta liittyi asiakasvaatimukseen. Liittyen vuosikausia sitten valitsemiseen, erään haastateltavan mukaan jQuery ja Bacon oli valittu vuosikausia sitten, jonka vuoksi niitä käytetään edelleen projekteissa. Toteutukseen liittyvät ratkaisut koskivat infonäyttö-sovellus-ratkaisuja ja pilvitoteutuksia. Eräässä yrityksessä Electronia käytetään sovelluksissa, joissa on fyysinen rauta, eli muun muassa infonäyttö-sovellus-ratkaisuissa.

5.1.3 CSS-sovelluskehykset ja käyttöliittymän muotoilu

Tutkimuksesta ilmeni, että yrityksissä käytetään CSS-sovelluskehyksiä, asettelumallia, muotokieltä, valmiita komponentteja sekä omia tuotoksia, kuten esimerkiksi komponenttikirjastoja (kuvio 8). Haastatteluista kävi myös ilmi, että noin puolet yrityksistä eivät käytä tai eivät mielellään käytä sovelluskehyksiä.



Kuvio 8. Ohjelmistoyrityksissä käytössä olevat työkalut ja ratkaisut käyttöliittymän muotoiluun (n = 8)

Bootstrapin käyttö yrityksissä liittyi tunnepohjaisiin kokemuksiin, verkostoon, ominaispiirteisiin ja projektin lähtökohtiin. Tunnepohjaiset kokemukset liittyivät kokemukseen ja viehättävyyteen. Kokemukseen liittyen Bootstrap oli havaittu hyväksi, ja se valittiin eri henkilöiden kokemuksien myötä. Viehättävyyteen liittyen Bootstrapin ulkoasu koettiin nätin näköiseksi. Verkosto koski suosiota. Vastaajien mukaan Boostrap on suosituin CSS-sovelluskehys ja monelle tuttu. Juuri työkalun tunnettavuuden takia Bootstrap on vaivatonta ottaa käyttöön.

Ominaispiirteet koskivat nopeutta ja helppoutta. Liittyen nopeuteen, sovelluskehystä käytetään aikataulullisista syistä, koska sillä pääsee nopeasti liikkeelle ja saa nopeasti tehtyä. Helppouteen liittyen, haastateltavien mukaan Bootstrapilla on helppo tehdä käyttöliittymää. Syyt projektin lähtökohtiin liittyivät asiakkaan päätökseen ja siihen, että työkalu oli ennestään käytössä. Vastaajien mukaan osa asiakastapauksista sanelee mitä toivovat käyttävän tai asiakas päättää suoraan mitä tulisi käyttää. Erään vastaajan mukaan Bootstrap oli jo ennestään käytössä, jonka vuoksi se jäi projektiin.

> *”Varmaan aika aikataulullisista syistä, että sillä aika äkkiä saa tehtyä… ihan nättiäkin leiskaa. Ja aika monellehan se on tuttu.”*

Haastateltavien mukaan *Bootstrapin hyvät puolet* liittyivät ominaispiirteisiin ja hintaan. Ominaispiirteet liittyivät selkeyteen, helppokäyttöisyyteen ja kattavuuteen. Liittyen selkeyteen, vastaajat kokivat Bootstrapin sivut selkeinä ja informatiivisina. Liittyen helppokäyttöisyyteen, vastaajien mukaan Bootstrap on vaivatonta ottaa käyttöön. Lisäksi sovelluskehyksessä on vaivatonta tehdä sivusto eri selaimille sopivaksi. Kattavuuteen liittyen, valikoimasta löytyy paljon erilaisia ominaisuuksia käyttöliittymän komponenteille. Liittyen hintaan, sovelluskehys koettiin halpana. Vastaajien mukaan kehyksen käyttäminen projektissa on halpaa.

Vastaajien mukaan *syyt Foundationin käyttöön* liittyivät ominaispiirteisiin, tunnepohjaisiin kokemuksiin ja projektin lähtökohtiin. Ominaispiirteet koskivat helppoutta. Eräs haastateltava mainitsi, että valmis käyttöliittymä on helpompi lähtökohta, jos tehdään mainostoimiston tai UX/UI-tiimin suunnitelmaa. Tunnepohjaiset kokemukset liittyivät kokemukseen. Foundation valittiin käytettäväksi eri henkilöiden kokemuksien myötä. Lopuksi projektin lähtökohdat koskivat asiakkaan päätöstä. Osa asiakastapauksista olivat sanelleet mitä toivovat käyttävän, jolloin Foundation on otettu käyttöön.

Materiaalista kävi ilmi, että on monia syitä *miksi yrityksissä ei käytetä sovelluskehyksiä tai tehdään mahdollisesti omia ratkaisuja*. Vastauksista ilmeni ominaispiirteisiin ja hyödyttömyyteen liittyviä asioita. Ominaispiirteet liittyivät tehottomuuteen, sovelluskehysten saneluun ja ulkoasusta huomaamiseen. Liittyen tehottomuuteen, vastaajien mukaan kehysten mukana tulee paljon ylimääräistä tavaraa mukaan, joista kaikkea ei tule käytettyä. Kehyksien mukana tulevan suuren koodimäärän vuoksi toteutus toimii ja latautuu hitaasti.

> *”Mä en tue niitä. Mä oon käyttäny joskus, mä oon käyttäny Foundationia aikanaan, puhutaan kaks kolme vuotta sitten, mutta nykyään en käytä, koska 90 prosenttia kamasta on turhaa mitä sieltä tulee.”*

> *”…mä näkisin et suurin ongelma näissä useissa suosituissa freimiksissä, otetaan nyt vaikka Bootstrappi ja Angular ja Reactki… niissä on aika suuri cloudbase. Elikkä jos sä teet simppelinki fronttiprojektin, sul tulee helposti satoja kiloja tai jopa megaa koodia. Pari megaakaan ei oo mikään harvinaisuus… et se on musta tosi ikävä tämmönen kehityssuunta… - Ja sit se näkyy lähinnä siinä, että toimii hitaasti, latautuu hitaasti, varsinkin mobiilissa.”*

Sovelluskehysten saneluun liittyen vastaajat olivat sitä mieltä, että CSS-sovelluskehyksiä käyttäessä ohjelmoija pysyy tiukasti kiinni niiden metodologioissa. Sovelluskehykset ovat haastateltavien mukaan ongelmallisia, koska ne istuvat harvoin tekemisen kanssa yhteen. Liittyen ulkoasusta huomaamiseen, erään haastateltavan mukaan on helposti havaittavissa, jos on käytetty valmista sovelluskehystä.

Hyödyttömyys liittyi merkityksen menettämiseen, riskin ottamiseen ja siihen, että ei opi CSS:ää ja että on helpompi tehdä itse. Liittyen merkityksen menettämiseen, vastaajien mukaan ne ovat hävittäneet merkityksensä Flex Boxin ja komponenttipohjaisten työkalujen aikakaudella. Kehitys ja tyylitys on enenevissä määrin siirtynyt komponenttikirjastoilla ja JavaScript-työkaluilla tehtäviksi. Esimerkiksi Reactilla on UI-kirjastoja, joista saa tyylejä. Liittyen riskin ottamiseen, erään vastaajan mukaan riskinä on kehysten päivittäminen, joka vie paljon aikaa ja rahaa.

Liittyen siihen, että ei opi CSS:ää, erään haastateltavan mukaan sovelluskehyksien käyttö on helppo oikotie onneen. Käyttäessä kehyksiä ohjelmoija ei välttämättä opi kirjoittamaan perinteistä CSS:ää. Lopuksi liittyen siihen, että on helpompi tehdä itse, vastaajien mukaan on helpompi tehdä itse kuin opetella sovelluskehys ulkoa. Tyylejä ja komponentteja on helpompi tehdä itse kuin kustomoida valmiita sovelluskehyksiä halutun ulkoasun mukaiseksi. Kaiken lisäksi, itsetekemällä saa juuri tarvittavan toiminnallisuuden.

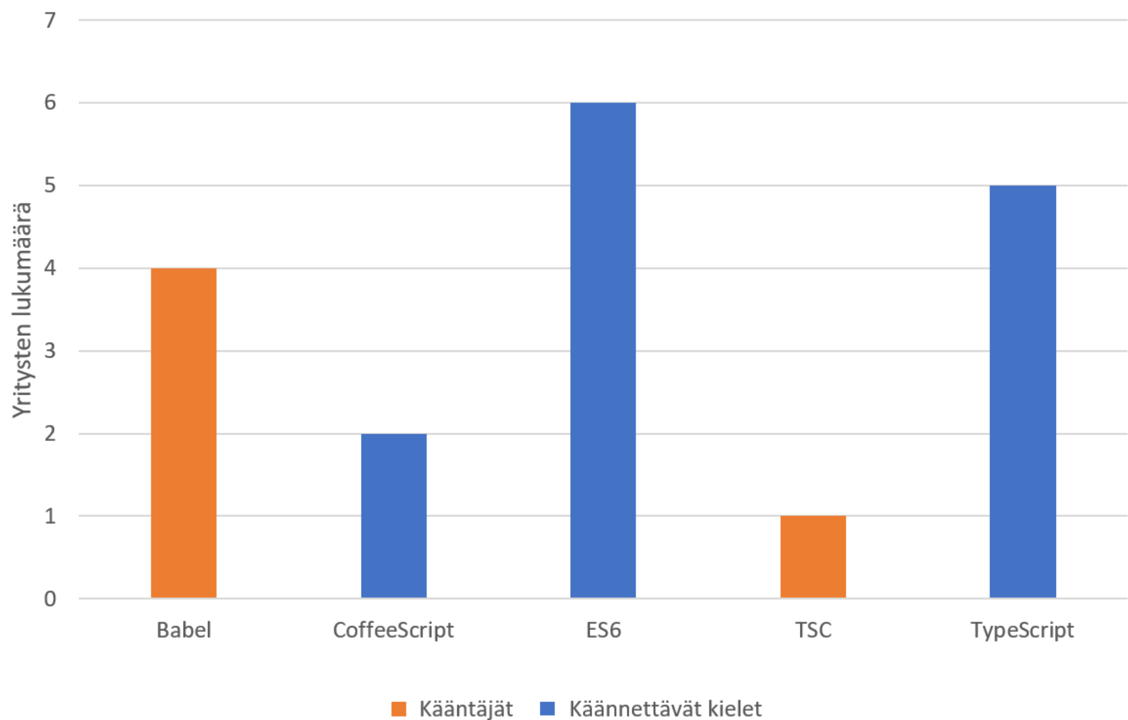
> *”Ja käytännön syyt on ihan sellaisia että… se on usein melkein nopeampaa tehdä ite noita… tyylejä tai komponentteja… kuin alkaa kustomoimaan noita valmiita komponentteja tai tyyliframeworkkejä leiskan mukaiseksi. Ja sit jos sä teet ite, ni sä saat just sen toiminnallisuuden mitä tarvitaan. Ei tuu sellaista ylimääräistä taakkaa mukaan.”*

> *”…ne on helpompi tehdä itse, niin se on ehkä se. Ja mun mielest kehitys on menny enemmän komponenttikirjastoihin, että netistähän löytyy nykyään esimerkiks Reactille, ni tollasi UI-kirjastoja, mistä sä saat nappeja ja tollasia, ni niit on.”*

Lopuksi *syyt Material Designin käyttöön* koskivat projektin lähtökohtia, tunnepohjaisia kokemuksia ja aikaa. Projektin lähtökohdat koskivat asiakkaan päätöstä. Haastateltavan mukaan osa asiakastapauksista sanelee mitä toivovat projektissa käytettävän. Tunnepohjaiset kokemukset koskivat kokemusta. Valinta liittyi eri henkilöiden kokemuksiin siitä, että työkalu on hyvä. Lopuksi aika liittyi tulevaisuuteen. Eräs haastateltava oli sitä mieltä, että maailma on menossa enemmän Material Designin -suuntaan.

5.1.4 Kääntäjät ja käännettävät kielet

Vastauksista ilmeni, että yrityksissä käytetään JavaScriptiksi kääntyviä ohjelmointikieliä ja kääntäjiä. Eniten oli käytössä ES6:sta ja TypeScriptiä sekä Babel-kääntäjää (ks. kuvio 9). Eräs haastateltavista mainitsi, että TypeScriptin kääntämiseen käytetään TypeScriptille tarkoitettua kääntäjää nimeltä TypeScript compiler (TSC).



Kuvio 9. Ohjelmistoyrityksissä käytössä olevat kääntäjät ja JavaScriptiksi käännettävät kielet (n = 8)

TypeScriptin käyttö koski suositusta, ominaispiirteitä ja tunnepohjaisia kokemuksia. Liittyen suositukseen, eräässä yrityksessä käytetään TypeScriptiä, koska sitä suositellaan Angularin kanssa käytettäväksi. Vastaajan mukaan tutoriaalit liittyen Angulariin on kirjoitettu TypeScriptillä, jonka vuoksi kieli on ollut luontevaa ottaa käyttöön. Ominaispiirteet koskivat virheiden vähyyttä, helpottamista ja luettavuutta. Koskien virheiden vähyyttä, yrityksissä käytettiin TypeScriptiä, jotta virheitä saataisiin vähennettyä. TypeScriptin tyypityksen vuoksi samanlaisia virheilmoituksia kuten JavaScriptissä ei tule. Haastateltavien mukaan vääränlainen data jää kiinni jo kompilointivaiheessa. Liittyen helpottamiseen, vastaajien mukaan TypeScript laajentaa ja helpottaa JavaScriptin kirjoittamista. Erään haastateltavan mukaan ohjelmointikieli helpottaa erityisesti laadullisia virheitä. Luettavuuteen liittyen kieli nähtiin myös helposti luettavana apufunktioidensa johdosta. Lopuksi tunnepohjaisiin kokemuksiin liittyen, erään vastaajan mukaan kielen valinta johtui siitä, että ohjelmoijat tykkäävät tyypitetyistä kielistä.

> *”…toiset ihmiset tykkää tyyppipohjaisesta tai tyypitetyistä kielistä enemmän kuin täysin ei-tyypitetyistä, niin on sitten valittu sen takia, että on saatu JavaScript-maailmaan tyypitystä mukaan. Et toinen vaihtoehto olis ollu käyttää Flowta, mutta se ei oo ihan yhtä vahva ton tyypityksen kanssa kuin TypeScripti, ni siks on valittu.”*

TypeScriptin hyvät puolet koskivat ominaispiirteitä ja tunnepohjaisia kokemuksia. Liittyen ominaispiirteisiin, kielen hyvät puolet koskivat helpottamista. Erään vastaajan mukaan tyypityksen lisääminen vähentää testauksen tarvetta. Tunnepohjaiset kokemukset liittyvät kokemuksiin. Haastateltavien mukaan TypeScript on selkeämpi kuin JavaScript, ja sitä pidettiin myös parempana versiona JavaScriptistä. Erään vastaajan mukaan kieli on erittäin hyvä verrattuna ES6:seen.

Kielen huonot puolet liittyivät ominaispiirteisiin. Erään vastaajan mukaan TypeScriptissä on koodin rakenteeseen liittyviä asioita, jotka aiheuttavat päänvaivaa. Haastateltavan mukaan tietyissä asioissa tyypittämällä ei pysty tekemään asioista riittävän geneerisiä. Vastaaja myös lisäsi, että tyypityksen johdosta joissain tapauksissa joutuu tekemään saman asian useamman kerran.

Taulukossa 20 on listattuna TypeScriptin hyviä ja huonoja puolia. Listauksen mukaan työkalu koettiin paremmaksi JavaScriptiin ja sen eri versioihin verrattuna.

Taulukko 20. TypeScriptin hyvät ja huonot puolet

Hyvät puolet	**Huonot puolet**
• Tyypityksen lisääminen vähentää testauksen tarvetta • Selkeämpi kuin JavaScript • Parempi versio JavaScriptistä • Erittäin hyvä ES6:seen verrattuna	• Tyypittämällä ei pysty aina tekemään asioista riittävän geneerisiä • Joskus saman asian joutuu tekemään useasti johtuen tyypityksestä

Syyt ES6:en käyttöön liittyivät projektin lähtökohtiin, ominaispiirteisiin ja uutuuteen. Projektin lähtökohtiin liittyen asiakas oli määritellyt ES6:n käytettäväksi teknologiaksi. Ominaispiirteet liittyivät ominaisuuksiin, jotka koskivat ES6:en apufunktioita ja muita uusia ominaisuuksia. Uutuus liittyi kielen uutuuteen. Vastaajien mukaan ES6 on tuorein vaihtoehto, jolla voi ohjelmoida.

> *”No se on se tuorein vaihtoehto millä voi tehä… ja… nii siinäpä se on. Et siinä on niitä uusia ominaisuuksia mitä voi käyttää niinni, miks kirjoittaa vanhoilla ku voi kirjoittaa uudella.”*

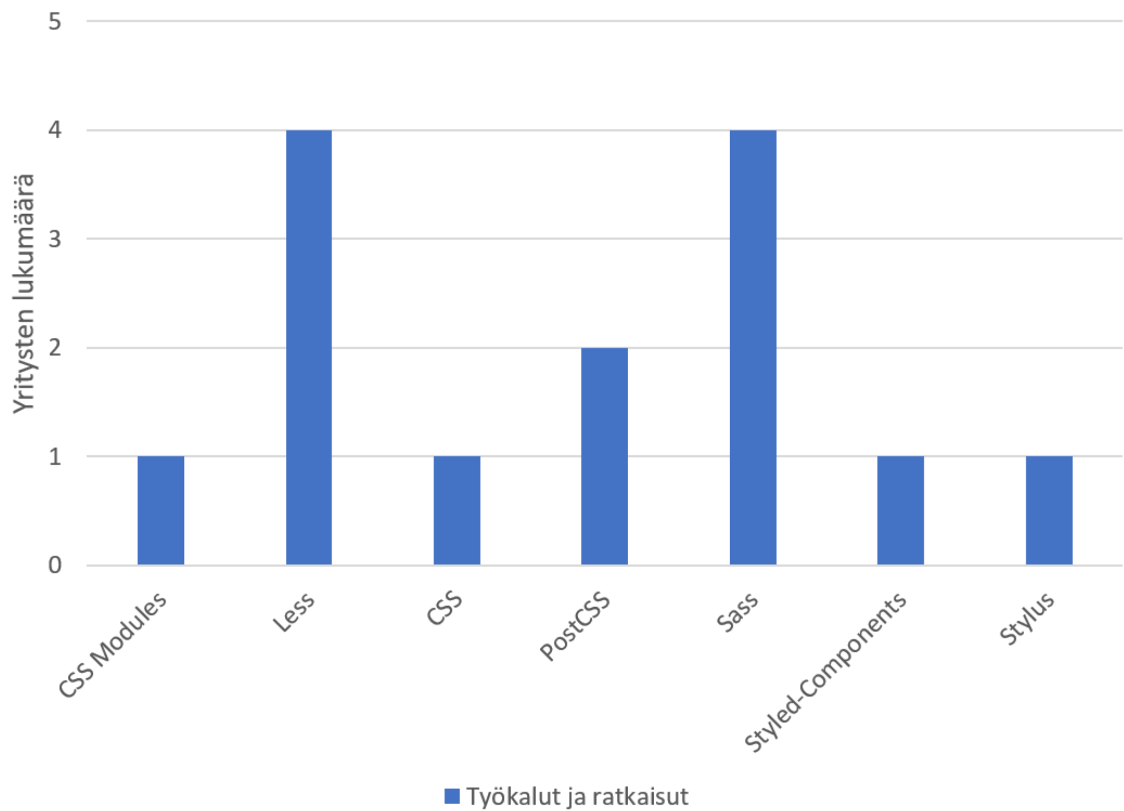
Yrityksissä käytetään Babelia ES6:en kääntämiseen. *Syyt Babelin käyttöön* liittyivät projektin lähtökohtiin ja tunnepohjaisiin kokemuksiin. Koskien projektin lähtökohtia, haastateltavien mukaan asiakas oli määritellyt Babelin käytettäväksi. Liittyen tunnepohjaisiin kokemuksiin, vastaajien mukaan muita kääntäjiä ei ole nykyään, ja että ne ovat kaikki hävinneet. Eräs haastateltavista mainitsi Babelin olevan ’de facto’, eli ainut oikea.

Vastaajien mukaan *ES6:en hyvät puolet* liittyivät ominaispiirteisiin, jotka koskivat ominaisuuksia ja helppokäyttöisyyttä. Ominaisuuksiin liittyen haastateltavien mukaan ES6:ssa modulaarisuus on hyvä ominaisuus. Lisäksi muuttujien määrittely nähtiin hyvänä lisänä. Helppokäyttöisyyteen liittyen, kielessä on vastaajien mukaan koodaamista helpottavia ominaisuuksia. Eräs vastaaja lisäsi, että kieli yksinkertaistaa ja helpottaa asioita. Myös *ES6:n huono puoli* koski ominaispiirrettä, joka liittyi ominaisuuksien turhuuteen. Eräs vastaajista näki, että monet ES6:en ominaisuudet ovat turhia, kuten esimerkiksi luokat.

CoffeeScriptin käyttö yrityksissä liittyi verkostoon ja tunnepohjaisiin kokemuksiin. Verkosto liittyi suosioon. Eräässä yrityksessä kieli otettiin käyttöön laajasti noin viisi vuotta sitten, kun oli CoffeeScriptin aallonhuippu. Tunnepohjaiset kokemukset liittyivät mieltymykseen. Yrityksessä käytetään CoffeeScriptiä osittain sen takia, koska siellä on paljon Python-ohjelmoijia. Vastaajan mukaan Python-ohjelmoijalle on helppo myydä CoffeeScript, koska kielillä on samantyyppinen syntaksi.

5.1.5 CSS:n tyylittely

CSS:n tyylittelyyn yrityksissä käytetään CSS-esikäsittelijöitä ja muita tyylittelyyn liittyviä työkaluja. Yrityksissä oli eniten käytössä Sass- ja Less-esikäsittelijöitä (ks. kuvio 10). Yhdessä yrityksessä ei käytetä lainkaan CSS:n tyylittelyyn liittyviä työkaluja.



Kuvio 10. Ohjelmistoyrityksissä käytössä olevat työkalut ja ratkaisut CSS:n tyylittelyyn (n = 8)

Haastateltavien vastauksien mukaan *syyt Sassin käyttöön* koskivat ominaispiirteitä, tunnepohjaisia kokemuksia ja projektin lähtökohtia. Ominaispiirteet koskivat monipuolisuutta ja ominaisuuksia. Liittyen monipuolisuuteen, vastaajien mukaan Sassissa pystyy käyttämään muuttujia, kuten esimerkiksi vaihtamaan värejä, jotka muuttuvat läpi sovelluksen. Sassia käyttäessä koodi on yleisesti ylläpidettävämpää ja rakenteellista. Liittyen ominaisuuksiin, erään vastaajan mukaan työkaluun siirryttiin Lessin jälkeen, sillä Sass on funktionaalisempi verrattuna Lessiin. Tunnepohjaiset kokemukset liittyivät mieltymykseen. Vastaajien mukaan riippuu tekijästä, mitä esikäsittelijää on miellyttävä käyttää. Lopuksi projektin lähtökohtia koskien, Sass oli sekä asiakkaan että kolmannen osapuolen valitsema työkalu.

Sassin hyvä puoli koski ominaispiirteitä, jotka liittyivät toiminnallisuuksiin. Sassin Mixinit ovat erään vastaajan mukaan käteviä.

Syyt Lessin käyttöön liittyivät tunnepohjaisiin kokemuksiin ja projektin lähtökohtiin. Tunnepohjaisiin kokemuksiin liittyen, vastaajien mukaan riippuu tekijästä, mitä esikäsittelijää ohjelmoija haluaa käyttää. Liittyen projektin lähtökohtiin, erään vastaajan mukaan Lessin valinta tuli kolmannelta osapuolelta, jonka lisäksi se oli asiakkaan valitsema.

Haastateltavien mukaan *Lessin hyvät puolet* koskivat ominaispiirteitä, jotka liittyivät toiminnallisuuksiin, joustavuuteen, monipuolisuuteen ja ylläpidettävyyteen. Toiminnallisuudet koskivat muuttujien ja matemaattisten funktioiden määrittämistä. Liittyen joustavuuteen, Less on perinteiseen CSS:ään verrattuna dynaamisempi. Koskien monipuolisuutta, erään vastaajan mukaan Less on tuonut paljon laajennusta perinteiseen CSS:ään. Ylläpidettävyyteen liittyen Less on helpommin ylläpidettävä kuin tavallinen CSS.

Styluksen käyttöön liittyvät syyt koskivat ominaispiirteitä ja projektin lähtökohtia. Ominaispiirteisiin liittyen, käyttö koski samankaltaisuutta. Vastaajan mukaan Styluksella on samantyyppinen syntaksi kuin Pythonilla, jonka vuoksi sitä on helppo suositella Python-ohjelmoijille. Styluksen käyttö koskien projektin lähtökohtia liittyi siihen, että se oli asiakkaan valitsema. Haastateltavan mukaan työkalu on projektista riippuvainen sillä tavalla, että asiakas voi valita mitä tahansa työkaluja käytettäväksi. Lisäksi *Styluksen huono puoli* koski verkostoa, joka liittyi tukeen. Vastaajan mukaan Stylukselle löytyy heikoimmin tukea CSS-esikäsittelijöistä.

Syyt PostCSS:n, CSS Modulesin ja Styled-Componentsin käyttöön liittyivät ominaispiirteisiin ja aikaan. Ominaispiirteet liittyivät tehokkuuteen, sillä kyseenomaiset työkalut olivat säästäneet aikaa. Aikaan liittyvät syyt koskivat trendikkyyttä. Erään haastateltavan mukaan kyseiset työkalut ovat jo pari vuotta kasvattaneet suosiota markkinoilla. Vastaajan mukaan työkalut ovat parempia kuin Sass ja Less. Lisäksi komponenttipohjaisen ajan myötä tarve CSS:n esiprosessoinnille on vastaajan mukaan vähentynyt.

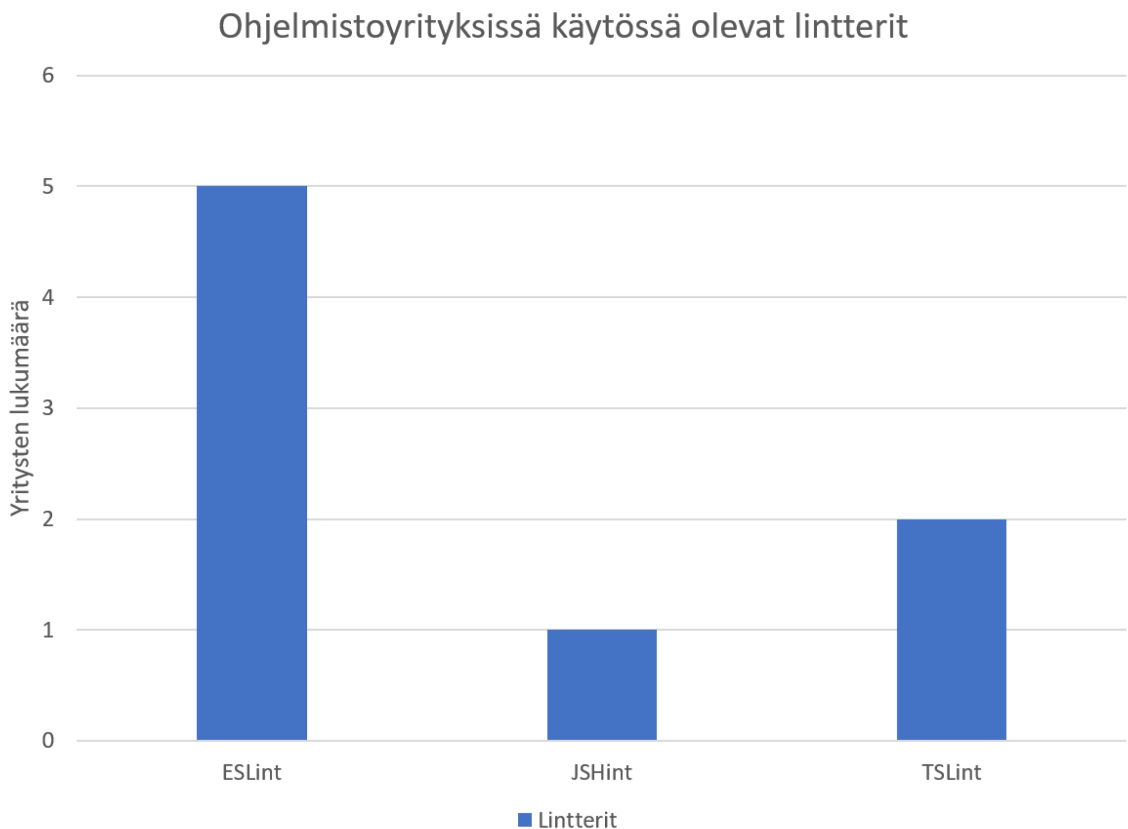
> *"Joo se on ollut nyt pari vuotta, pari vuotta syöny ton markkinan, et ei kukaan enää käytä Lessiä ja Sassia… Mut mun mielest se ylipäätään komponenttipohjaisen ajan myötä tarve CSS:nki esiprosessoinnille on vähentyny. Et se mikä mun mielestä on tärkeetä siin esiprosessoinnissa nykyään tai prosessoinnissa ylipäätään on se, että esimerkiks auto prefixaukset, kaikki selain-vendorit, sun ei tartte muistaa niitä, et sen kun vaan kirjoitat CSS:ää ja sanot et mitä selaimii sä haluut tukea ja joku tekee ne sulle. Ton tyyppiset ominaisuudet tietysti on säästänyt tosi paljon aikaa, koska eihän kukaan haluu opetella ulkoa jotain mitä jokin selain tukee ja mitä se ei tue."*

Styled-Componentsin hyvä puoli koski riittoisuutta. Haastateltava näki työkalun riittoisana, sillä se vie pois tarvetta CSS:n käytöstä ylipäätään. Lisäksi eräs haastateltava mainitsi, että jos he aloittaisivat uuden projektin puhtaalta pöydältä, he valitsisivat Styled-Componentsin.

Pelkkään CSS:n käyttöön liittyvät syyt koskivat aikaa ja tarpeettomuutta. Aikaan liittyi ajan puute. Vastaaja arveli, ettei ole ollut aikaa perehtyä esikäsittelijöihin. Tarpeettomuuteen liittyen, vastaajan mukaan he pärjäävät tavallisella CSS:llä.

5.1.6 Koodin tarkistaminen

Tutkimuksesta ilmeni, että yrityksissä käytettiin reilusti eniten ESLint-työkalua koodin tarkistamiseen (ks. kuvio 11).



Kuvio 11. Ohjelmistoyrityksissä käytössä olevat lintterit (n = 8)

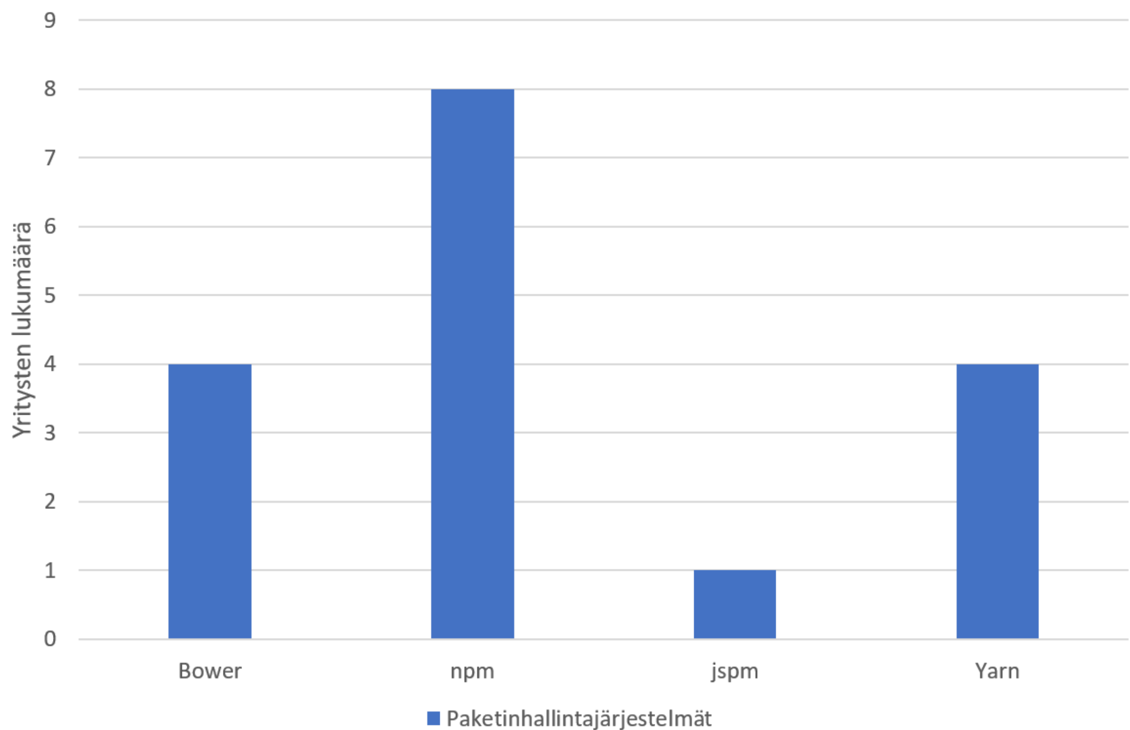
ESLintin käytön syyt liittyivät ominaispiirteisiin, verkostoon ja optimointiin. Ominaispiirteisiin liittyen ESLint koettiin toimivaksi ja helppokäyttöiseksi. Liittyen toimivuuteen, erään vastaajan mukaan ESLint toimii hyvin Sublime Text -editorin kanssa. Helppokäyttöisyyteen liittyen työkalu koettiin helppokäyttöiseksi, sekä esimerkiksi asetusten määrittäminen nähtiin helppona. Liittyen verkostoon, työkalun käyttöön liittyi tunnettavuus. Vastaajien mukaan sitä käytetään, koska se on kaikille jollain asteella tuttu. Optimointiin liittyen ESLint on tullut yrityksen käyttöön tutkimustyöllä. Näiden lisäksi erään vastaajan mukaan työkalun valinnalle ei ollut mitään isompaa syytä.

Vastaajien mukaan *syyt TSLintin käyttöön* liittyivät verkostoon, ominaispiirteisiin ja tiimityöskentelyyn. Verkostoon liittyen, työkalua käytetään suosion takia. Erään vastaajan mukaan Atomissa TSLint on suosittu työkalu TypeScriptille. Ominaispiirteet liittyivät toimivuuteen ja yhteensopivuuteen. Toimivuuteen liittyen, TSLint toimii muun muassa Visual Studio Coden kanssa hyvin. Liittyen yhteensopivuuteen, TSLint on käytössä TypeScriptin takia. Tiimityöskentely taas liittyi yhtenäisyyteen. Erään vastaajan mukaan TSLintin avulla kaikki ohjelmoijat kirjoittavat samankaltaista koodia.

Haastateltava, joka mainitsi, että heidän yrityksessään käytetään JSHint-lintteriä, ei osannut sanoa tarkempaa *syytä JSHintin käyttöön*. Vastaajan mukaan työkalun valinta riippuu koko työkalupakin valinnasta, eli siitä mikä luonnostaan sopii parhaiten muiden työkalujen joukkoon.

5.1.7 Paketinhallinta

Haastateltavien vastauksista ilmeni, että jokaisessa yrityksessä käytetään npm-paketinhallintajärjestelmää (ks. kuvio 12). Boweria ja Yarnia käytetään yhtä paljon. Vain yhdessä yrityksessä käytetään jspm-paketinhallintajärjestelmää. Kappaleessa on jätetty tarkastelun ulkopuolelle syyt jspm:n käyttöön johtuen siitä, että haastateltava ei ottanut kantaa asiaan.



Kuvio 12. Ohjelmistoyrityksissä käytössä olevat paketinhallintajärjestelmät (n = 8)

Haastateltavien vastausten perusteella *syyt npm:n käyttöön* liittyvät ominaispiirteisiin, projektin lähtökohtiin ja aikaan. Ominaispiirteet koskivat kattavuutta ja yksinkertaisuutta. Liittyen kattavuuteen, vastaajat kokivat npm:n laajana kirjastona. Npm:stä löytyy kaikki tarvittavat paketit, joita voi ikinä tarvita - valmiina paketteina. Liittyen yksinkertaisuuteen, vastaajien mukaan pakettien asentaminen on todella yksinkertaista. Projektin lähtökohtiin liittyen, erään vastaajan mukaan projektissa käytetään npm-paketinhallintajärjestelmää silloin, kun projektissa on Webpack ja Browserify käytössä. Aika liittyi nykyaikaisuuteen. Vastaajan mukaan paketinhallinnan käyttö on nykyaikaista.

Haastateltavien mukaan *npm:n hyvät puolet* liittyivät ominaispiirteisiin, jotka koskivat tehokkuutta, toimivuutta, helppokäyttöisyyttä, yhteensopivuutta ja monipuolisuutta. Tehokkuuteen liittyen työkalu on nopea ja kevyt. Liittyen toimivuuteen, npm on pätevä ja toimiva, ja sen nähtiin toimivan tosi hyvin kaiken kanssa. Eräs haastateltava koki, että työkalu on toimiva sekä front end että back end -koodissa. Helppokäyttöisyyteen liittyen käyttäjät kokivat, että työkalu on selkeä ja helppokäyttöinen. Erään vastaajan mukaan päivittäminen on npm:ssä helppoa. Liittyen yhteensopivuuteen erään vastaajan mukaan npm:n paketit ovat CommonJS:n ja ES6:en kanssa yhteensopivia. Monipuolisuutta koskien vastaajien mukaan npm:ssä on todella laaja kirjo käytettäviä työkaluja. Erään vastaajan mukaan

npm:ssä on paljon työkaluja, jotka on tehty Node.js:n päälle ja joita voi käyttää selaimessa sitä kautta. Eräs vastaaja luetteli työkalun hyväksi puoleksi sen, että sitä voi käyttää myös tehtävänsuorittajana.

Paketinhallinnan huonot puolet liittyivät ominaispiirteisiin, jotka koskivat liiallista laajuutta, tehottomuutta ja alustariippumattomuutta. Liialliseen laajuuteen liittyen, npm on liiankin laaja ja sisältää turhia kirjastoja. Liittyen tehottomuuteen, paketinhallinta toimii hitaasti suuren sisällön vuoksi eikä asenna tarpeeksi nopeasti paketteja. Haastateltavat kokivat myös riippuvuuksien hallinnan olleen raastavaa. Työkalun käytön kanssa on ollut ongelmia, sillä jossain on ollut eri riippuvuuksien versio. Lopuksi liittyen alustariippumattomuuteen, erään käyttäjän mukaan npm:n käyttö tehtävänsuorituksessa ei ole alustariippumatonta.

Taulukossa 21 on tiivistettynä npm:n hyvät ja huonot puolet. Npm:llä on haastateltavien vastausten perusteella enemmän hyviä kuin huonoja puolia.

Taulukko 21. Npm-paketinhallintajärjestelmän hyviä ja huonoja puolia

Hyvät puolet	**Huonot puolet**
• Nopea • Kevyt • Toimiva • Pätevä • Toimi hyvin kaiken kanssa • Toimii sekä front end että back end koodissa • Päivittäminen helppoa • Helppokäyttöinen • Selkeä • npm-skriptejä voi käyttää tehtävän-suoritukseen • Laaja kirjasto • Sisältää paljon Noden päälle tehtyjä työkaluja • Paketit ovat CommonJS:n ja ES6:en kanssa yhteensopivia	• Liiankin laaja • Sisältää turhia kirjastoja • Hidas • Ongelmia riippuvuuksienhallinnassa • Käyttö tehtävänsuorituksessa ei ole alustariippumatonta

Syyt Yarnin käyttöön koskivat ominaispiirteitä, aikaa ja optimoimista. Ominaispiirteet liittyivät tehokkuuteen ja toiminnallisuuksiin. Tehokkuuteen liittyen Yarn on vastaajien mukaan huomattavasti nopeampi ja suorituskykyisempi verrattuna npm:ään. Erään haastateltavan mukaan Yarn osaa paremmin tallentaa muistiin, eli ”cachettaa” verrattuna npm:ään. Toiminnallisuuksiin liittyen Yarnia käytetään sen takia, että se lukitsee käytettyjen pakettien versiot. Erään haastateltavan mukaan Yarnin käyttöönottoaikaan npm:ssä ei ollut versioiden lukitsemisominaisuutta.

Aikaa koski nykyaikaisuus. Haastateltavat kokivat Yarnin modernimmaksi verrattuna npm:ään. Erään vastaajan mukaan Yarnilla on voinut tehdä joitakin asioita paremmin, koska se on uudempi. Toisen haastateltavan mukaan Yarn syrjäytti npm:n kahdessa päivässä. Lopuksi optimoiminen liittyi rinnakkain käyttöön. Yarnia käytetään rinnakkain npm:n kanssa npm-pakettien asentamisen hallintaan. Lisäksi Yarnilla käytetään yleensä samaa npm:n ekosysteemiä, koska vaihtoehtoja on vähän.

Yarnia käyttäneillä haastateltavilla oli myös *muita mietteitä koskien paketinhallintaa.* Haastateltavien ajatukset liittyivät tietoon, joka koski arvelemista ja tiedottomuutta. Liittyen arvelemiseen, vastaajien mukaan uusimmassa npm:n versiossa pitäisi toimia myös käytettyjen pakettien versioiden lukitus. Tiedottomuuteen liittyen, vastaajat eivät kuitenkaan olleet kokeilleet npm:n uusinta versiota, joka oli tullut hiljattain. He eivät myöskään tienneet, kumpi Yarnista ja npm:stä on nykyään nopeampi.

Haastateltavien mukaan *Yarnin hyvät puolet* liittyivät optimoimiseen ja ominaispiirteisiin. Liittyen optimoimiseen, Yarn nähtiin täydentävänä, sillä työkalu paikkasi npm:n puutteita. Ominaispiirteet liittyivät toiminnallisuuteen. Käyttäjät pitivät siitä, että Yarn lukitsee pakettien versiot oletuksena. Vastaavasti *Yarnin huono puoli* liittyi ominaispiirteeseen, joka koski yhteensopimattomuutta. Erään vastaajan mukaan Yarnissa on lievää yhteensopimattomuutta npm:n kanssa, koska Yarn on uusi työkalu. Yarnin hyvät ja huonot puolet on listattuna taulukossa 22.

Taulukko 22. Yarnin hyvät ja huonot puolet

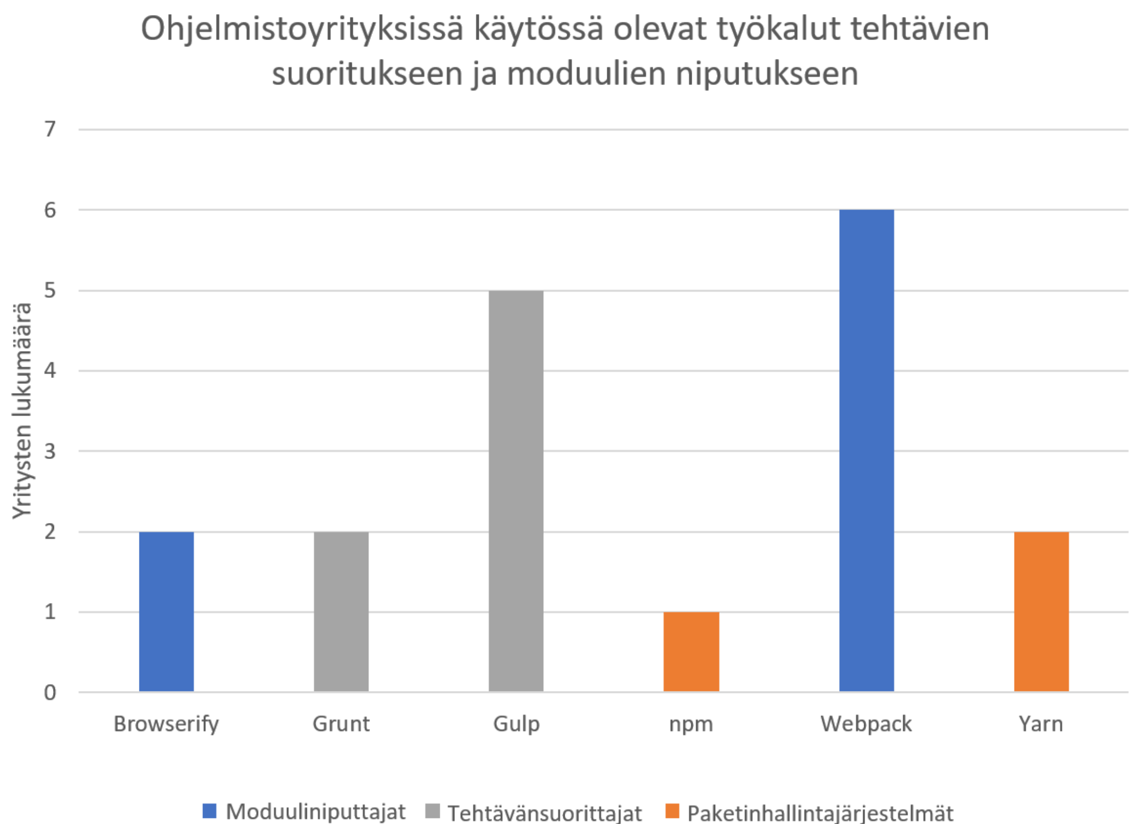
Hyvät puolet	Huonot puolet
• Paikannut npm:n puutteita • Lukitsee pakettien versiot oletuksena	• Lievää yhteensopimattomuutta npm:n kanssa

Syyt Bowerin käyttöön koskivat projektin lähtökohtia ja aikaa. Projektin lähtökohdat koskivat riippuvuutta projektista. Eräässä yrityksessä Bower on käytössä markkinointisivuilla, joihin ei tarvita paketoijaa (engl. module bundler). Toisen vastaajan mukaan Bower on käytössä vanhemmissa projekteissa. Liittyen aikaan, erään vastaajan mukaan paketinhallinnan käyttö on ylipäänsä nykyaikaista. *Bowerin hyvä puoli* liittyi ominaispiirteeseen, jossa työkalu nähtiin tehokkaana. Erään vastaajan mukaan Bower latautuu nopeasti ja tehokkaasti.

Lopuksi liittyen jspm-paketinhallintaan, erään haastateltavan mukaan on odotettu, että HTTP/2:en yleistyttyä jspm:stä tulisi käytetympi työkalu. Vastaaja ei kuitenkaan nähnyt sen olevan vielä ajankohtaista.

5.1.8 Tehtävänsuoritus ja moduulien niputus

Tehtävien suoritukseen yrityksissä käytetään tehtävien automatisointiin tarkoitettuja työkaluja ja paketinhallintajärjestelmiä. Suurin osa haastattelijoista mainitsi myös moduulien niputtajat tehtävänsuorituksen yhteydessä, jonka vuoksi ne käsiteltiin yhdessä. Vastauksista ilmeni, että yrityksissä käytetään eniten Webpackia ja Gulppia (ks. kuvio 13).



Kuvio 13. Ohjelmistoyrityksissä käytössä olevat työkalut tehtävien suoritukseen ja moduulien niputukseen (n = 8)

Yrityksissä käytetään tehtävänsuorittajia ja moduulien niputtajia rinnakkain ja vaihtelevasti. Eräässä yrityksessä Yarnia käytetään tehtävien suoritukseen ja Webpackia pyörittämään projektien kehitysympäristöä. Toisessa yrityksessä haastateltava pyrki suosimaan npm-skriptejä ja Browserifyta. Kolmannessa yrityksessä käytetään Gulppia tehtävien suoritukseen ja paketoimiseen Webpackia.

Haastateltavien mukaan *syyt Webpackin käyttöön* liittyivät projektin lähtökohtiin ja ominaispiirteisiin. Projektin lähtökohtiin liittyen syy työkalun käyttöön johtui siitä, että asiakas oli valinnut sen käytettäväksi. Koskien ominaispiirteitä syy työkalun käyttöön liittyi toimivuuteen. Erään vastaajan mukaan Webpackilla saa semmoista koodia mikä toimii kaikilla selaimilla.

Webpackin hyvät puolet koskivat ominaispiirteitä, johon liittyivät ajatukset toimivuudesta ja ominaisuuksista. Liittyen toimivuuteen, haastateltavat olivat sitä mieltä, että Webpack on toiminut hyvin. Ominaisuuksiin liittyen, työkalun Hot Reloading -ominaisuus koettiin erittäin hyvänä. Myös *työkalun huono puoli* koski ominaispiirteitä, joka liittyi monimutkaisuuteen. Eräs haastateltava oli sitä mieltä, että työkalu voi monimutkaistua vaativammassa käytössä.

Browserifyn käyttö koski projektin lähtökohtia, jotka liittyivät tarpeellisuuteen. Työkalua käytettiin moduulien niputukseen. Eräs haastateltava lisäsi, että Browserify ja Webpack ovat käytännössä samanlaisia, mutta Webpackille löytyy paremmin Hot Loadereita.

Syyt Gulpin käyttöön liittyivät tunnepohjaisiin kokemuksiin, ominaispiirteisiin, verkostoon ja projektin lähtökohtiin. Tunnepohjaiset kokemukset liittyivät viehättävyyteen ja kokemukseen. Viehättävyyteen liittyen erään vastaajan mukaan Gulpin syntaksi on nätimpää verrattuna Grunttiin. Kokemuksiin liittyen, monet ihmiset olivat kehuneet ja suositelleet sitä. Ominaispiirteet koskivat toimivuutta. Vastaajien mukaan Gulp on ollut hyvä työkalu, ja sillä pystyy putkittamaan paremmin kuin Gruntilla. Lisäksi eräs vastaaja mainitsi käytön syyksi sen, että työkalulla koodi toimii kaikilla selaimilla. Verkostoa koskien, vastaajien mukaan Gulppiin löytyy enemmän osaamista. Lopuksi projektin lähtökohtiin liittyen, syy Gulpin käyttöön johtui siitä, että se oli asiakkaan valitsema.

Gulpin hyvät puolet liittyivät ominaispiirteisiin, jotka koskivat toimivuutta, ominaisuuksia ja monipuolisuutta. Liittyen toimivuuteen, haastateltavien mukaan työkalu on toiminut hyvin. Ominaisuuksiin liittyen, erään haastateltavan mukaan Hot Reload -ominaisuus on erittäin hyvä lisä. Lisäksi työkalu on alustariippumaton. Liittyen monipuolisuuteen, Gulpilla pystyy tekemään paljon asioita, kuten muun muassa kustomoimaan pelkästään koodia kirjoittamalla. Lisäksi Gulppiin löytyy valtava määrä lisäosia.

Gulpin huonot puolet koskivat ominaispiirteitä: työkalu koettiin monimutkaiseksi ja tehottomaksi. Monimutkaisuuteen liittyen, erään vastaajan mukaan työkalu voi monimutkaistua vaativammassa käytössä. Tehottomuuteen liittyen, Gulp koettiin raskaaksi systeemiksi.

Syyt Gruntin käyttöön liittyivät projektin lähtökohtiin. Erään haastateltavan mukaan Grunt otettiin käyttöön asiakkaan vaatimuksesta. Sen sijaan *Gruntin hyvät puolet* liittyivät ominaispiirteisiin, jotka koskivat monipuolisuutta, toimivuutta ja ominaisuuksia. Liittyen monipuolisuuteen, erään vastaajan mukaan Grunt on monipuolinen, sillä sille löytyy paljon lisäosia ja sitä pystyy kustomoimaan koodia kirjoittamalla. Toimivuuteen liittyen, työkalu on toiminut hyvin. Ominaisuuksiin liittyen, Grunt on alustariippumaton. Myös *Gruntin huono puoli* liittyi ominaispiirteisiin: Grunt koettiin tehottomaksi, sillä se on raskas järjestelmä.

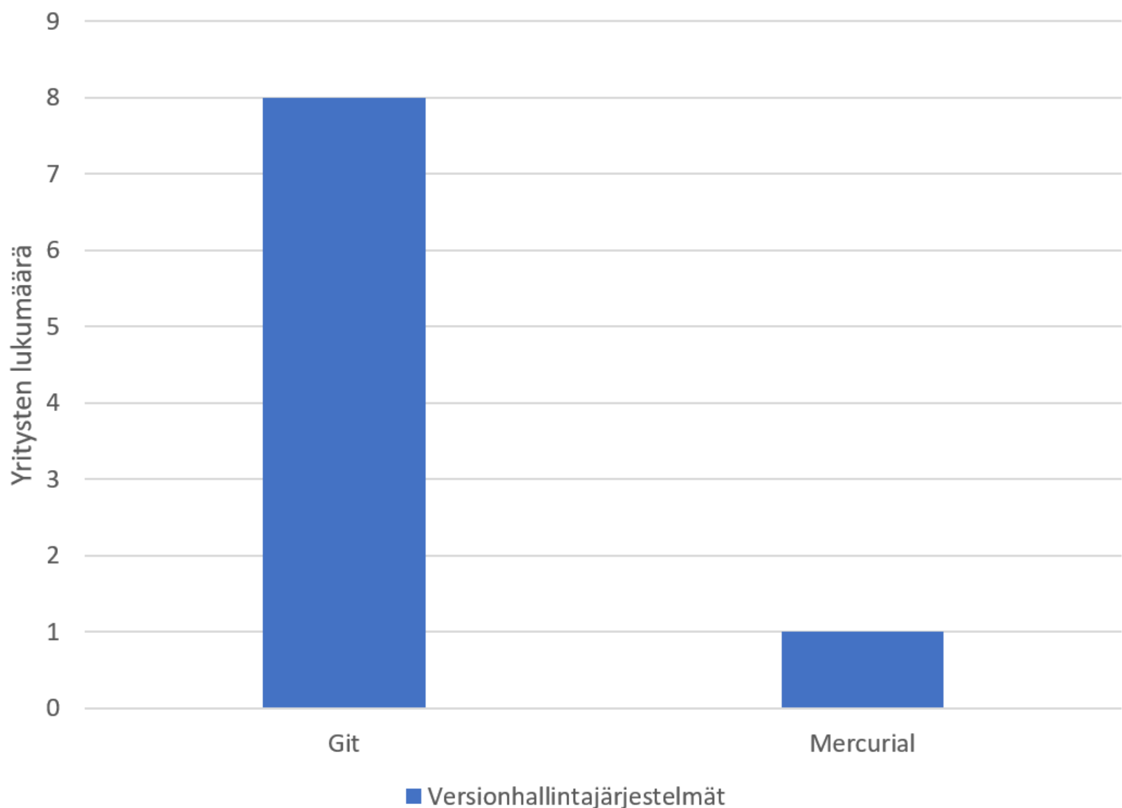
Haastateltavien mukaan *syyt npm:n käyttöön* tehtävien suoritukseen liittyivät ominaispiirteisiin. Vastaajat kokivat työkalun monipuolisena: npm on tehtävänsuorittaja samalla kun se on paketinhallintajärjestelmä, jonka vuoksi ei ole tarvetta erilliselle tehtävänsuorittajalle. Erään haastateltavan mukaan tehtävänsuorittajat ovat tarpeettomia npm:n takia. Gulpin ja Gruntin monimutkaiset työnkulut ovat poistuneet npm:n käytön myötä.

> *"Npm on task runneri… samalla kun se on paketinhallinta, niin niille ei ole enää mitään tarvetta. Et koska käytännös nois proggiksissa ainoo mitä se erilaisten buildaus-skriptien ja tollasten lisäksi enää tekee on se, et se käynnistää Webpackin, niin kaikki ne monimutkaiset työnkulut mitä Gruntissa ja Gulpissa oli, ni ne on siirtyny pois sieltä. Ei niil tee mitään enää."*

Työkalun hyvä puoli koski ominaispiirrettä, joka liittyi minimaalisuuteen. Erään vastaajan mukaan npm-skriptien ylivoimainen etu on se, että se ei vaadi ylimääräistä, sillä se on joka tapauksessa mukana projektissa. Myös *npm:n huono puoli* koski ominaispiirrettä, joka liittyi rajallisuuteen. Erään vastaajan mukaan npm on jossain määrin rajoittunut liittyen Windows-tukeen.

5.1.9 Versionhallinta

Haastateltavien vastausten perusteella jokaisessa yrityksessä käytetään Git-versionhallintajärjestelmää (ks. kuvio 14). Gitin lisäksi yhdessä yrityksessä oli erään vastaajan mukaan Mercurial käytössä.



Kuvio 14. Ohjelmistoyrityksissä käytössä olevat työkalut versionhallintaan (n = 8)

Syyt Gitin käyttöön liittyivät tunnepohjaisiin kokemuksiin ja verkostoon. Tunnepohjaiset kokemukset liittyivät tosiasiallisuuteen. Vastaajien mukaan yhtä hyvää versionhallintajärjestelmää ei ole. Haastateltavat olivat sitä mieltä, että Git on ’de facto’, eli ainut oikea versionhallinnanjärjestelmä. Verkosto liittyi suosioon ja osaamineen. Suosioon liittyen, haastateltavien mukaan Git on suosittu, kaikille tuttu ja käytetyin versionhallintajärjestelmä. Kaikki tuntevat Gitin ja se on siksi helppo ottaa käyttöön. Osaamiseen liittyen, vastaajien mukaan kaikki osaavat käyttää Gittiä.

Gitin hyvät puolet liittyivät ominaispiirteisiin ja tunnepohjaisiin kokemuksiin. Ominaispiirteet liittyivät helppokäyttöisyyteen, selkeyteen, monipuolisuuteen ja toimivuuteen. Koskien helppokäyttöisyyttä, vastaajien mielestä Git on kätevä, koska sitä käytetään komentoriviltä. Lisäksi Gitissä pystyy käsittelemään isommankin tiimin toimintaa. Vastaajien mukaan uutta ominaisuutta tehtäessä on helppo tehdä oma haara (engl. branch), ja niitä vaihtamalla on helppo tehdä korjauksia. Selkeyteen liittyen, versionhallintajärjestelmä nähtiin selkeänä ja vastaajat kokivat, että Gitistä löytyy helposti tietoa askarruttaviin kysymyksiin.

Liittyen monipuolisuuteen, vastaajat kokivat työkalun myös monipuoliseksi. Toimivuuteen liittyen, Git koettiin toimivana. Lisäksi Git tekee lupaavansa asiat ja tekee ne hyvin. Tunnepohjaiset kokemukset liittyivät kokemuksiin. Gitin koettiin olevan todella hyvä työkalu. Lisäksi haastateltavat pitivät Gittiä parempana verrattuna muihin versionhallintajärjestelmiin. Muihin versionhallintajärjestelmiin verratessa, erään vastaajan mukaan haarojen hallitseminen on nopeaa ja kätevää. Erään toisen haastateltavan mukaan Git on parempi kuin SVN-versionhallintajärjestelmä.

Gitin huonot puolet liittyivät ominaispiirteisiin. Vastaajien kokemukset liittyivät haastavuuteen. Haastateltavien mukaan Gitissä on paljon asioita, joita on vaikea tehdä. Haastateltavat kokivat, että kaikkien toiminnallisuuksien muistaminen saattaa olla myös haastavaa. Erään haastateltavan mukaan Gitin sisäistäminen vaatii aikaa. Seuraavassa taulukossa on lueteltuna Gitin hyviä ja huonoja puolia (ks. taulukko 23).

Taulukko 23. Git-versionhallintajärjestelmän hyviä ja huonoja puolia

Hyvät puolet	Huonot puolet
• Helppo tehdä omia haaroja • Haaroja vaihtamalla helppoa tehdä korjauksia • Komentoriviltä käyttö kätevää • Isomman tiimin toiminnan käsitteleminen • Selkeä • Löytyy helposti tietoa • Monipuolinen • Toimiva • Todella hyvä • Tekee lupaamansa asiat hyvin • Parempi verrattuna muihin versionhallintajärjestelmiin	• Sisäistäminen vaatii aikaa • Paljon asioita, joita on vaikea tehdä • Kaikkien toiminnallisuuksien muistaminen saattaa olla haastavaa

Syy Mercurialin käyttöön liittyi projektin lähtökohtiin. Mercurialia käytetään yleensä asiakkaan toiveista johtuen. Asiakas voi esimerkiksi toivoa Mercurialin käytettäväksi, jos se sattuu heiltä itseltään löytymään.

5.1.10 Testaustyökalut

Haastateltavilta kysyttiin, että käytetäänkö yrityksissä muita front end -työkaluja, joita teemahaastattelurungossa ei mainittu. Haastateltavat lisäsivät erilaisia työkaluja, joista suurin osa rajattiin pois opinnäytetyön kannalta. Rajatuista työkaluista mainitaan tarkemmin työn johdannossa. Työkalut, jotka eivät jääneet rajauksen ulkopuolelle, olivat testaustyökalut. Kaksi haastateltavaa mainitsi, että heidän yrityksissään käytetään niitä. Yhdessä yrityk-

sessä oli käytössä Mocha, Selenium ja PhantomJS. Toisessa yrityksessä Jest ja Entzyme.

Syyt Mochan, Seleniumin ja PhantomJS:n käyttöön liittyivät projektin lähtökohtiin. Haastateltavan mukaan kyseisten työkalujen käyttö on projektista riippuvaista. Sen sijaan s*yyt Jestin ja Enzymen käyttöön* liittyivät toimivuuden varmistukseen, joka koski toimivuutta ja tärkeyttä. Liittyen toimivuuteen haastateltavan mukaan työkaluilla varmistetaan, että komponentit toimivat oikein. Tärkeyttä koskien, vastaajan mukaan yksikkötestit ovat todella tärkeitä.

5.2 Työkalujen käyttö ja tulevaisuus

Tässä luvussa käsitellään front end -työkalujen arvioitua käyttöikää ja vastauksia liittyen työkalujen vaihtoon tulevaisuudessa. Materiaalista nousi esiin myös ajatuksia alan nopeasta kehityksestä työkalujen käyttöiän yhteydessä, ja ajatuksia front end -alan tulevaisuudesta työkalujen vaihtoon liittyvien vastauksien yhteydessä.

5.2.1 Työkalujen arvioitu käyttöikä

Haastateltavien *vastaukset liittyen front end -työkalujen arvioituun käyttöikään* koskivat arvioita työkalujen käyttöi'istä, arvioita tiheimmin vaihtuvista työkaluista, ajatuksia projekteista sekä mietteitä työkalujen käytöstä.

Liittyen arvioihin työkalujen käyttöi'istä, haastateltavien vastaukset vaihtelivat noin vuoden ja viiden vuoden välillä. Eräs vastaajista ei kokenut, että tämän päivän käyttämät työkalut olisivat päteviä vuoden päästä. Moni vastaajista oli sitä mieltä, että työkalujen käyttöikä on muutama vuosi. Vastaajien mukaan he tulevat käyttämään nykyisiä työkaluja muutaman vuoden, jonka jälkeen ne korvataan uusilla. Erään haastateltavan mukaan heidän nykyisillä työkaluilla on useamman vuoden käyttöikä. Muutama haastateltava oli sitä mieltä, että työkalujen käyttöikä on suunnilleen viisi vuotta. Muutamassa yrityksessä mainittiin, että heillä on projekteja alkuperäisillä työkaluilla, jotka ovat viisi vuotta vanhoja. Eräässä yrityksessä viidestä vuodesta ylöspäin vanhoja projekteja uudistetaan kokonaan. Eräs haastateltava mainitsi, että front end -sovellukset ovat harvoin kymmenvuotiaita.

Materiaalista nousi myös muita aiheita liittyen käyttöikään, kuten käyttöiän vaihtelemista, käyttöiän suunnittelemattomuutta sekä ajatuksia käyttöiättömyydestä. Liittyen käyttöiänvaihtelemiseen, erään haastateltavan mukaan tietyn työkalun käyttöikä vaihtelee paljon. Käyttöiän suunnittelemattomuuteen liittyen, eräs toinen vastaajista arveli, että työkalujen käyttöikää ei yrityksessä suunnitella. Liittyen ajatuksiin käyttöiättömyydestä, eräs haasta-

teltavista ei halunnut antaa työkaluille käyttöikää. Vastaaja koki, että työkalua käytetään niin kauan kuin se on käyttökelpoinen ja vaihdetaan mikäli tarve vaatii.

> *”Mä en anna työkaluille käytännössä käyttöikää. Mä enemmänkin aattelen sen niin, että sitä käytetään niin kauan kun se toteuttaa, antaa ratkaisun tai toteutuksen siihen tarpeeseen mikä meillä on. Elikkä jos tulee parempi, otetaan se käyttöön. Jos tulee parempi, mutta me pärjätään vanhalla, se toimii yhtä hyvin tai tässä paremmassa on joku semmoinen feature mitä me ei tarvita, ni ei oteta sitä käyttöön.”*

Tiheimmin vaihtuviksi työkaluiksi mainittiin JavaScript-sovelluskehykset, tehtävänsuorittajat ja pienet kirjastot.

> *”Sovelluskehykset millä sitä softaa rakennetaan, niin on vaihtunu tosi usein. Ja usein noi kirjastot, pienet kirjastot siel, ne vaihtuu tosi usein ja ne kehittyy tosi usein.”*

> *”Mä melkein sanoisin, että Taski Managerit. Se on ehkä ollu, se on vaihtunu kerran tässä vuosien varrella. Se on nimenomaan hypänny tosta, no itseasiassa seitsemän vuoden aikana kolme kertaa. Että Code Kitistä Grunttiin, Gruntista Gulppiin. Että siellä ollu tihein se.”*

Vastaajien ajatukset projekteista koskivat projektista riippuvuutta, projektin päivittämistä ja projektin ikää. Liittyen projektista riippuvuuteen, haastateltavien mukaan työkalujen käyttöikä riippuu projektista. Yhden vastaajan mukaan on paljon projekteja, joissa työkalut jäävät koko projektin ajaksi käyttöön. Koskien projektin päivittämistä, haastateltavien mukaan työkaluja on niin paljon, että jossain vaiheessa tietyn työkalun kehitys ja tuki on lopetettu, jolloin työkalu on järkevää vaihtaa. Haastateltavien mukaan sovellusta saatetaan käyttää viidenkin vuoden päästä, ellei ole tilaisuutta kirjoittaa sitä uudelleen. Erään vastaajan mukaan heillä on projekteja, joissa on muun muassa AngularJS käytössä, koska ei ole ollut tilaisuutta päivittää. Liittyen projektin ikään, erään haastateltavan mukaan viisi vuotta vanha front end -sovellus on yleensä vanhentuneen näköinen. Vanhentumisen myötä asiakas haluaa päivittää ulkoasua modernimmaksi.

Erään haastateltavan ajatus liittyen työkalujen käyttöä koski työkalujen kokeilemista. Vastaaja oli sitä mieltä, että työkalujen kokeileminen on hyvä asia. Työkalujen testaaminen antaa näkökulmaa ja tietotaitoa. Vastaaja oli sitä mieltä, että erilaisia työkaluja tulisi kokeilla joka tapauksessa.

Haastateltavien *ajatukset liittyen front end -alan kehitykseen* koskivat alan nopeaa kehitystä, ajatuksia teknologioista ja syitä selainsovellusten suosioon. Ajatukset alan nopeasta kehityksestä liittyivät nopeaan kehitykseen sekä työkalujen lyhytikäisyyteen. Liittyen nopeaan kehitykseen, monet vastaajat olivat sitä mieltä, että front end -alalla kehitys on to-

della nopeaa. Uusia työkaluja tulee koko ajan ja todella paljon sekä kaikkea uutta keksitään koko ajan. Vastaajat olivat yhtä mieltä siitä, että työkaluihin liittyvä myllerrys on käynnissä jatkuvasti. Liittyen työkalujen lyhytikäisyyteen, vastaajien mukaan front end -sovellusten käyttöikä on kohtuullisen rajallinen. Työkalut ovat lyhytikäisiä ja vaihtuvat yllättävän usein. Erään haastateltavan mukaan selainalustojen käyttöikä on lyhyt muihin alustoihin verrattuna, sillä selaimet kehittyvät kovaa vauhtia ja niiden myötä kaikki front end -teknologiat.

Haastateltavien vastaukset liittyen ajatuksiin teknologioista koskivat tietoisuutta uusista teknologioista ja teknologioiden valitsemista. Liittyen tietoisuuteen uusista teknologioista, erään haastateltavan mukaan heidän yritys on nopeasti perillä uusista teknologioista. Liittyen teknologioiden valitsemiseen, eräässä toisessa yrityksessä yritetään valita teknologiat parhaan mukaan, jos uusi projekti on alkamassa.

> *"…tällä alalla on aika nopeaa tämä kehitys. Ja näitä frameworkkeja tulee ja menee aika nopealla aikataulullakin joskus. Ja se nyt ei oo *****:n sisällä mitenkään tietynlainen, mutta sanoisin että aika nopeasti ollaan aina aallonharjalla näissä uusissa teknologioissa."*

Haastateltavien vastaukset liittyen selainsovellusten suosioon koskivat selainalustojen suosiota, kehityksen kustannustehokkuutta ja kehityksen uudistamisen helppoutta. Liittyen selainalustojen suosioon, erään vastaajan mukaan selainalustat ovat suosittuja, koska natiivisovelluksia ei tarvita eri alustoille, jolloin on edullista kehittää sovelluksia selaimelle. Liittyen kehityksen kustannustehokkuuteen, vastaajan mukaan asiakkaan näkökulmasta on kustannustehokasta tehdä front end -sovelluksia. Liittyen kehityksen uudistamisen helppouteen, front end on erillään back endistä, jolloin front end on helppo uudistaa kokonaan.

5.2.2 Työkalut tulevaisuudessa

Haastateltavien *vastaukset liittyen työkalujen vaihtoon tulevaisuudessa* koskivat työkalujen vaihtoa, muita ajatuksia työkalujen vaihdosta, yrityksen menettelytapoja, tulevaisuuden näkymiä sekä projektin lähtökohtia.

Työkalujen vaihto liittyi ajatuksiin vaihdon suunnittelemattomuudesta, Angulariin vaihdosta, päivittämisen halusta ja tiedottomuudesta liittyen vaihtoon. Liittyen vaihdon suunnittelemattomuuteen moni haastateltavista kertoi, ettei yrityksessä ole suunnitteilla vaihtaa työkaluja tällä hetkellä eikä lähitulevaisuudessa. Yksi vastaajista kertoi, että heillä ei ole suunnitteilla vaihtaa työkaluja, mutta ei myöskään pysyä missään. Liittyen Angulariin vaihtoon, erään vastaajan mukaan he siirtyivät puoli vuotta sitten Angulariin eikä heillä ole

aikeita siirtyä pois lähitulevaisuudessa. Päivittämisen haluun liittyen, erään haastateltavan mukaan heidän yrityksessä on hyvin vanhoja projekteja, joita he haluavat päivittää. Lopuksi vaihdosta tiedottomuuteen liittyen, eräs haastateltavista ei ollut tietoinen työkalujen mahdollisesta vaihdosta.

Liittyen muihin ajatuksiin työkalujen vaihtoon liittyen, haastateltavien vastaukset koskivat työkalujen kokeilemista, asiakkaan päätöstä työkaluja koskien sekä maltillisuutta työkalujen suhteen. Liittyen työkalujen kokeilemiseen, eräässä yrityksessä suositaan kokeilukulttuuria työkalujen suhteen. Yksi toinen vastaajista oli sitä mieltä, että heillä olisi ollut kivinen tie, jos he eivät olisi käyttäneet kokeilukulttuuria. Kolmannen vastaajan mukaan yrityksen työntekijät saavat kokeilla työkaluja vapaasti. Liittyen asiakkaan päätökseen työkaluja koskien, erään haastateltavan mukaan asiakas ei aina innostu vaihtamaan työkaluja. Asiakas saattaa pelätä, että työkaluille ei löydy tekijöitä tulevaisuudessa. Yhden toisen vastaajan mukaan osa työskentelytiimistä tulee asiakkaan puolelta, jolloin asiakkaan front end -ohjelmoijat tekevät paljon muutakin, jolloin heillä ei välttämättä ole innostusta perehtyä kaikkiin muoti-ilmiöihin.

Monessa yrityksessä suosittiin myös maltillisuutta työkalujen suhteen. Yrityksissä mietitään mitä työkaluja kannattaisi käyttää ja kuunnellaan sopivassa suhteessa kehittäjien mielipiteitä. Eräässä yrityksessä työkalujen vaihtoon liittyy harkittu konservatiivisuus, jossa jokaista uusinta trendiä ei oteta käyttöön, sillä niiden elinkaari saattaa jäädä hyvinkin lyhyeksi. Uusien trendien ilmestyessä yrityksessä katsotaan rauhassa kannattaako työkalu ottaa käyttöön. Vastaajien mukaan pitää keskittyä nykyhetkeen, mutta kuitenkin tunnustellen tulevaisuuden tuulia. Eräs vastaajista oli sitä mieltä, että ei pidä stressata niitä asioita, joihin ei tällä hetkellä pysty vaikuttamaan. Yhden vastaajan mukaan on hyvä odotella vuoden verran, että näkee mihin suuntaan ollaan menossa, ennen kuin päättää mitä tekee.

> *”…toki firman sisällä suositaan kokeilukulttuuria eikä meillä oo mitään sellaista käytäntöö, että annettais valmiina, että näillä se pitää tehdä. Et toki me myös samaan aikaan ymmärretään, että ei joka kerta voi sovelluksia rakentaa eri tavoin vaan sen takia et se on kivaa, mutta kyllä me sopivassa suhteessa koko aika tutkitaan ja sit kuunnellaan nimenomaan meidän kehittäjien mielipiteitä, ja mietitään että mihin suuntaan tätä hommaa kannattais viedä…”*

Yrityksen menettelytapoja koskien, haastateltavien vastaukset liittyivät työkalujen liittyvään käytännöttömyyteen. Yhdessä yrityksessä ei ole käytäntöä liittyen siihen, että tiettyjä työkaluja pitäisi käyttää. Eräässä toisessa yrityksessä ei ole yhteistä linjaa liittyen teknologioihin.

Koskien tulevaisuuden näkymiä, erään haastateltavan vastaus koski edelläkävijyyttä. Haastateltavan mukaan heidän yritys on edelläkävijä uusien teknologioiden käyttöönotossa.

Haastateltavien ajatukset liittyen projektin lähtökohtiin koskivat projekti-, kehittäjä- ja asiakaskohtaisuutta. Liittyen projektikohtaisuuteen vastaajien mukaan teknologiat tulevat asiakkaalta tai projektin sisältä, jolloin työkalujen vaihto riippuu projektista. Erään haastateltavan mukaan oli myös vaikea sanoa ovatko työkalut vaihtumassa, koska ne ovat niin projektikohtaisia. Liittyen kehittäjäkohtaisuuteen, vastaajien mukaan työkalujen vaihto riippuu ohjelmoijista, sillä he päättävät millä työkaluilla haluavat ohjelmoida. Eräässä yrityksessä ohjelmoijat saavat valita työkalut vapaasti harkintakykynsä mukaan. Lopuksi asiakaskohtaisuutta koskien työkalujen vaihto riippuu siitä, mitä asiakas haluaa ja mikä yritykselle sopii. Työkalujen vaihto on riippuvainen myös asiakkaan ohjelmoijien osaamisesta, jolloin valitaan osaamista vastaavat teknologiat.

Haastateltavien *ajatukset liittyen front end -alan tulevaisuuteen* koskivat tulevaisuuden näkymiä, ajatuksia natiivisovelluksista, tulevaisuuden työkaluja, ajatuksia Reactista sekä JavaScriptin dynaamisuutta.

Vastaukset liittyen tulevaisuuden näkymiin koskivat web-sovelluksiin siirtymistä, Progressive Web App eli PWA-sovelluksia sekä uudistumista. Liittyen web-sovelluksiin, muutaman haastateltavan mukaan selainsovellukset tulevat tulevaisuudessa kasvamaan entistä merkittävimmiksi, korvaten natiivisovelluksia. Haastateltavien mukaan on monta vuotta ollut trendi, että siirrytään enenevässä määrin selainalustoille natiiveista sovelluksista. Monet ennusteet puhuvat sen puolesta, että ollaan siirtymässä pelkästään web-sovelluksiin. Haastateltavat arvelivat, että selainpuolella mennään vahvemmin PWA-maailmaan, jolloin PWA on se, mitä tullaan tekemään tulevaisuudessa.

> *”…webbipuolellahan mennään vahvemmin tänne PWA-maailmaan, eli tämmöiseen Progressive Web Apit, jotka tuo sitten… niitähän on jo olemassa, eli tuo mukanaan sen mahdollisuuden, että webbisaitit voi tallentaa käynnistyskuvakkeiksi puhelimeen - Et siinä vaiheessa kun se maailma muuttuu, niin se on sitten mielenkiintoista, että painottuuko se tekeminen yhä enemmän vaan webbiin ja sovellusten määrä sitten entisestään laskee.”*

> *”...jos pitää tulevaisuutta ennustaa, ni veikkaan et tää (PWA) tulee kyllä olemaan sellainen juttu mitä tehdään.”*

PWA-sovelluksiin liittyen, vastaajien mukaan mobiilipuolella web-sovellukset koetaan vähemmän mobiilikäyttöliittymään integroituvina verrattuna natiivisovelluksiin. Asia on kuitenkin muuttunut viime aikoina PWA:n takia. Vastaajien mukaan PWA on aika uusi juttu,

jossa web-sovellus toimii kuten natiivisovellus. Käyttäjän näkökulmasta PWA:n ja natiivisovelluksen välillä ei ole eroa. Vastaajien mukaan iOS-tuki on tällä hetkellä heikompi PWA:lle. Erään haastateltavan mukaan työkalut tulevat olemaan samat, joilla PWA-sovelluksia tehdään, koska kyse on edelleen selainsovelluksista. Lopuksi uudistumiseen liittyen, erään haastateltavan mukaan viime aikoina näyttää siltä, että rajapinnat uudistuvat. Vastaajan mukaan REST-rajapinnasta luovutaan ja GraphQL tulisi tilalle.

Haastateltavien vastaukset liittyen ajatuksiin natiivisovelluksista koskivat natiivisovellusten hyödyllisyyttä. Vastaajien mukaan natiiveja työpöytäsovelluksia alkaa olemaan jo harvassa. Yrityspäättäjille toitotetaan, että he miettisivät kannattaako natiivia sovellusta tehdä. On kuitenkin paljon sovelluksia, joissa tarvitaan natiiviteknologioita, eivätkä ne ole haastateltavien mukaan korvattavissa selainsovelluksilla. Haastateltavien mukaan sovelluskaupoista ladattaville sovelluksille löytyy paikkansa jatkossakin, jos ne ovat hyödyllisiä.

Vastaukset liittyen tulevaisuuden työkaluihin koskivat Reactin ja Angularin tulevaisuutta, tulevaisuudessa tekemistä sekä työkalujen käytön ennustamattomuutta. Liittyen Reactin ja Angularin tulevaisuuteen, erään haastateltavan mukaan React ja Angular tulevat säilymään pitkälle tulevaisuuteen. Tulevaisuudessa tekeminen liittyi arveluun työkalujen käytöstä. Eräs vastaaja kertoi, että viiden vuoden päästä he luultavasti tekevät joillain uusilla työkaluilla, jotka ovat sukua tämän päivän työkaluille. Lopuksi työkalujen käytön ennustamattomuuteen liittyen, haastateltava oli sitä mieltä, että tällä alalla ei näy vuoden päähän millä tehdään, paitsi JavaScriptillä.

Haastateltavien ajatukset Reactista koskivat Reactin ikää, käyttämistä ja samankaltaisia työkaluja. Liittyen Reactin ikään, vastaajien mukaan React on ollut pitkäikäinen. Reactin käyttämiseen liittyen, erään haastateltavan mukaan heillä ei ole puheita Reactista pois siirtymisestä. Vastaaja arveli, että he saattaisivat käyttää Reactin uusinta versiota tulevaisuudessa. Liittyen Reactin kaltaisiin työkaluihin, erään haastateltavan mukaan Reactin rinnalle on ilmestynyt samaa ideologiaa käyttäviä sovelluskehyksiä ja kirjastoja, jotka ovat avointa lähdekoodia ja joissa ei ole Facebookin tuomaa lisenssi-ongelmaa.

Lopuksi erään haastateltavan mietteet liittyen JavaScriptin dynaamisuuteen koskivat JavaScriptin monipuolisuutta. Vastaajan mukaan JavaScript on äärettömän dynaaminen kieli, jonka vuoksi halutaan kokeilla, mitä kaikkea sillä voi tehdä. Esimerkiksi Hot Module Replacement (HMR) on osoitus siitä, mitä kaikkea JavaScriptillä voidaan tehdä.

> *”…esimerkiks tää Hot Module Replacement, niin on aika hieno tämmöinen osoitus, että mihin kaikkeen saadaan taipumaan tämmöinen dynaaminen kieli. Et voidaan*

> *lennossa päivittää ajossa olevaa sovellusta. Et se ei ihan kaikilla ohjelmointikielillä välttämättä onnistu tai ei todellakaan onnistu."*

5.3 Mitä haastateltavat suosittelevat opettelemaan?

Haastateltavien vastaukset liittyen asioihin, joita he suosittelevat opettelevan koskivat kieliä, työkaluja sekä muita asioita liittyen front end -kehitykseen. Edellisten lisäksi haastateltavilla oli osaamiseen liittyviä ajatuksia.

Kielistä haastateltavat suosittelivat opettelemaan JavaScriptiä, HTML:ää ja CSS:ää. Moni vastaajista kehotti opettelemaan kyseisten kielten syvimmät olemukset alkuun, ennen JavaScript-sovelluskehyksiä ja -kirjastoja. Erään haastateltavan mukaan ohjelmointia on helpotettu työkaluilla, mikä on johtanut siihen, että puhdas ymmärrys kyseisistä kielistä on heikentynyt huomattavasti. Kielten heikko ymmärrys on johtanut siihen, että ei välttämättä ymmärretä kaikkien asioiden taustoja, kuten esimerkiksi miksi jokin asia tehdään tietyllä tavalla ja miksi selainten välillä aiheutuu ongelmia.

Liittyen pelkästään JavaScriptiin, haastateltavat olivat sitä mieltä, että JavaScript on front end -ohjelmoinnissa suuressa vedossa. Kielen opetteleminen on hyvästä, sillä se antaa riittävän pohjan opetella kaikkea muuta, kuten esimerkiksi JavaScript-sovelluskehyksiä ja -kirjastoja. JavaScriptin ymmärtäminen ja osaaminen on tärkeämpää kuin tietyn sovelluskehyksen tai kirjaston osaaminen. Vastaajien mukaan kielen syvin olemus kannattaa opetella ennen sovelluskehyksiin ja kirjastoihin tutustumista, koska ne vaihtuvat, mutta JavaScript pysyy aina. Vastaajien mukaan kieli kannattaa opetella kunnolla ja erityisesti uusinta versiota, ES6:sta.

Haastateltavien *vastaukset liittyen työkaluihin, joita kannattaa opetella* koskivat JavaScript-sovelluskehyksiä ja -kirjastoja, versionhallintajärjestelmää, koodieditoreja ja kehitysympäristöjä sekä kääntäjiä.

Vastaukset liittyen sovelluskehyksiin ja kirjastoihin koskivat yleisesti sovelluskehyksiä ja kirjastoja, Reactia, Angularia sekä sovelluskehyksiin keskittymättömyyttä. Yleisesti sovelluskehyksiin ja kirjastoihin liittyen, haastateltavat olivat sitä mieltä, että JavaScriptin jälkeen kannattaa tutustua JavaScript-sovelluskehyksiin ja -kirjastoihin, koska ne ovat paremmin ymmärrettävissä silloin. Vastaajat suosittelivat opettelemaan suositun sovelluskehyksen tai kirjaston, kuten esimerkiksi Reactin tai Angularin. Liittyen Reactiin, haastateltavat suosittelivat opettelemaan sen JavaScript-kirjastoista. Haastateltavat suosittelivat opettelemaan lisäksi Reactia tukevat työkalut. Eräs haastateltavista suositteli opettelevan Reactia ja Reduxia, koska niitä käytetään paljon. Liittyen Angulariin, haastateltavat suosit-

telivat opettelemaan sen, jonka lisäksi eräs vastaajista mainitsi Angular Clientin opettelun sovellusten tekemiseen.

Sovelluskehyksiin keskittymättömyyteen liittyen, haastateltavat olivat sitä mieltä, että JavaScript-sovelluskehyksiin ja -kirjastoihin ei kannata keskittyä liikaa. Erään vastaajan mukaan ne tulevat muuttumaan uran aikana, siksi kannattaa profiloitua mieluummin hyväksi front end -ohjelmoijaksi. Jos niitä opettelee, kannattaa keskittyä enintään kahteen.

> *"…ei kannata mun mielestä mennä sellaiseen mikä oli muutama vuosi sitten takaperin, etsittiin jQuery-ninjoja, ja siitä vähän eteenpäin, niin etsittiin AngularJS-osaajia. Ni ei kannata profiloida mun mielest itseään missään tapauksessa jonkun tällaisen kovin syvällisen frameworkin tai kirjaston osaajaksi, vaan kannattaa ennemmin profiloida itsensä siihen et on hyvä frontti-devaaja, koska ne kirjastot tulee muuttumaan uran aikana ihan perkeleellisen paljon."*

Versionhallintajärjestelmistä haastateltavat suosittelivat opettelemaan Gitin. Vastaajat pitivät Gittiä ehdottomana, ja suosittelivat opettelemaan perusteet. Liittyen työpaikkahaastatteluun, eräs haastateltavista suositteli tekemään Git-harjoituksia esimerkiksi GitHubiin, johon laittaa muutamia koodiesimerkkejä. Vastaajan mukaan esimerkkien ei tarvitse olla isoja, vaan tärkeintä on se, että on kokeillut oikeita asioita. Työkokemuksen puuttuessa haastatteluun on helpompi lähtökohta, jos pystyy osoittamaan omaa osaamistasoonsa kyseisillä harjoituksilla.

Koodieditoreista ja kehitysympäristöistä vastaajat suosittelivat opettelemaan Atomin, Visual Studio Coden ja WebStormin. Erään haastateltavan mukaan Atom ja Visual Studio Code ovat hyviä vaihtoehtoja, joilla voi ohjelmoida. Erään toisen vastaajan mukaan hyviä vaihtoehtoja front end -ohjelmoimiseen ovat Atom ja WebStorm. Vastaaja piti WebStormin hyvänä puolena sitä, että se on opiskelijoille ilmainen. Lopuksi liittyen kääntäjiin, eräs haastateltavista suositteli opettelemaan TypeScriptin.

Muut asiat, joita haastateltavat suosittelivat opettelemaan, koskivat arkkitehtuuria, teoriaan perehtymistä, käyttöliittymää ja mobiilia. Arkkitehtuuri koski SPA-arkkitehtuuria, REST-rajapintaa sekä irrallisuutta. Liittyen SPA-arkkitehtuuriin, eräs haastateltavista suositteli tutustumaan siihen. SPA-sovelluksien myötä vastaaja suositteli tutustumaan myös Boweriin, Gulppiin ja Sassiin. Koskien REST-rajapintaa, eräs haastateltavista suositteli tutustumaan myös siihen. Liittyen irrallisuuteen, erään haastateltavan mukaan on hyvä ymmärtää, että front end on täysin irrallinen osa, joka pohjautuu johonkin sovelluskehykseen.

Eräs vastaajista suositteli perehtymään front end -puolen teoriaan lukemalla muutaman hyvän kirjan, joka sisältää perusperiaatteita JavaScriptistä. Vastaajan mukaan on hyvä

lukea muun muassa suunnittelukuvioista, luokista ja eri tavoista ohjelmoida objekteja, ja ymmärtää mitä ne ovat ja miten toimivat. Jos haluaa olla sovellusohjelmoija, haastateltavan mukaan on tärkeää ymmärtää miten asiat toimivat.

> *"Mä suosittelen lukemaan muutaman kirjan, jos ei oo jo lukenu. Mä tiiän, kirjojen lukeminen on tylsää, se on varmasti sitä. Voi myöskin totta kai käyttää jotain online-palvelua, kuten Code Academya ynnä muuta, mutta mun mielestä ne ei anna ihan täydellistä, kattavaa kuvaa. Mutta Eric Elliottin kirja JavaScriptistä, erittäin lämpimästi suosittelen lukee, ihan perusperiaatteita koko kielestä. Ja varsinki ku kuitenki frontti nykyään on todella paljon juurikin sitä JavaScriptiä."*

Erään haastateltavan suosittelu liittyen käyttöliittymää koski responsiivisuutta. Vastaaja suositteli ymmärtämään responsiivisuuden merkityksen. Responsiivisuuteen liittyen vastaaja mainitsi myös grid-mallin ymmärtämisen. Mobiiliin liittyen, eräs vastaajista suositteli React Nativea, mikäli mobiiliohjelmointi kiinnostaa.

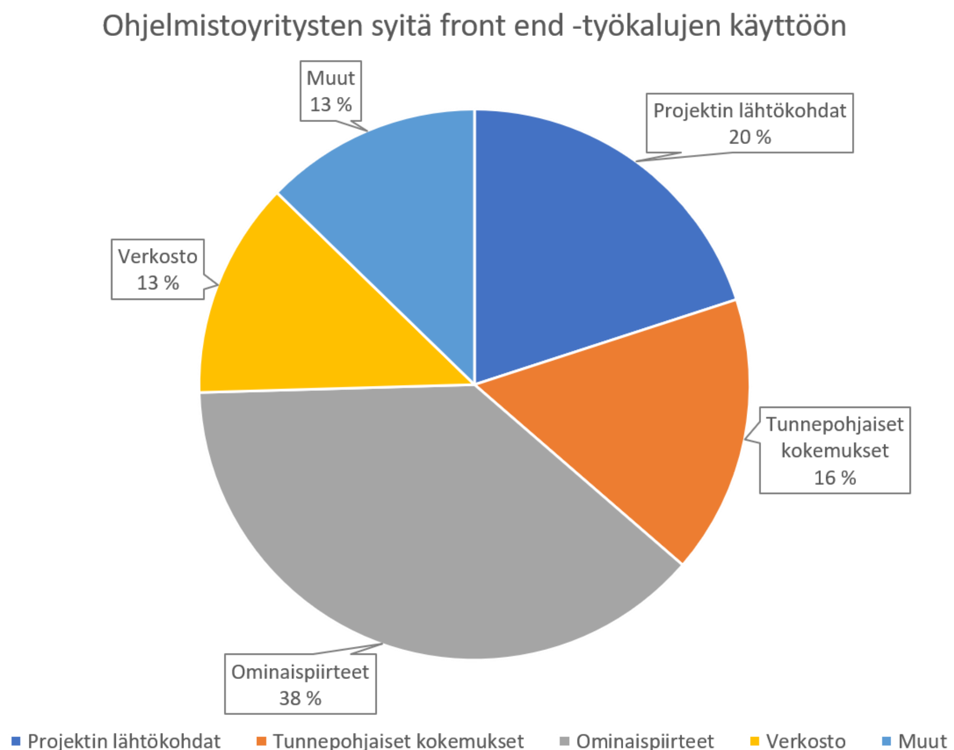
Haastateltavien osaamiseen liittyvät ajatukset koskivat suosituimpia työkaluja ja kieliä sekä työssä osaamista. Liittyen suosituimpiin työkaluihin ja kieliin, erään haastateltavan mietteet koskivat työkalujen yhdistelmiä. Haastateltavan mukaan Reactin, Webpackin ja ES6:n osaamisen kautta löytyy varmasti töitä. Vastaajan mukaan kyseinen yhdistelmä on yleisin, jolla front end -ohjelmointia tehdään. Lopuksi työssä osaaminen liittyi markkinointipuoleen ja työkalujen käyttöön. Mikäli markkinointipuolesta on kiinnostusta, ohjelmoijan pitää osata PHP:tä sisällönhallintajärjestelmien, kuten WordPressin, Drupalin ja Joomlan takia. Työkalujen käyttöön liittyen, erään vastaajan mukaan työkaluilla ei ole hirveästi väliä. Työhön pääseminen ei ole riippuvaista työkaluista, ellei käytä Notepadia.

6 Pohdinta

Tässä luvussa tarkastellaan tuloksia sekä tutkimuksen luotettavuutta ja eettisyyttä. Luvussa esitellään myös johtopäätökset sekä muodostetaan niiden pohjalta työkalupakki, jonka jälkeen esitellään jatkotutkimusehdotukset. Lopuksi arvioidaan opinnäytetyöprosessia ja omaa oppimista.

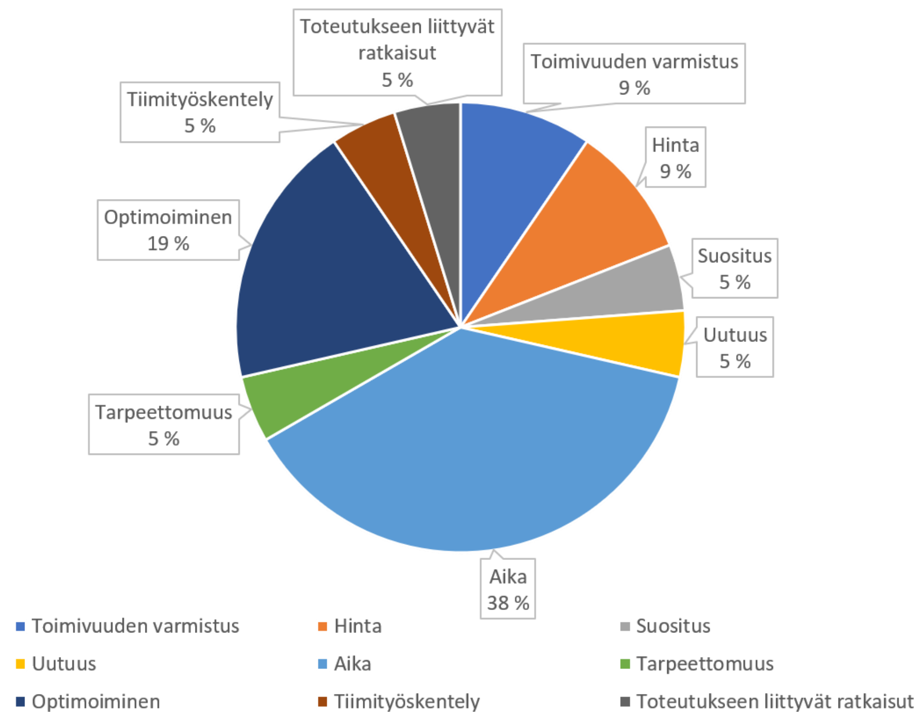
6.1 Tulosten tarkastelu

Tutkimustuloksista kävi ilmi, että suurin syy työkalujen käyttöön ohjelmistoyrityksissä oli työkalujen ominaispiirteet (ks. kuvio 15). Työkalujen valintaan vaikutti siis eniten työkalujen ominaisuudet ja piirteet, kuten esimerkiksi nopeus. Toinen syy oli projektin lähtökohdat, jossa useimmiten asiakas on vaikuttanut päätökseen. Kolmantena vaikutti tunnepohjaiset kokemukset, jossa työkalut valittiin erilaisten hyvien kokemuksien ja mieltymyksien perusteella. Neljäntenä syynä oli verkosto, johon liittyi muun muassa suosio ja ylläpidettävyys.



Kuvio 15. Ohjelmistoyritysten syitä front end -työkalujen käyttöön

Seuraavassa kuviossa on kuvailtuna muiden syiden jakautuminen (ks. kuvio 16). Eniten oli vaikuttanut aikaan liittyvät asiat ja toiseksi optimoiminen. Aikaan liittyvät asiat koskivat esimerkiksi nykyaikaisuutta. Loput syyt jakautuivat suhteellisen tasaisesti. Liitteestä 5 löytyy työkalujen käytön syyt kvantifioituna, joiden perusteella kuviot muodostettiin.



Kuvio 16. Ohjelmistoyritysten syitä front end -työkalujen käyttöön (muut)

Liittyen koodin kirjoitusvälineisiin, ohjelmistoyrityksissä käytettiin ohjelmoimiseen koodieditoreita, kehitysympäristöjä ja komentorivipohjaisia editoreja, eli laidasta laitaan. Kuten Mikkosen ja Nummen (2016b, 50) mukaan komentorivipohjaiset editorit ovat edelleen suosiossa, myös tämä tutkimus vahvistaa asian siltä osin, että niitä edelleen käytetään. Lisäksi The State of JavaScriptin (2017g) selvityksen mukaan viidenneksi käytetyin editori oli Vim, joka on komentorivipohjainen.

Yrityksissä ohjelmoijat saivat pääasiassa itse päättää, millä ohjelmoivat. Haastateltavista seitsemän kertoi ohjelmoivansa työssään, ja he käyttivät tasaisesti Atomia, Sublime Textiä ja Visual Studio Codea. Myös The State of JavaScriptin (2017g) selvityksestä ilmeni, että kolme suosituinta koodieditoria oli VS Code, Atom ja Sublime Text, joista reilusti suosituin oli VS Code. Myös Stack Overflown vuoden 2017 selvityksessä kyseisiä editoreja oli käytössä, joista eniten Sublime Textiä (Stack Overflow 2017d).

Tutkimuksesta kävi myös ilmi, että haastateltavat eivät pitäneet kehitysympäristöistä. Moni vastaajista oli kokeillut niitä ja todennut, että ne ovat raskaita. Kehitysympäristöjen hitautta korostaa myös Adams (2015, luku 1), jonka mukaan kehitysympäristöt saattavat olla ajoittain hitaita ja niitä voi olla sen takia tuskallista käyttää.

Liittyen JavaScript-sovelluskehyksiin ja -kirjastoihin, tutkimuksesta kävi ilmi, että yrityksissä oli käytössä sekä sovelluskehyksiä että kirjastoja. Eniten oli käytössä React-kirjastoa sekä AngularJS ja Angular -sovelluskehyksiä, eli suosittuja ja keskenään kilpailevia työkaluja (Williamson 2015, luku 1; Mikkonen 2016a; Vänskä 2016, 40). Tutkimuksesta kävi myös ilmi, että Reactia käytettiin yritysten sisällä eniten. Reactin suosiota tukee myös Nolanin (29.11.2016) selvitys, jonka mukaan React oli loppuvuodesta 2016 jQueryn jälkeen suosituin JavaScript-kirjastoista ja -sovelluskehyksistä. Myös Stack Overflown vuoden 2017 kehittäjäselvityksestä käy ilmi, että AngularJS oli suosituin ja React toiseksi suosituin sovelluskehyksistä ja kirjastoista (Stack Overflow 2017c).

Syyt Reactin käyttöön koskivat eniten ominaispiirteitä, tunnepohjaisia kokemuksia ja verkostoa, joiden lisäksi toiseksi tärkeimpänä syynä tuli projektin lähtökohdat. Haastateltavat luettelivat Reactille myös paljon hyviä puolia. Reactin hyvyyteen viittaa myös The State of JavaScriptin (2017b) selvitys vuodelta 2017, jonka mukaan ohjelmoijat käyttivät Reactia eniten ja käyttäisivät sitä myös uudestaan. Lisäksi Stack Overflown vuoden 2017 kehittäjäselvityksestä käy ilmi, että React oli tykätyin kirjastoista, sovelluskehyksistä ja muista teknologioista (Stack Overflow 2017b).

Tutkimuksesta myös ilmeni, että haastateltavat pitävät Reactin joustavuudesta eli siitä, että se ei sido liikaa tiettyyn kaavaan tehdä asioita. Toisaalta heidän mielestään Reactin huono puoli oli se, että se on liian joustava ja sen takia eri ohjelmoijien tai tiimien tekemät projektit voivat olla todella eri näköisiä. Näitä tuloksia tukee myös Thomas (2015, luku 9) ja Waikar (2015, luku 1), joiden mukaan kirjaston voi jäsentää omiin vaatimuksiin joustavuuden takia, mutta sen myötä koodin yhteydenmukaisuus voi puuttua, sillä ohjelmoijilla saattaa olla erilainen tapa kirjoittaa koodia. Vastauksista ilmeni myös, että React on tehokas DOM:n manipuloinnissa. Tätä tukee myös Jakobus ja Marah (2016, kuku 9), joiden mukaan React on tunnettu tehokkaasta DOM:n käsittelemisestä.

Sekä AngularJS:n että Angularin kohdalla työkalujen käytön syiksi painottuivat projektin lähtökohdat. AngularJS:n kohdalla myös verkosto oli syynä käyttöön. Angularin kohdalla yksi vastaus liittyi tunnepohjaisiin kokemuksiin. Tutkimuksesta myös ilmeni, että AngularJS:ää käytettiin vanhoissa projekteissa ja uudempaa Angularia oli jo monessa yrityksessä käytössä. Liittyen hyviin ja huonoihin puoliin, molempien Angularien kohdalla oli enemmän huonoja kuin hyviä puolia. AngularJS:n huonoja puolia tukee myös The State of JavaScriptin (2017b) selvitys, jonka mukaan lähes 10 000 AngularJS:ää käyttänyttä ohjelmoijaa ei käyttäisi sitä uudelleen. Lisäksi Stack Overflown vuoden 2017 selvityksen mukaan AngularJS tuli neljänneksi liittyen siihen, että työkalua käyttävät ohjelmoijat eivät olleet kiinnostuneet käytön jatkamisesta (Stack Overflow 2017b).

Liittyen CSS-sovelluskehyksiin ja käyttöliittymän muotoiluun, ohjelmistoyrityksissä käytettiin eniten Bootstrap-kehystä. Työkalun suosion vahvistaa myös Bootstrapin verkkosivut (2017a), joiden mukaan Bootstrap on maailman suosituin CSS-sovelluskehys. Syyt Bootstrapin käyttöön liittyivät tasaisesti ominaispiirteisiin, projektin lähtökohtiin, tunnepohjaisiin kokemuksiin ja verkostoon.

Tutkimuksesta myös ilmeni, että puolissa yrityksissä ei kuitenkaan käytetty mitään kehyksiä ja tehdään myös omia tuotoksia. Vastaajilla oli paljon erilaisia syitä kehyksien käyttämättömyyteen. Yksi syy liittyi tehottomuuteen, jossa vastaajien mukaan kehyksien mukana tulevan koodimäärän vuoksi toteutus toimii hitaasti. Tätä tukee myös Kyrnin (2015, luku 1), jonka mukaan ohjelmoija ei aina käytä kaikkia kehyksien antimia, mikä saattaa tuoda raskautta sovellukseen.

Kehykset nähtiin myös ongelmallisina, koska ohjelmoija pysyy tiukasti kiinni niiden metodologioissa. Myös Kyrninin (2015, luku 1) mukaan kehyksien huonoihin puoliin lukeutuu se, että ohjelmoijalla on vähemmän kontrollia niitä käyttäessä. Vastaajien mukaan on helpompi tehdä itse kuin kustomoida kehyksiä halutun ulkoasun mukaiseksi. Lisäksi vastaajien mukaan ne ovat menettäneet merkityksensä komponenttipohjaisella aikakaudella.

Liittyen kääntäjiin ja käännettäviin kieliin, tutkimuksesta kävi ilmi, että JavaScriptiin kääntyvistä kielistä ES6:sta käytettiin kuudessa yrityksessä ja TypeScriptiä viidessä. Kääntäjistä suosituin oli Babel-kääntäjä, joka on myös Chaun (2017, luku 3) mukaan hyvin suosittu kääntäjä. ES6:n ja TypeScriptin suosio näkyy The State of JavaScriptin (2017) selvityksessä, jonka mukaan ES6:sta oli käyttänyt 21 000 ohjelmoijaa, jonka lisäksi he käyttäisivät kieltä uudelleen. TypeScriptin kohdalla 8 000 ohjelmoijaa oli käyttänyt kieltä ja käyttäisi uudelleen. Lisäksi 8 800 ohjelmoijaa oli kuullut TypeScriptistä ja haluaisi oppia sitä. (The State of JavaScript 2017c.)

Liittyen CSS:n tyylittelyyn, yrityksissä käytettiin eniten Sass- ja Less-esikäsittelijöitä. Sassin käyttö koski ominaispiirteitä, tunnepohjaisia kokemuksia ja projektin lähtökohtia. Lessin käyttö liittyi tunnepohjaisiin kokemuksiin ja projektin lähtökohtiin. Näiden lisäksi yrityksissä käytettiin tasaisesti CSS Modulesia, PostCSS:ää, Stylusta, tavallista CSS:ää ja Styled-Componentsia. Myös The State of JavaScriptin (2017e) ja Nolanin (29.11.2016) selvitysten mukaan Sass ja Less olivat käytetyimmät työkalut tyylittelyyn. Kuitenkin CSS-JavaScriptissä -tyylittelytapa näytti The State of JavaScriptin (2017e) selvityksen mukaan olevan käytössä jo jonkin verran.

Tutkimuksesta ilmeni, että koodin tarkistamiseen käytettiin eniten ESLint-työkalua. Myös Nolanin (29.11.2016) selvityksen mukaan loppuvuodelta 2016 ESLint-työkalua käytettiin reilusti eniten. Tässä tutkimuksessa toiseksi eniten käytettiin TSLint-työkalua. Tutkimuksesta kävi ilmi, että TSLint-työkalua käytettiin TypeScriptin takia, ja se oli myös suosittu työkalu TypeScriptille Atom-koodieditorissa. Myös TSLintin verkkosivujen (2017) mukaan työkalu on tarkoitettu TypeScript-kielelle ja se on saatavilla monissa nykyaikaisissa editoreissa.

Liittyen paketinhallintaan, jokaisessa ohjelmistoyrityksessä käytettiin npm-paketinhallintaa. Npm:n suosiota vahvistaa myös npm:n (2017) verkkosivut, joiden mukaan työkalu on maailman käytetyin paketinhallintajärjestelmä. Tutkimuksesta ilmeni, että npm-työkalulla on paljon hyviä puolia. Vastaajien mukaan sitä pystyy käyttämään myös tehtävänsuorittajana ja sen vahvistaa myös Haviv (2016, luku 11). Lisäksi tutkimuksesta ilmeni, että ohjelmistoyrityksissä käytettiin Yarnia paikkaamaan npm:n puutteita, kuten esimerkiksi riippuvuuksienhallintaa ja hitautta. Tätä puoltaa se, että McKenzien ym. (2016) mukaan Yarn on kehitetty nimenomaan korvaamaan npm:n puutteita, kuten esimerkiksi hitautta ja johdonmukaisuutta riippuvuuksien hallinnassa.

Liittyen tehtävänsuorittajiin ja moduulien niputtajiin, tutkimuksesta ilmeni, että ohjelmistoyrityksissä käytettiin tehtävänsuorittajista Gulp- ja Grunt-työkaluja. Moduulien niputtajista käytettiin Webpackia ja Browserifyta. Tehtävänsuoritukseen käytettiin myös paketinhallintajärjestelmiä, kuten npm:ää ja Yarnia. Yrityksissä oli eniten käytössä Webpackia ja Gulppia. Myös Nolanin (29.11.2016) selvitys osoitti, että Webpack oli käytetyin työkalu moduulien niputukseen, jonka lisäksi Webpackin suosio oli kasvanut peräti 30 prosenttia edellisvuoteen verrattuna. Nolanin selvityksen lisäksi myös The State of JavaScriptin (2017f) vuoden 2017 selvityksen mukaan Webpack oli suosituin moduulien niputtaja. Nolanin (29.11.2016) selvityksen mukaan myös Gulp oli käytetyin työkalu tehtävien suoritukseen 43:lla prosentilla. Samaa näyttää The State of JavaScriptin (2017f) selvityksessä, jonka mukaan Gulp oli suosituin työkalu tehtävänsuorittajista.

Työn tutkimuksesta ilmeni myös, että tehtävänsuorittajat koettiin tarpeettomina, koska npm-skripteillä voi hoitaa myös tehtävänsuoritusta. Lisäksi tehtävänsuorittajat koettiin tehottomina ja npm-skriptit nähtiin kevyempänä vaihtoehtona. Myös Nolanin (29.11.2016) selvityksestä ilmenee, että npm-skriptien käyttö tehtävänsuorituksessa oli toiseksi yleisintä Gulpin jälkeen. Lisäksi npm-skriptien käyttö oli noussut peräti 22 prosenttia edellisvuoteen verrattuna. (Nolan 29.11.2016.)

Liittyen versionhallintajärjestelmiin, selvästi suosituin työkalu oli vastaajien mukaan Git, ja se oli jokaisessa yrityksessä käytössä. Tutkimuksesta ilmeni, että Gitin käyttö liittyi tunnepohjaisiin kokemuksiin ja verkostoon. Git oli vastaajien mukaan 'de facto', eli ainut oikea versionhallintajärjestelmä. Verkostoon liittyen työkalu oli kaikille tuttu ja käytetyin luokassaan. Myös RhodeCoden (2016) selvityksen perusteella Git oli vuonna 2016 suosituin versionhallintajärjestelmä. Saman Google Trends -haun perusteella, jota RhodeCodessa käytettiin, Git oli edelleen suosituin myös vuonna 2018 tammikuussa.

Liittyen testaustyökaluihin, yrityksissä käytettiin Mochaa, Jestiä, Enzymeä, Seleniumia ja PhantomJS:ää. Vastauksista myös ilmeni, että yksikkötestaus on todella tärkeää. Verratessa Nolanin (29.11.2016) ja The State of JavaScriptin (2017d) selvityksiin, Mocha oli suosituin testaustyökalu ja se on tarkoitettu yksikkötestaukseen.

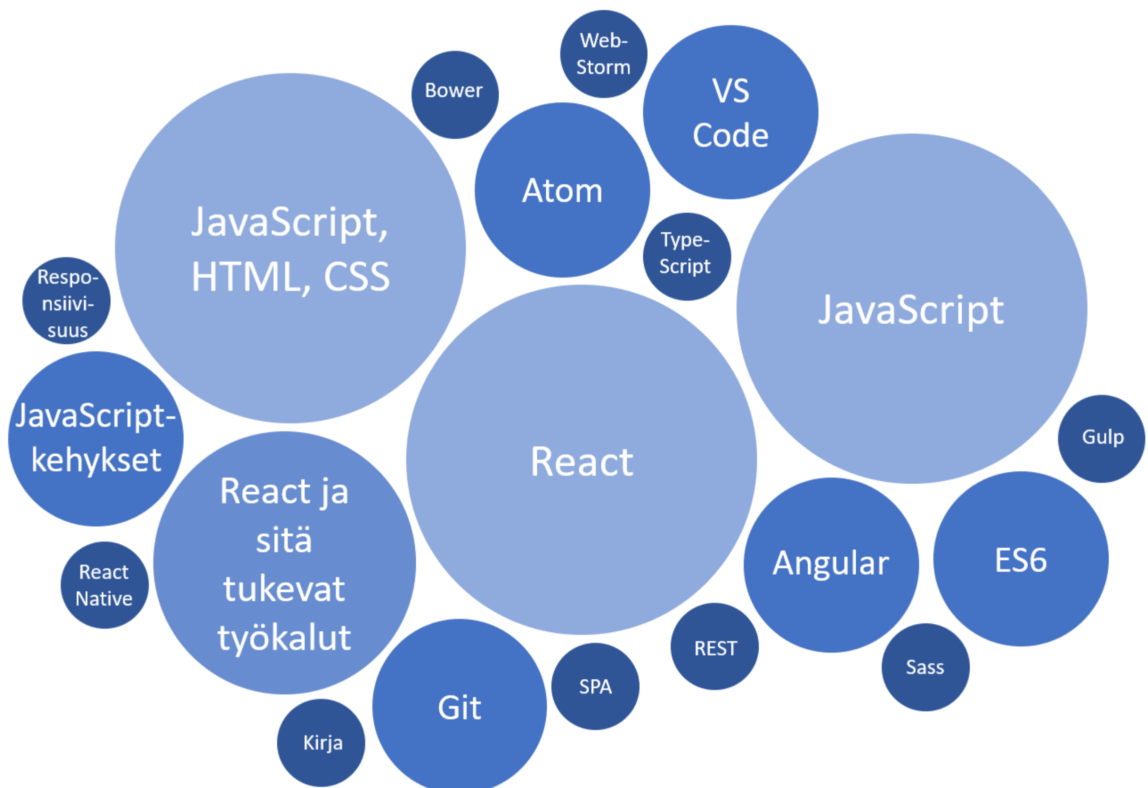
Liittyen työkalujen käyttöikään, tutkimuksesta ilmeni, että front end -työkalujen käyttöikä vaihtelee vuoden ja viiden vuoden välillä. Tiheimmin vaihtuu JavaScript-kehykset sekä tehtävänsuorittajat. Ilmeni myös, että haastateltavien mukaan front end -alalla kehitys on todella nopeaa. Myös Aquinon ja Gandeen (2016, luku 1) mukaan alan kehitys on nopeaa, sillä lukuisia työkaluja on tarjolla kehityksen tueksi ja niitä tulee koko ajan lisää.

Ohjelmistoyrityksissä oltiin kuitenkin maltillisia työkalujen vaihtamisen suhteen: yritykset katsoivat rauhassa mitä tekevät. Yrityksissä keskityttiin nykyhetkeen ja samalla tarkkailtiin tulevaisuutta. Lisäksi yritysten sisällä suosittiin kokeilukulttuuria työkalujen suhteen. Ilmeni myös, että haastatteluaikaan ohjelmistoyrityksissä ei ollut suunnitelmia vaihtaa työkaluja lähitulevaisuudessa. Kehittäjien kesken kuitenkin tutkittiin, mitä työkaluja kannattaisi käyttää. Monen vastaajan mukaan ohjelmistoyrityksissä oltiin myös ajan tasalla uusista teknologioista. Lisäksi ilmeni, että työkalujen vaihto riippuu hyvin paljon projektista. Projektiin liittyen eniten painottui se, mitä asiakas haluaa.

Tutkimuksesta kävi ilmi, että haastateltavien mukaan ennusteet puhuvat sen puolesta, että alalla ollaan siirtymässä natiivisovelluksista web-sovelluksiin. Vastaajat arvelivat, että PWA-sovellukset tulevat olemaan tulevaisuuden tapa tehdä sovelluksia. Vastaajien mukaan yrityspäättäjille toitotetaan, että he miettisivät kannattaako natiivisovelluksia tehdä. Haastateltavat eivät kuitenkaan nähneet, että natiivisovellusten tekeminen loppuisi kokonaan. Vastaajien mukaan on paljon sovelluksia, joissa tarvitaan natiiviteknologioita. Lisäksi niille löytyy paikka jatkossakin, jos ne ovat hyödyllisiä. Tuloksia puoltaa myös Ater (2017), jonka mukaan PWA-sovellukset ovat mullistava harppaus kehityksessä. Myös esimerkiksi Uotilan (13.4.2017) mukaan PWA-sovellukset haastavat jo tänä päivänä natiivisovellukset, ja ne tulevat yleistymään muutamassa vuodessa.

Tutkimuksesta myös ilmeni, että vastaajien mukaan React ja Angular tulevat säilymään pitkälle tulevaisuuteen. Myös Wilcox (16.5.2017) ennustaa, että React ja Angular taistelevat kehityksen tulevaisuudesta, ja tulevat todennäköisesti kasvamaan entisestään seuraavina vuosina. Lisäksi ilmeni, että haastateltavien mukaan React on ollut pitkäikäinen ja sen rinnalle on ilmestynyt paljon saman kaltaisia kehyksiä. Tuloksia puoltaa myös Azaustre (2016), jonka mukaan komponenttipohjaisuus näkyy jo nyt ja tulee määrittämään web-kehityksen tulevaisuutta. Myös Vänttisen (7.3.2017) mukaan komponenttipohjaiset ratkaisut ovat tulevaisuuden tapa kehittää sovelluksia.

Tutkimuksesta ilmeni, että haastattelevat painottivat opettelemaan kieliä ja erityisesti JavaScriptiä ennen JavaScript-kehyksiin perehtymistä. Seuraava kuvio 17 on muodostettu tutkimuksen vastausten pohjalta. Kuviossa on työkalut, kielet sekä muut asiat, joita haastateltavat suosittelivat opettelemaan. Iso alue vastaa neljän henkilön suositusta ja pienin yhden henkilön.



Kuvio 17. Haastateltavien suosittelemat asiat, joita kannattaa opetella

Tutkimuksesta ilmeni, että JavaScriptin osaaminen antaa riittävän hyvän pohjan JavaScript-kirjastojen ja -sovelluskehysten opetteluun. Jos JavaScript on hallussa, pystyy oppimaan minkä tahansa kehyksen. Vastauksista kävi myös ilmi, että ohjelmoijan ei kannata profiloitua tietyn JavaScript-kehyksen osaajaksi, vaan sen sijaan hyväksi JavaScript-osaajaksi. Vastaajien mukaan JavaScript-kehykset tulevat vaihtumaan, mutta JavaScript

pysyy aina. Vastaajat myös lisäsivät, että kannattaa opetella JavaScriptin uusinta versiota, ES6:sta. Haastateltavat mainitsivat myös, että JavaScriptin lisäksi kannattaa opetella myös HTML ja CSS -kielet perusteellisesti ennen muita asioita. Haastateltavat suosittelivat vasta JavaScriptin jälkeen tutustumaan sovelluskehyksiin ja kirjastoihin.

6.2 Tutkimuksen luotettavuus

Hirsijärven, Remeksen ja Sajavaaran (2009) mukaan laadullisen tutkimuksen luotettavuutta parantaa se, että tutkimuksen toteuttamisesta kirjoitetaan yksityiskohtaisesti liittyen kaikkiin tutkimuksen vaiheisiin. Näitä ovat aineiston kerääminen, aineiston analysointi ja tulosten kirjoittaminen. Aineiston keräämisestä tulisi kertoa muun muassa haastattelutilanteista ja paikoista, käytetystä ajasta, häiriöistä ja virhetulkinnoista. Aineiston analysointiin liittyvät päätökset tulisi selostaa, ja kirjoittaessa tuloksia, tulkinnat tulisi perustella. Tulkintojen perustelemista auttaa esimerkiksi suorat lainaukset. (Hirsijärvi ym. 2009, 232-233.) Tässä tutkimuksessa aineiston keräämisestä ja analysoinnissa on kerrottu luvussa 4. Lisäksi tutkimuksessa on esitetty suoria lainauksia monipuolisesti.

Tuomen ja Sarajärven (2018) mukaan laadullisessa tutkimuksessa ei ole täsmällistä luotettavuuden arviointitapaa, mutta he ovat listanneet muutamia huomioon otettavia asioita. Näistä ensimmäinen on tutkimuksen kohde ja tarkoitus, jossa tulisi kertoa mitä tutkitaan sekä syyt tutkimiselle. (Tuomi & Sarajärvi 2018, 163.) Tämän tutkimuksen kohde ja tarkoitus on kerrottu luvussa 1.

Tuomi ja Sarajärvi (2018) kirjoittavat, että luotettavuutta tarkastellessa tulisi ottaa huomioon myös omat sitoumukset tutkijana. Tutkimuksessa tulisi selostaa, miksi kyseinen tutkimus on tärkeä sekä mahdollisten olettamusten muutokset (Tuomi & Sarajärvi 2018, 163). Tutkija kokee tutkimuksen olevan tärkeä sen ajankohtaisuuden vuoksi, ja haluaa helpottaa aloittelevien ohjelmoijien työkaluihin liittyvää päätöksentekoa. Tutkijan ajatukset eivät ole muuttuneet, vaan entisestään vahvistuneet tutkimuksen loppua kohden.

Luotettavuutta voi tarkastella myös aineiston keräämisen kautta. Tutkimuksessa tulisi kertoa menetelmät, tekniikat, erityispiirteet, ongelmat sekä muut huomattavat asiat. Lisäksi tulisi kertoa perustelut osallistujien valinnalle sekä arvioida tutkimukseen osallistujien ja tutkijan suhde. (Tuomi & Sarajärvi 2018, 164.) Aineiston keruuta koskevista asioista sekä syy osallistujien valinnoille on kerrottu luvussa 4. Haastateltavien ja tutkijan suhde toimi oikein hyvin, sillä kommunikointi oli sujuvaa eikä ongelmia ilmennyt missään vaiheessa.

Lopuksi tulisi kertoa tutkimuksen kestosta, aineiston analysointitavoista, tutkimuksen luotettavuudesta ja raportoinnista (Tuomi & Sarajärvi 2018, 164). Työn tekeminen alkoi maaliskuun loppupuolella 2017 ja on kestänyt melkein yksitoista kuukautta. Työ tullaan todennäköisesti julkaisemaan keväällä 2018. Aineiston analysointitavoista ja raportoinnista on kerrottu luvussa 4 ja 5. Lopuksi liittyen luotettavuuteen, tutkija on pyrkinyt parhaansa mukaan raportoimaan tutkimuksen eri vaiheet niin, että ne olisivat helposti ymmärrettävissä ja toistettavissa.

6.3 Tutkimuksen eettisyys

Hirsijärven ym. (2009) mukaan tutkijan on huomioitava tutkimuksen tekoon liittyvät eettiset kysymykset. Suomessa on monia tahoja, jotka ohjeistavat ja valvovat tutkimukseen liittyviä eettisiä periaatteita. Näistä yksi on tutkimuseettinen neuvottelukunta, jonka mukaan tämän tutkimuksen eettisiä periaatteita tarkastellaan. (Hirsijärvi ym. 2009, 23.) Lisäksi tutkimusaiheen valinta sekä ihmisoikeudet liittyvät eettisiin ratkaisuihin, joiden mukaan tämän tutkimuksen eettisyyttä tarkastellaan myös (Tuomi & Sarajärvi 2018, 153-155; Hirsijärvi ym. 2009, 24-25).

Tutkimuseettisen neuvottelukunnan (2012) mukaan hyvään tieteelliseen käytäntöön kuuluu, että tutkimuksessa noudatetaan tarkkuutta, huolellisuutta ja rehellisyyttä tutkimuksen eri vaiheessa (TENK 2012, 6). Tässä tutkimuksessa on luettu ja tarkistettu moneen kertaan koko tutkimusprosessiin liittyvät asiat. Näitä olivat muun muassa litteroidut aineistot, sisällönanalyysin tulokset, tutkimustulosten esittäminen ja arviointi, aineiston kvantifiointiluvut, kuviot ja taulukot, sekä tietoperustaan liittyvä tieto ja lähteet. Erityisesti tutkimuksen suuruuden vuoksi näistä asioista on pyritty pitämään huolta, jotta tutkimusprosessi ei monimutkaistuisi.

Hyvään tieteellisen käytäntöön kuuluu myös se, että tutkimuksessa sovelletaan tieteelliselle tutkimukselle tyypillisiä menetelmiä, jotka ovat eettisesti kestäviä. Lisäksi tuloksia julkaistaessa ollaan avoimia. (TENK 2012, 6.) Tässä tutkimuksessa on käytetty sellaisia tutkimusmenetelmiä, jotka ovat tyypillisiä tutkimukselle, kuten laadullinen tutkimus, teemahaastattelu, nauhoitus, litterointi, sisällönanalyysi ja aineiston kvantifiointi. Menetelmistä on kerrottu ja niitä on avattu tarkemmin luvussa 4.

Hyvää tieteellistä käytäntöä tukee myös se, että muiden tutkijoiden työ ja saavutukset huomioidaan ja kunnioitetaan. Muille tutkijoille tulisi antaa niille kuuluva arvo ja merkitys omassa tutkimuksessa. (TENK 2012, 6.) Tässä tutkimuksessa on viitattu parhaan mukaan kaikkiin tutkimuksessa käytettyihin lähteisiin ja selvityksiin. Tutkimuksessa on myös pyritty

käyttämään ensisijaisesti primäärilähteitä. Lisäksi sekä tutkija että ulkopuoliset henkilöt ovat tarkistaneet moneen otteeseen lähteiden oikeanlaisen viittauksen, jotta virheiltä vältyttäisiin.

Tutkimus tulisi myös suunnitella, toteuttaa, raportoida sekä tallentaa tieteellisyyden periaatteita noudattaen. (TENK 2012, 6.) Tutkija ei ole aiemmin tehnyt vastaavaa tutkimusta, jolloin eteneminen ei ole tapahtunut täysin järjestelmällisesti. Tämä tutkimus on kuitenkin pyritty parhaan mukaan suunnittelemaan, toteuttamaan ja raportoimaan lukemalla etukäteen ja koko tutkimuksen ajan alan kirjoja tutkimuksen eri vaiheista. Tarvittaessa asioihin on palattu ja tutkittu lisää. Tutkimuksessa syntyneet aineistot on säilytetty omassa osiossaan tietokoneella sekä varmuuskopioitu, jotta ne säilyvät käyttökelpoisina tutkimuksen ajan. Aineistojen tietoturva on varmistettu salasanojen ja tietoturvaohjelmien turvin.

Tutkimuseettisen neuvottelukunnan (TENK 2012, 6) mukaan tarvittavat tutkimusluvat tulisi hankkia ja tutkimukselle tulisi suorittaa eettinen ennakkoarviointi. Tässä tutkimuksessa ei koettu tarpeelliseksi hankkia tutkimuslupia, sillä tutkija ei kokenut aiheen olevan arkaluontoinen eikä yrityksillä ollut tiettävästi omia menettelytapoja koskien asiaa. Tutkimuksen edetessä kysymys tutkimusluvasta nousi kuitenkin uudelleen pintaan ja tutkija pohti jälkeenpäin olisiko lupa pitänyt pyytää. Tässä tutkimuksessa on pidetty anonymiteetti sekä yritysten että haastateltavien osalta. Lisäksi muutamat haastateltavat olivat keskustelleet johtohenkilökunnan kanssa asiasta, joiden mielestä tutkimus vaikutti hyväksyttävältä idealta.

Eettinen ennakkoarviointi suoritettiin siten, että tutkittavia tiedotettiin haastattelukutsussa (ks. liite 2) kertomalla tutkimuksen aihe, aineistonkeruumenetelmät, aineiston käyttötarkoitus ja anonymiteetti koskien sekä haastateltavaa että yritystä. Liittyen aineiston jatkokäyttöön, kutsussa ilmoitettiin, että kaikki tutkimusmateriaali poistetaan asianmukaisesti tutkimuksen loputtua. Kutsussa ilmoitettiin myös tutkijan yhteystiedot mahdollisia kysymyksiä varten. Ennen haastattelutilannetta tutkimukseen osallistuville annettiin tai lähetettiin kirjallinen suostumuslomake (ks. liite 3) allekirjoitettavaksi, jossa ilmoitettiin osallistumisen vapaaehtoisuus ja sen, että haastattelun voi keskeyttää milloin tahansa.

Hyvään tieteelliseen käytäntöön kuuluu myös kaikkien tutkimuksen osapuolten kesken sopiminen käytännöistä, jotka liittyvät oikeuksiin, vastuisiin, velvollisuuksiin, käyttöoikeuksiin, tekijyyden periaatteisiin ja aineistojen säilyttämiseen (TENK 2012, 6). Tässä tutkimuksessa ei ollut toimeksiantajaa eikä muita osallisia lukuun ottamatta haastateltavia. Liittyen aineistojen säilyttämiseen, haastatteluun osallistuvia tiedotettiin, että kaikki tutkimusmateriaali hävitetään tutkimuksen loputtua.

Hyvään käytäntöön kuuluu myös tutkimuksen rahoituslähteiden ja muiden sidonnaisuuksien ilmoittaminen julkaisussa sekä asianosaisille ja tutkimukseen osallistuville (TENK 2012, 6). Tässä tutkimuksessa ei ollut rahoittajia, vaan tutkija kustansi haastatteluista palkkioksi tarkoitetut elokuvaliput itse. Sitä, että tutkija kustansi liput itse, ei ilmoitettu tutkimukseen osallistuville erikseen johtuen tutkijan kokemattomuudesta tutkimuksen teossa. Tässä tutkimuksen julkaisussa se tulee kuitenkin esille muutamaan otteeseen.

Tutkijan ollessa esteellinen, hänen täytyy pidättäytyä kaikista päätökseen ja arviointiin liittyvistä tilanteista (TENK 2012, 7). Tämän tutkimuksen arviointiin liittyviin asioihin tutkija ei osallistu ollenkaan, sillä vastuu arvioinnista on Haaga-Helian ammattikorkeakoululla. Tutkija on järjestänyt tutkimusprosessiin liittyviä kokouksia Haaga-Helian nimittämän opinnäytetyön ohjaajan kanssa, joissa on suunniteltu tutkimuksen etenemiseen liittyviä asioita.

Tutkijan tulisi myös huomioida tietosuojaan liittyvät asiat sekä noudattaa hyvää talous- ja henkilöstöhallintoa (TENK 2012, 7). Kuten aiemmin mainittu, tämän tutkimuksen aikana on pidetty tutkimusprosessiin liittyviä kokouksia, joissa on suunniteltu tutkimuksen etenemiseen liittyviä asioita. Liittyen tietosuojaan, haastateltaville ilmoitettiin haastattelukutsussa (ks. liite 2), että sekä haastateltavien että yritysten kohdalla anonymiteetti säilytetään. Tutkimus on suunniteltu ja kirjoitettu niin, että yrityksiä eikä haastateltavia voi tunnistaa. Tutkimusaineisto on myös käsitelty siten, ettei kukaan ulkopuolinen ole voinut päästä niihin käsiksi. Lisäksi vain tutkimuksen tekijä on ollut tietoinen tutkimukseen osallistuneista ja kohdeyrityksistä.

Liittyen tutkimusaiheen valinnan eettisyyteen, tulisi miettiä, miksi tutkimusaihe valitaan ja miksi tutkimus tehdään (Tuomi & Sarajärvi 2018, 154; Hirsijärvi ym. 2009, 24). Tämän tutkimuksen aihe valittiin ja tutkimukseen ryhdyttiin tutkijan omasta kokemuksesta liittyen työkalujen paljouteen. Tutkija huomasi myös muiden opiskelijoiden pohtivan samaa asiaa sekä luki muun muassa asiaan liittyviä artikkeleita. Tutkimukseen ryhdyttiin, jotta aloittelevien ohjelmoijien olisi helpompi päästä alkuun urassa.

Hirsijärven ym. (2009, 25) mukaan tutkimuksessa tulisi olla lähtökohtana se, että ihmisarvoa kunnioitetaan. Tutkittaville tulisi kertoa ymmärrettävästi tutkimuksen tarkoitus, tutkimustavat ja potentiaaliset riskit. Osallistujien tulisi myös tietää, että osallistuminen on vapaaehtoista ja sen voi keskeyttää milloin vain. Tutkittavien hyvinvointi tulisi olla ensisijalla eikä heille pidä aiheuttaa vahinkoa. Osallistujien anonymiteetti tulisi säilyttää ja tutkimuksesta saatua tietoa tulisi pitää yksityisenä. Lopuksi tutkijan on toimittava luvatusti. (Tuomi & Sarajärvi 2018, 155-156.)

Tässä tutkimuksessa osallistujille ilmoitettiin haastattelukutsussa tutkimuksen tavoitteesta ja tutkimusmenetelmistä. Lisäksi ennen haastattelutilannetta osallistujilta varmistettiin, että he ymmärtävät tutkimuksen kulun menetelmät. Tähän tutkimukseen ei tiettävästi liity riskejä, joten niistä ei keskusteltu. Lisäksi tutkittaville ei tiettävästi ole aiheutunut vahinkoa tutkimuksen seurauksena. Osallistujien vapaaehtoisuus varmistettiin siten, että heille annettiin tai lähetettiin suostumuslomake allekirjoitettavaksi (ks. liite 3). Suostumuslomakkeessa luki, että haastattelun voi keskeyttää milloin tahansa. Osallistujille ilmoitettiin haastattelukutsussa, että anonymiteetti säilytetään sekä tutkittavan että yrityksen osalta (ks. liite 2). Tutkimuksessa haastateltavien tietoja ei ole luovutettu eikä niistä ole kerrottu ulkopuolisille.

6.4 Johtopäätökset, työkalupakin muodostaminen ja jatkotutkimusehdotukset

Tässä tutkimuksessa oli tarkoitus saada vastaus tutkimusongelmaan ”Mitä front end -kehityksessä käytettäviä työkaluja kannattaa opetella tulevaisuuden työtä ajatellen?”. Tutkimusongelmaan ja sen alakysymyksiin saatiin vastaukset. Tämän tutkimuksen tulokset pääosin vahvistavat aikaisempaa kirjallisuutta ja selvityksiä aiheesta. Vaikka tutkimuksen kohteena oli vain kahdeksan ohjelmistoyritystä, tuloksissa näkyi paljon samankaltaisuutta käytetyissä työkaluissa, kun vertailupohjana käytettiin esimerkiksi aiempia maailmanlaajuisia selvityksiä sekä kirjallisuutta. Tämä tutkimus tuottaa myös uutta tietoa liittyen syihin työkalujen käyttöön ja siihen, mitä asioita kannattaa opetella työelämää varten.

Johtopäätöksenä voi todeta, että ohjelmistoyrityksissä ohjelmoijat eivät aina päätä mitä työkalua käyttävät, sillä usein myös asiakas vaikuttaa päätökseen. Työkalun käyttöä ei toisin sanoen voi aina suunnitella. Front end -ohjelmointia aloittaessa on kuitenkin hyvä opetella ensin JavaScriptin, HTML:n ja CSS:n perusteet ennen muita asioita, mikä todennäköisesti helpottaa myös tuntemattomiin työkaluihin tutustumista. Perusteiden osaaminen koskee myös tulevaisuutta, johtuen työkalujen tiheästä vaihtelusta. Työkalut vaihtuvat ennen pitkään, jolloin perusteiden osaaminen on tärkeintä. Vasta perusteiden jälkeen kannattaa siirtyä JavaScript-kehyksiin. Lisäksi taustalla tapahtuva siirtymä komponenttipohjaiseen kehitykseen kannustaa valitsemaan kehyksen, joka tukee kyseenomaista kehitystapaa.

Haastateltavien yrityksissä käytetään samoja työkaluja mitä muualla maailmassa, joten tärkeimmäksi työkalun valintakriteeriksi nousee suosio ja toiseksi hyvät kokemukset. Muutaman työkalun kohdalla oli kuitenkin epäselvää, mitä kannattaa käyttää. Näitä olivat CSS-sovelluskehykset ja CSS:n tyylittely. Bootstrap on suosittu ja paljon käytetty, mutta vastaajien mukaan käyttöliittymän muotoilu on siirtymässä komponenttikirjastoilla tehtäviksi. Ja

vaikka CSS-JavaScriptissä (engl. CSS-in-JS) on saanut jalansijaa, Sass-esikäsittelijää käytetään edelleen. Tästä voi päätellä, että CSS-työkalujen suhteen saattaa olla tapahtumassa muutos.

Johtopäätösten pohjalta luotiin työkalupakki (ks. liite 6), joka tukee komponenttipohjaista kehitystä ja koostuu pääosin suosituimmista työkaluista. Työkalupakkiin on valittu sekä Bootstrap että Sass, sillä niitä edelleen käytetään tällä hetkellä eniten. On kuitenkin hyvä pitää silmällä CSS-työkaluihin liittyvää muutosta, ja tarvittaessa korvata kyseiset työkalut. Koodieditoriksi on valittu VS Code, sillä työkalulla ei ollut vastaajien mukaan suorituskyvyllisiä ongelmia ja se toimii kaikin puolin hyvin. Myös erään haastateltavan kokemuksen mukaan editorissa on parhaat puolet kaikista editoreista. Tehtävänsuoritukseen valittiin npm-skriptit, sillä työkalupakki pysyy siten yksinkertaisena ja kevyempänä. Testaustyökaluista otettiin vain yksikkötestaukseen tarkoitettu työkalu, sillä yksikkötestejä painotettiin tutkimuksessa tärkeiksi. Yarnia ei otettu työkalupakkiin mukaan, koska työkalupakki haluttiin pitää yksinkertaisena, mutta sen voi tarvittaessa ottaa käyttöön npm:n rinnalle.

Koska front end -alan kehitys on nopeaa, tämän tutkimuksen johdosta muodostunut työkalupakki todennäköisesti vanhentuu muutaman vuoden päästä. Siksi jatkotutkimusehdotuksena on tutkimuksen toisto, mutta esimerkiksi määrällisessä muodossa. Jatkotutkimuksessa voisi kysyä pelkästään käytössä olevia työkaluja, sillä tästä tutkimuksesta on selvinnyt syyt työkalujen käyttöön ja valintaan. Lisäksi määrällinen tutkimus voisi mahdollistaa sen, että ei välttämättä tarvitse tehdä tiukempaa rajausta työkalujen suhteen. Määrällisen tutkimuksen voisi lähettää esimerkiksi sähköpostitse ohjelmistoyritysten front end -kehityksestä vastaaville. Samanlaisen määrällisen tai laadullisen tutkimuksen voi myös tehdä esimerkiksi back end -työkaluille.

6.5 Opinnäytetyöprosessin ja oman oppimisen arviointi

Opinnäytetyöprosessi oli ajallisesti odotettua pidempi johtuen tutkimisen ensikertaisuudesta ja töistä prosessin ohella. Ensikertaisuuteen liittyviä haasteita ilmaantui toinen toisensa jälkeen, joista piti ottaa selvää, mutta hyvin kiinnostava aihe piti mielenkiintoa yllä koko tutkimusprosessin ajan. Työn idea tuli omasta ja muiden opiskelijoiden kokemuksista, jonka takia koin aiheen todella tärkeäksi. Aluksi ei ollut varmuutta osallistuuko kukaan tutkimukseen, mutta pitkäjänteinen yrittäminen ja huolellinen valmistuminen luultavasti auttoivat haastattelijoiden keräämisessä. Moni haastateltavista innostui aiheesta, mikä sai ottamaan tutkimuksen entistä vakavammin.

Opinnäytetyöprosessin aikana pidin tuntikirjanpitoa, josta pystyin seuraamaan tutkimuksen kulkua ja siihen käytettyä aikaa, mikä helpotti kokonaisuuden hallintaa. Prosessin kuluessa työtä oli kuitenkin hankala pitää siedettävän kokoisena johtuen työkalujen paljoudesta. En kuitenkaan tiedä miten hyvin työkalupakin rakentaminen olisi onnistunut ilman kaikkea tutkimuksesta saatua tietoa. Lisäksi aiheen jatkuvan muutoksen ja uutuuden takia oli hankala löytää sellaisia lähteitä, joissa olisi käsitelty samassa paikassa montaa asiaa. Tämän johdosta jouduin käyttämään laajalti erilaisia lähteitä ja ottamaan lauseita eri paikoista, jonka takia lähteitä kertyi runsaasti. Näiden asioiden takia tutkimus kasvoi jokseenkin suureksi. Koen silti, että sen tekeminen kokonaisuudessaan oli loistavaa harjoitusta mahdollisia maisterinopintojani varten.

Opin myös todella paljon erilaisia asioita tutkimusprosessin aikana. Opin paljon uusia työkaluja ja erityisesti niiden teknistä puolta. Jokaisen kirjoittamani työkalun kohdalla oli ensin ymmärrettävä mihin tarkoitukseen sitä käytetään ja mitä se tekee, jotta siitä osaisi kertoa. Opin myös paljon tutkimuksen kulusta ja erilaisista käytänteistä. Laadullinen tutkimus piti sisällään paljon erilaisia vaiheita, mikä oli myös piristävää vaihtelua ja uuden odotusta. Myös lähes päivittäinen kirjoittaminen paransi tiedon omaksumista sekä kirjoitusnopeutta ja -tyyliä. Oli myös hienoa, kun tutkimuksen aikana sattumalta itse koin ensimmäistä kertaa PWA-sovelluksen. Kaiken kaikkiaan minulle jäi oikein positiivinen kokemus projektista.

Lähteet

Adams, C. 2015. Mastering JavaScript High Performance. Packt Publishing. Birmingham.

Airbnb 2017. Introduction. Luettavissa: http://airbnb.io/enzyme/. Luettu: 7.12.2017.

Ajzele, B. 2015. Magento 2 Developer’s Guide. Packt Publishing. Birmingham.

Alabes, T. & Tarkus, K. 2017. Isomorphic JavaScript web Development. Packt Publishing. Birmingham.

A M, V. & Sonpatki, P. 2016. ReactJS by Example – Building Modern Web Applications with React. Packt Publishing. Birmingham.

Ambler, T. & Cloud, N. 2015. JavaScript Frameworks for Modern Web Dev. Apress. New York.

Angular 2017a. Architecture Overview. Luettavissa: https://angular.io/guide/architecture. Luettu. 9.11.2017.

Angular 2017b. What is Angular? Luettavissa: https://angular.io/docs. Luettu: 9.11.2017.

Angular University 25.9.2015. Introduction to the Jspm package manager and the SystemJS module loader. Angular University. Luettavissa: https://blog.angular-university.io/introduction-to-es6-modularity-the-jspm-package-manager-and-the-systemjs-loader/. Luettu: 6.1.2018.

Antani, V., Timms, S. & Mantyla, D. 2016. JavaScript: Functional Programming for JavaScript Developers. Packt Publishing. Birmingham.

Atkinson, B. 2015. Custom SharePoint Solutions with HTML and JavaScript: For SharePoint 2013 and SharePoint Online. Apress.

Atom 2017. Atom – A hackable text editor for the 21th Century. Luettavissa: https://atom.io/. Luettu: 9.11.2017.

Aquino, C. & Gandee, T. 2016. Front-End Web Development: The Big Nerd Ranch Guide. Big Nerd Ranch Guides. Atlanta.

Ater, T. 2017. Building Progressive Web Apps. O'Reilly Media. Sebastopol.

Aurelia 2017. What is Aurelia? Luettavissa: http://aurelia.io/docs/overview/what-is-aurelia#what-is-aurelia. Luettu: 9.11.2017.

Azaustre, C. 2016. Web components: present and future in the web development. Luettavissa: https://bbvaopen4u.com/en/actualidad/web-components-present-and-future-web-development. Luettu: 6.1.2018.

Backbone 2017. Backbone.js. Luettavissa: http://backbonejs.org/. Luettu: 21.11.2017.

Bacon.js 2017. What is Bacon.js? Luettavissa: https://baconjs.github.io/. Luettu: 21.11.2017.

Ballard, P. 2015. JavaScript in 24 Hours, Sams Teach Yourself, Sixth Edition. Sams. Indianapolis.

Baumgartner, S. 2016. Front-End Tooling with Gulp, Bower, and Yeoman. Manning Publications. Shelter Island.

Bertoli, M. 2017. React Design Patterns and Best Practices. Packt Publishing. Birmingham.

Bibeault, B., Katz, Y. & De Rosa, A. 2015. jQuery in Action, Third Edition. Manning Puplications. Shelter Island.

Bootstrap 2017a. Bootstrap. Luettavissa: https://getbootstrap.com/. Luettu: 28.11.2017.

Bootstrap 2017b. History. Luettavissa: https://getbootstrap.com/docs/4.0/about/history/. Luettu: 28.11.2017.

Bootstrap 2017c. Team. Luettavissa: https://getbootstrap.com/docs/4.0/about/team/. Luettu: 28.11.2017.

Bower 2017. Bower. A package manager for the web. Luettavissa: https://bower.io/. Luettu: 6.5.2017.

Branas, R., Chardermani., Frisbie, M. & Haviv, A. 2016. AngularJS: Maintaining Web Applications. Packt Publishing. Birmingham.

Browserify 2017. Hello World With Browserify. Luettavissa: http://browserify.org/. Luettu: 13.12.2017.

Burchard, E. 2017. Refactoring JavaScript. O'Reilly Media. Sebastopol.

Casciaro, M. & Mammino, L. 2016. Node.js Design Patterns – Second Edition. Packt Publishing. Birmingham.

Cassio de Sousa, A. 2015. Pro React. Apress.

Chau, G. 2017. Vue.js 2 Web Development Projects. Packt Publishing. Birmingham.

Chaudhary, M. & Kumar, A. 2015. Practical jQuery. Apress.

Chauhan, S. 2014. Understanding MVC, MVP and MVVM Design Patterns. Luettavissa: http://www.dotnettricks.com/learn/designpatterns/understanding-mvc-mvp-and-mvvm-design-patterns. Luettu: 23.11.2017.

CoffeeScript 2017. CoffeeScript. Luettavissa: http://coffeescript.org/#top. Luettu: 3.12.2017.

Dahlstrom, R. 2015. Using Linters For Faster, Safer Coding With Less Javascript Errors. Luettavissa: https://raygun.com/blog/using-linters-for-faster-safer-coding-with-less-javascript-errors/. Luettu: 8.5.2017.

Daityari, S. 2015. Jump Start Git. SitePoint. Melbourne.

Dasa, R. 2016. Leran CakePHP: With Unit Testing, Second Edition. Apress. New York.

Dayley, B. & Dayley, B. 2015. Sams Teach Yourself AngularJS, JavaScript and jQuery. Sams. Indianapolis.

Eisenman, B. 2016. State of the JavaScript Landscape: A Map for Newcomers. Luettavissa: https://www.infoq.com/articles/state-of-javascript-2016. Luettu: 27.4.2017.

Electron 2017. About Electron. Luettavissa: https://electronjs.org/docs/tutorial/about. Luettu: 21.11.2017.

Ember 2017. What is Ember? Luettavissa: https://guides.emberjs.com/v2.16.0/. Luettu: 22.11.2017.

ESLint 2017. About. Luettavissa: https://eslint.org/docs/about/. Luettu: 3.12.2017.

Facebook 2017a. Introducing JSX. Luettavissa: https://facebook.github.io/react/docs/introducing-jsx.html. Luettu: 16.4.2017.

Facebook 2017b. React. Luettavissa: https://facebook.github.io/react/. Luettu: 15.4.2017.

Feldman, R., Bagnardi, F., Hojberg, S. & Hall, J. 2016. Developing a React Edge, Second Edition. Bleeding Edge Press.

Foundation 2017a. Easy-to-use HTML, CSS and Javascript to help create all kinds of awesome. Luettavissa: https://foundation.zurb.com/showcase/about.html. Luettu: 28.11.2017.

Foundation 2017b. Foundation. Luettavissa: https://foundation.zurb.com/. Luettu: 28.11.2017.

Foundation 2017c. Responsive design gets a whole lot faster. Luettavissa: https://foundation.zurb.com/. Luettu: 28.11.2017.

Gackenheimer, C. 2015. Introducing to React. Apress. New York.

Git 2017a. Small and Fast. Luettavissa: https://git-scm.com/about/small-and-fast. Luettu: 16.12.2017.

Git 2017b. Branching and Merging. Luettavissa: https://git-scm.com/about/branching-and-merging. Luettu: 16.12.2017.

GitHub 2017a. Ember.js. Luettavissa: https://github.com/emberjs/ember.js. Luettu: 22.11.2017.

GitHub 2017b. Introduction. Luettavissa: https://github.com/browserify/browserify-handbook. Luettu: 13.12.2017.

GitHub 2017c. Introduction. Luettavissa: https://github.com/webpack/webpack. Luettu: 12.12.2017.

Gonzalez, A. 2016. Comparing the 4 Most Popular Client-Side Javascript Frameworks. Luettavissa: https://blog.appdynamics.com/product/comparing-the-4-most-popular-client-side-javascript-frameworks/. Luettu: 29.3.2017.

Google 2018. Progressive Web Apps. Luettavissa: https://developers.google.com/web/progressive-web-apps/. Luettu: 5.1.2018.

Grunt 2017. Why use a task runner? Luettavissa: https://gruntjs.com/. Luettu: 14.12.2017.

Gupta, R., Prajapati, H. & Singh, H. 2015. Test-Driven JavaScript Development. Packt Publishing. Birmingham.

Haney, D. 2015. The Modern JavaScript Developer’s Toolbox. Luettavissa: https://www.infoq.com/articles/modern-javascript-toolbox. Luettu: 8.5.2017.

Hauser, D. 2016. Test-Driven iOS Development with Swift. Packt Publishing. Birmingham.

Haviv, A. 2016. MEAN Web Development – Second Edition. Packt Publishing. Birmingham.

Hirsijärvi, S., Remes, P. & Sajavaara, P. 2009. Tutki ja kirjoita. Tammi. Helsinki.

Horton, A. & Vice, R. 2016. Mastering React. Packt Publishing. Birmingham.

Jain, N. 2014. Review of different responsive CSS front-end frameworks. Journal of Gloval Research in computer science, 5, 11, s. 5.

Jakobus, B. & Marah, J. 2016. Mastering Bootstrap 4. Packt Publishing. Birmingham.

Jest 2017a. Jest. Luettavissa: http://facebook.github.io/jest/. Luettu: 7.12.2017.

Jest 2017b. Testing Web Frameworks. Luettavissa: http://facebook.github.io/jest/docs/en/testing-frameworks.html#content. Luettu: 7.12.2017.

Jones, D. 2017. JavaScript: Novice to Ninja, 2nd Edition. SitePoint. Melbourne.

jQuery 2017. What is jQuery? Luettavissa: https://jquery.com/. Luettu: 28.11.2017.

JSHint 2017. JSHint, A Static Code Analysis Tool for JavaScript. Luettavissa: http://jshint.com/about/. Luettu: 3.12.2017.

Jspm 2018. Frictionless browser package manager. Luettavissa: https://jspm.io/. Luettu: 6.1.2018.

Järvinen, J. 2014. TypeScript on olioystävällinen. Tietokone, s. 48.

Kananen, J. 2014. Laadullinen tutkimus opinnäytetyönä: miten kirjoitan kvalitatiivisen opinnäytetyön vaihe vaiheelta. Jyväskylän ammattikorkeakoulu. Jyväskylä.

Kasagoni, S. 2017. Building Modern Web Applications Using Angular. Packt Publishing. Birmingham.

Knockout 2017a. Knockout. Luettavissa: http://knockoutjs.com/index.html. Luettu: 23.11.2017.

Knockout 2017b. Introduction. Luettavissa: http://knockoutjs.com/documentation/introduction.html. Luettu: 23.11.2017.

Kotilainen, S. 2016. Koodiseppä valitsee työkalunsa. Tietoviikko, joulukuu, s. 49.

Krill, P. 2014. React: Making faster, smoother Uis for data-driven Web apps. Luettavissa: http://www.infoworld.com/article/2608181/javascript/react--making-faster--smoother-uis-for-data-driven-web-apps.html. Luettu: 14.4.2017.

Kumar, S. 2015. Difference Between Library and Framework. Luettavissa: http://www.c-sharpcorner.com/UploadFile/a85b23/framework-vs-library/. Luettu: 29.3.2017.

Kyrnin, J. 2016. How to Choose the Right CSS Framework for Your Website. Luettavissa: http://www.informit.com/articles/article.aspx?p=2464969. Luettu: 16.4.2017.

Kyrnin, J. 2015. Sams Teach Yourself Bootstrap in 24 Hours. Sams. Indianapolis.

Lamia 30.11.2017. PWA-natiivisovelluksen hyödyt selaimeen. Lamian blogi. Luettavissa: https://lamia.fi/blog/pwa. Luettu: 6.1.2018.

Laster, B. 2016. Professional Git. John Wiley & Sons.

Less 2017a. Getting Started. Luettavissa: http://lesscss.org. Luettu: 11.12.2017.

Less 2017b. History. Luettavissa: http://lesscss.org/about/. Luettu: 11.12.2017.

Material Design 2017a. Enviroment. Luettavissa: https://material.io/guidelines/material-design/environment.html#. Luettu: 28.11.2017.

Material Design 2017b. Introduction. Luettavissa: https://material.io/guidelines/material-design/introduction.html#. Luettu: 28.11.2017.

Maynard, T. 2017. Getting Started with Gulp – Second Edition. Packt Publishing. Birmingham.

McKenzie, S., Nakazawa, C. & Kyle, J. 2016. Yarn: A new package manager for JavaScript. Luettavissa: https://code.facebook.com/posts/1840075619545360. Luettu: 1.11.2017.

Mercurial 2017. Mercurial source control management. Luettavissa: https://www.mercurial-scm.org/about. Luettu: 17.12.2017.

Mew, K. 2015. Learning Material Design. Packt Publishing. Birmingham.

Mikkonen, J. 2016a. Pitkä odotus päättyi – Googlen Angular 2 on julkaistu. Luettavissa: http://www.tivi.fi/Vinkit/pitka-odotus-paattyi-googlen-angular-2-on-julkaistu-6583112. Luettu: 9.11.2017.

Mikkonen, J. 2016b. Facebook teki javascript-paketinhallinnastaan avointa koodia. Luettavissa: http://www.tivi.fi/Vinkit/facebook-teki-javascript-paketinhallinnastaan-avointa-koodia-6589838. Luettu: 1.11.2017.

Mikkonen, J. & Nummi, N. 2016a. Javascript-kehittäjä valitsee työkalupakin. Tietoviikko, maaliskuu, s. 44-46.

Mikkonen, J. & Nummi, N. 2016b. Koodiseppä valitsee työkalunsa. Tietoviikko, joulukuu, s. 50-52.

Minar, I. 19.7.2012. MVC vs MVVM vs MVP. Google Plus -päivitys. Luettavissa: https://plus.google.com/+IgorMinar/posts/DRUAkZmXjNV. Luettu: 23.11.2017.

Mocha 2017. Mocha. Luettavissa: http://mochajs.org/. Luettu: 7.12.2017.

Nielsen, J. 2012. Usability 101: Introduction to Usability. Luettavissa: https://www.nngroup.com/articles/usability-101-introduction-to-usability/. Luettu: 21.3.2017.

Nolan, A. 29.11.2016. The State of Front-End Tooling – Results. Blog. Luettavissa: https://ashleynolan.co.uk/blog/frontend-tooling-survey-2016-results. Luettu: 26.01.2018.

Npm 2017. Npm. Luettavissa: https://www.npmjs.com/. Luettu: 28.4.2017.

Nummi, N. & Mikkonen, J. 2016. Kumman valitset, kehitysympäristön vai tekstieditorin? Luettavissa: http://www.mikrobitti.fi/2016/11/kumman-valitset-kehitysymparisto-vai-tekstieditorin-ohjelmointi-koodi/. Luettu: 22.3.2016.

Pajunen, J. 27.12.2013. Automatisoidut testit vs. kliksuttelu. Fraktion blogi. Luettavissa: https://fraktio.fi/blogi/automatisoidut-testit-vs-kliksuttelu/. Luettu: 7.12.2017.

PhantomJS 2017a. PhantomJS. Luettavissa: http://phantomjs.org/. Luettu: 8.12.2017.

PhantomJS 2017b. Headless Testing. Luettavissa: http://phantomjs.org/headless-testing.html. Luettu: 8.12.2017.

Polymer Project 2017a. Polymer library. Luettavissa: https://www.polymer-project.org/. Luettu: 22.11.2017.

Polymer Project 2017b. Who we are. Luettavissa: https://www.polymer-project.org/about. Luettu: 22.11.2017.

Pop, A. 2015. Learning Underscore.js. Packt Publishing. Birmingham.

Porcello, E. & Banks, A. 2017. Learning React, 1st. Edition. O'Reilly Media. Sebastopol.

Prabhu, A. 2015. Beginning CSS Preprocessors: With SASS, Compass.js, and Less.js. Apress. New York.

Pääkkö, T. 21.3.2017. Tyylittely Reactissa on hankalaa. Symbion blogi. Luettavissa: https://www.symbio.com/fi/tyylittely-reactissa-hankalaa. Luettu: 28.12.2017.

Reyna, M. 2015. Meteor Design Patterns. Packt Publishing. Birmingham.

Reynolds, D. 2016. Learning Grunt. Packt Publishing. Birmingham.

RhodeCode 2016. Version Control Systems Popularity in 2016. Luettavissa: https://rhodecode.com/insights/version-control-systems-2016. Luettu: 26.1.2018.

Rossel, S. 2017. Continuous Integration, Delivery, and Deployment. Packt Publishing. Birmingham.

Rozema, M. 2015. IDE vs. Text Editor: Choosing the Right Tool at the Right Time. Luettavissa: https://spin.atomicobject.com/2015/12/22/ide-vs-text-editor/. Luettu: 23.3.2017.

Sass 2017a. Sass (Syntactically Awesome StyleSheets). Luettavissa: http://sass-lang.com/documentation/file.SASS_REFERENCE.html#Syntax. Luettu: 11.12.2017.

Sass 2017b. Syntax. Luettavissa: http://sass-lang.com/documentation/file.SASS_REFERENCE.html#Syntax. Luettu: 11.12.2017.

Seitz, A., Rometty, J., Brown, M., Jelisejevs, P., Green, D., Buckler, C., Mardan, A. & Kolce, J. 2017. Modern JavaScript. SitePoint. Melbourne.

Selenium 2017. Introduction. Luettavissa: http://www.seleniumhq.org/docs/01_introducing_selenium.jsp. Luettu: 7.12.2017.

Sheppard, D., Miller, C. & Liptak A. 2015. AngularJS for .NET Developers in 24 Hours. Sams. Indianapolis.

Simpson, K. 2015. You Don’t Know JS: Up & Going. O’Reilly Media. Sebastopol.

Smashing Magazine 2015. Practical Approaches For Designing Accessible Websites. Smashing Magazine. Freiburg.

Stack Overflow 2017a. Stack Overflow Annual Developer Survey. Luettavissa: https://insights.stackoverflow.com/survey. Luettu: 26.01.2018.

Stack Overflow 2017b. Most Loved, Dreaded, and Wanted Frameworks, Libraries and Other Technologies. Luettavissa: https://insights.stackoverflow.com/survey/2017#technology-most-loved-dreaded-and-wanted-frameworks-libraries-and-other-technologies. Luettu: 27.01.2018.

Stack Overflow 2017c. Frameworks, Libraries, and Other Technologies. Luettavissa: https://insights.stackoverflow.com/survey/2017#technology-frameworks-libraries-and-other-technologies. Luettu: 27.01.2018.

Stack Overflow 2017d. Most Popular Developer Enviroments by Occupation. Luettavissa: https://insights.stackoverflow.com/survey/2017#technology-most-popular-developer-environments-by-occupation. Luettu: 27.01.2018.

Stangarone, J. 1.9.2015. Web Development: What's changed and where is it going? Mrc's Cup of Joe Blog. Luettavissa: https://www.mrc-productivity.com/blog/2015/09/web-development-whats-changed-and-where-is-it-going/. Luettu: 29.1.2018.

The State of JavaScript 2017a. The JavaScript world is richer and messier than ever. Luettavissa: https://stateofjs.com/. Luettu: 26.01.2018.

The State of JavaScript 2017b. Front-End Frameworks – Results. Luettavissa: https://stateofjs.com/2017/front-end/results. Luettu: 26.01.2018.

The State of JavaScript 2017c. JavaScript Flavors – Results. Luettavissa: https://stateofjs.com/2017/flavors/results. Luettu: 26.01.2018.

The State of JavaScript 2017d. Testing Tools – Results. Luettavissa: https://stateofjs.com/2017/testing/results. Luettu 26.01.2018.

The State of JavaScript 2017e. CSS & Styling – Results. Luettavissa: https://stateofjs.com/2017/css/results. Luettu: 27.01.2018.

The State of JavaScript 2017f. Build Tools – Results. Luettavissa: https://stateofjs.com/2017/build-tools/results. Luettu: 27.01.2018.

The State of JavaScript 2017g. Text Editors. Luettavissa: https://stateofjs.com/2017/other-tools/. Luettu: 26.01.2018.

Stefanov, S. 2016. React: Up & Running. O'Reilly Media. Sebastopol.

Styled-Components 2017a. Getting Started. Luettavissa: https://www.styled-components.com/docs/basics#getting-started. Luettu: 12.12.2017.

Styled-Components 2017b. Styled-Components. Luettavissa: https://www.styled-components.com/. Luettu: 12.12.2017.

Stylus 2017. Try Stylus online! Luettavissa: http://stylus-lang.com/try.html#. Luettu: 11.12.2017.

Sublime Text 2017. Buy. Luettavissa: https://www.sublimetext.com/buy?v=3. Luettu: 25.3.2017.

Taylor, K. & Smith, B. 2015. Getting a Web Development Job For Dummies. John Wiley & Sons. New York.

Thomas, S. 2015. Data Visualization with JavaScript. No Starch Press. San Francisco.

TSLint 2017. TSLint. Luettavissa: https://palantir.github.io/tslint/. Luettu: 3.12.2017.

Tuomi, J. & Sarajärvi, A. 2009. Laadullinen tutkimus ja sisällönanalyysi. Tammi. Helsinki.

Tuomi, J. & Sarajärvi, A. 2018. Laadullinen tutkimus ja sisällönanalyysi. Tammi. Helsinki.

Turner, W. & Leonard, S. 2017. JavaScript for Sound Artists. Focal Press. Boca Raton.

TENK 2012. Hyvä tieteellinen käytäntö ja sen loukkausepäilyjen käsitteleminen Suomessa. Luettavissa: http://www.tenk.fi/sites/tenk.fi/files/HTK_ohje_2012.pdf. Luettu 3.2.2018.

Uotila, M. 13.4.2017. Progressive Web Appit haastavat jo mobiilisovellukset. Qvikin blogi. Luettavissa: https://qvik.com/news/progressive-web-app/. Luettu: 6.1.2018.

Visual Studio Code 2017. Visual Studio Code. Luettavissa: https://code.visualstudio.com/. Luettu: 26.3.2017.

Vue.js 2017a. The Progressive JavaScript Framework. Luettavissa: https://vuejs.org/. Luettu: 27.11.2017.

Vue.js 2017b. What is Vue.js? Luettavissa: https://vuejs.org/v2/guide/. Luettu: 27.11.2017.

Vuokola, J. 2014. Kaikki tekee ux:ia. Tietoviikko, 3, s. 7.

Vänskä, O. 2016. Javascript-kehittäjä valitsee työkalupakin. Tietoviikko, maaliskuu, s. 40.

Vänttinen, R. 7.3.2017. Viisi syytä, miksi sinunkin verkkopalvelusi tulisi pohjautua Reactiin. Nord Softwaren blogi. Luettavissa: https://www.nordsoftware.com/viisi-syyta-miksi-sinunkin-verkkopalvelusi-tulisi-pohjautua-reactiin/. Luettu: 6.1.2018.

Wagner, J. 2016. Web Performance in Action: Building Fast Web Pages. Manning Publications. Shelter Island.

Waikar, M. 2015. Data-oriented Development with AngularJS. Packt Publishing. Birmingham.

W3C 2017. W3C Candidate Recommendation. Luettavissa: https://www.w3.org/TR/css-flexbox-1/. Luettu: 30.11.2017.

Westhuizen, P. 2016. Bootstrap for ASP.NET MVC – Second Edition. Packt Publishing. Birmingham.

Weyl, E. 2017. Flexbox in CSS. O’Reilly Media. Sebastopol.

Wilcox, M. 16.5.2017. Angular vs React: Battle for the future of front-end web development? Developer Economics Blog. Luettavissa: https://www.developereconomics.com/angular-react-front-end-web-development. Luettu: 6.1.2018.

Williamson, K. 2015. Learning AngularJS. O’Reilly Media. Sebastopol.

Yhteiskuntatieteellinen tietoarkisto 2017. Litterointi. Luettavissa: http://www.fsd.uta.fi/aineistonhallinta/fi/kvalitatiivisen-datan-kasittely.html#litterointi. Luettu: 11.1.2018.

You, E. 11.2.2014. First week of launching Vue.js. Evan You Blog. Luettavissa: http://blog.evanyou.me/2014/02/11/first-week-of-launching-an-oss-project/. Luettu: 27.11.2017.

Zea, R. 2015. Mastering Responsive Web Design with HTML5 and CSS3. Packt Publishing. Birmingham.

Liitteet

Liite 1. Lista pois rajatuista työkaluista

- BrowserStack
- BrowserSync
- BitBucket
- Chart.js
- Chokidar
- Create React App
- Docker
- Dygraphs
- Emacs
- GitHub
- GitLab
- Handlebars
- Illustrator
- Ionic
- Jenkins
- JetBrains IDE:t
- Makefile
- Nano
- Ngrok
- Odoo
- Photoshop
- PHP Storm
- React Native
- Reactin selaimeen asennettava työkalupakki
- Redux
- Selaimen kehitystyökalut
- Seo Site Checkup
- Serverless Framework
- Tampermonkey
- Uglify
- Vim
- Visual Studio
- Visual Team Services
- W3 Validator

Liite 2. Saatekirje

Hei [etunimi]!

Olen Kristina Wiik, Haaga-Helia ammattikorkeakoulun tietojenkäsittelyn koulutusohjelman viimeisen vuoden opiskelija.

Kutsun teidät osallistumaan opinnäytetyöni tutkimukseen, josta saatte yhden elokuvalipun Finnkinon teatteriin.

Opinnäytetyöni tutkimuksen tarkoituksena on selvittää, mitä front end -työkaluja ohjelmistoyritykset käyttävät ja miksi, sekä vastauksien pohjalta koota suositeltava front end -työkalupakki ohjelmoijille. Tutkimuksesta hyötyvät erityisesti alaa opiskelevat ja alalta vastavalmistuneet, sillä he voivat harjoitella tutkimuksen työkaluja tulevaisuuden työtä varten. Lisäksi, tutkimustuloksista hyötyvät alan opettajat sekä ohjelmistoyritykset.

Tutkimuksessa on tarkoitus haastatella ohjelmistoyritysten edustajia, jotka vastaavat front end -työkalujen käytöstä.

Toteutan tutkimuksen teemahaastatteluna kesä - elokuun 2017 aikana. Haastattelu kestää noin 1 - 1,5 tuntia ja se pidetään teille mieluisassa paikassa. Haastattelu voidaan järjestää myös esimerkiksi Skypen välityksellä. Tulen nauhoittamaan haastattelun analysointia varten ja käyttämään sitä vain tutkimustarkoituksiin. Raportoin haastattelussa esille tulleet asiat tutkimusjulkaisuun tavalla, jossa yritystä eikä haastateltavaa pysty tunnistamaan. Tutkimuksen julkaisussa voidaan sisällyttää suoria lainauksia haastatteluista. Tutkimuksen loputtua kaikki tutkimusmateriaali hävitetään asianmukaisesti. Voin myös ilomielin lähettää opinnäytetyöni teille sen valmistuttua!

Mikäli haluatte osallistua tutkimukseen, voitte olla minuun yhteydessä sähköpostitse tai puhelimitse. Annan mielelläni lisätietoja ja vastaan askarruttaviin kysymyksiin.

Liitteenä löytyy haastattelurunko, jota tulen käyttämään haastattelussa.

Ystävällisin terveisin ja avusta kiittäen,

Kristina Wiik
Haaga-Helia amk, Pasila
Tietojenkäsittelyn ko.
Puh. *** *** ****
S-posti. **********
Kotisivut: **********

Liite 3. Suostumuslomake

KIRJALLINEN SUOSTUMUS TUTKIMUKSEEN OSALLISTUMISESTA

Opinnäytetyön tutkimuksen tarkoituksena on selvittää, mitä front end -työkaluja ohjelmistoyritykset käyttävät ja miksi, sekä vastauksien pohjalta koota suositeltava front end -työkalupakki ohjelmoijille.

Haastattelu nauhoitetaan analysointia varten ja käytetään vain tutkimustarkoituksiin. Haastattelussa esille tulleet asiat raportoidaan tutkimusjulkaisuun tavalla, jossa yritystä eikä haastateltavaa pysty tunnistamaan. Tutkimusjulkaisuun voidaan sisällyttää suoria lainauksia haastatteluista. Tutkimuksen loputtua kaikki tutkimusmateriaali hävitetään asianmukaisesti.

Tutkimukseen osallistuminen on vapaaehtoista ja haastateltava voi keskeyttää haastattelun milloin tahansa.

Annan suostumukseni tutkimuksen tekemiseen ja haastattelun nauhoittamiseen:

____
Tutkimukseen osallistuvan allekirjoitus

____
Nimenselvennys

____
Päiväys

Kristina Wiik
Haaga-Helia amk, Pasila
Tietojenkäsittelyn ko.
Puh. *** *** ****
S-posti. **********
Kotisivut: **********

Liite 4. Teemahaastattelurunko

TEEMAHAASTATTELUN RUNKO

FRONT END -TYÖKALUT

Teema 1. Yrityksen taustatiedot

- Yrityksen toimiala
- Perustamisvuosi
- Liikevaihto
- Henkilöstön lukumäärä

Teema 2. Haastateltavan tausta ja kokemus alalta

- Asema yrityksessä
- Kokemus
- Osaaminen

Teema 3. Käytössä olevat front end -kehitystyökalut

- Koodin kirjoitusvälineet
- Sovelluskehykset ja kirjastot *(engl. frameworks, libraries)*
  - JavaScript *(esim. React, Angular)*
  - CSS ja HTML *(esim. Bootstrap)*
- Koodin esikäsittely ja kääntäminen *(engl. preprocessors, transpilers)*
  - JavaScript (*esim. ES6, CoffeeScript, TypeScript)*
  - CSS *(esim. Sass, Less)*
  - HTML *(esim. Haml)*
- Koodin tarkistaminen *(engl. linters)*
  - JavaScript
  - CSS
  - HTML
- Paketinhallinta *(engl. package managers)*
- Tehtävänsuoritus *(engl. task managers)*
- Versionhallinta *(engl. version control)*
- Muita front end -työkaluja

Teema 4. Työkalujen käyttö ja tulevaisuus

- Työkalujen arvioitu käyttöikä
- Työkalut tulevaisuudessa

Teema 5.* Mitä suosittelet opettelemaan tulevaisuuden työtä ajatellen?

*Lisätty jälkeenpäin

Liite 5. Työkalujen käytön syyt kvantifioituna

Työkalu	Aika	Hinta	Ominaispiirteet	Optimoiminen	Projektin lähtökohdat	Suositus	Tarpeettomuus	Tiimityöskentely	Toimivuuden varmistus	Toteutukseen liittyvät ratkaisut	Tunnepohjaiset kokemukset	Uutuus	Verkosto
Atom		2	9								1		
Sublime Text			5								1		
VSC			4								2		
TypeScript			5			1					1		
ES6			2		1							1	
CoffeeScript											2		1
Babel					1						2		
Sass			3		1						1		
Less					2						1		
Stylus			1		1								
PostCSS, CSS Modules, Styled Components	3		1										
Pelkkä CSS	1						1						
ESLint			4	1									1
TSLint			3					1					1
Npm	1		5		1								
Yarn	1		6	3									
Bower	1				2								
React			6		3						5		5
AngularJS					3								3
Angular					5						1		
Muut*					2					1			
Bootstrap			3		3						3		3
Foundation			1		1						1		
Material Design	1				1						1		
Webpack			1		1								
Grunt					1								
Npm			1										
Browserify					1								
Gulp			3		1						2		1
Git											3		6
Mercurial					1								
Jest, Enzyme									2				
Mocha, Selenium, PhantomJS					1								
Yht.	**8**	**2**	**63**	**4**	**33**	**1**	**1**	**1**	**2**	**1**	**27**	**1**	**21**

* Muut JavaScript-kirjastot ja -sovelluskehykset

Liite 6. Työkalupakki ja työkalujen väliset suhteet