Riia Tarvainen

JAVASCRIPT JA TAKAISINKUTSUT SUOMALAISILLA VERKKOSIVUSTOILLA

Takaisinkutsumenetelmän yleisyys ja ongelmat

JYVÄSKYLÄN YLIOPISTO
TIETOJENKÄSITTELYTIETEIDEN LAITOS
2016

TIIVISTELMÄ

Tarvainen, Riia
JavaScript ja takaisinkutsut suomalaisilla verkkosivuilla - Takaisinkutsumenetelmän yleisyys ja ongelmat
Jyväskylä: Jyväskylän yliopisto, 2016, 84 s.
Tietojenkäsittelytiede, pro gradu –tutkielma
Ohjaaja: Sakkinen, Markku

JavaScript on hyvin paljon käytetty ohjelmointikieli ja verkkosivuilla vallitsevassa asemassa. JavaScript-kieliset ohjelmat sisältävät kuitenkin paljon virheitä. Verkkokeskusteluissa ”takaisinkutsuhelvetti”, eli tilanne jossa ohjelmakoodi on monien sisäkkäisten takaisinkutsullisten funktioiden vuoksi hyvin vaikesti ymmärrettävä ja hallittava, nostetaan yhdeksi kielen kipupisteistä. Tässä tutkimuksessa tutkittiin kirjallisuudesta JavaScriptin takaisinkutsumenetelmään liitettyjä ongelmia sekä sitä korvaavia asynkronisen toiminnan menetelmiä. Empiirisessä tutkimuksessa kartoitettiin suomalaisten verkkosivujen JavaScript-ohjelmakoodia. Ohjelmakoodin määriä, sijaintia ja ominaisuuksia mitattiin. Erityisesti tutkittiin takaisinkutsumenetelmän hyödyntämistä ja siihen liittyvien sisäkkäisyyksien syvyyksiä. Tutkimusta varten kehitettiin mittausohjelmisto, jonka avulla tutkimus suoritettiin. Tiedostot haettiin sivustoittain, tiedot mitattiin ja havainnot kirjattiin havaintomatriisiin. Tutkimukseen valittiin suomalaisia verkkopalvelun sisältäviä verkkosivustoja jaoteltuina tarkoituksen mukaan viiteen ryhmään ja tutkittuja sivustoja oli yhteensä 134. Tutkimus osoitti, että suomalaisten verkkosivujen sisältämän JavaScript-ohjelmakoodin ominaisuudet vaihtelevat sivuston tarkoituksen mukaan. Tutkimuksessa havaittiin myös, että sivustoilla hyödynnetään erittäin paljon sekä yleensä HTML-tiedostoihin sisällytettyä JavaScriptiä että asynkronisen toiminnan menetelmää, jossa HTML-elementtien attribuuteiksi asetetaan JavaScriptiä. Tutkimusaineistosta oli löydettävissä monitasoisia sisäkkäisyyksiä ja takaisinkutsumenetelmän hyödyntäminen muutenkin oli yleistä. Sen sijaan tutkitut takaisinkutsuja korvaavat asynkronisen toiminnan menetelmät olivat aineistossa vähäisiä. Tästä huolimatta tutkimuksessa havaittiin sivustoryhmittäin tarkasteltuna viitteitä riippuvuudesta lupausabstraktion löytymisen ja vähäisemmän sisäkkäisyyden välillä. Tuloksia vertailtiin julkaistuun tutkimukseen, jossa joitakin samoja asioita oli tutkittu avoimen lähdekoodin projekteista. Julkaistun tutkimuksen tulosten havaittiin olevan osin kyseenalaisia ja jopa virheellisiä. Vertailukelpoisten tietojen osalta tulokset olivat varsin yhteneviä.

Asiasanat: JavaScript, takaisinkutsu, asynkronisuus, takaisinkutsuhelvetti, verkkopalvelut

ABSTRACT

Tarvainen, Riia
JavaScript and callbacks on Finnish websites - Prevalence and disadvantages of the callback mechanism
Jyväskylä: University of Jyväskylä, 2016, 84 p.
Computer science, Master's thesis
Supervisor: Sakkinen, Markku

JavaScript is a very widely used programming language. It is used in almost all websites and all modern web browsers support it. However, applications developed with JavaScript contain a lot of errors. Callback functions have been generally considered to cause errors. Callback hell is a situation where there are deeply nested, usually anonymous and asynchronously called, callback functions. This makes the program code hard to understand and to maintain. In this study we performed a literature review of problems related to JavaScript callback mechanisms and also some surrogate asynchronous methods. An empirical study was performed to analyze JavaScript code used in Finnish websites. Code characteristics, usage of callback mechanism and depth of nested functions were measured. A measurement tool was created to collect measurements from selected web sites. For each site all necessary files were first downloaded, then analyzed, and the results were saved to a data matrix. The total number of analyzed web sites was 134 in five different categories. Our research indicates that the style of JavaScript code varies between the categories. We also found that many web sites embed JavaScript directly in HTML and also within HTML attributes. Frequent usage of callback functions was found from the collected data set as well as deep nesting of these functions. Only few surrogate methods were used. Despite this, we found indication of correlation between use of the promise abstraction and shallower levels of nesting. Results were also compared with a published research where same things were analyzed from Open Source -projects. We found that some of their results were questionable or even incorrect. For comparable parts the results were similar.

Keywords: JavaScript, callback, asynchrony, callback-hell, web-based services

ESIPUHE

Käsillä oleva työ on toteutettu keskellä kiireisiä ruuhkavuosia, eikä se olisi ollut mahdollista ilman sitä kaikkea upeaa tukea jota minulle ja perheelleni on suotu. Olen erityisesti kiitollinen äidilleni Ritva Sinisalolle, jonka apu on ollut työn edistymisen kannalta korvaamatonta. Olen kiitollinen myös kaikille teille muille läheisille jotka olette suurilla tai pienillä teoilla perhe-elämäämme helpottaneet. Kiitokset Miljalle, Alissalle ja Niilakselle, jotka ovat joutuneet sietämään ajoittain täysin ajatuksiinsa vajonnutta, hajamielistä tai villisti tietokonetta naputtavaa äitiä.

Erityiset kiitokset professori Markku Sakkiselle työn kärsivällisestä, tarkkanäköisestä ja loputtoman ystävällisestä ohjauksesta. Olen hyvin onnellinen, että työtäni ohjasi hänen kaltaisensa viisas, rehellinen ja toisista hyvää etsivä ammattilainen.

Kiitos Tero Tarvaiselle siitä kumppanuudesta ja rakkaudesta, jonka kannattelemana työ oli mahdollista toteuttaa perhe-elämämme keskellä. Lisäksi kiitän häntä siitä viisaudesta jolla hän osasi olla puuttumatta opiskelutyöhöni, kuitenkin valaen minuun uskoa silloin kun sitä tuntui puuttuvan.

KUVIOT

TAULUKOT

LISTAUKSET

SISÄLLYS

1 JOHDANTO

JavaScriptin kehitys hallitsevaksi ohjelmointikieleksi on aikamme menestystarina. Se kehitettiin Netscape-yrityksessä 90-luvulla nopeasti, tavoitteena yksinkertainen ja helposti opittava kieli pieniä verkkosivujen ulkoasuja parantavia toiminnallisuuksia varten (Severance, 2012). JavaScriptillä on moninainen historia. Vuosien ajan sitä pidettiin huonona ja epäturvallisena kielenä ja sen käyttöä neuvottiin välttämään. JavaScript saavutti kuitenkin nopeasti suosiota, mikä johti sen standardisoimiseen; standardi sai nimekseen ECMAscript (ECMA, 2015). ECMAscript-standardiin pohjautuu nykyisin muitakin toteutuksia JavaScript-kielen lisäksi (Herman, 2012). JavaScript on kuitenkin selkeästi tunnetuin toteutus tälle standardille.

JavaScript-kielen alkuperäinen toimintaympäristö on verkkoselainten sisällä, tarkoituksena lisätä interaktiivisuutta verkkosivuihin ja vähentää ylimääräisiä palvelinkutsuja. Nykyisin sen asema verkkosivuilla onkin vertaansa vailla. JavaScript löytyy kaikista merkittävistä verkkoselaimista, joissa se mahdollistaa nykyaikaisten verkkosovellusten toteuttamisen (Haverbeke, 2014). Sen ja muiden selaintekniikoiden kehitys on mahdollistanut siirtymisen perinteisistä erikseen jokaiselle tietokoneelle asennettavista työpöytäsovelluksista verkossa sijaitseviin selaimella toimiviin sovelluksiin. Nykyisin JavaScript on kasvavassa määrin myös palvelimissa, jossa se toimii esim. Node.js -ympäristön sisällä (Syed, 2014). Node.js ympäristöä on hyödynnetty jo monissa kriittisissäkin järjestelmissä ja pian se löytyy myös satelliitista (Reaktor, 2016).

JavaScript-kielen aktiivinen kehitystyö on tuottanut tulosta ja se onkin tällä hetkellä erittäin suosittu ja ehkä jopa yleisin ohjelmointikieli. Stack Overflown laaja kysely osoitti sen olevan heidän yhteisönsä suosituin kieli (StackOverflow, 2015). JavaScript-ohjelmakoodin on todettu olevan erittäin yleistä verkkosivuilla. Vuonna 2013 tehdyssä tutkimuksessa (Yue & Wang, 2013) 96.9 % sivustoista sisälsi JavaScript-ohjelmakoodia. Valitettavasti JavaScript-ohjelmat sisältävät paljon virheitä (Ocariza, Pattabiraman & Zorn, 2011; Ocariza, Bajaj, Pattabiraman & Mesbah, 2013; Fard & Mesbah, 2013). Virheet voivat olla hankalia, sillä ohjelma voi toimia kehitysympäristössä ja testiympäristöissä aivan halutulla tavalla, mut-

ta aiheuttaa ongelmia loppukäyttäjille koska esimerkiksi laitteiston suorituskyky tai verkon nopeus poikkeavat merkittävästi (Parker, 2015). Virheiden syitä voi olla monia ja kielessä on ehkä perusrakenteena piirteitä jotka altistavat virheille kuten heikko tyypitys. Verkkokeskusteluissa kuitenkin nostetaan esille käsite takaisinkutsuhelvetistä, joka osaltaan vaikeuttaisi JavaScript-ohjelmien hallintaa ja sen myötä edesauttaisi virheiden syntymistä. Takaisinkutsuhelvetti on tunnettu käsite JavaScript-ohjelmoijien keskuudessa (Boduch, 2015). Takaisinkutsumenetelmä tarkoittaa tapaa, jossa funktiokutsulle annetaan parametrina funktio, joka tulee kutsutuksi myöhemmin. Takaisinkutsuhelvetillä tarkoitetaan erilaisia asioita eri yhteyksissä, mutta useimmiten tarkoitetaan ohjelmakoodia joka on monien sisäkkäisten takaisinkutsullisten funktioiden vuoksi hyvin vaikesti ymmärrettävä ja hallittava.

JavaScript herätti kiinnostukseni palatessani vajaan vuosikymmenen mittaiseksi venähtäneeltä perhevapaalta. Lähtiessäni kieli oli varsin erilaisessa asemassa kuin mitä se on nyt. Vaikka en tästä kehityksestä täysin tietämätön ollutkaan, niin kielen kehityskulku viehätti epätavallisuudellaan. Tehtyäni muutamia ohjelmointikokeiluja, kävi verkkokeskusteluista tunnettu takaisinkutsuhelvetti nopeasti tutuksi ja pro gradu -työn aihepiiri alkoi hahmottua. Empiirisen tutkimuksen idea tuli innostavasta tutkimuksesta ”Don’t Call Us, We’ll Call You: Characterizing Callbacks in JavaScript” (Gallaba, Mesbah & Beschastnikh, 2015). Siinä on tutkittu takaisinkutsumenetelmän käyttöä joidenkin avoimen lähdekoodin -projektien JavaScript-tiedostoissa. Heidän kehittämänsä työkalut olivat niin ikään vapaasti saatavilla. Näin sain idean tutkia näillä välineillä suomalaisia verkkosivustoja siltä osin kuin niitä on mahdollista ladata tutkittaviksi. Valitettavasti työ ei sujunut aivan näin jouhevasti, vaan työkalujen hyödyntämisessä oli ongelmia alusta asti ja lopulta niiden hyödyntäminen jäi virheiden vuoksi hyvin vähäiseksi. Toteutin korvaavat sovellusosat itse ja näin ajauduin toteuttamaan huomattavasti mittavamman ohjelmointikokonaisuuden kuin aluksi suunniteltiin.

Tutkimuksen kirjallisuuteen pohjautuvassa osassa tarkastellaan kielen samanaikaisuuden mallia ja sitä kuinka takaisinkutsumalli sisältyy hyvin perustavanlaatuisesti JavaScript-kieleen. Tutkimuksessa etsitään tietoa siitä millaisia ongelmia takaisinkutsumenetelmään on liitetty sekä millaisia korvaavia asynkronisen toiminnan menetelmiä on kehitetty. Tutkimuksessa todettiin, että takaisinkutsumenetelmään on liitetty lukuisia ongelmia, kuten luottamukseen, ohjelman ymmärtämiseen ja virheiden käsittelyyn liittyvät seikat. Korvaavista asynkronisen toiminnan välineistä käsiteltiin lupausabstraktiota sekä lyhyesti myös joitakin kirjastoja ja reaktiivista ohjelmointia. Lupausabstraktion todettiin auttavan joihinkin takaisinkutsumenetelmän aiheuttamiin ongelmiin.

Tutkimuksen empiirisessä osassa suoritettiin ohjelmallinen ja automaattinen tutkimus, jossa kartoitettiin verkkosivustojen sisältämiä JavaScript-ohjelmakoodiin liittyviä menetelmiä ja niiden yleisyyttä. Tutkimusta varten kehitettiin räätälöity mittausohjelmisto, jonka avulla sivustojen hakeminen, mittaaminen ja tulosten kirjaaminen suoritettiin. Tutkimukseen valittiin verkkopalvelun sisältäviä verkkosivuja luokiteltuna tarkoituksensa mukaan eri kategorioihin. Nämä ovat

finanssisektori, pikavipit, rahapelit, julkinen sektori ja verkkokaupat. Sivustoja tutkittiin yhteensä 134 kpl.

Tutkimuksen empiirisessä osassa todettiin, että JavaScript-koodia löytyy verkkosivustoilta myös HTML-tiedostoista merkittävissä määrin. Tapahtuma-attribuuttien käyttö on edelleen yleistä, joskin hyödyntämisessä on suuria eroja sivustojen kesken. Tapahtuma-attribuuteilla tarkoitetaan HTML-elementtien attribuuteiksi asetettavia asynkronisia toimintoja. Menetelmä ei ole enää suositeltava, sillä se sitoo esitystavan ja toiminnallisuuden vahvasti yhteen. Uusimmilla sivustoilla tämän tyylin käyttö onkin mahdollisesti vähentynyt. Tutkimus vahvistaa, että jQuery-kirjasto on hyvin paljon hyödynnetty kirjasto suomalaisilla verkkosivustoilla. Sen sijaan async.js-kirjaston käyttö oli tässä aineistossa hyvin vähäistä.

Tutkimuksessa todettiin myös, että anonyymeihin takaisinkutsullisiin funktiokutsuihin liittyvä sisäkkäisyys on keskimäärin kohtuullista, joskin sivustoilta on löydettävissä jopa sisäkkäisyystasolle kuusi meneviä ketjuja. Lupausabstraktiota käytettiin jossain määrin kolmasosassa sivustoja, mutta niiden hyödyntämisen määrät oli vielä vähäisiä. Tutkimuksessa kuitenkin havaittiin viitteitä siitä, että lupausabstraktio esiintyisi yhdessä matalampien sisäkkäisyyksien kanssa. On siten mahdollista, että juuri monimutkaisimpia toimintoja olisi jo toteutettu lupausabstraktiota hyödyntämällä ja perustoiminnallisuus edelleen perinteisellä tavalla. Tämän vahvistaminen vaatii kuitenkin jatkotutkimusta.

Luvut 2 ja 3 esittävät tutkimuksen kirjallisuusosuuden. Niissä käsitellään paljon takaisinkutsumenetelmää asynkroniselta kannalta. Empiirisessä osuudessa oli tarkoitus tutkia myös asynkronisten takaisinkutsullisten funktiokutsujen esiintyvyyttä. Tämä oli kuitenkin valitettavasti jätettävä pois apuvälineiden virheiden vuoksi. Luku 2 käsittelee takaisinkutsumenetelmää asynkronisen toiminnan välineenä ja kokoaa kirjallisuudesta löytyneet ongelmat joita takaisinkutsumenetelmään liitetään JavaScriptissä. Luku 3 esittelee muutamia menetelmiä, joilla takaisinkutsumenetelmän ongelmia on pyritty ratkaisemaan. Näistä tärkeimpänä käsitellään lupausabstraktiota, eli JavaScriptin Promise-mallia. Luvussa 4 esitellään empiirinen tutkimus, jossa kartoitettiin suomalaisten verkkosivujen JavaScript-koodin ominaisuuksia. Luku 5 esittelee saavutetut tulokset ja luvussa 6 pohditaan niiden merkityksiä sekä käsitellään tutkimusmenetelmän vaikutuksia niihin. Luku 7 on yhteenvetoluku. Tutkimuksen liitteenä on tutkimuksen aineistosta mitattujen muuttujien arvot sivustoittain.

2 TAKAISINKUTSUMENETELMÄ JA ASYNKRONINEN TOIMINTA

Tässä luvussa luodaan yleiskuva JavaScript-kielen takaisinkutsumenetelmästä ja asynkronisesta toiminnasta sekä siitä miksi takaisinkutsumenetelmä on tarpeen ja miksi sen voidaan olettaa aiheuttavan jossain tapauksissa ongelmia. Ensimmäisessä alaluvussa esitellään takaisinkutsumenetelmä ja sulkeuma-ominaisuus. Toinen alaluku kuvaa tietojärjestelmien samanaikaisuutta ja sitten poraudutaan kohti tätä JavaScript-kielen asynkronisen toiminnan perusrakennetta. Kolmas alaluku kokoaa kirjallisuudesta löydetyt ongelmat, joita JavaScript-kielen takaisinkutsumenetelmään on liitetty.

2.1 Takaisinkutsumenetelmä

Kaikenlaisten ohjelmistojen suunnittelussa eräs yleisperiaate on pyrkiä rakenteeseen jossa komponentit ovat sisäisesti yhtenäisiä, mutta niiden väliset riippuvuudet ovat mahdollisimman heikkoja (Koskimies & Mikkonen, 2005). Periaate on lähtöisin komponenttiajattelun alkuajoilta (Parnas, 1972; Dijkstra, 1976) ja on edelleen tärkeä ohjenuora pyrittäessä uudelleenkäytettäviin ja ylläpidettäviin järjestelmiin. Komponenttien välisten riippuvuuksien minimointi ei tule ohjelmistoihin itsestään ja sitä voidaankin pitää hyvänä tavoitteena arkkitehtuurisuunnittelulle. Ohjelmistojen kehittämistä varten on vuosien saatossa kehitetty ja löydetty paljon menetelmiä, joilla riippuvuuksia voidaan vähentää.

Takaisinkutsumenetelmä on eräs tapa, jossa komponentti voi hyödyntää toisen komponentin tarjoamia palveluita kuitenkaan tuntematta tätä ja siten olematta tästä suoraan riippuvainen (Koskimies & Mikkonen, 2005). Takaisinkutsufunktio (engl. callback function) on aliohjelma, joka välitetään esimerkiksi kirjastolle rekisteröitäväksi. Jossain myöhemmässä vaiheessa kirjasto sitten toteuttaa takaisinkutsun kutsumalla tätä takaisinkutsufunktiota. (Szyperski, Gruntz & Murer, 1998.) Takaisinkutsuja on kahden tyyppisiä: ne voivat olla synkronisia, jolloin takaisinkutsu toteutetaan ennen kuin kutsuva funktio palaa, tai asynkronisia, jolloin takaisinkutsu toteutetaan jossain vaiheessa myöhemmin, kun kutsuva

funktio on palannut (Gallaba ym., 2015). Näiden operaatioiden hyödyntäminen ja mahdolliset ongelmat ovat erilaisia.

Takaisinkutsumenetelmää käytetään monissa tilanteissa, etenkin kehyksiin perustuvissa arkkitehtuureissa, jolloin koko ohjelman suorituksen kontrolli on kehyksellä. Tästä käytetään nimitystä käänteinen ohjelmakontrolli (engl. inversion of control) (Johnson & Foote, 1988). Tavanomaisesti ohjelmalla on itsellään kontrolli siitä mitä suoritetaan ja milloin. Näin on myös silloin, kun sovellus hyödyntää kirjastoja toiminnassaan, sovellus vain kutsuu kirjaston palvelua ja sovellus itse tietää missä vaiheessa minkäkin toiminnon olisi syytä tapahtua (Koskimies & Mikkonen, 2005). Käänteisen ohjelmakontrollin tapauksessa tilanne on toinen. Silloin kehys hallitsee toimintoja ja niiden suoritusjärjestystä ja sovellukset ovat ikäänkuin komponentteja kehysten sisällä. Näiden välinen rajapinta on sopimus siitä millainen komponentin tulee olla että se sopii liitettäväksi kehykseen. Käänteisestä ohjelmakontrollista käytetään usein myös termiä Hollywood-periaate (engl. Hollywood principle: "don't call us, we'll call you"). Komponentilla ei siten ole itsellään kontrollia sen suhteen milloin operaatiot tulevat kutsutuiksi, vaan toiminta perustuu sopimuksiin ja luottamukseen.

Kerrosarkkitehtuurilla tarkoitetaan arkkitehtuurisuunnittelua, jossa ohjelmiston osat nähdään abstraktiotasonsa mukaan kerroksittaisina, alhaalla karkeammat ja ylhäällä abstraktimmat. Näin yleiskäyttöisen ohjelman osan voidaan ajatella kuuluvan alemmalle tasolle, jonka tarjoamia palveluita ylempi taso hyödyntää. Kerrosarkkitehtuurin yksi kultainen sääntö on, että alempi taso ei voi kutsua ylempää tasoa, ettei tulisi tästä riippuvaiseksi. Takaisinkutsuja hyödynnettäessä tätä kerrosarkkitehtuurin tärkeää periaatetta rikotaan tietoisesti, sillä silloin yleiskäyttöinen, alempi komponentti kutsuu ylempää komponenttia (Szyperski ym., 1998). Takaisinkutsumekanismin vuoksi yleiskäyttöinen komponentti ei kuitenkaan tule riippuvaiseksi kutsuttavasta komponentista (Koskimies & Mikkonen, 2005).

Takaisinkutsurakenne muodostaa JavaScriptissä (kuten yleensä funktio-ohjelmointikielissä) aina sulkeuman (engl. closure), riippumatta siitä kutsutaanko takaisinkutsufunktiota asynkronisesti vai synkronisesti. Sulkeumaan liittyy käsitteet näkyvyysalue ja leksikaalinen näkyvyysalue. Näkyvyysalue (engl. scope) tarkoittaa sitä aluetta, jossa määritellyt muuttujat ovat käytettävissä. JavaScript on funktioiden näkyvyysalueisiin perustuva kieli. Se tarkoittaa, että mikäli muuttuja määritellään funktion sisällä, se ei ole käytettävissä funktion ulkopuolella. Toisaalta se myös tarkoittaa sitä, että JavaScript ei ole lohkojen (engl. block) näkyvyyksiin perustuva kieli, eli esim. silmukkarakenteen sisällä määritellyt muuttujat ovat samalla tavalla näkyvissä muualla funktiossa kuin muutkin sen muuttujat. (Simpson, 2014.) Tämä on eri tavalla kuin monissa muissa yleisissä kielissä. Näkyvyysalueet toimivat hierarkisesti siten, että funktion sisällä määritellyn funktion näkyvyysalueeseen kuuluu myös ulompi funktio, mutta ei toisinpäin (Harmes, Diaz & Willison, 2008). Globaalitaso näkyy kaikille funktioille.

Käytännössä sulkeuma syntyy kun funktio esimerkiksi välitetään parametrina tai annetaan paluuarvona toiselle funktiolle. Takaisinkutsuttavaa funktio-

ta kutsutaan sen leksikaalisen näkyvyysalueen ulkopuolelta. Tästä huolimatta funktiolla on käytettävissään se näkyvyysalue joka sillä olisi mikäli sitä kutsuttaisiin määrittelyalueeltaan. Näin käy vaikka funktion leksikaalisen näkyvyysalueen suoritus olisikin jo päättynyt siinä vaiheessa kun takaisinkutsu tapahtuu. Näistä takaisinkutsufunktion ulkopuolelle jäävistä muuttujista tulee silloin ns. vapaita muuttujia (engl. free variables), joihin pääsee käsiksi enää vain sulkeuman kautta. Ne ovat myös roskienkeruun (engl. garbage collection) ulottumattomissa niin kauan kuin sulkeuma on olemassa. Takaisinkutsujafunktion toimiminen asynkronisesti ei tee tähän poikkeusta.

2.2 Samanaikaisuus tietojärjestelmissä

Synkronisesti toimiva operaatio, joka hyödyntää takaisinkutsua, suorittaa takaisinkutsun ennen kuin palaa kutsujalleen takaisin. Asynkronisesti toimiva operaatio palaa sen sijaan kutsujalleen takaisin, mutta takaisinkutsu tulee kutsutuksi jossain vaiheessa, riippuen käyttäjän toiminnoista, toteutuksesta ja järjestelmän muusta kuormituksesta. Operaation kutsujan tulee ottaa tämä toteutuksessa huomioon. Alaluvussa käsitellään samanaikaisuuden malleja sekä yleisemmin että tarkemmin JavaScriptiin ja takaisinkutsumalliin liittyen.

2.2.1 Samanaikaisuuden mallit

Yksi prosessoriydin voi suorittaa aidosti vain yhden tehtävän kerrallaan. Tietokoneiden ja käyttöjärjestelmien rakenne mahdollistaa kuitenkin näennäisen samanaikaisuuden (engl. concurrency). Ohjelmien ja käyttäjien näkökulmasta toimintoja voi tapahtua näennäisesti samaan aikaan, kun prosessorin suorituksessa olevia ohjelman osia vaihdetaan toisiin riittävän nopeasti. Useamman prosessoriytimen järjestelmässä kyseeseen voi tulla rinnakkaiset (engl. parallel) suoritukset, jolloin toiminnot voivat tapahtua aidosti samaan aikaan. Prosessoriytimet voivat sijaita saman fyysisen laitteen sisällä ja ne voivat esim. jakaa yhteistä muistialuetta ja hyödyntää yhdessä laitteen resursseja. Järjestelmä voi olla myös hajautettu, jolloin prosessorit toimivat erillään toisistaan ja yhteisten muistialueiden jakaminen ei onnistu, vaan on käytettävä muita keinoja esim. viestinvälitysmenetelmää (engl. message passing). Näiden tapojen erilaiset hybridit ovat yleisiä (Andrews, 1991).

Samanaikaisuuden hyödyntäminen ohjelmissa sai alkunsa käyttöjärjestelmien kehittymisen myötä, kun ne alkoivat tarjota mahdollisuuden suorittaa useita tehtäviä samaan aikaan (Andrews, 1991). Nykyiset käyttöjärjestelmät poikkeavat toisistaan, mutta ne perustuvat samanaikaisuudelle ja tavoille hallita sitä. Omalla logiikallaan ne hoitavat suorituksessa olevien ohjelmanosien vaihtamisen toisiin, niin että näennäinen samanaikaisuus toteutuu. Sittemmin myös sovellukset ovat voineet hyödyntää samanaikaisuutta toiminnassaan. Samanaikaisuus voi toteutua erilaisilla tavoilla, riippuen siitä millainen laitteisto- ja ohjelmistokokoonpano on kyseessä.

Samanaikaisuuden malli (engl. concurrency model) kuvaa sitä arkkitehtuuria,

jonka mukaan ohjelmassa hoidetaan tarve näennäisesti tai aidosti samanaikaisille tapahtumille. Samanaikaisuuden malli luo puitteet ja mahdollistaa asynkronisen toiminnan ohjelmille. Asynkronisella toiminnalla tarkoitetaan sitä, että ohjelman näkökulmasta jokin toiminto voidaan asettaa suoritukseen, kun samalla ohjelman muu kulku jatkuu toisaalla. Keskeisimmät mallit samanaikaisuuden luomiseksi ovat useihin säikeisiin perustuva malli ja tapahtumaperustainen malli. Molemmilla arkkitehtuureilla on vankat kannattajansa edelleen (Arpaci-Dusseau & Arpaci-Dusseau, 2015), vaikka väittely paremmuudesta on vuosikymmeniä vanha (Ousterhout, 1996; Welsh, Culler & Brewer, 2001; von Behren, Condit & Brewer, 2003; Adya, Howell, Theimer, Bolosky & Douceur, 2002). Kaikki eivät tosin ajattele, että nämä olisivat toisiaan poissulkevia näkökulmia, vaan että molemmista maailmoista olisi mahdollista kerätä parhaat puolet (Adya ym., 2002; Welsh ym., 2001).

Prosessit (engl. processes) ja loogisesti niiden sisällä olevat kevyemmät ja nopeammin vaihdettavat prosessit eli säikeet (engl. threads) ovat peruskäsitteitä, joilla samanaikaisuutta mahdollistetaan. Prosesseilla on yleensä omat muistiavaruutensa, mutta säikeet jakavat yhteisen muistialueen ja voivat käsitellä samoja muuttujia toisten saman prosessin säikeiden kanssa.

Sovellukset voivat hyödyntää käyttöjärjestelmän tarjoamia ja vuorontamia (engl. scheduled) säikeitä omassa toteutuksessaan, mikäli ohjelmointiarkkitehtuuri sen sallii. Monisäikeiset ohjelmat ovatkin tyypillinen ja perinteinen tapa toteuttaa samanaikaisuutta ohjelmissa (Arpaci-Dusseau & Arpaci-Dusseau, 2015). Nykyisin tietokoneet sisältävät hyvin usein useita ytimiä. Monisäikeisyyttä hyödyntävät ohjelmat pääsevätkin hyödyntämään usean ytimen tuomaa rinnakkaisuutta luonnollisesti (Zeldovich, Yip, Dabek, Morris, Mazieres & Kaashoek, 2003). Monisäikeisen ohjelman ohjelmoijan tulee ottaa aina ohjelmaa rakentaessaan huomioon, että ohjelmalla ei ole kontrollia säikeiden suoritusjärjestykseen ja siihen, milloin säikeen suoritus keskeytyy toisen säikeen vuoksi. Kun säikeet ylläpitävät yhdessä ohjelman tilaa käsittelemällä mahdollisesti samoja tietorakenteita, on huolehdittava siitä, että ohjelma toimii oikein riippumatta säikeiden suoritusjärjestyksestä tai siitä milloin säikeen toiminta keskeytyy vuorontimen (engl. scheduler) vaihtaessa suoritettavaa ohjelmalohkoa. Ohjelmointi monisäikeisessä ympäristössä ei ole triviaalia (Arpaci-Dusseau & Arpaci-Dusseau, 2015).

Tapahtumiin perustuva samanaikaisuus on malli, jossa samanaikaisuus toteutetaan yksisäikeisessä järjestelmässä. Tapahtuma (engl. event) on ennalta määritelty tilanne, joka voi sattua ohjelman suorituksen aikana ja johon jonkin järjestelmän osan tulisi reagoida. (Koskimies & Mikkonen, 2005.) Käyttäjän kanssa vuorovaikutteiset sovellukset painikkeineen ja muine toimintoineen ovat loogisia tapahtumapohjaisen arkkitehtuurin hyödyntäjiä ja sitä onkin tyypillisesti hyödynnetty graafisissa käyttöliittymissä. Tapahtumapohjaisen samanaikaisuuden juuret kuitenkin löytyvät C/UNIX maailmasta (Arpaci-Dusseau & Arpaci-Dusseau, 2015). Tapahtumat voivat olla monenlaisia muitakin tilanteita joihin jonkin komponentin tulee reagoida. Esimerkiksi tiedostojärjestelmän ja tätä visuaalisesti esittävien komponenttien välillä olisi hyvä olla ilmoitusmenettely sen

sijaan, että komponentit kävisivät ajastetusti tutkimassa muutoksia tiedostojärjestelmästä (Szyperski ym., 1998).

Tapahtumiin perustuvat järjestelmät rakentuvat tapahtumasilmukkarakenteen (engl. event loop) varaan. Yksinkertaisimmillaan se on rakenne, jossa ikuisessa silmukassa käydään läpi jonoon asetettuja tapahtumia ja käynnistetään tapahtumakäsittelijä jokaiselle yksi toisensa jälkeen. Tapahtumiin perustuvan rakenteen suurin etu on se, että sen toiminta vastaa käyttöjärjestelmän tekemää ajastusta säikeille, mutta tapahtumasilmukan avulla ohjelmalla on itsellään sen kontrolli. (Arpaci-Dusseau & Arpaci-Dusseau, 2015.) Ohjelmoija voi vaikuttaa eri ohjelmalohkojen suoritusjärjestykseen huolehtimalla siitä, että ne tulevat tehtäväjonoon halutussa järjestyksessä. Ohjelmoijan ei tarvitse myöskään varautua ohjelmalohkojen keskeytyksiin, vaan ohjelmoija tietää että ohjelmanosat suoritetaan kokonaan eikä ohjelman toiminnan näkökulmasta mikään keskeytä niiden suorittamista. Arpaci-Dusseau ja Arpaci-Dusseau (2015) pitävät tapahtumapohjaisen mallin suurimpina ongelmina estäviä (engl. blocking) järjestelmäkutsuja, manuaalista pinonhallintaa ja toimintaa useiden prosessorien kanssa. Näitä käsitellään seuraavissa kappaleissa sekä yleisesti että JavaScript-kieleen liittyen.

Estävillä järjestelmäkutsuilla tarkoitetaan kutsuja, jotka vaativat aikaa ja jäädyttävät muun toiminnan siksi aikaa. Tapahtumapohjaisessa mallissahan käytössä on vain yksi säie, joka hoitaa koko järjestelmän toiminnan. On hyvin tärkeää ettei mikään toiminto varaa säiettä pitkäksi aikaa ja siten estä muiden toimintojen suoritusta (Arpaci-Dusseau & Arpaci-Dusseau, 2015). JavaScriptissä on pyritty siihen, että lähes kaikki järjestelmäkutsut toimivat asynkronisesti, eli ympäristön tarjoamat rajapinnat on toteutettu asynkronista toimintatapaa hyödyntäen. Tarjolla on kuitenkin myös synkronisia rajapintoja eikä mikään estä toteuttamasta muita suuria toiminnallisuuksia, kuten vaikka laskentaa, joka estäisi ohjelman muun suorituksen. Viime kädessä tällaisen välttäminen on ohjelmoijien vastuulla.

Manuaalisen pinonhallinnan (engl. stack management) ongelma on ollut keskeinen kompastuskivi ylipäätään tapahtumapohjaisessa mallissa ja vaatii paljon ohjelmoijan suorittamaa työtä hoituakseen (Adya ym., 2002). Monisäikeisessä ohjelmassa eri säikeillä on omat pinot, joissa ohjelman tilan tiedot ovat käytössä. Tapahtumapohjaisessa mallissa näin ei ole, vaan asynkronisesti takaisinkutsuttu funktio ei tiedä mitään alkuperäisen ohjelmakoodin tilasta kun sitä kutsutaan. (Arpaci-Dusseau & Arpaci-Dusseau, 2015.) Ongelman hoitamiseen on käytetty manuaalisen pinonhallinnan tekniikkaa (Arpaci-Dusseau & Arpaci-Dusseau, 2015), joka hyödyntää kontinuaatioita (engl. continuations) (Friedman, Haynes & Kohlbecker, 1984). Pinonhallinnan automatisoinnin kehittyminen ohjelmointikieliin on helpottanut tapahtumapohjaisen ohjelmointityylin hyödyntämistä (Bierman, Russo, Mainland, Meijer & Torgersen, 2012). JavaScriptissä ja muissa ohjelmointikielissä, jotka sisältävät sulkeumat ei tarvitse huolehtia manuaalisesta pinonhallinnasta, eikä tätä kompastuskiveä siten ole (Adya ym., 2002).

Perinteisesti tapahtumapohjaiset järjestelmät eivät voi hyödyntää monta ydintä, sillä ne perustuvat yhden säikeen varaan (Gaud, Geneves, Lachaize, Lepers,

Mottet, Muller & Quéma, 2010). Useamman ytimen järjestelmissä tapahtumasilmukan hoitavan komponentin olisi osattava toimia myös useamman ytimen kanssa, jotta tapahtumapohjaisessa järjestelmässä voitaisiin hyödyntää todellista rinnakkaisuutta (Zeldovich ym., 2003). Mikäli järjestelmät siirtyvät yhden ytimen sijasta hyödyntämään useita ytimiä, osa tapahtumapohjaisen lähestymistavan yksinkertaisuudesta häviää. Voidakseen hyödyntää useampaa kuin yhtä prosessoriydintä tapahtumankäsittelijän tulee voida käynnistää useita tapahtumia samaan aikaan. Silloin monisäikeisestä ympäristöstä tunnetut ongelmat kuten kriittiset alueet (engl. critical sections) ilmaantuvat ja niitä on hallittava aivan vastaavasti esimerkiksi lukkojen (engl. locks) avulla. Siten useita prosessoreita hyödyntävän järjestelmän ei ole enää mahdollista hoitaa yksinkertaista tapahtumankäsittelyä ilman lukkoja. (Arpaci-Dusseau & Arpaci-Dusseau, 2015.) Tapahtumapohjaisena kielenä myös JavaScript on kärsinyt vaikeudesta hyödyntää useita prosessoriytimiä. Tähän ongelmaan kehitettiin verkkotyöläiset (engl. web workers), joiden avulla on mahdollista hyödyntää myös JavaScript-kielisissä ohjelmissa nykyaikaisten laitteiden mahdollistamaa todellista rinnakkaisuutta (Boduch, 2015). Verkkotyöläinen käynnistetään omaan säikeeseen, jolloin todellinen rinnakkaisuus on moniytimisessä ympäristössä mahdollista.

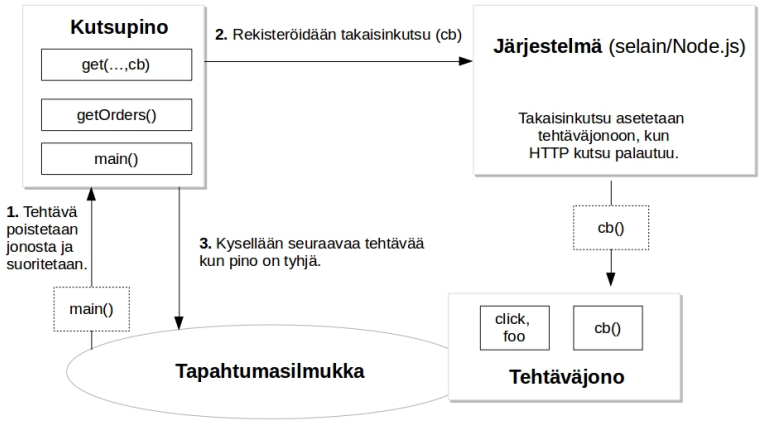
Tapahtumapohjainen arkkitehtuuri hyödyntää yleensä takaisinkutsuja sekä tarkkailija-suunnittelumallia (engl. observer design pattern) (Koskimies & Mikkonen, 2005). Takaisinkutsumenetelmä onkin yleinen malli monissa komponenteissa, joiden tehtävänä on hallita asynkronisia tapahtumia (Szyperski ym., 1998). Perusajatus asynkronisessa toiminnassa on palastella aikaavievät, ns. estävät operaatiot pienempiin osiin, siten että ripeän alustuksen jälkeen ohjelman kontrolli palaa kutsujalle. Alustusmetodille voidaan välittää takaisinkutsufunktio, joka tulee kutsutuksi operaation toteuttamisen jälkeen tai alustusmetodi voi palauttaa kahvan (engl. handle), jonka avulla kutsuja voi rekisteröidä takaisinkutsufunktion. (Bierman ym., 2012.)

2.2.2 JavaScript ja asynkronisuus

JavaScript-ohjelmia kuvataan usein yksisäikeisiksi ja tapahtumapohjaisiksi (Mickens, Elson & Howell, 2010). Ennen tuoretta ECMAScript 6 -standardia tätä samanaikaisuuden mallia ei kuitenkaan oltu määritelty missään. JavaScript-kielen toteutukset eli eri ympäristöihin kehitetyt JavaScript-moottorit kuitenkin jakavat yleensä yhteisen samanaikaisuuden mallin standardin aiemmista puutteista huolimatta. (Herman, 2012.) JavaScript-ohjelmia ajetaan soveltuvan ympäristön, esimerkiksi selaimen sisässä. Ohjelma tulkataan ajonaikaisesti ympäristön tarjoaman JavaScript-moottorin avulla. Esimerkiksi Chrome-verkkoselain sisältää V8-nimisen JavaScript-moottorin. Tapahtumapohjaisuus on toteutettu siten, että varsinaisen JavaScript-moottorin ulkopuolella on tarvittava toiminnallisuus tapahtumien hallinnalle. Ympäristö, esimerkiksi selain, tarjoaa JavaScript-moottorin käytettäväksi yhden säikeen. Yhdellä säikeellä toimittaessa on mahdollista tehdä vain yhtä asiaa kerrallaan, on vain yksi kutsupino (engl. call stack) jonka mukaan suoritus etenee. Toisaalta ei tarvitse pelätä että muut säikeet esimerkiksi

käsittelisivät samoja muistialueita ja muuttujia. Useiden säikeiden aiheuttamista mahdollisista ongelmatilanteista ei tarvitse JavaScript-ohjelmakoodissa huolehtia (Simpson, 2015).

Tapahtumapohjaisuus perustuu tehtäväjonon (engl. task queue) ja tapahtumasilmukan (engl. event loop) varaan. Näiden toimintaa kuvataan kuviossa 1.



KUVIO 1: JavaScriptin tapahtumasilmukka (Gallaba ym., 2015, 2).

Tehtäväjono on tietorakenne, johon talletetaan suoritusta odottavat ohjelmanosat. Toiset säikeet viestivät JavaScript-ohjelman kanssa tehtäväjonon avulla, johon ne voivat asettaa töitä suoritettavaksi (Parker, 2015). Tapahtumasilmukka hakee tehtäväjonosta tapahtumia suoritettavaksi yksi toisensa jälkeen järjestyksessä. Tapahtumat ovat joitakin ennalta määrättyjä tilanteita, joihin ohjelmassa halutaan reagoida. Ne voidaan ohjelmoida saamaan alkunsa yhtä hyvin käyttäjän toiminnasta kuin jonkin muun toiminnon tilan muutoksesta. Tapahtumasilmukka asettaa kutsupinon suorittamisen tehtäväjonon tapahtumien edelle eli vasta kun kutsupino on tyhjä, tapahtumasilmukka käsittelee tehtäväjonossa seuraavan tapahtuman (Gallaba ym., 2015). Jokainen tapahtuma luvataan suorittaa kerrallaan loppuun asti (engl. run-to-completion). Näin ollen ohjelmissa voidaan luottaa siihen, että ohjelmakoodi tulee suoritetuksi loppuun eikä tapahtumien käsittely sitä koskaan keskeytä (Simpson, 2015; Mickens ym., 2010). Toisaalta mikä tahansa pidempi ja aikaa vievä operaatio voi estää ohjelman suorituksen jouhevan etenemisen ja siten pilata käyttäjäkokemuksen hitaalla vasteella.

Tehtäväjonoon asetetaan funktioita, jotka tulevat kutsutuiksi omalla vuorollaan. Näin esimerkiksi tapahtumien kuuntelijoille voidaan asettaa takaisinkutsufunktio, jota toivotaan kutsuttavaksi kun tapahtuma tapahtuu. Kuuntelijan tehtävänä on asettaa tämä takaisinkutsufunktio tehtäväjonoon sitten kun haluttu tapantuma on tapahtunut. Tämä on ympäristön, kuten selaimen tarjoamaa palvelua. Esimerkiksi listaus 1 esittää yksinkertaisen JavaScript-ohjelman osan, jossa muodostetaan click-tapahtumalle kuuntelija, joka kutsuu hei -funktiota tapahtuman tapahduttua. Hei-funktio on takaisinkutsufunktio ja kuuntelija on takaisinkutsuja. AddEventListener -funktio on selaimen tarjoama rajapinta, jolla voidaan asettaa kuuntelija tietylle tapahtumalle.

LISTAUS 1: Kuuntelijan asettaminen click-tapahtumalle

```
1       function hei() {
2         alert("Heipä hei!");
3       }
4       document.addEventListener("click", hei);
```

Toinen vaihtoehto on käyttää tapahtuma-attribuutteja (engl. event attributes, on-events tai inline events), jolloin suoritettava koodi asetetaan suoraan HTML-elementin attribuutin arvoksi. Usein tähän asetetaan globaalisti määritelty funktio, kuten esimerkin hei-funktio voisi olla. Listauksessa 2 esitetään että arvoksi voi kuitenkin laittaa mitä vain koodia. Attribuutille annettu merkkijono käsitellään takaisinkutsuttavana funktiona (Flanagan, 2006). Näin elementille muodostetaan kuuntelija, ja arvoksi asetettu ohjelmakoodi tulee sitä kautta suoritetuksi tapahtuman myötä kun se suoritetaan tehtäväjonosta omalla vuorollaan.

LISTAUS 2: Tapahtuma-attribuuttityyppisen kuuntelijan asettaminen

```
1    <html>
2    ..
3    <input type="button" value="Tulos" onclick="x=3+4; alert('Tulos on'+ x);"/>
4    ..
5    </html>
```

JavaScript- ja HTML-koodit ovat jälkimmäisessä vaihtoehdossa hyvin tiukasti toisiinsa sidottuja mikä voi aiheuttaa ylläpito-ongelmia (Zakas, 2009; Fard & Mesbah, 2013). Valinnalla on vaikutusta myös näkyvyysalueisiin, sillä tapahtuma-attribuuttina asetetun tapahtumakäsittelijän näkyvyysalue on erilainen ja monimutkaisempi (Flanagan, 2006). Jälkimmäistä toteutustapaa ei enää yleisesti suositella käytettäväksi (Mozilla, 2014), mutta yhteensopivuuden vuoksi se on vielä olemassa ikäänkuin jäänteenä aikaisemmista toteutustavoista (W3C, 2011).

Tällaiset asynkroniset tapahtumakutsut ovat olleet mukana kielen toteutuksissa alusta asti mahdollistaen käyttöliittymään interaktiivisia toimintoja (Simpson, 2015). Palvelinkutsujen osalta tilanne on toinen. Verkkosivutekniikoiden kehittymisen alkupäässä, kun JavaScript oli vasta tullut selaimiin mukaan, JavaScript-ohjelmat toimivat synkronisesti tehdessään palvelinkutsuja. Tekniikka vaatii koko sivun lataamisen uudelleen jokaisen kutsun yhteydessä. (Zheng, Bao & Zhang, 2011.) Perinteinen synkroninen palvelinkutsu toimii siten että HTML-linkin tai -lomakkeen avulla muodostetaan palvelinkutsu. Palvelin suorittaa pyydetyn toiminnon ja lähettää sitten selaimelle takaisin uuden sivun kaikkine tietoineen. Vastauksen saavuttua selaimelle koko sivu rakennetaan ja piirretään uudelleen. Käyttäjälle tämä näkyy selaimen tilana, jossa mitään ei voi tehdä ennenkuin työ on suoritettu.

AJAX, joka on lyhenne termistä ”Asynchronous JavaScript + XML” (Garrett, 2005), mahdollisti asynkroniset palvelinkutsut (Zheng ym., 2011) ja ns. yhden sivun ohjelmat (engl. Single Page App), jossa verkkosivua ei ladata kokonaan uudelleen missään vaiheessa, vaan ainoastaan tarvittava osa tulee päivitetyksi palvelimelta saatujen tietojen mukaan. Nimestään huolimatta rajapinta tarjoaa myös synkronisen tavan ladata sivuston osia, mutta tämä menetelmä on kui-

tenkin käyttäjäkokemukselle yhtä huono kuin perinteinen synkroninen palvelukutsu, eikä sitä siksi suositella käytettäväksi kuin erikoistilanteissa (Mozilla, 2015). AJAX -käsitteellä tarkoitetaan useiden eri tekniikoiden yhteiskäyttöä (Garrett, 2005). Olennaisin osa sitä kuitenkin on XMLHttpRequest -rajapinta, jonka selaimet tarjoavat JavaScript-ohjelmien käyttöön. Rajapinta mahdollistaa ohjelmalle asynkronisten palvelukutsujen hyödyntämisen. Rajapinnan avulla ohjelma voi käyttää myös palvelinoperaatioissaan samaa tapahtumapohjaista toimintamallia kuin muuallakin. Siten, aivan samaan tyyliin kuin tapahtuman kuuntelija, palvelinkutsujen kuuntelija saa tiedon valmistuneesta palvelinkutsusta ja asettaa halutun ohjelmakoodin samaan tehtäväjonoon muiden tapahtumien kanssa ja tapahtumasilmukka kutsuu tätäkin takaisinkutsufunktiota omalla vuorollaan (Herman, 2012).

Takaisinkutsujen hyödyntäminen asynkronisessa toiminnassa ei ole uusi asia ja niiden herkästi muodostama spagettikoodikin on tiedostettu vuosikymmeniä sitten (Myers, 1991). Siitä huolimatta ongelmat ovat näkyvissä myös JavaScript-kielessä. Näitä käsitellään seuraavassa alaluvussa.

2.3 Asynkronisten takasinkutsujen ongelmia

Takaisinkutsujen on havaittu aiheuttavan monenlaista harmia, etenkin AJAX-toiminnallisuuden myötä, kun takaisinkutsuja hyödyntävien ohjelmapätkien kompleksisuus on kasvanut. Takaisinkutsut muodostavatkin hyvin nopeasti ohjelmarakenteen jota on vaikea ymmärtää ja hallita (Boduch, 2015). Erillisten, asynkronisesti toisiaan ristiin rastiin kutsuvien takaisinkutsufunktioiden hallittavuus on heikkoa, ja tätä kuvaa hyvin termi *”asynkroninen spagetti”*. Termillä viitataan yleisempäänkin ongelmaan websovelluksissa, joissa rikotaan vanhoja ohjelmistotuotannon periaatteita ja siten kohdataan myös niitä spagettikoodiksi nimettyjä ongelmia joita vuosikymmeniä sitten pyrittiin rakenteisen ohjelmoinnin periaateilla ratkaisemaan (Mikkonen & Taivalsaari, 2007).

Asynkronisiin takaisinkutsuihin liitettyjä ongelmia on kerätty kirjallisuudesta ja esitetty taulukossa 1. Ongelmat ovat osin toisiinsa liittyviä tai toisista ongelmista johtuvia. Merkittävimmät löydetyistä ongelmista käsitellään alaluvuissa.

2.3.1 Sisäkkäiset funktiot

Asynkronisen spagetin yhdeksi ratkaisuksi kehitettiin sisäkkäiset anonyymit funktiot, joiden avulla kokonaisuus pysyy samassa paikassa (Azegami, Fukuda & Leger, 2014). Lukuisat sisäkkäiset takaisinkutsut muodostavat helposti ”oikealle kaartuvaa”, usein kolmion muotoon hakeutuvaa ohjelmakoodia. Tästä syystä rakennetta kutsutaan myös termillä ”tuomion pyramidi” (engl. pyramid-of-doom) (Syed, 2014). Kuten nimikin jo kertoo, tällaisen rakenteen on havaittu aiheuttavan ohjelman kehittäjälle tai viimeistään sen ylläpitäjälle ongelmia. Listaus 3 esittää melko vaikeaselkoisen takaisinkutsurakenteen, jossa on monta sisäkkäistä, asynkronista ja nimetöntä funktiota.

TAULUKKO 1: JavaScriptin asynkronisiin takaisinkutsuihin liitettyjä ongelmia

Esitetty ongelma	**Lähteet**
Sisäkkäiset funktiot	Simpson, 2015; Syed, 2014; Gallaba ym., 2015; Ogden, 2015
Virheidenkäsittely	Simpson, 2015; Syed, 2014; Gallaba ym., 2015; Parker, 2015
Suoritusjärjestyksen hallinta	Simpson, 2015; Gallaba ym., 2015; Kambona, Boix & De Meuter, 2013; Azegami ym., 2014; Alimadadi, Sequeira, Mesbah & Pattabiraman, 2014; Parker, 2015; Mogk, 2015
Ohjelman tilan hallinta	Bainomugisha, Carreton, van Cutsem, Mostinckx & De Meuter, 2013; Alimadadi ym., 2014; Mogk, 2015; Boduch, 2015
Luottamuskysymykset	Simpson, 2015; Syed, 2014; Parker, 2015
Ohjelmakoodin määrä (boilerplate)	Kambona ym., 2013; Parker, 2015; Boduch, 2015
Perustuu sivuvaikutuksiin	Bainomugisha ym., 2013; Mogk, 2015
Silmukkarakenteet	Syed, 2014
Uudelleenkäyttö vaikeaa	Gallaba ym., 2015

LISTAUS 3: Takaisinkutsuhelvetti (Gallaba ym., 2015, 1)

```
1   var db = require("somedatabaseprovider");
2   http.get("/recentposts",function(req,res){
3     db.openConnection("host",creds,function(err,conn){
4       res.param["posts"].forEach (function(post){
5         conn.query("select * from users where id = " + post
6         ["user"], function(err , results ){
7           conn.close ();
8           res.send(results[0]);
9         }) ;
10      }) ;
11    }) ;
12  }) ;
```

Tuomionpyramidi -käsitteen ohella kyseinen ongelma tunnetaan myös termillä ”takaisinkutsuhelvetti” (engl. callback hell). Takaisinkutsuhelvetti ei ole vakiintunut käsite: siihen viitataan lukuisissa artikkeleissa ja kirjoissa, mutta se määritellään eri tavoin ja siihen liitetään erityyppisiä asioita ja ongelmia. Ilmeisin ja helpoimmin havaittava ongelma löytyy useimmista lähteistä, eli juuri tämä ilmiöön liittyvä ohjelmakoodin epämiellyttävä ulkoasu. Tähän ongelmaan keskittyy myös Max Ogdenin kirjoittama, usein viitattu, takaisinkutsuhelvettiä esittelevä verkkosivu (Ogden, 2015). Sisäkkäin koostettujen funktioiden ongelma tulee esille monissa lähteissä. Erityisesti tätä ongelmaa korostetaan erilaisilla verkkosivuilla ja -keskusteluissa joissa käsitellään JavaScriptin takaisinkutsumallin tuomia ongelmia. Sisäkkäiset funktiot ovat ehkä yleisimmin tunnettu ja näkyvin ongelma jonka takaisinkutsumenetelmä saa helposti aikaan. Keskittyminen ohjelmakoodin ulkoasuun voi kuitenkin olla harhaanjohtavaa, sillä takaisinkutsut voivat muodostaa huomattavasti ohjelmakoodin sisentymistä syvempiä ongelmia (Simpson, 2015).

2.3.2 Virheidenkäsittelymekanismin puuttuminen

Syedin (2014) mukaan virheiden käsittely, tai sen puuttuminen, on takaisinkutsuhelvettiin liittyvä pahin ongelma. Samaan tapaan aikanaan go to -ohjelmarakenteen vakavampia puutteita oli se, ettei se mahdollistanut tietoa siitä mistä virhetilanteeseen oli tultu (Dijkstra, 1968). Kutsumalla asynkronista funktiota, joka ottaa parametrikseen takaisinkutsuttavan funktion, ohjelmoija menettää mahdollisuuden käyttää try-catch -virheidenkäsittelyominaisuutta tämän toiminnon suhteen. Siten ohjelmoija ei voi virheen sattuessa heittää virhettä yhdessä paikassa ja ottaa sitä kiinni toisessa paikassa, ohjelmahierarkian ylemmällä tasolla. Sen sijaan ohjelmoijien on rakennettava virheiden varalle virheilmoitusten välittämis- ja tarkistusmenettely jokaiselle takaisinkutsutilanteelle erikseen. Virheidenkäsittelyrakenteen luominen on toki mahdollista, mutta se on suuritöistä ja virhealtista (Simpson, 2015). Virheilmoitusten välittämistä varten ohjelmoijayhteisö on kehittänyt virhe ensin (engl. error-first) -sopimuksen, missä sovitaan että jokaisen takaisinkutsun ensimmäinen parametri on varattu virheilmoitusten välittämiseen (Gallaba ym., 2015). Tämä on kuitenkin vain suositus, eikä mikään tae siitä että näin tapahtuisi.

2.3.3 Suoritusjärjestyksen ymmärtämisen vaikeus

Ohjelman kulun ymmärtäminen on Alimadadin ym. (2014) mukaan haasteellista JavaScript-ohjelmissa jotka perustuvat tapahtumapohjaiselle vuorovaikutukselle. Suoritusjärjestyksen ymmärtämisen vaikeus luonnehtiikin Simpsonin (2015) mukaan takaisinkutsuhelvettiongelmaa parhaiten. Ohjelman toiminnan järkeily ja suorituksen kulun ymmärtäminen ei ole triviaalia lukuisten asynkronisten takaisinkutsujen muodostamassa ohjelmassa. Ihminen lukee staattista ohjelmatekstiä, mutta suoritus tapahtuu dynaamisesti ja tässä tapauksessa dynaaminen tila voi olla hyvin erilainen kuin kooditiedostossa näkyvä teksti. Ihmisen vaikeus ymmärtää ohjelman suoritusta, mikäli staattisen ja dynaamisen tilan ero oli suuri, tunnistettiin jo lähes 50 vuotta sitten (Dijkstra, 1968).

2.3.4 Tilan hallinta

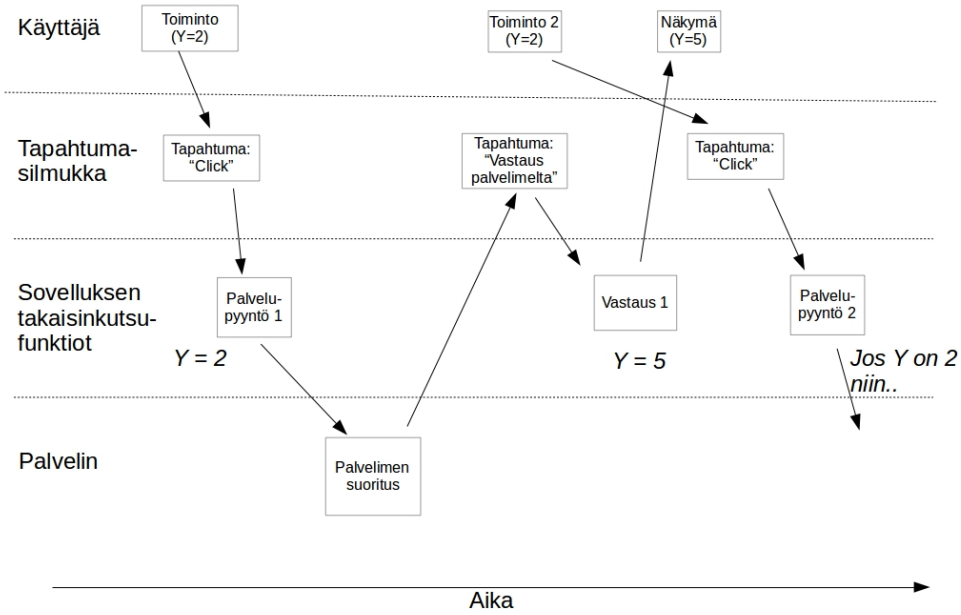
Ohjelman tilan hallinta voi olla vaativa tehtävä, kun erillään olevat ohjelman osat voivat muokata samaa dataa ja niiden suoritusjärjestys voi vaihdella (Bainomugisha ym., 2013). Lisäksi takaisinkutsujen toiminta perustuu sivuvaikutuksille. Se tekee vaikeaksi ymmärtää toisistaan riippuvaisten takaisinkutsujen muodostamaa ohjelman tilaa. (Mogk, 2015.)

Sulkeumat ovat monella tapaa erittäin hyödyllisiä asynkronisessa toiminnassa mahdollistaessaan pääsyn funktion leksikaalisen ympäristön muuttujiin. Sovelluksen tilan hallintaan ne tuovat kuitenkin omat haasteensa kun samaan leksikaaliseen ympäristöön kuuluvat funktiot voivat toisistaan tietämättä käsitellä samoja vapaita muuttujia.

JavaScript-ympäristöissä tehtäväjono on sama kaikille tapahtumille ja tapahtumat käsitellään jonosta yksi toisensa jälkeen järjestyksessä. Samassa tapahtumasilmukassa käsitellään siis sekä käyttäjän toimintoihin liittyviä että asynkronisiin palvelinkutsuihin liittyviä takaisinkutsuja. Tehtäväjonoon tapahtumat voi-

vat tulla eri järjestyksessä eri suorituskerroilla, riippuen käyttäjän toiminnoista ja palvelimen kuormituksesta. Tyypillisesti eri tapahtumien takaisinkutsufunktioissa tehdään myös uusia asynkronisia kutsuja ja palvelimen vastausaika voi tietenkin vaihdella. JavaScriptissä yksittäinen DOM-puuhun liittyvä tapahtuma voi tuottaa myös lukuisia uusia tapahtumia (Alimadadi ym., 2014). Tapahtumankäsittely on siten epädeterministinen: ohjelmoija ei voi luottaa siihen, että tapahtumien käsittelyjärjestys olisi aina samanlainen (Zheng ym., 2011).

Esimerkiksi Zhengin ym. (2011) kuvasta muokattu kuvio 2 esittää epäjohdonmukaisuustilannetta (engl. inconsistency), jossa käyttäjä pääsee tekemään ja tekee uusia käyttäjätoimia ennen kuin aikaisemman toimenpiteen palvelinpyyntö on valmistunut. Kuvassa sovelluksen takaisinkutsufunktiot ”Vastaus 1” ja ”Pal-



KUVIO 2: Takaisinkutsujen aiheuttama epäjohdonmukaisuustilanne.

velupyyntö 2” käsittelevät samaa tietoa. Aikaisempi operaatio on muuttanut tietoa, jota tarvitaan seuraavassa operaatiossa. Esimerkkitapauksessa ohjelma suorittaa operaation perustuen siihen että Y on 5, vaikka käyttäjä on käynnistänyt operaation ajatellen että Y on 2. Yhtälailla voisi olla niin että käyttäjä haluaa poistaa esimerkiksi jonkin kuvan, mutta mikäli järjestelmän tietorakenne muuttuu kesken kaiken, poistettavaksi voisi joutua väärä kuva. Aivan vastaava epäjohdonmukaisuustilanne syntyy myös silloin kun uusi tapahtuma tulee kutsutuksi ennen kuin edellisen vastaus on saapunut. Tällöin Zheng ym. (2011) kutsuvat ongelmaa atomisuuden loukkaukseksi (engl. atomicity violation).

Mikäli ohjelman suorituksen oikeellisuus riippuu tapahtumien tietystä suoritusjärjestyksestä, sanotaan ohjelman kärsivän kilpailutilanteen (engl. race condition) ongelmasta. Tällöin monet samanaikaiset toiminnat voivat muuttaa jaettujen tietorakenteiden tilaa eri tavalla riippuen siitä missä järjestyksessä ne suori-

tetaan. (Herman, 2012.) Monisäikeisessä ympäristössä kilpailutilanteiden hallinta on tunnettu ja tärkeä osa ohjelmointia. Vähemmän tunnettua on, että kilpailutilanteet ovat samalla tavalla olemassa myös tapahtumapohjaisessa ympäristössä liittyen mahdolliseen epäjohdonmukaisuuteen tapahtumien käynnistämisissä (Raychev, Vechev & Sridharan, 2013; Billes, 2015). Vaikka tapahtumapohjainen malli on vapaa joistain monisäikeiselle ohjelmoinnille tyypillisistä ongelmista, niin tapahtumiin liittyvä kilpailutilanne on kuitenkin olemassa myös JavaScript-kielellä ohjelmoitaessa (Petrov, Vechev, Sridharan & Dolby, 2012). Billesin (2015) mukaan suurin osa kilpailutilanteisiin liittyvistä ohjelmavirheistä liittyy AJAX-kutsuihin.

Epädeterministisyyteen liittyvien virheiden löytäminen voi olla vaikeaa. Esimerkiksi yleinen testausautomaatiosovellus Selenium ei kykene toistamaan montakaan epädeterministisyydestä johtuvista virheistä. (Mickens ym., 2010.)

2.3.5 Luottamus

Takaisinkutsuhelvetissä on myös kysymys luottamuksesta tai oikeammin sen puutteesta (Simpson, 2015). Tyypillisesti takaisinkutsuja-funktio on kolmannen osapuolen toteuttama rajapinta jollekin asynkroniselle toiminnalle. Tällöin on sanaton 'sopimus' oman ohjelmakoodin ja kolmannen osapuolen ohjelman välillä, että rajapinnan oletetaan toimivan tietyllä tavalla (Simpson, 2015). Simpson löytää luottamukseen perustuvia monenlaisia ongelmia. Näitä ovat

- takaisinkutsun kutsuminen liian aikaisin
- takaisinkutsun kutsuminen liian myöhään tai ei koskaan
- takaisinkutsun kutsuminen liian harvoin tai liian useasti
- välitettävien parametrien puutteellisuus
- virheilmoitusten pimittäminen.

Luottamuskysymystä kuvaakin osuvasti seuraava siteeraus: ”Ei voi tietää milloin takaisinkutsufunktiota kutsutaan, mitkä takaisinkutsufunktiot on jo kutsuttu, mitä takaisinkutsuja tullaan kutsumaan ja mitkä muut takaisinkutsut käynnistyvät takaisinkutsun yhteydessä.” (Edwards, 2009, 2). Ei ole mahdollista tietää varmasti, miten takaisinkutsujafunktio käyttää takaisinkutsuvaltaansa, ja takaisinkutsujafunktio voi myös sisältää virheitä. Huolellinen ohjelmoija joutuukin ottamaan tämän ongelman huomioon ja toteuttamaan suojarakenteita ja tarkistuksia esim. sen varalle että kerran toteutettavaksi tarkoitettu takaisinkutsu toteutettaisiinkin monta kertaa. Tämä lisää ohjelman kompleksisuutta ja ohjelmakoodin määrää.

Ohjelmoijan on myös voitava luottaa siihen, että asynkroninen funktio toimii aina asynkronisesti, sillä heterogeenisen API:n hyödyntäminen ilman virheitä on vaikeaa (Herman, 2012). Heterogeenisella API:lla tarkoitetaan rajapintaa, joka toimii toisinaan asynkronisesti ja toisinaan synkronisesti. Tämä aiheuttaa ohjelmakoodin haarautumisen erilaisiin suorituspolkuihin, joka taas vaikeuttaa ymmärtämistä ja testaamista sekä siten vaarantaa ohjelman luotettavan toiminnan (Parker, 2015). Toisinaan asynkronisesti ja toisinaan synkronisesti toimivan palvelun toteuttaminen voi olla palvelurajapinnan toteuttajasta houkuttelevaa,

sillä usein jokin tieto olisi palautettavissa heti, ilman palvelukutsua esim. välimuistista. Tässäkin tapauksessa tieto kuitenkin tulee välittää tapahtumasilmukan kautta, esimerkiksi asynkronista setTimeout-funktiota hyödyntämällä, jotta rajapintaa hyödynnettäessä ohjelmat voidaan toteuttaa hallitusti. Parkerin (2015) mukaan heterogeenisia rajapintoja on kuitenkin olemassa esim. erittäin laajalti käytetyssä jQuery-kirjastossa (jQuery, 2016).

2.3.6 Ohjelmakoodin määrä (boilerplate)

Kambonan ym. (2013) mukaan asynkronisuuden hoitaminen takaisinkutsumenetelmällä pakottaa ohjelmoijan kirjoittamaan paljon turhaa ohjelmakoodia pystyäkseen kontrolloimaan tapahtumien kulkua. Erityisesti virheidenkäsittelyn puuttuminen lisää ohjelmakoodin määrää, kun virheitä pitää tarkistella jokaisella askeleella erikseen (Parker, 2015). Ohjelmakoodin lisääntyessä ja monimutkaistuessa lisääntyvät vastaavasti myös inhimillisten virheiden mahdollisuudet.

3 ASYNKRONISEN TOIMINNAN ABSTRAKTIOITA

Edellisessä luvussa käsiteltiin mahdollisia syitä siihen miksi takaisinkutsumenetelmä asynkronisessa toiminnassa voi aiheuttaa ongelmia. Takaisinkutsujen hallitseminen voi olla hyvin vaikea ongelma jopa kokeneille ohjelmoijille ja tarkoituksenmukaisten abstraktioiden tarve on selvästi olemassa (Bainomugisha ym., 2013). Tämä luku esittelee joitakin kehitettyjä abstraktioita, joiden avulla virheettömämmän ohjelman kirjoittamisen on esitetty olevan ohjelmoijalle helpompaa. Ensimmäinen alaluku käsittelee abstraktiotason nostamista yleisesti JavaScript-kielessä. Tässä yhteydessä käsitellään lyhyesti myös JavaScript-ohjelmia varten kehitettyjä kirjastoja ja kehyksiä. Toinen ja kolmas alaluku keskittyvät lupausabstraktioon ja sen esitettyihin hyötyihin. Neljäs alaluku käsittelee joitakin muita kirjallisuudessa esitettyjä menetelmiä joilla takaisinkutsumenetelmän tuomia ongelmia voitaisiin välttää.

3.1 Abstraktiotason nostaminen ja kirjastot

JavaScript-ohjelmille tarkoitettuja kehyksiä ja kirjastoja on valtavasti ja asynkronisen toiminnallisuuden avuksi löytyy tuhansia komponentteja (Gallaba ym., 2015). Kehysten ja kirjastojen välinen ero on häilyvä. Perinteisesti erona on pidetty kehysten tuottamaa käänteistä kontrollia. JavaScript-kielen yhteydessä tämä jaottelu on ehkä ongelmallinen, kun kielen ohjelmat suoritetaan aina jonkin ympäristön sisällä ja siten koko ohjelma perustuu käänteiseen kontrolliin, jossa ympäristössä on aina kontrolli. Selvyyden vuoksi tässä työssä käytetään sekä kirjastoista että kehyksistä termiä kirjasto.

JavaScript-ohjelmia on mahdollista kirjoittaa ilman minkäänlaisia kirjastoja. Kirjastojen suureen määrään on kuitenkin syynsä. Niiden on todettu helpottavan ohjelmointityötä mm. abstraktoimalla pisimpiä ja monimutkaisimpia operaatioita, lyhentämällä operaatioiden kirjoittamiseen tarvittavaa koodimäärää ja varmistamalla ohjelman toimimisen eri selainympäristöissä (Graziotin & Abrahamsson, 2013). Muiden monimutkaisten operaatioiden ohella myös asynkroni-

sia toimintoja on pyritty helpottamaan hyödyntämällä niitä kirjastojen kautta.

Async.js (McMahon, 2016) on avoimen lähdekoodin JavaScript-kirjasto joka tarjoaa rajapintoja asynkronisten, takaisinkutsutyylisten funktiokutsujen hallitsemiseen (Gallaba ym., 2015). Listaus 4 esittää kuinka sama ohjelmakoodi on toteutettu tavallisella takaisinkutsumallilla, async.js kirjastoa hyödyntäen sekä seuraavassa alaluvussa tarkemmin esitettyä lupausabstraktiota hyödyntäen.

jQuery on yleisesti käytössä oleva ja hyvin tunnettu kirjasto JavaScript-kielelle (jQuery, 2016). Kirjasto tarjoaa ohjelmien rakentamiseen korkeamman abstraktiokerroksen, piilottaen monia työteliäitä ja virheherkkiä vaiheita ohjelmointityössä. Kirjaston tarjoamien apuvälineiden laaja kirjo on ainakin yhtenä syynä sen suureen suosioon (Chaffer & Swedberg, 2013). Kirjasto tarjoaa myös lupausmenetelmän mukaisia rajapintoja asynkronisten toimintojen suorittamiseen. Valitettavasti nämä eivät kuitenkaan ole nykyisen lupausstandardin mukaisia.

LISTAUS 4: Toiminnallisuus toteutettuna takaisinkutsumallilla, async.js-kirjastolla ja Promise-mallilla. (Gallaba ym., 2015, 3)

```
1   // Takaisinkutsumallilla toteutettu toiminnallisuus.
2   $("# button "). click (function() {
3     promptUserForTwitterHandle (function( handle ) {
4       twitter . getTweetsFor ( handle , function( tweets ) {
5         ui . show ( tweets );
6       }) ;
7     }) ;
8   }) ;
9
10  // Sama toiminnallisuus toteutettuna Async.js kirjaston watrefall-metodilla.
11  $("# button "). click (function() {
12    async . waterfall ([
13      promptUserForTwitterHandle ,
14      twitter . getTweetsFor ,
15      ui . show
16    ]
17    , handleError );
18  }) ;
19
20  // Sama toiminnallisuus toteutettuna Promise-mallilla.
21  $("# button "). clickPromise ()
22    . then ( promptUserForTwitterHandle )
23    . then ( twitter . getTweetsFor )
24    . then ( ui . show );
```

3.2 Lupaukset ja Promises/A+

Lupaus-malli on abstraktio jolla takaisinkutsujen kääntämä kontrolli (engl. inversion of control) käännetään takaisin (ns. un-inversion of control), jolloin päädytään kontrollin osalta lähemmäs synkronista ohjelman kulkua. Lupausmallia onkin esitetty helpottamaan takaisinkutsumallin muodostamia ongelmia (Kambona ym., 2013; Brodu, Frénot & Oblé, 2015; Gallaba ym., 2015). Mallia on Kambonan ym. (2013) mukaan hyödynnetty monissa ohjelmointikielissä ja ympäristöissä, nimityksen hieman vaihdellen. Lupausmenetelmän juuret ovatkin vuosikymmenien takana (Friedman & Wise, 1976; Liskov & Shrira, 1988).

Promises/A ja Promises/A+ ovat kaksi spesifikaatiota jotka määrittelevät Java-

Script-kielen lupausmenetelmän (Li, Mickens, Nath & Ravindranath, 2015). JavaScript-kieleen lupaus-abstraktio tuli ensin kirjastojen kautta, kun erilaiset kirjastot ryhtyivät tarjoamaan lupaustyylisiä rajapintoja erilaisten asynkronisten toiminnallisuuksien toteuttamiseen. Vast'ikään virallistettu ECMAScript 6 -standardi toi lupaus-abstraktion JavaScript-kielen natiiviksi ominaisuudeksi (ECMA, 2015). Kielen ominaisuudeksi valittiin Promises/A+ -spesifikaation mukainen malli (Promises/A+, 2015).

Lupausmallin heikkoutena JavaScript-kielessä ovat moninaiset, hieman toisistaan poikkeavat toteutukset (Kambona ym., 2013). Vanhempi Promises/A -spesifikaatio on valitettavan lyhyt ja monitulkintainen. Sitä vaihtelevasti tulkitsemalla on tehty lukuisia toteutuksia lupausabstraktiosta eri kirjastoihin. Esimerkiksi erittäin yleinen jQuery-kirjasto toteuttaa lupaukset sitä omalla tavallaan tulkiten (Otero & Larsen, 2012). Domenic Denicola esittää väärinymmärrettyjen lupausten ongelman seikkaperäisessä verkkoartikkelissaan (Denicola, 2012). Vaikka uudempi Promises/A+ -spesifikaatio määrittelee lupaukset tarkasti, väärinymmärryksiin perustuvat toteutukset ovat liian laajalti hyödynnettyjä, mm. jQuery-kirjaston myötä, että virhettä voisi helposti korjata. Parkerin (2015) mukaan eri kirjastojen lupausabstraktioiden eroavaisuudet ovat kuitenkin triviaaleja ja helposti ymmärrettäviä. Ne eivät kuitenkaan ole välttämättä suoraan yhteentoimivia ja niiden virheidenkäsittely voi poiketa toisistaan. Toteutukset ovat voineet kokonaan kadottaa lupausabstraktion pointin (Denicola, 2012) ja tämä on ehkä aiheuttanut suurta vahinkoa JavaScript-kielen lupausabstraktion omaksunnassa ja leviämisessä.

Lupaus on asynkronisen toiminnallisuuden abstraktio, eli se ei tarjoa mitään sellaista ominaisuutta JavaScript-kieleen, mitä ei saisi aikaan myös perinteisesti pelkkiä takaisinkutsuja hyödyntämällä (Simpson, 2015). Abstraktiossa asynkronisen toiminnallisuuden toteuttava rajapinta palauttaa erityisen objektin, jonka avulla takaisinkutsufunktio voidaan välittää ja joka sisältää hyödyllisiä ominaisuuksia asynkronisen toiminnallisuuden hallitsemiseen. Tämä erityinen objekti on nimeltään Promise. Lupaus-abstraktiossa takaisinkutsumalli siis säilyy, mutta sitä hyödynnetään kuitenkin hieman toisella tavalla (Simpson, 2015). Käyttäjälle malli on yksinkertainen, kun sen on ensin ymmärtänyt. Taustalla on kuitenkin kohtalaisen monimutkainen abstraktio, jonka ympäristö tarjoaa sovellusten käyttöön.

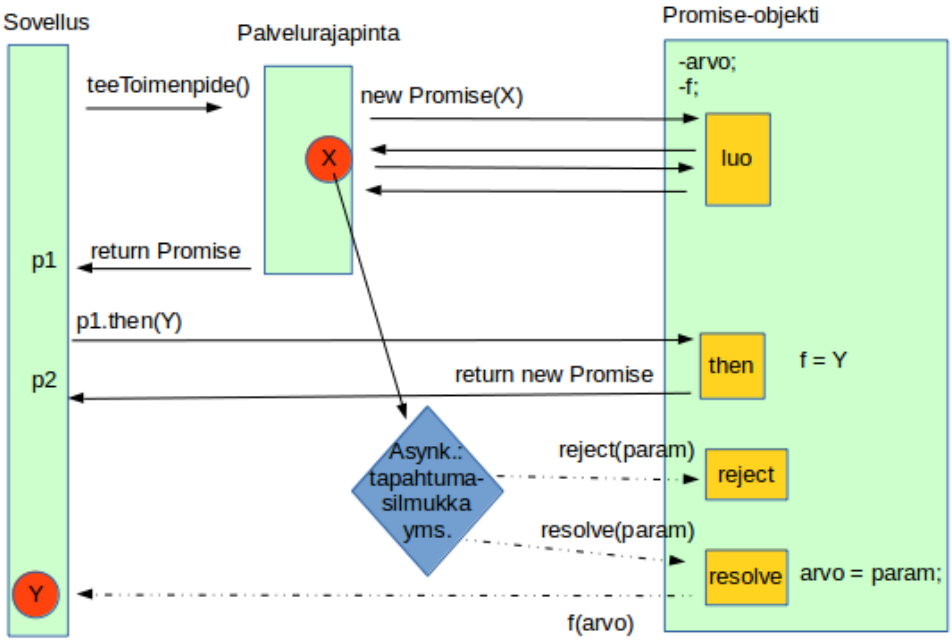
Promise on objekti, joka käärii sisäänsä sekä arvon että takaisinkutsufunktion. Takaisinkutsufunktio asetetaan objektille synkronisesti then-funktiolla. Talletettu takaisinkutsufunktio tulee kutsutuksi, kun asynkroninen toiminto on valmistunut. Samalla Promise saa myös arvon, joka ei voi enää muuttua (Kambona ym., 2013). Näin tätä objektia voi huoletta välittää toisille objekteille jotka tarvitsevat siihen talletettua tulosta, sillä sitä ei voi vahingossa tai tahallaan muuttaa. Promise-objektilla on myös tila, joka voi olla kolmenlaisessa tilassa. Se voi olla määrittelemätön (undefined) jossa se on ennen valmistumistaan. Sitten tuloksen saavuttua se voi olla hylätty (rejected) tai ratkaistu (resolved).

Kuvio 3 havainnollistaa karkealla tasolla tätä hieman aivoja venyttävää mal-

lia. Kuviossa esitetään yksinkertaistettu näkymä listauksen 5 kutsujen etenemisestä, eikä siihen ole otettu virheenkäsittelyä mukaan. Kuvioon on pyritty kuvaamaan joitakin kutsuja joita tuollaisessa tapauksessa voisi tapahtua, varsinainen asynkroninen toiminta on yksinkertaistettu salmiakkikuvion sisään.

LISTAUS 5: Yksinkertainen Promise-rajapinta

```
1   //Sovellus suorittaa asynkronisen palvelukutsun
2   teeToimenpide()
3       .then(function(tulos) {    // kuvan Y-funktio
4               alert(tulos);
5           })
6
7   //Funktiomäärittely simuloi jotain palvelurajapintaa, joka
8   //toteuttaa jonkin aikaavievän prosessin, kuten palvelinkutsun. Esimerkin
9   //vuoksi tässä kuitenkin vain odotetaan 2 sekuntia.
10  function teeToimenpide(){
11    return new Promise(function(resolve, reject) { // kuvan X-funktio
12        setTimeout(function(){      // ei näy kuvassa
13            resolve("Valmis");
14        },2000);
15    });
16  }
```



KUVIO 3: Yksinkertaistettu Lupausmalli

Kuvion 3 funktiot X ja Y ovat sovelluksen tai palvelurajapinnan luomia, anonyymeja takaisinkutsufunktioita. Funktio X tulee kutsutuksi synkronisesti, Promise-objektin luomisen yhteydessä ja se määrittelee asynkronisesti kutsuttavan takaisinkutsufunktion. Asynkronisen takaisinkutsun vaativa toimenpide käynnistetään (setTimeout-operaatio) ja palataan synkronisesti takaisin sovelluksen koodiin. Siellä kutsutaan synkronisesti then-funktiota, jossa asetetaan funktio Y Promiselle tiedoksi tulevaa varten. Tapahtumien kuuntelijan ja tapahtumasilmukan

myötä työn valmistuttua kutsutaan parametrina saatua resolve-funktiota, eli tehdään takaisinkutsu Promise-ympäristöön. Parametrina viedyn tiedon myötä Promise saa arvon, joka ei voi enää koskaan muuttua. Resolve-funktio suorittaa aiemmin asetetun then-funktioon liittyvän funktioY:n ja välittää arvon parametrina. Funktio Y tulostaa tässä tapauksessa arvon näytölle. Yhtä hyvin se voisi suorittaa mitä tahansa muita toimintoja. Uuden asynkronisen toiminnon suorittaminen edellisen tiedon perusteella onnistuu tästä hyvin, tällöin funktio muodostaisi uuden Promise-objektin ja tieto välittyisi then-funktioita ketjuttamalla kuten listauksessa 4.

Kuvion 3 esimerkissä sovelluksessa määritellään yksi takaisinkutsu, then-funktiolle asetettu funktioparametri. Taustalla takaisinkutsuja muodostetaan kuitenkin useampia. Lupaus-abstraktion ja toimenpiteen tarjoavan rajapinnan välillä on kaksisuuntainen takaisinkutsujen menettely. Ensin Promise-objektin luomistilanteessa suoritetaan synkroninen takaisinkutsu jolloin asynkroninen toiminnallisuus käynnistetään. Tässä yhteydessä takaisinkutsuja asettaa parametreiksi resolve- ja reject-funktiot, joista toinen tulee asynkronisesti takaisinkutsutuksi asynkronisen operaation jälkeen. Näin lupausmalli sulkee sisäänsä natiivin asynkronisen kutsun tarjoten tätä hyödyntävälle sovellukselle vastaavan Promise-rajapinnan. Lupausmallissa takaisinkutsuja hyödynnetään toimivana, mutta hieman monimutkaisena työkaluna (Simpson, 2015).

ES6-spesifikaatio määrittelee pari apufunktiota. Näitä ovat Promise.all, joka ottaa parametrina listan Promiseja ja suoritus siirtyy then-funktioon, kun kaikki on suoritettu. Toinen on Promise.race, joka toimii muuten vastaavasti, mutta siirtyy then-funktioon kun ensimmäinen valmistuu. Eri kirjastot ovat toteuttaneet monenlaisia vastaavia apufunktioita, mutta toistaiseksi standardi määrittelee nuo kaksi.

Promise-objekteja voidaan siis ketjuttaa siten että edellisen funktion tuloksena saadut tiedot ovat seuraavan käytössä. ES6-spesifikaatio ei määrittele mitään luonnollista mekanismia keskeyttää ketjua. Jos esimerkiksi jokin muuttuja saa sellaisen arvon, ettei ketjua olisi enää järkevää jatkaa, on ainut mahdollisuus toteuttaa virhemenettely. Tämän voidaan ajatella olevan virhemenettelyn väärinkäyttöä kun kyse on tavallisesta ohjelman toiminnan haarasta.

3.3 Promise-abstraktioon liitettyjä hyötyjä

Tässä alaluvussa käsitellään joitakin JavaScript-kielen lupausabstraktion esitettyjä hyötyjä suhteessa alaluvussa 2.3 esitettyihin takaisinkutsumenetelmän muodostamiin ongelmiin.

3.3.1 Sisäkkäiset funktiot

Promise-malliin on sisäänrakennettu näppärä ketjuttamistapa. Tämä tarkoittaa sitä, että asynkronisia kutsuja voi ketjuttaa siten että seuraava saa edellisen paluuarvon parametrina ja että then-funktio palauttaa uuden Promise-objektin joilloin ohjelmakoodi ei vaadi lukuisia sisäkkäisiä funktioita. Listaus 4:n promise-

esimerkissä näkyy että koodi ei ketjutuksen vuoksi lähde sisentymään eikä siten muodosta tuomionpyramidia, vaan pysyy helpommin ymmärrettävänä ja luettavana. Mallin tuomaa helpotusta sisäkkäisyyksiin on kuitenkin myös kritisoitu siitä, että tietyissä tapauksissa myös Promise-ketjut joudutaan rakentamaan sisäkkäisiksi jolloin takaisinkutsumallia vastaava ongelma toistuu (Kambona ym., 2013).

3.3.2 Virheidenkäsittely

Virheidenkäsittely on takaisinkutsujen kanssa hyvin hankalaa ja aiheuttaa paljon ohjelmointia ja manuaalisesta työstä johtuvia virheiden mahdollisuuksia. Asynkronisten kutsujen kanssa ei voi hyödyntää synkronisessa ohjelmoinnissa käytettyä try-catch-mallia. Promisemalliin on kuitenkin rakennettu oma virheidenkäsittelymekanisminsa, joka jäljittelee toiminnallisuudeltaan try-catch-mallia. Ohjelmoija voi määritellä promisen then-funktioon myös takaisinkutsufunktion virhetilanteen varalle. Toinen tapa on määritellä kutsuketjun viimeiseksi catch-funktio, joka nappaa kaikista ketjun Promise-objekteista muodostuneet virheet. Aivan samoin kuin try-catch-mallissa, myös tässä on mahdollisuus virheisiin ja käyttää rakennetta väärin. Esimerkiksi virheiden huomiotta jättäminen on helppoa, mikäli ne otetaan kiinni, mutta niitä ei käsitellä asianmukaisesti (Simpson, 2015).

Promise-mallin virheidenkäsittely aiheuttaa kuitenkin helposti virheellisesti toimivia ohjelmia (Simpson, 2015). Esimerkiksi listauksen 6 (Simpson, 2015) esimerkissä näyttäisi siltä että virhe tulee käsitellyksi, mutta näin ei kuitenkaan ole. Rejected-funktioon tullaan, mikäli ensimmäisen Promise-objektin muodostamisessa (rivi 1) tapahtuisi virhe. Esimerkissä virhe tapahtuu then-funktiossa (rivi 7) jota kutsutaan kun Promise on jo valmis. Tämän virheen voi ottaa kiinni seuraavassa then tai catch -funktiossa, jota ei kuitenkaan tässä tapauksessa ole olemassa. Promisen virheidenkäsittelyn yksi heikkouksista onkin juuri se että virheet on liian helppoa kadottaa (Simpson, 2015).

LISTAUS 6: Promise-mallin virheidenkäsittelyn ongelma. (Simpson, 2015)

```
1   var p = Promise.resolve( 42 );
2
3   p.then(
4     function fulfilled(msg){
5         // numbers don't have string functions,
6         // so will throw an error
7         console.log( msg.toLowerCase() );
8     },
9     function rejected(err){
10        // never gets here
11    }
12  );
```

3.3.3 Luottamuskysymykset

Luottamusongelman parantuminen on ehkä yksi tärkeimmistä parannuksista joita Promise-malli Simpsonin (2015) mielestä tuo. Promise voidaan ratkaista (resolve) vain kerran, ja sitten sen arvo on muuttumaton (engl. immutable) ja sen voi tarkastaa kuinka monta kertaa vain. Tämän arvon voi siten välittää muille

komponenteille ja kolmansille osapuolille tietäen ettei sitä voida vahingossakaan muuttaa. (Simpson, 2015.)

Luottamukseen liittyy olennaisesti se miten takaisinkutsu tulee kutsutuksi. Kutsutaanko sitä liian aikaisin, liian myöhään, liian harvoin, liian useasti tai ei ehkä ollenkaan (Simpson, 2015). Promise-malli takaa että joko takaisinkutsu tai virhekutsu tulevat kutsutuksi kerran ja vain kerran (Promises/A+, 2015). Promises/A+ määrittelee myös että takaisinkutsujen täytyy tulla kutsutuksi asynkronisesti. Promises/A:n mukaan takaisinkutsu voi sen sijaan tulla kutsutuksi synkronisesti tai asynkronisesti (Li ym., 2015).

3.3.4 Perustuu sivuvaikutuksiin

Promise-mallissa asynkronisesti toimiva funktio palauttaa arvon, promise-objektin. Tällöin toiminta ei perustu sivuvaikutuksille, vaan funktion palauttamalle arvolle. (Simpson, 2015.)

3.4 Muita menetelmiä

Reaktiivinen ohjelmointi on malli joka sopii hyvin tapahtumapohjaisten sovellusten kehittämiseen (Bainomugisha ym., 2013). Se on ikäänkuin tapahtumapohjaisen mallin erikoistapaus, jossa keskiössä on tieto, jonka muuttuminen aiheuttaa tapahtuman. Sen hyödyntämistä on esitetty takaisinkutsumenetelmän tuomien ongelmien helpottajaksi (Kambona ym., 2013; Bainomugisha ym., 2013).

Yksinkertainen esimerkki reaktiivisesta ohjelmoinnista on lause ”x = a + b”. Ei-reaktiivinen ohjelmointiympäristö toimii siten, että kun x on kerran määritelty, sen arvo ei muutu mikäli a tai b muuttuu. Reaktiivinen ohjelmointiympäristö taasen toimii siten, että a:n tai b:n muuttuessa myös x muuttuu. Jos vielä lisäksi olisi määritelty ”y = x - 5”, niin esim. a:n muuttuessa muuttuisi x, jonka muuttuessa muuttuisi myös y. Muuttujat ovat toisistaan riippuvaisia ja muodostavat riippuvuuksien verkon. Vastaava toiminnallisuus on olemassa taulukko-ohjelmissa (engl. spreadsheets), jotka ovat ehkä maailman yleisin loppukäyttäjien hyödyntämä ohjelmointikieli (Bainomugisha ym., 2013). Nämä tyypillisesti koostuvat soluista jotka sisältävät arvon tai kaavan. Kaavat lasketaan automaattisesti uudelleen, mikäli arvot muuttuvat.

Reaktiivinen ohjelmointi onkin pohjimmiltaan taulukko-ohjelmatyylisen mallin sisällyttämistä ohjelmointiympäristöön (Bainomugisha ym., 2013). Reaktiivista ohjelmointia voidaan toteuttaa käyttämällä reaktiivista ohjelmointikieltä tai hyödyntämällä reaktiivista kirjastoa apuna. Eri reaktiiviset kirjastot eivät kuitenkaan ole välttämättä yhteneväisiä käyttämiensä termien tai toiminnallisuuksien osalta. (Kambona ym., 2013.) JavaScript-kielen määrittely ei tunne reaktiivisuutta eli se ei ole reaktiivinen ohjelmointikieli. Vaihtoehtoina on joko kirjoittaa ohjelma soveltuvalla kielellä joka käännetään JavaScriptiksi tai hyödyntää soveltuvia kirjastoja mahdollistamaan reaktiivisen ohjelmoinnin (Mogk, 2015). Reaktiivisilla sovelluksilla sen sijaan voidaan yleisesti tarkoittaa monenlaisia sovelluksia, eikä niiden toteutuksessa ole välttämättä mitenkään hyödynnetty reaktiivista ohjel-

mointia.

Muita kirjallisuudesta tavattuja menetelmiä ovat esimerkiksi generaattorit, joka on ECMAScript 6 -standardin (ECMA, 2015) määrittelemä uusi funktiotyyppi. Se tuo kieleen yield-toiminnon, jolla tällaisen funktion toiminnan voi keskeyttää kesken suorituksen siten että funktion tila säilyy uudelleenkäynnistystä varten (Mickens, 2014). Tämä menetelmä yhdistettynä lupausabstraktioon mahdollistaa Simpsonin (2015) mukaan mahdollisuuden ohjelmoida asynkroniset ohjelmat helpommin ymmärrettävällä tavalla, ikäänkuin ohjelmoisi peräkkäin tapahtuvaa (engl. sequential), synkronista ja estävää ohjelmakoodia.

4 SUOMALAISTEN WEB-SIVUSTOJEN AUTOMATISOITU TUTKIMUS

Tutkimuksen tavoitteena on kartoittaa verkkosivustojen sisältämän JavaScriptin ominaisuuksia takaisinkutsuongelmiin liittyen. Verkkosivuilta tutkittiin JavaScript-ohjelmakoodia HTML- ja JavaScript-tiedostoista. Tätä varten räätälöitiin tietojärjestelmä, joka suoritti aineiston keräyksen ja mittauksen. Seuraavissa alaluvuissa kerrotaan tarkemmin tutkimusmenetelmästä, tutkimusaineiston keräämisestä ja tutkimuksen toteuttamisesta.

4.1 Tutkimusmenetelmä

Tutkimuksen menetelmäksi valittiin kartoittava ja kvantitatiivinen tutkimusote ja ohjelmallinen aineistonhankinta. Vilkan (2007) mukaan kartoittavan tutkimuksen tavoitteena on etsiä aihealueesta uusia näkökulmia. Sen avulla voidaan tutkia vähän tunnettuja asioita sekä kehittää hypoteeseja. Kvantitatiivinen menetelmä on tutkimustapa jossa tutkittavia asioita ja niiden ominaisuuksia käsitellään numeerisesti. (Vilkka, 2007.)

Verkkosivujen sisältämän JavaScript-ohjelmakoodin määrä ja sisältö on vain vähän tutkittu alue. Kartoittavalla ja kvantitatiivisella tutkimuksella pyritiin lisäämään tietoa verkkosivujen sisältämän ohjelmakoodin määrällisistä ominaisuuksista. Aineiston ohjelmallisella keräyksellä pyrittiin mahdollisimman suureen yhteismitallisuuteen havaintoyksiköiden välillä.

Tutkimuksen perusjoukko on sellaiset suomenkieliset verkkopalveluita sisältävät verkkosivustot, jotka ovat tutkimuksen kannalta riittävän saavutettavia. Riittävällä saavutettavuudella tarkoitetaan sivustoja jotka on mahdollista ainakin osittain ladata avoimesta internetistä automaattisin ja sallituin menetelmin. Tutkimukseen valittiin sivustoja, joita on mahdollista käyttää suomen kielellä ja jotka tarjoavat käyttäjille muun sisällön lisäksi myös operatiivista verkkopalvelua. Operatiivisella verkkopalvelulla tarkoitetaan sellaista palvelua jota hyödyntämällä tapahtuu jokin muutos internet-järjestelmän ulkopuolella, kuten esimerkiksi rahan tai tuotteen siirtyminen paikasta toiseen.

Tutkittavat sivustot jaoteltiin muutamiin kategorioihin niiden tarkoituksen mukaan. Kategorioiden valinnassa keskityttiin sellaisiin sivustoryhmiin jotka voisivat väärin toimiessaan tuottaa taloudellisia tai yksityisyyden suojaan liittyviä ongelmia. Jokaiseen kategoriaan pyrittiin hankkimaan noin 30 sivustoa, jotka olivat vapaasti saavutettavissa ja tuottivat tutkimuksen menetelmillä riittävästi tutkittavaa aineistoa. Tutkittavat kategoriat olivat finanssisektori, rahapelit, pikavipit, verkkokaupat ja julkisen sektorin palvelut. Näiden tarkemmat kuvaukset löytyvät taulukosta 2. Julkisen sektorin palveluiden osalta saavutettavien sivustojen lukumäärä jäi vaatimattomaksi, vaikka sivustoja yritettiin ladata vastaavasti kuin muitakin kategorioita. Tämän kategorian sivustot olivat selkeästi muita useammin sellaisia, joita ei voinut tutkimuksen menetelmillä saavuttaa.

TAULUKKO 2: Tutkittujen sivustojen kategoriat ja niiden kuvaukset

Kpl	**Kategoria**	**Kuvaus kategorian sivustoista**
33	Finanssi-sektori	Suomen finanssivalvonnan valvomia toimijoita, kuten talletuspankit, vakuutuslaitokset ja pääomamarkkinoilla toimivat tahot. Sivustot valittiin siten että haettiin finanssivalvonnan valvomia toimijoita, joiden verkkosivut sisälsivät verkkopalvelun. Esimerkiksi pankin verkkopalvelu on verkkopankki, joten tutkimukseen voitiin valita sellaisten pankkien sivustoja, joilla on sivustoillaan verkkopankki.
28	Pikavipit	Verkossa toimivia suomenkielisiä pikavippisivustoja, joita haettiin sekä Googlen hakupalvelusta että pikavippisivustoja listaavilta sivustoilta. Pienlainoittajat eivät yleensä tarvitse toimilupaa ja siten eivät kuulu finanssivalvonnen piiriin. Pienlainoittajien tulee kuitenkin nykyisin rekisteröityä luotonantajarekisteriin. Tutkimuksessa tutkittiin verkkosivuja, mutta niiden yhteyttä luotonantajarekisteriin ei tarkistettu. Sivuston todellisia toimijoita ei tarkastettu ja on mahdollista että sama toimija ylläpitää useita erinimisiä pikavippisivustoja. Tutkimukseen otettiin sivustoja jotka toteuttivat luototuspalvelua tai kilpailuttivat niitä siten että hakemus henkilötietoineen tehdään sivuston kautta.
32	Rahapelit	Verkossa toimivia suomenkielisiä rahapelisivustoja, joita haettiin mm. Google-hakukoneella. Tutkimuksessa ovat mukana kaikki suomalaisen monopolin kolme rahapelisivustoa: RAY, Veikkaus ja Fintoto. Muiden rahapelisivustojen alkuperää ei tarkastettu.
8	Julkinen sektori	Suomalaisia julkisen sektorin sähköisiä palveluita tarjoavia sivustoja. Tutkitut sivustot jäivät määrältään vähäisiksi koska suurin osa sivustoista ei tuottanut riittävästi aineistoa tutkimuksen aineistonkeräysmenetelmällä.
33	Verkko-kaupat	Verkkokauppoja joissa on myös suomenkielinen käyttöliittymä. Tutkimukseen on pyritty valitsemaan erityyppisiä verkkokauppoja.
134	Yhteensä	

Tutkimuksen havaintoyksikkö on sivusto. Havaintoyksikölle asetettiin rajaus, jonka mukaan hyvin vähän funktiokutsuja sisältäviä sivustoja ei valittu tutkimukseen mukaan vaikka ne olisi saatu ladattua. Rajauksen määrityksessä harkittiin tiedostojen lukumääriä. Käytännössä kuitenkin havaittiin, että joillain sivustoilla oli hyvin suuria tiedostoja jotka sisälsivät paljon koodia ja joillain sivustoilla oli vähäinen määrä koodia jaoteltuna moniin tiedostoihin. Toinen mahdollinen rajaus olisi ollut koodirivien määrät. Yhteismitallistavien toimien vuoksi tutkimuksen sivustojen koodirivit ovat verrattavissa toisiinsa ja niiden perusteella rajaus olisi mahdollinen. Tutkimuksessa kuitenkin pääasiassa mitattiin funktioita ja niiden esiintyvyyttä. Näin päädyttiin käyttämään rajauksessa funktiokutsujen määrää. Mikäli sivuston mittauksessa saatiin tulokseksi alle 1000 funktiokutsua, se hylättiin tutkimuksesta. Lähes poikkeuksetta tällaiset tilanteet olivat sivustoja joilta ei saatu ladattua kuin esim. yksi index.html.

4.2 Tutkimuksen muuttujat

Kvantitatiivinen tutkimus on mittaamista ja mittareiden luominen on tutkimuksen onnistumisen edellytys. Muuttuja tarkoittaa mitattavaa käsitettä. (Kananen, 2015) Taulukoissa 3, 4, 5 ja 6 kuvataan tutkimuksen muuttujat ja niihin liittyvät indikaattorit. Taulukossa 7 kuvataan muuttujat jotka olisi haluttu mitata, mutta joiden mittaaminen ei onnistunut. Taulukossa 8 kuvataan muuttujista johdetut arvot ja niiden laskentakaavat. Taulukossa 9 esitetään sisäkkäisyyksiin liittyviä arvoja.

Taulukossa 3 kuvataan havaintoyksikön perustietoja kuten nimi, tiedosto- ja rivimäärät. Yuen ja Wangin (2013) tutkimuksen mukaan HTML-sivuihin sisällytetty JavaScript on erittäin yleistä (yli 90 %:ssa sivustoista). Tästä syystä tutkimuksessa ollaan kiinnostuneita myös HTML-sivuihin sisällytetyn JavaScript-koodin ominaisuuksista. Ennen ohjelmakoodirivien laskemista tiedostoille on suoritettu ohjelmakoodin yhteismitallistavia toimia, kuten yhtenäinen rivitys. Rivimääriin ei lasketa tyhjiä rivejä eikä pelkkiä kommentteja.

Taulukossa 4 esitetään muuttujia jotka kuvaavat havaintoyksiköstä laskettujen funktiokutsujen määriä. Määrät lasketaan erikseen JavaScript-tiedostoista ja HTML-tiedostoihin sisällytetystä JavaScriptistä. Funktiokutsujen laskennassa hyödynnetään osittain Gallaban ym. (2015) toteuttamia komponentteja, joihin viitataan myös heidän artikkelissaan (Gallaba ym., 2015). Näistä komponeneista käytetään jatkossa myös termiä GMB-komponentit.

Funktiokutsukohtien osalta tutkimuksessa mitataan kaikki löydetyt funktiokutsukohdat. Näistä lasketaan sellaiset funktiokutsukohdat jotka lähettävät parametrina funktion, eli takaisinkutsulliset funktiokutsukohdat. Edelleen näistä lasketaan sellaiset funktiokutsukohdat jotka lähettävät parametrina anonyymin funktion. Lisäksi on laskettu kaikki funktioparametrit ja anonyymit funktioparametrit; niiden määrä on isompi kuin vastaavien funktiokutsukohtien määrä mikäli funktiokutsun parametrina annetaan useampia funktioita. Nämä funktiokutsujen laskennat on toteutettu uutena toiminnallisuutena mittausohjelmistoon.

TAULUKKO 3: Sivuston perustietoja kuvaavat muuttujat

No.	Muuttujan kuvaus	Indikaattori
1	Havaintoyksikön eli sivuston yksilöivä nimi	Sivuston URL eli merkkijono jolla tutkittava sivusto on haettu verkosta.
2	Tutkittujen JavaScript-tiedostojen määrä.	Tutkittujen JavaScript-tiedostojen lukumäärä. Mukana on kaikki JavaScript-tiedostot jotka olivat saavutettavissa sivustolta.
3	Tutkimusta varten ladattujen HTML-sivujen kokonaismäärä.	Tutkimusta varten tutkimuskoneelle ladattujen HTML-sivujen kokonaismäärä. Tästä joukosta on otettu satunnaisotanta tutkimusta varten.
4	Tutkittujen HTML-tiedostojen määrä.	Tutkittujen HTML-tiedostojen määrä. Tutkitut HTML-sivut on valittu satunnaisotannalla kaikista ladatuista HTML-sivustoista. Arvo on 1000, mikäli HTML- sivuja on 1000 tai enemmän.
5	JavaScript-koodirivien määrä (loc) JavaScript-tiedostoissa.	JavaScript-ohjelmakoodirivien lukumäärä. Arvo ei sisällä HTML-tiedostoihin sisällytettyä JavaScriptiä.
6	JavaScript-koodirivien määrä (loc) HTML-tiedostoissa.	JavaScript-ohjelmakoodirivien lukumäärä. JavaScript-ohjelmakoodi on erotettu HTML-ohjelmakoodista ohjelmallisesti. Arvo ei sisällä HTML-koodia.
7	HTML-koodirivien määrä (loc) HTML-tiedostoissa.	HTML-tiedoston ohjelmakoodirivien lukumäärä, josta on ensin erotettu pois JavaScript-ohjelmakoodi. Arvo ei sisällä JavaScript-koodia.

Kuitenkin toteutuksessa on joissain kohdissa myös hyödynnetty Gallaban ym. (2015) ohjelmakoodia.

Taulukossa 5 kuvataan havaintoyksikköön liitettyjä muuttujia, jotka kuvaavat funktioiden määrittelyjen lukumääriä sivustoilla. Näistä lasketaan erikseen sellaiset funktioiden määrittelyt, jotka toteuttavat takaisinkutsun eli kutsuvat parametrinaan saamaansa funktiota. Funktioiden laskenta perustuu Gallaban ym. (2015) toteuttamalle komponentille.

Taulukossa 6 kuvataan muuttujia, jotka esittävät promise-toiminnallisuuden sekä async.js- ja jQuery- kirjastojen esiintymismääriä. Esiintymät on haettu merkkijonohauilla, jotka kuvataan taulukossa. Promise-mallin osalta on selvää, että mikäli then()-funktiokutsuja ei löydy, ei Promise-mallia ole käytetty. Toisinpäin tulkinta ei ole yhtä selvää, sillä ohjelmakoodissa voi olla then -nimisiä funktioita jotka eivät liity promise-toiminnallisuuteen. Kirjastojen osalta löydös sen sijaan tarkoittaa todennäköisesti sitä että kirjastoa on käytetty. Esiintymisten suuri lukumäärä kertoo todennäköisesti sivustossa käytetyn kirjastoa melko paljon. Toi-

TAULUKKO 4: Funktiokutsuihin liittyvät muuttujat

No.	Muuttujan kuvaus	Indikaattori
8	Funktiokutsujen määrä JavaScript-tiedostoissa.	Funktiokutsujen kokonaislukumäärä JavaScript-tiedostossa.
9	Funktiokutsujen määrä HTML-tiedostoissa.	Funktiokutsujen kokonaislukumäärä HTML-tiedostossa. Funktiokutsut lasketaan HTML-tiedostoihin sisällytetystä JavaScript-koodista.
10	Takaisinkutsullisten funktiokutsujen määrä JS-tiedostoissa	Sellaisten funktiokutsujen määrä sivuston JS-tiedostoissa, jotka lähettävät parametrina vähintään yhden funktion.
11	Takaisinkutsullisten funktiokutsujen määrä HTML-tiedostoissa	Sellaisten funktiokutsujen määrä sivuston HTML-tiedostoissa, jotka lähettävät parametrina vähintään yhden funktion.
12	Anonyymien takaisinkutsullisten funktiokutsujen määrä JS-tiedostoissa	Sellaisten funktiokutsujen määrä sivuston JS-tiedostoissa, jotka lähettävät parametrina vähintään yhden anonyymin funktion.
13	Anonyymien takaisinkutsullisten funktiokutsujen määrä HTML-tiedostoissa	Sellaisten funktiokutsujen määrä sivuston HTML-tiedostoissa, jotka lähettävät parametrina vähintään yhden anonyymin funktion.
14	Tapahtuma-attribuuttien määrä	HTML-tiedostoissa HTML-elementtien attribuuttina määriteltyjen tapahtumakutsujen määrä. Esim. onclick="alert(’Hello’)". Muodostaa anonyymin, asynkronisen funktiokutsun.
15	Funktioparametrien määrä JS-tiedostoissa	Sivuston funktiokutsujen sisältämien kaikkien funktioparametrien määrä JavaScript-tiedostoissa.
16	Funktioparametrien määrä HTML-tiedostoissa	Sivuston funktiokutsujen sisältämien kaikkien funktioparametrien määrä HTML-tiedostoissa.
17	Anonyymien funktioparametrien määrä JS-tiedostoissa	Sivuston funktiokutsujen sisältämien anonyymien funktioparametrien kokonaismäärä JavaScript-tiedostoissa.
18	Anonyymien funktioparametrien määrä HTML-tiedostoissa	Sivuston funktiokutsujen sisältämien anonyymien funktioparametrien kokonaismäärä HTML-tiedostoissa.

TAULUKKO 5: Funktioiden määrittelyihin liittyvät muuttujat

No.	Muuttujan kuvaus	Indikaattori
19	Takaisinkutsuvien funktiomäärittelyjen määrä JavaScript-tiedostoissa	Niiden funktioimäärittelyjen lukumäärä JavaScript-tiedostoissa, joissa suoritetaan takaisinkutsu.
20	Takaisinkutsuvien funktiomäärittelyjen määrä HTML-tiedostoissa	Niiden funktiomäärittelyjen lukumäärä HTML-tiedostoihin sisällytetyssä JavaScript-koodissa, joissa suoritetaan takaisinkutsu.
21	Funktiomäärittelyjen määrä JavaScript-tiedostoissa	Kaikkien funktiomäärittelyjen määrä JavaScript-tiedostoissa.
22	Funktiomäärittelyjen määrä HTML-tiedostoissa	Kaikkien funktiomäärittelyjen määrä HTML-tiedostoihin sisällytetyssä JavaScript-koodissa.

TAULUKKO 6: Asynkronisiin abstraktioihin ja kirjastoihin liittyvät muuttujat

No.	Muuttujan kuvaus	Indikaattori
23	Promise toiminnallisuuden esiintyminen	Promise -toiminnallisuuteen liittyvien ”.then(” merkkijonojen määrä sivustolla.
24	Async.js-kirjaston esiintyminen	Async.js-kirjastoon liittyvien ”async.” -merkkijonojen määrä sivustolla.
25	jQuery-kirjaston esiintyminen	jQuery-kirjastoon liittyvien ”jquery” -merkkijonojen määrä sivustolla.

sinpäin tulkinta ei kuitenkaan ole mahdollista, sillä yksikin löytö voi tarkoittaa kirjaston olevan oikeasti laajalti käytössä. Lisäksi on mahdollista että kirjaston hyödyntäjä olisi nimennyt kirjaston uudelleen, jolloin kirjasto ei tulisi tässä tutkimuksessa ehkä lainkaan löydetyksi.

Taulukossa 7 esitetään muuttujia jotka jäivät tämän tutkimuksen ulkopuolelle. Nämä ovat sellaisia, joita olisi ehdottomasti haluttu tutkia, mutta joiden mittaaminen osoittautui liian työlääksi toteuttaa tämän tutkimuksen puitteissa. Erityisen harmillista on asynkronisten funktiokutsujen mittaamisen osoittautuminen GMB:n komponenteilla niin epäluotettavaksi, ettei menetelmää haluttu toistaa, eikä uuden kehittäminen ollut mahdollista enää tämän tutkimuksen puitteissa.

Taulukossa 8 on kuvattu tutkimuksen muuttujista johdetut, laskennalliset arvot ja niiden laskutavat. Useissa kaavoissa esiintyvä tekijä M3/M4 kompensoi sen, että kustakin sivustosta tutkitaan vain 1000 HTML-tiedostoa, jos niitä on enemmän kuin 1000, ja monissa sivustoissa niitä on hyvin paljon enemmän. Johdetuille arvoille on annettu kirjain-numero -tunnus. A: Takaisinkutsullisten funktiokutsujenosuus kaikista funktiokutsuista. B: Sellaisten funktiokutsujen osuus takaisinkutsullisista funktiokutsuista, jotka lähettävät parametrina vähintään yhden anonyymin funktion. Näitä sanotaan myös anonyymitakaisinkutsullisiksi funktiokutsuiksi. C: Tapahtuma-attribuuttien osuus kaikista funktiokutsuista. D:

TAULUKKO 7: Kiinnostavat, mutta puuttuvat muuttujat

Puuttuva muuttuja	**Muuttujan kuvaus**
Asynkronisten funktiokutsujen määrä JavaScript ja HTML-tiedostoissa	Sellaisten takaisinkutsullisten funktiokutsujen kokonaislukumäärä, jotka toteuttavat takaisinkutsun asynkronisesti.
Virhe ensin -menetelmän yleisyys	Sellaisten funktiomäärittelyjen määrä, jossa funktiolle tulee asettaa virhekoodi ensimmäiseksi parametriksi. Vastaavasti kiinnostavaa olisi myös tällaisten funktiokutsujen määrä.

Tapahtuma-attribuuttien osuus takaisinkutsullisista funktiokutsuista. E: Tapahtuma-attribuuttien osuus anonyymitakaisinkutsullisista funktiokutsuista. F: Anonyymien funktioparametrien osuus kaikista takaisinkutsullisista funktioparametreista. G: Takaisinkutsullisten funktiomäärittelyjen osuus funktiomäärittelyistä. H: Funktioparametrien määrä per takaisinkutsullinen funktiokutsu. Numero kuvaa sitä tiedostojen joukkoa josta arvo on laskettu: 1 JavaScript, 2 HTML tai 3 molemmat.

Taulukossa 9 kuvataan anonyymien takaisinkutsullisten funktiokutsujen sisäkkäisyyksiä. Funktiokutsu, jolle annetaan parametrina anonyymi funktio, lasketaan tason 1 funktiokutsuksi. Mikäli tämän anonyymin funktiomäärittelyn sisällä on funktiokutsu, joka sisältää anonyymin funktioparametrin, lasketaan tämä tason 2 funktioksi jne. Näin kaikki anonyymit funktioparametrit tulevat lasketuiksi jollekin tasolle. Sivustoilta on laskettu maksimisisäkkäisyys, sisäkkäisyyksien keskiarvo ja mediaani. Tuloksissa aineistosta on myös esitetty sisäkkäisyyksien jakaumat sivustoryhmittäin.

4.3 Tutkimusaineiston hankinta ja mittaus

Tutkimusaineiston hankinnassa tavoiteltiin tutkimuksen tarkkuutta. Kun kaikilta sivustoilta kerättiin samat tiedot samalla tavalla, voitiin minimoida satunnaisvirheiden määrä, joita usein muodostuu inhimillisten virheiden vuoksi manuaalisissa työvaiheissa. Tutkittavat sivustot olivat vapaasti saatavilla olevia verkkosivustoja, jotka olivat saavutettavissa mistä tahansa internettiin liitetystä tietokoneesta. Näistä seikoista johtuen tutkimusaineiston keräys toteutettiin ohjelmallisesti tutkimusta varten kehitetyllä tietojärjestelmällä eli mittausohjelmalla. Mittausohjelma kehitettiin hyvin automaattiseksi: sille voitiin antaa sivuston URL-osoite, jonka jälkeen mittausohjelma suoritti sivuston hakemisen, valmistelun, mittauksen ja tulosten kirjaamisen havaintomatriisiin automaattisesti.

Mittausohjelma mahdollisti myös työvaiheiden suorittamisen erikseen ja useille havaintoyksiköille peräkkäin. Tämä ominaisuus osoittautui erityisen hyödylliseksi, sillä ohjelmistoa kehitettiin koko tutkimusprosessin ajan ja oli hyödyllistä voida tutkia kaikki sivustot uudelleen, mikäli jokin mittausohjelmiston osa muuttui. Näin varmistettiin se, ettei joitakin sivustoja tutkittu erilaisella ohjelmistolla

TAULUKKO 8: Tutkimuksen muuttujista johdetut arvot

	Kuvaus	**Laskenta**
A	Takaisinkutsullisten funtiokutsujen osuus kaikista funktiokutsuista	
A1	JS-tiedostot	M10/M8
A2	HTML-tiedostot	M11/M9
A3	JS ja HTML-tiedostot	(M10+M11×M3/M4)/(M8+M9 × M3/M4)
B	Anonyymeja funktioparametreja lähettävien funktiokutsujen osuus takaisinkutsullisista funktiokutsuista	
B1	JS-tiedostot	M12/M10
B2	HTML-tiedostot	M13/M11
B3	JS ja HTML-tiedostot	(M12+M13×M3/M4)/(M10+M11 × M3/M4)
C	Tapahtuma-attribuuttien osuus kaikista funktiokutsuista	
C2	HTML-tiedostot	M14/M9
C3	JS ja HTML-tiedostot	M14×M3/M4/(M8+M9 × M3/M4)
D	Tapahtuma-attribuuttien osuus takaisinkutsullisista funktiokutsuista	
D2	HTML-tiedostot	M14/M11
D3	JS ja HTML-tiedostot	M14×M3/M4/(M10+M11 × M3/M4)
E	Tapahtuma-attribuuttien osuus anonyymitakaisinkutsullisista funktiokutsuista	
E2	HTML-tiedostot	M14/M13
E3	JS ja HTML-tiedostot	M14×M3/M4/(M12+M13 × M3/M4)
F	Anonyymien funktioparametrien osuus kaikista takaisinkutsullisista funktioparametreista	
F1	JS-tiedostot	M17/M15
F2	HTML-tiedostot	M18/M16
F3	JS ja HTML-tiedostot	(M17+M18×M3/M4)/(M15+M16 × M3/M4)
G	Takaisinkutsullisten funktiomäärittelyjen osuus funktiomäärittelyistä	
G1	JS-tiedostot	M13/M15
G2	HTML-tiedostot	M14/M16
G3	JS ja HTML-tiedostot	(M13+M14×M3/M4)/(M15+M16 × M3/M4)
H	Funktioparametrien määrä per takaisinkutsullinen funktiokutsu	
H1	JS-tiedostot	M15/M10
H2	HTML-tiedostot	M16/M11
H3	JS ja HTML-tiedostot	(M15+M16×M3/M4)/(M10+M11 × M3/M4)

TAULUKKO 9: Funktiokutsujen sisäkkäisyyksiin liittyvät muuttujat

No.	Muuttujan kuvaus
I	Takaisinkutsullisten funktiokutsujen löydetty maksimisisäkkäisyys.
I1	JavaScript-tiedostoissa
I2	HTML-tiedostoissa
J	Takaisinkutsullisten funktiokutsujen löydetty keskiarvosisäkkäisyys.
J1	JavaScript-tiedostoissa
J2	HTML-tiedostoissa
K	Takaisinkutsullisten funktiokutsujen löydetty mediaanisisäkkäisyys.
K1	JavaScript-tiedostoissa
K2	HTML-tiedostoissa

kuin toisia. Sivustoja ei haettu kuin kerran onnistuneesti. Tämän jälkeen uudelleenmittaukset kohdistettiin näihin aiemmin haettuihin, tutkimuskoneessa säilytettyihin tiedostoihin. Näin tutkimus kuormitti organisaatioiden palvelimia ja verkkoympäristöä mahdollisimman vähän.

4.3.1 Haku

Hakuvaiheessa mittausohjelma latasi ja tallensi sivuston tutkimustietokoneen kiintolevylle. Sivustoilta ladattiin rekursiivisesti kaikki saatavilla olevat HTML- ja JavaScript-tiedostot. Hakemisessa kunnioitettiin robottien ohjeistamista varten kehitettyä pienistä komentoryhmistä koostuvaa protokollaa (engl. The Robots Exclusion Protocol), eli käytännössä sivuston sisältämää robots.txt-tiedostoa ja sen sisältämiä ohjeistuksia. Robots.txt-tiedostoihin sivustoista vastaavat toimijat voivat ohjeistaa mitä tietoja sivusto sallii automaattisesti sivustoja lukevien ohjelmien käsittelevän.

Sivustosta tunnistetaan joitakin päätteettömiä tiedostoja joko HTML- tai JavaScript-tiedostoiksi. Tämä tehdään pääosin automaattisesti, mutta joitain yksittäisiä tiedostoja muutettiin myös manuaalisesti. Sivuston haku tehdään onnistuneesti vain kerran.

4.3.2 Valmistelu

Sivuston lataamisen jälkeen, ennen varsinaista mittausta, tiedostoille tehdään valmistelevia toimia edesauttamaan mittauksen onnistumista ja lisäämään mittauksen oikeellisuutta. Ladatun sivuston tiedostoihin kohdistetaan seuraavia toimia:

HTML-sivujen otanta. Havaittin, että sivustoilta saatavien tiedostojen määrät vaihtelevat hyvin paljon. Osa sivustoista tuottaa yli satatuhatta tiedostoa, jonka jälkeen haku oli keskeytettävä tilan- ja ajanpuutteen vuoksi. JavaScript-tiedostojen määrät olivat kaikissa tapauksissa hallittavia, yleensä joitakin kymmeniä, mutta enimmillään muutamia satoja tiedostoja. Tutkimukseen päätettiin ottaa mukaan kaikki kohtuudella saatavissa olevat JavaScript-tiedostot ja tehdä 1000 alkion satunnaisotanta sivuston HTML-tiedostoista. Käytännössä wget-operaatiolla haettiin HTML- ja JS-tiedostoja sivustoilta. Jos saatavilla olevia tie-

dostoja oli enemmän kuin 30 000, haku keskeytettiin jossain vaiheessa tuon määrän jälkeen. Tarkasti tätä ei voinut määrittää, koska wget ei mahdollista tiedostojen määrän rajaamista. Tutkimukseen valittiin kaikki JS-tiedostot ja 1000:n tiedoston satunnaisotos HTML-tiedostoja. Mikäli sivustolta ladattujen HTML-tiedostojen määrä oli tätä vähäisempi, otettiin tutkimukseen kaikki HTML-sivut.

Kopiointi. Tutkittavat tiedostot kopioitiin uuteen hakemistoon.

JavaScript-koodin erottaminen HTML-tiedostoista. HTML-tiedostojen sisältämät JavaScript-ohjelmakoodit erotettiin omiksi tiedostoikseen varsinaista mittausta varten. HTML-tiedoston script-elementtien sisällöstä muodostetiin uusi JS-tiedosto. Samaan tiedostoon luotiin myös tapahtuma-attribuuttien eli HTML-elementtien sisältämien attribuuttien määrittelevistä arvoista uudet funktiot. Löydös talletettiin siten että attribuutin arvo kirjoitettiin generoidun funktion sisään. Löydökseksi laskettiin tilanne, jossa elementillä on jokin seuraavista attribuuteista: onblur, onchange, onclick, ondblclick, onfocus, onkeydown, onkeypress, onkeyup, onload, onmousedown, onmousemove, onmouseout, onmouseover, onmouseup, onreset, onselect, onsubmit, onunload, onscroll, onresize, onerror, onabort, ondragstart, ondrag, ondragenter, ondragleave, ondragover, ondrop tai ondragend. Näin attribuutin arvon sisältämät funktiokutsut ja funktiomäärittelyt tulivat lasketuksi varsinaisessa laskennassa. Tapahtumaparametriin liittyvä funktiokutsu, joka vastaa ”addEventListener” -funktion kutsua, laskettiin tässä yhteydessä erikseen. Nämä eivät tule lasketuksi varsinaisessa mittauksessa ja siksi tämä luku lisättiin tarvittaviin muuttujiin erikseen. M14:n sisältämä tulos on siten laskettu mukaan M9, M11, M13, M16 ja M18 -muuttujiin.

Ohjelmakoodin esiformatointi. Ohjelmakoodille suoritettiin esiformatointi, joka muokkasi ohjelmakoodit yhtenäiseen muotoon. Näin saatiin myös aukaistua minifoidut osat ja vähennettyä muutenkin erilaisista koodin kirjoitustavoista johtuvaa mittausvirhettä ohjelmarivien laskennassa.

Ohjelmakoodirivien laskenta. Käytännön syistä tässä käsittelyn yhteydessä mitattiin sekä HTML- että JavaScript-ohjelmakoodin rivimäärät. Koodiriveihin ei laskettu mukaan kommentti- eikä tyhjiä rivejä.

Tiedostojen jako pienempiin hakemistoihin. Sivuston tiedostot jaoteltiin 100 kappaleen ryhmiin omiin hakemistoihinsa mittauksen onnistumisen varmistamiseksi. Muutaman sivuston kohdalla jaottelu oli tehtävä 10 tiedoston ryhmiin, jotta analysointi tutkimuskoneella onnistui.

4.3.3 Mittaus

Mittaus toteutettiin käyttämällä sekä tutkijan kehittämää mittausohjelmaa että Gallaban ym. (2015) toteuttamia mittauskomponentteja. Mittausohjelmisto tuotti tuloksena tietoa sivustojen JavaScript-koodin sisältämien kaikkien funktiokutsujen määristä, takaisinkutsullisten funktiokutsujen määristä, anonyymien takaisinkutsuvien funktiokutsujen määristä, kaikkien funktiomäärittelyjen määristä, takaisinkutsuvien funktiomäärittelyjen määristä, takaisinkutsullisten funktiokutsujen sisäkkäisyyksistä sekä promise-ominaisuuden ja async.js ja jQuery-kirjastojen yleisyydestä. JavaScript-koodin osalta tiedot laskettiin erikseen HTML-tie-

dostossa sijaistsevista JavaScriptistä ja erillisissä JavaScript-tiedostoissa olevasta koodista.

4.3.4 Havaintomatriisi

Tiedot koottiin ja talletettiin kategorioittain havaintomatriiseihin, joiden havaintoyksikkö on sivusto. Havaintomatriisin yksi rivi kuvaa yhtä havaintoyksikköä. Sarakkeet kuvaavat muuttujia, jotka on esitelty taulukoissa 3, 4, 5 ja 6. Sivustojen osalta anonymisoidut havaintomatriisit ovat liitteenä (Liite 1).

4.4 Mittausohjelmisto

Tietojärjestelmä toteutettiin JavaScript-kieltä, Node.js-ympäristöä, Bash-skriptausta ja taulukkolaskentaa hyödyntäen. Mittauslogiikka luotiin JavaScript-kielellä useisiin moduuleihin, joita käytettiin Node.js -ympäristön avulla. Bash-skriptit olivat helpottamassa tiedostojen käsittelyjä ja luomassa automatisoituja työprosesseja. Moduulit tuottivat tuloksena csv-tiedostoja, joissa sivustoihin liittyvä data oli esitetty pilkuilla erotettuna. Nämä koostettiin ja linkitettiin taulukkolaskentatiedostoon, jossa tieto muokattiin haluttuun esitysmuotoon.

Sivustojen lataaminen tutkimustietokoneelle toteutettiin rekursiivisesti wget-komentoriviohjelmaa hyödyntäen. HTML-tiedostojen satunnaisotosta varten hyödynnettiin shuf-operaatiota, jolla valittiin sivuston HTML-tiedostojen hakemistosta 1000 satunnaista tiedostoa tutkimukseen. HTML-tiedostoissa oleva JavaScript-koodi paikannettiin script-elementtien sisältä ja talletettiin uusiin JS-tiedostoihin laskentaa varten. Niitä tutkitiin vastaavasti kuin valmiiksi erillisissä tiedostoissa sijaitsevaa JavaScript-koodia. Lisäksi funktioiden laskennassa huomioitiin HTML-elementtien attribuuttina annettavat tapahtumakutsut. Esimerkiksi ”onclick” -attribuutilla voi asettaa vaikka ”button” -elementille kuuntelijan napin painallusta varten. Toiminta on täysin sama kuin addEventListener-funktiolla toteutettuna.

Muuttujien mittaamisen logiikka ohjelmoitiin JavaScript-kielellä pääosin osana tätä tutkimustyötä. Ainoastaan funktioiden määrittelyihin liittyvä mittaus tapahtuu GMB-komponentilla, vaikka aluksi oli tarkoitus hyödyntää GMB-komponentteja huomattavasti laajemmin. Käytännössä havaittiin että ainakaan tutkimustietokoneella GMB-komponentit eivät suoriutuneet suurien hakemistojen käsittelystä. Tämän vuoksi valmisteluvaiheeseen lisättiin sivuston pilkkominen pienempiin hakemistoihin mittauskomponentteja varten. Lisäksi HTML-tiedostojen script-elementtien sisäisen ohjelmakoodin analysointia edelsi mahdollisten JSON-tietojen ja HTML-koodin poistaminen, jotta analysointi ei päättyisi virheeseen.

Asynkronista toimintaa edesauttavien muiden ratkaisujen tutkiminen perustui puhtaasti merkkijonoihin. Etsittyjä muita ratkaisuita olivat Promise-toiminnallisuus sekä async.js-kirjasto. Lisäksi etsittiin jQuery-kirjaston esiintyvyyttä. JavaScript-ohjelmakoodista etsittiin merkkijonoja ja löydökset raportoitiin havaintomatriisiin. Promise-abstraktio vaatii määritelmänsä mukaan then-funktion. Näi-

tä etsittiin JavaScript-koodista hakemalla ".then(" -merkkijonoa, siten että ennen ja jälkeen then-sanan saattaa olla tyhjiä merkkejä. Muista kirjastoista tutkittiin async.js-kirjaston ja jquery-kirjaston esiintyvyyttä (Taulukko 6).

Mittausohjelmiston totetutuksessa hyödynnettiin sekä Bash-ympäristössä toimivia valmiita operaatioita että Node-ympäristölle kehitettyjä moduuleita ja kirjastoja. Näitä ovat seuraavat:

- wget: Bash-ympäristön tarjoama operaatio jolla sivustot voitiin ladata verkosta tutkimusta varten.
- shuf: Bash-ympäristön tarjoama operaatio jolla suoritettiin satunnaisotanta.
- dir_usage.js: GMB-komponentti joka laski funktiomäärittelyjen kokonaismäärän ja sellaisten funktiomäärittelyjen määrän, jotka toteuttavat takaisinkutsun parametrinaan saamalla funktiolla.
- syntax-error: Node.js moduuli, joka tekee JavaScript-koodin syntaksitarkistuksen. Käytetään HTML-tiedostoista erotetun JavaScript-koodin tarkistamiseen ennen kuin se kirjoitetaan uuteen tiedostoon josta funktiot tutkitaan.
- js-beautify: Node.js moduuli, joka yhteismitallistaa koodit tiettyyn muotoon. Toimii erikseen HTML-koodille ja JavaScript-koodille.
- html-parser: Node.js moduuli. Käytetään tässä työssä hakemaan tapahtuma-attribuutit HTML-sivulta.
- html-minifier: Node.js moduuli. Hyödynnetään tässä tutkimuksessa HTML-kommenttien poistamisessa asiallisesti.
- sloc: Node.js moduuli, joka laskee koodin rivimäärät ilman kommentteja ja tyhjiä rivejä.
- fs,fs-promise ja graceful-fs: Node.js moduulit. Käytetään tiedostojen lukemista ja kirjoitusta varten.
- util, jQuery, string, path, jsonfile, json2csv: Node.js moduulit. Käytetään pienissä aputehtävissä.

4.5 Tutkimusprosessi

Tutkimusprosessin alkuvaiheessa havaittiin, että tutkimusaineiston hankintaan tulisi menemään runsaasti aikaa. Sivustojen lataus tutkimuskoneelle kotiverkkoyhteyksien nopeuksilla vei vaihtelevasti aikaa. Hyvin pienien sivustojen osalta aikaa kului muutamia minuutteja ja toisaalta suurien ja/tai hitaasti latautuvien sivustojen osalta jopa useita päiviä.

Aineistoa ryhdyttiin lataamaan, vaikka mittausohjelmisto oli muutoin vielä hyvin alkeellisessa vaiheessa. Järjestelmä rakennettiin siten, että kaikki mittausta varten tehdyt tiedostojen käsittelyt tehtiin aina kopioituihin tiedostoihin ja alkuperäiset ladatut olivat aina saatavilla uudelleenkäsittelyä varten. Näin sivustoja voitiin käsitellä lukuisia kertoja ilman alkuperäisten tuhoutumista esimerkiksi ohjelmointivirhetilanteessa.

Alkuvaiheessa oletettiin, että tätä tutkimusta varten kehitetyn ohjelmakoodin osuus keskittyisi lähinnä sivustojen hakemiseen sekä HTML-tiedostojen sisältä-

män JavaScript-koodin erottelemiseen mittaamista varten. Varsinaisen mittaustyön suunniteltiin tapahtuvan GMB-komponenteilla. Lopulta kuitenkin suurin osa mittauslogiikastakin jouduttiin ohjelmoimaan uudelleen. Tämä varsin paljon työtunteja vienyt arviointivirhe johtui puutteellisesta testauksesta tutkimuksen suunnitteluvaiheessa. Tutkimuksen suunnittelussa luotettiin aivan liikaa Gallaban ym. (2015) julkaistussa tutkimuksessa käytettyjen komponenttien toimivuuteen ja luotettavuuteen, eikä niille tehty pieniä kokeiluita lukuunottamatta asianmukaisia testauksia. Jälkikäteen voisi todeta että myös komponenttien lähes täydellisesti puuttuva dokumentaatio ja puuttuvat kommentit ohjelmakoodissa olisivat viitanneet siihen, että komponenttien käyttöönotossa tulisi olemaan ongelmia. Aluksi kuitenkin tulkittiin että ohjelmakoodi oli kehitetty tiettyyn kertakäyttöiseen tilanteeseen, eikä sitä siksi oltu kehitetty tuotantosovellusten vaatimusten mukaisesti. Ohjelmakoodin selkeistä puutteista huolimatta mittauslogiikan toimivuutta ei aluksi osattu epäillä. Näiden komponenttien ongelmia käsitellään tarkemmin seuraavassa alaluvussa.

Mittausohjelmistolla käsiteltiin isoja tietomassoja. Hankaluutena oli virheet, jotka ilmaantuivat vasta joidenkin yksittäisten tiedostojen kohdalla tutkimusprosessin loppuvaiheissa. Näin ohjelmaa jouduttiin korjaamaan joitakin kertoja ja tämän jälkeen ajamaan kaikki mittaukset uudelleen reliabiliteetin varmistamiseksi. Kaikkien mittausten uudelleen ajaminen kesti aina useita päiviä.

Mikäli mitattava JavaScript-koodi sisälsi syntaksi- tms. virheitä, ei GMB-koodi käsitellyt niitä. Näin sama toiminnallisuus periytyi myös tämän tutkimuksen tuloksiin. Tätä tietoa ei tutkimuksessa mitattu, mutta tutkijan tekemien havaintojen perusteella syntaksivirheet näyttivät keskittyvän HTML-tiedostojen script-elementtien sisäiseen JavaScript-koodiin.

4.6 GMB-komponentit

Funktioiden mittausta varten tutkimuksessa hyödynnettiin Gallaban ym. (2015) omaa tutkimustaan varten kehittämää ohjelmistoa. Ohjelmisto on vapaasti ladattavissa Github-palvelussa (callmeback, ladattu tutkijan koneelle elokuussa 2015). GMB-komponentit oli kirjoitettu JavaScript-kielellä ja niitä voitiin käyttää Node.js -ympäristöä hyödyntäen. Ohjelmisto sisältää useita komponentteja, mutta dokumentaatio niistä pääosin puuttuu, eikä ohjelmakoodissa ole kommentteja. Komponentteja pyrittiin hyödyntämään niin paljon kuin mahdollista, mutta lopulta hyödynnettiin vain stats_dir.js -komponenttia. Se näytti mittaavan funktioiden määrittelyjä ja testiaineistolla komponentin todettiin toimivan oikein. Komponentti mittaa funktiomäärittelyjä siten että se laskee takaisinkutsun suorittavat funktiot erikseen ja muut erikseen. Komponentit toimivat siten, että niille tuli antaa parametrina hakemisto josta mittaus haluttiin tehtäväksi. Suorituksen jälkeen komponentit tuottivat numeromuotoisen mittaustuloksen.

Gallaban ym. (2015) komponentteja jouduttiin muuttamaan, jotta niitä oli mahdollista hyödyntää uudessa tutkimusympäristössä. Komponentit sisälsivät käyttämättömiä riippuvuuksia, jotka aiheuttivat kääntäjän virheitä. Nämä kommen-

toitiin pois. Absoluuttisia hakemistopolkuja muutettiin siten että komponentit toimisivat tutkimustietokoneella. Lisäksi mittaustulokset muutettiin kirjoitettavaksi tiedostoon konsolin sijasta. Komponenttien virheidenkäsittelyä jouduttiin muuttamaan, jottei virheitä kadotettu hiljaisesti.

GMB-ohjelmakoodin hyödyntäminen oli työlästä. Se oli ilmeisesti toteutettu yhden tutkimuksen kertakäyttöiseksi apuvälineeksi jossa täytyi ehkä myös osata kommentoida ohjelmakoodia oikeista kohdista tiettyjen analyysien toteuttamiseksi. Vaikka komponentit ovat voineet toimia kontekstissaan riittävästi, ne eivät olleet valmiusasteeltaan sellaisia joita ulkopuolinen olisi vain voinut ottaa suoraan käyttöön. Lisäksi ne sisälsivät monia puutteita, kuten minifoitujen koodinosien ohittaminen ja se että ne kaatuvat hiljaisesti virhetilanteissa. Komponenttien tuomien ongelmien kanssa vietettiin luvattoman paljon aikaa. GitHub-palvelusta saatavan ohjelmakoodin todettiin myös olevan vaillinainen. Kaikkia artikkelin mukaisia mittauksia ei näillä välineillä voitu mitata. Joitakin nyt puuttuvia muuttujia ehkä olisi voinut mitata, mikäli sitä olisi jotenkin ohjeistettu.

GMB-koodia pyrittiin myös muokkaamaan luotettavampaan suuntaan. Esimerkiksi osoittautui että GMB-mittauskomponentit kaatuivat hiljaisesti monessa tapauksessa. Koodi sisälsi useita try-catch -lauseita, joiden catch -osiossa ei ollut mitään toteutusta. Mikäli jonkin tiedoston käsittely päätyi jossain vaiheessa virheeseen, se ei tullut koskaan tietoon vaan käsittely jatkui seuraavasta tiedostosta ja tämän tiedoston kaikki tulokset jäivat tallentamatta. Komponenttia muutettiin tulostamaan virheet näkyville. Havaittiin että monet näistä virheistä eivät olleet varsinaisia virheitä ohjelmakoodissa vaan sallittua JavaScript-koodia jota komponentti ei osannut tulkita. Virheiden esiintymistä pyrittiin vähentämään muokkaamalla komponentille annettavaa sisältöä siltä osin kun se ei vaikuttaisi mittaukseen. Sisällöstä mm. poistettiin JSON-tiedot ja sellaiset script-elementtien sisällä olevat tiedot, jotka alkavat ”<” -merkillä ja ovat ehkä HTML-koodia.

GMB-komponentit ohittivat minifoidut koodit, eli ohjelmanosat jotka oli ohjelmallisesti tiivistetty hyvin pieneen tilaan. Nämä koodinosat hypättiin komponenteissa yli ja jätettiin mittaamatta. Tässä tutkimuksessa se ei tuottanut ongelmia, koska minifointi purettiin jo aikaisemmassa vaiheessa koodirivien yhteismitallista laskemista varten.

Asynkronisten funktiokutsujen mittaus oli GMB-komponenttiin rakennettu. Tätä mittausta hyödynnettiin tutkimuksessa lähes loppuun asti, kunnes viime hetkillä ymmärrettiin miten harhaisesta mittauksesta on kyse ja se päätettiin jättää pois. Tulosten analysoinnissa havaittiin, että verrattuna mitattuihin, todellisiin takaisinkutsullisiin funktiokutsuihin asynkroniset löydökset saattoivat olla tuhansia prosentteja. Komponentin tarkemmassa tarkastelussa havaittiin, että esimerkiksi kaikki funktiokutsut, jotka jotenkin sisälsivät merkkijonon ”net”, tulivat lasketuksi asynkronisiksi funktiokutsuiksi. Asynkronisten funktiokutsujen mittaus perustuu merkkijonoille hyvin vastuuttomasti. Sellaisiakaan tarkistuksia ei ole tehty, jotka olisi ollut hyvin helppoa tehdä, kuten etsiä merkkijonoja vain takaisinkutsullisista funktiokutsuista, jolloin tulokset olisivat olleet jo paljon parempia. Artikkelissaan Gallaba ym. (2015) kertovat löytävänsä DOM-

tapahtumat, kuten ”addEventListener”, verkkokutsut kuten ”XMLHTTPRequest” ja ajastimet kuten ”setImmediate”,”setTimeout” ja ”setInterval”. Komponentti ilmeisesti kyllä löytää nämä asynkroniset funktiokutsut, mutta tulkitsee myös suuren määrän muita funktiokutsuja asynkronisiksi. Asynkronisten funktiokutsujen mittaaminen jäi näin pois tämän tutkimuksen mittaustuloksista.

5 TULOKSET

Tutkimuksessa kerättiin mittava aineisto suomalaisilta verkkosivuilta. Aineisto kerättiin maalis-toukokuussa 2016. Seuraavissa alaluvuissa esitellään tutkimuksen tulokset ja vertaillaan tuloksia sekä kategorioittain että Gallaban ym. (2015) tuloksiin.

5.1 Tutkimuksen tulokset

Tutkimuksessa tutkittiin 134 sivustoa, jotka sisälsivät yhteensä 13,8 miljoonaa JavaScript-funktiokutsua ja 4,4 miljoonaa funktiomäärittelyä. Sivustoja oli viidestä eri kategoriasta, jotka on esitetty taulukossa 2. Sivustoilta tutkituista JavaScript-koodista 4,6 miljoonaa riviä oli JavaScript-tiedostoista ja 31,9 miljoonaa riviä HTML-tiedostoista. Tutkimustulokset on esitetty koostetusti taulukoissa 10, 11, 12 ja 13. Havaintoyksiköittäin tutkimustuloksia voi tarkastella liittestä 1. Tutkimuksessa tutkittiin 33 finanssisektorin sivustoa, 28 pikavippisivustoa, 32 rahapelisivustoa, 33 verkkokauppaa ja vain 8 julkisen sektorin palveluiden sivustoa. Julkisten palveluiden aineiston hankkiminen oli hankalaa, sillä kymmenien sivustojen joukosta vain muutamalla oli riittävä saavutettavuus tutkimuksen kannalta. Suurin osa hylätyistä sivustoista ei tuottanut latauksen tuloksena yhtään tai juuri yhtään tiedostoa.

Tutkimuksessa käytetään käsitteitä, joiden tarkka selitys muodostuu helposti pitkäksi. Siksi muuttujat on numeroitu ja esitelty taulukoissa 3, 4, 5 ja 6 . Lisäksi johdetut arvot on merkitty kirjaimilla (A-L), jotka on selitetty tarkasti taulukoissa 8 ja 9. Kirjaimien perässä oleva numero (1-3) kuvaa sitä joukkoa josta arvo on laskettu. Numero yksi kuvaa JavaScript-tiedostojen sisältämää JavaScriptiä, kakkonen kuvaa HTML-tiedostojen sisältämää JavaScriptiä ja kolmonen kuvaa näitä molempia yhteensä. Taulukossa 8 esitetään laskentakaavat näille johdetuille arvoille.

Taulukossa 10 esitetään tutkittujen tiedostojen määrään ja ohjelmakoodin rivimäärään liittyviä tunnuslukuja kategorioittain. Lisäksi esitetään johdettu arvo kuinka suuri osuus HTML-tiedostojen ohjelmariveistä on JavaScript-koodia (L2).

TAULUKKO 10: Yleisiä tietoja mitatuista tiedostoista

		JS tiedostot (lkm)	HTML tiedostot (lkm)	JS tiedostot JS-loc	HTML tiedostot JS-loc	HTML tiedostot HTML-loc	HTML tiedostot JS-loc/ loc (%)
		M2	**M3**	**M5**	**M6**	**M7**	**L2**
Finanssisek.	Min	0	42	0	1428	4731	2,0
	Max	1447	151 306	971 804	591 481	8 904 397	61,3
	Med	14	1107	16 910	58 150	500 486	15,8
	Ka	68	8683	70 106	138 661	840 735	17,1
Pikavipit	Min	0	9	0	0	1488	0,0
	Max	116	1351	119 597	212 718	347 411	86,9
	Med	11	50	10 576	8869	18 884	28,4
	Ka	19	138	16 682	25 317	55 239	32,6
Rahapelit	Min	0	2	0	0	348	0,0
	Max	66	101 367	167 723	1 750 562	3 403 262	95,3
	Med	3	269	3601	49 846	172 371	32,3
	Ka	8	3908	15 203	175 723	442 431	32,8
Julkiset	Min	7	2	12 092	42	278	1,4
	Max	47	6109	209 357	301 529	589 161	64,0
	Med	12	388	39 817	29 898	86 479	25,2
	Ka	19	1183	70 005	84 334	162 656	29,7
Kaupat	Min	0	5	0	806	1046	2,4
	Max	63	104 839	147 529	2 617 765	5 134 663	82,5
	Med	14	4619	10 587	297 633	1 259 787	24,9
	Ka	18	16 361	24 763	562 542	1 409 929	26,6

Taulukossa 11 esitetään miten suuri osuus kaikista funktiokutsuista antaa parametrikseen vähintään yhden funktion (A). Lisäksi esitetään kuinka suuri osuus tällaisista takaisinkutsullisista funktiokutsuista on sellaisia joissa parametrina lähetetään vähintään yksi anonyymi funktio (B). Edelleen näistä lasketaan HTML-elementtien sisälle määriteltyjen funktiokutsujen eli tapahtuma-attribuuttien osuus kaikista funktiokutsuista (C), takaisinkutsullisista funktiokutsuista (D) ja anonyymitakaisinkutsullisista funktiokutsuista (E).

Tapahtuma-attribuuttien esiintyminen ylipäätään sivustoilla esitetään taulukossa 12. Siinä esitetään kuinka monessa sivustossa näitä asynkronisia mekanismeja esiintyy. Vastaavasti esitetään kuinka monessa sivustossa esiintyy Promise-toiminnallisuutta sekä async.js- ja jQuery-kirjastojen hyödyntämistä. Kirjastojen määrät sivustoilla on mitattu yksinkertaisella merkkijonohaulla, eivätkä sivustokohtaiset esiintymismäärät kerro paljoakaan kirjaston hyödyntämisen todellisista määristä sivustoilla. Tämän vuoksi esiintymät on esitetty karkeammalla tasolla kuin seuraavissa taulukoissa esitettävät tiedot.

TAULUKKO 11: Funktiokutsuihin liittyviä tuloksia

		A1	A2	A3	B1	B2	B3	C2	C3	D2	D3	E2	E3
Finanssisek.	Min	5,2	0,3	0,3	23,9	25,0	35,8	0,0	0,0	0,0	0,0	0,0	0,0
	Max	26,4	44,6	40,1	78,5	100,0	99,9	33,3	32,0	100,0	97,6	100,0	99,9
	Med	14,5	14,6	16,7	48,1	100,0	86,6	3,4	2,4	26,4	22,1	26,6	24,5
	Ka	15,3	16,1	16,9	49,5	89,5	79,1	7,3	6,6	35,9	30,7	37,2	33,0
Pikavipit	Min	7,4	0,0	7,6	15,4	10,0	19,4	0,0	0,0	0,0	0,0	0,0	0,0
	Max	31,2	28,1	26,9	76,9	100,0	92,3	12,5	7,3	100,0	46,0	100,0	72,7
	Med	19,2	10,2	16,0	34,4	75,7	41,6	0,4	0,0	6,8	0,2	12,5	0,5
	Ka	18,2	11,5	17,1	37,1	69,6	46,3	2,0	0,9	21,2	6,1	25,3	10,3
Rahapelit	Min	5,2	0,0	6,6	22,6	7,9	13,4	0.0	0.0	0,0	0,0	0,0	0,0
	Max	29,1	31,1	31,0	100,0	100,0	100,0	27,3	27,2	100,0	87,7	100,0	88,0
	Med	14,1	14,2	16,8	48,1	82,3	63,9	1,8	0,9	15,2	6,3	18,7	9,4
	Ka	16,6	14,7	18,1	54,4	73,3	65,5	4,4	3,5	25,5	16,1	35,8	21,3
Julkiset	Min	5,7	8,0	9,4	28,3	60,5	28,9	0,0	0,0	0,0	0,0	0,0	0,0
	Max	25,9	50,0	25,2	69,0	100,0	94,0	50,0	7,3	100,0	49,3	100,0	80,1
	Med	11,1	16,5	15,0	36,6	100,0	61,1	8,8	3,2	54,3	16,3	62,9	27,1
	Ka	13,6	22,3	16,5	42,8	92,9	59,3	13,3	3,5	42,6	21,6	47,7	31,7
Kaupat	Min	2,8	6,3	7,0	26,0	14,3	16,8	0,0	0,0	0,0	0,0	0,0	0,0
	Max	21,6	69,7	69,6	100,0	100,0	100,0	67,1	67,1	96,5	96,4	98,8	98,6
	Med	13,0	15,4	15,8	48,9	91,9	87,7	1,4	1,3	9,7	7,8	10,1	9,0
	Ka	12,9	21,5	21,7	55,3	78,3	75,3	11,4	11,2	35,3	33,8	37,1	36,0

A=Takaisinkutsullisten osuus kaikista funktiokutsuista, B=Anonyymitakaisinkutsullisten osuus takaisinkutsullisista, C=Tapahtuma-attribuuttien osuus funktiokutsuista, D=Tapahtuma-attribuuttien osuus takaisinkutsullisista funktiokutsuista, E=Tapahtuma-attribuuttien osuus anonyymitakaisinkutsullisista, 1=JavaScript-tiedostoista, 2=HTML- tiedostoista, 3=JavaScript ja HTML-tiedostoista.

TAULUKKO 12: Asynkronisten mekanismien yleisyys

Kategoria	**Sivustoja lkm**	**Tap.attr. lkm (%)**	**Promise lkm (%)**	**Async.js lkm (%)**	**jQuery lkm (%)**
Finanssisektori	33	26 (78,8)	16 (48,5)	0 (0,0)	30 (90,9)
Pikavipit	28	16 (57,1)	4 (14,3)	3 (10,7)	26 (92,9)
Rahapelit	32	24 (75,0)	8 (25)	2 (6,3)	25 (78,1)
Julkiset	8	7 (87,5)	5 (62,5)	0 (0,0)	7 (87,5)
Kaupat	33	25 (75,8)	10 (30,3)	2 (6,1)	32 (97,0)
Yht	134	98 (73,1)	43 (32,1)	7 (5,2)	120 (89,6)

Taulukossa 13 esitetään anonyymien funktioparametrien osuus kaikista funktioparametreista (F). Tämä eroaa johdetusta arvosta (B) jossa lasketaan vain funktioiden kutsukohtien eikä parametrien määrää. Lisäksi taulukossa esitetään niiden funktiomäärittelyjen osuus, jotka toteuttavat takaisinkutsun parametrina saamalleen funktiolle (G) sekä esitetään keskimääräinen funktioparametrien määrä takaisinkutsullista funktiokutsua kohden (H). Tutkimuksessa mitattiin myös funktiokutsuihin liittyvien anonyymien funktioparametrien sisäkkäisyyksiä. Ensimmäinen taso on se kun funktiokutsulle annetaan parametriksi anonyymi funktio. Mikäli tämän sisällä on funktiokutsu, joka saa parametriksi anonyymin funktion, saa se sisäkkäisyyden arvoksi tason kaksi. Näin edeten kaikki anonyymit funktioparametrit laskettiin jollekin sisäkkäisyystasolle. Taulukossa esitetään näistä laskettuja tunnuslukuja; maksimisisäkkäisyys (I), keskiarvosisäkkäisyys (J) ja mediaanisisäkkäisyys (K). Taulukossa 14 esitetään näiden tasojen jakauma jokaiselle kategorialle ja tiedostotyypille erikseen.

TAULUKKO 13: Funktioparametrien sisäkkäisyyksiin liittyviä tuloksia

		F1	**F2**	**F3**	**G1**	**G2**	**G3**	**H1**	**H2**	**H3**	**I1**	**I2**	**J1**	**J2**	**K1**	**K2**
Finanssisek.	Min	21,2	25,0	31,7	1,4	0,0	0,0	1,0	1,0	1,0	2,0	1,0	1,11	1,00	1,0	1,0
	Max	78,4	100,0	99,9	7,3	14,3	11,4	1,2	1,4	1,3	6,0	4,0	2,34	2,00	2,0	2,0
	Med	44,9	100,0	85,1	3,3	0,0	0,6	1,1	1,0	1,0	4,0	2,0	1,71	1,08	2,0	1,0
	Ka	47,1	89,5	78,1	3,3	1,1	1,9	1,1	1,0	1,0	4,0	2,0	2,00	1,00	2,0	1,0
Pikavipit	Min	13,2	10,0	15,9	1,1	0,0	0,1	1,0	1,0	1,0	1,0	1,0	1,0	0,0	1,0	0,0
	Max	75,5	100,0	92,3	37,5	19,2	17,9	1,2	1,3	1,3	6,0	3,0	2,7	2,1	2,0	2,0
	Med	32,2	75,7	39,0	3,1	0,1	3,1	1,1	1,0	1,1	5,0	2,0	1,8	1,3	2,0	1,0
	Ka	34.7	68.6	43.7	4.7	4.5	4.5	1,1	1,0	1,1	4,4	2,1	1,8	1,3	1,7	1,3
Rahapelit	Min	19,3	7,9	13,4	0,0	0,0	0,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
	Max	100,0	100,0	100,0	12,8	26,7	23.7	1,2	1,3	1,3	6,0	4,0	2,7	1,7	2,0	2,0
	Med	42,9	77,8	59,0	3,3	0,1	3,4	1,1	1,0	1,1	4,0	2,0	1,8	1,2	2,0	1,0
	Ka	52,5	71,3	62,5	4,2	4,7	5,3	1,1	1,1	1,1	4,0	2,1	1,7	1,2	1,6	1,2
Julkiset	Min	25,4	60,5	26,3	0,9	0,0	0,1	1,0	1,0	1,0	4,0	1,0	1,5	1,0	1,0	1,0
	Max	67,3	100,0	93,7	5,5	3,8	5,2	1,2	1,0	1,2	5,0	3,0	2,1	1,7	2,0	2,0
	Med	34,9	100,0	58,8	1,8	0,0	1,2	1,1	1,0	1,0	5,0	1,0	1,6	1,0	2,0	1,0
	Ka	40,4	92,4	57,1	2,1	0,5	1,9	1,1	1,0	1,1	4,8	1,5	1,7	1,2	1,8	1,1
Kaupat	Min	24,1	14,3	16,8	1,6	0,0	0,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0	1,0
	Max	100,0	100,0	100,0	6,1	27,9	27,7	1,1	1,4	1,4	6,0	4,0	2,1	1,9	2,0	2,0
	Med	46,6	92,0	86,6	3,1	0,9	1,6	1,0	1,0	1,0	4,0	2,0	1,6	1,1	1,0	1,0
	Ka	53,4	77,0	73,8	3,5	6,1	5,9	1,1	1,0	1,0	4,1	2,2	1,6	1,3	1,5	1,3

F=Anonyymiparametrien osuus kaikista funktioparametreista (%), G=Takaisinkutsuvien funktiomäärittelyjen osuus kaikista funktiomäärittelyistä (%), H=Funktioparametrien määrä per takaisinkutsullinen funktiokutsu, Sisäkkäisyyksien maksimi (I), keskiarvo (J) ja mediaani (K). 1=JavaScript-tiedostoista, 2=HTML-tiedostoista, 3=JavaScript ja HTML-tiedostoista.

TAULUKKO 14: Anonyymien funktioparametrien sisäkkäisyystasojen jakauma (%)

Kategoria	Tiedosto	1	2	3	4	5	6
Finanssisektori	JavaScript	46,20	41,75	9,80	2,16	0,08	0,01
	HTML	87,32	11,95	0,73	0,00		
	HTML*	57.68	39.89	2.43	0.00		
Pikavipit	JavaScript	33,34	43,70	18,91	3,40	0,62	0,02
	HTML	60,10	37,85	2,06			
	HTML*	53.29	44.30	2.41			
Rahapelit	JavaScript	19,61	61,27	16,52	2,28	0,27	0,04
	HTML	78,80	20,36	0,82	0,02		
	HTML*	59.95	38.47	1.56	0.03		
Julkiset	JavaScript	41,28	46,50	11,03	1,05	0,13	
	HTML	89,23	10,64	0,13			
	HTML*	67.66	31.95	0.40			
Kaupat	JavaScript	47,87	39,64	10,98	1,26	0,23	0,03
	HTML	89,39	8,80	1,59	0,22		
	HTML*	54.75	37.53	6.79	0.92		

(*)=Ensimmäisestä tasosta on poistettu tapahtuma-attribuuttien määrä

5.2 Tulosten arviointia ja vertailua

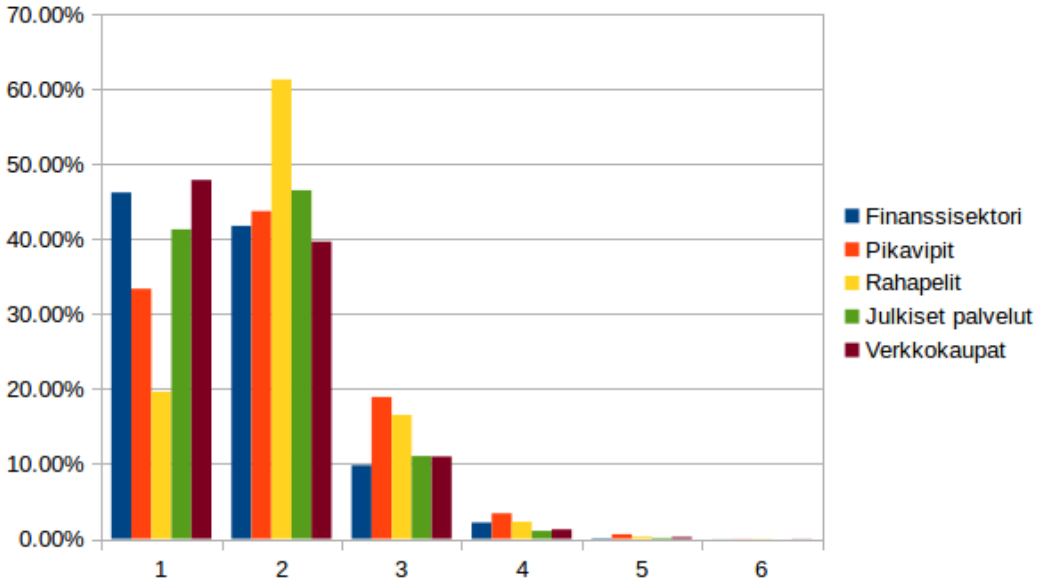
Tutkimuksessa mitattiin takaisinkutsullisten funktiokutsujen osuutta kaikista funktiokutsuista (A). Havaittiin, että kategorioiden (A3:n) keskiarvot olivat hyvin lähellä toisiaan (16,5-18,1 %), paitsi verkkokauppakategoriassa, jossa arvo oli hieman korkeampi (21,7 %). Tässä kauppojen ryhmässä esiintyy maksimiarvoiltaan lähes 70 %:n osuutta, joka kuulostaa hyvin korkealta luvulta. Tarkemmassa tarkastelussa voi havaita, että näiden korkeiden lukujen taustalla on hyvin korkeat tapahtuma-attribuuttien osuudet. Tällaisilla verkkosivuilla suurin osa takaisinkutsuista on siten toteutettu HTML-elementtien sisään, tapahtuma-attribuuttityyliä hyödyntäen ja muuta toiminnallisuutta on ollut vähemmän.

Anonyymitakaisinkutsullisten funktiokutsujen osuus takaisinkutsullisista funktiokutsuista (B3) tuottaa eroja kategorioiden välille. Arvot vaihtelevat 46,3 %:n ja 79,1 %:n välillä. Osassa sivustoista kaikki takaisinkutsulliset funktiokutsut ovat anonyymillisiä. Maksimiarvot ovat kaikissa kategorioissa yli 90 %. Koko aineiston minimi löytyy rahapeleistä, jossa siinäkin osuus on 13,4 %. Finanssisektorin minimi on 35,8 %, eli sivustoilla vähintään joka kolmas takaisinkutsullisista funktiokutsuista sisältää vähintään yhden anonyymin funktioparametrin. Anonyymien funktioparametrien käyttäminen on siten hyvin yleistä aineistossa.

Tapahtuma-attribuuttien osuus kaikista sivuston funktiokutsuista (C3) erottelee sivustot tehokkaasti. Myös kategorioiden välille syntyy selviä eroja. Verkkokaupoissa tapahtuma-attribuutteja on noin 11 % kaikista funktiokutsuista. Tämä on hyvin suuri osuus, kun ajattelee, että pienenkin toiminnallisuuden toteutta-

miseen tarvitaan usein monia funktiokutsuja, etenkin kun hyödynnetään erilaisia kirjastoja yms. Tapahtuma-attribuutteja on siten suhteessa muuhun toiminnallisuuteen hyvin paljon. Verkkokauppojen maksimi tässä on jopa 67,1 %! Muiden kategorioiden maksimit ja keskiarvot ovat paljon maltillisempia. On kuitenkin huomattavaa, että verkkokauppojen ryhmässä joka neljäs sivusto ei sisällä tapahtuma-attribuutteja ollenkaan. Pikavippiryhmässä tapahtuma-attribuutteja esiintyi selkeästi vähiten. Tutkituista pikavippisivustoista vajaa puolet ei käyttänyt tätä ohjelmointityyliä ollenkaan ja keskimäärin kaikista kategorian takaisinkutsullisista funktiokutsuista tapahtuma-attribuuttien osuus on selkeästi vähäisempi kuin muissa katrgorioissa.

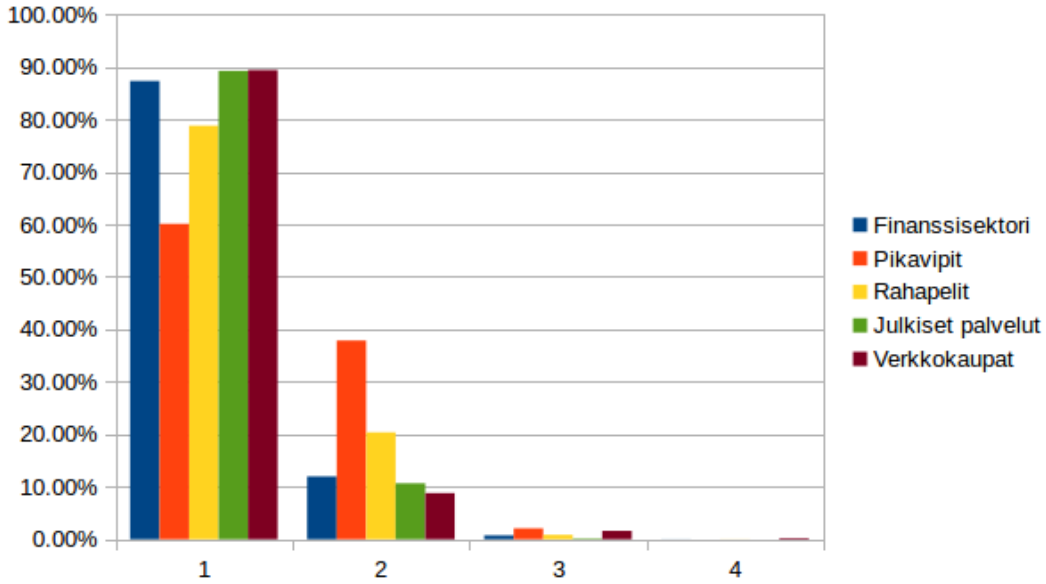
Sivustoilta löydettyjen maksimisisäkkäisyyksien arvot eroavat JavaScript-tiedostoissa ja HTML-tiedostoissa. JavaScript-tiedostoissa maksimisisäkkäisyys oli 6, HTML-tiedostoissa vastaava luku on 4. Kuviossa 4 näkyy sisäkkäisyyksien jakaumat kategorioittain JavaScript-tiedostoissa. Ryhmien välillä on nähtävissä



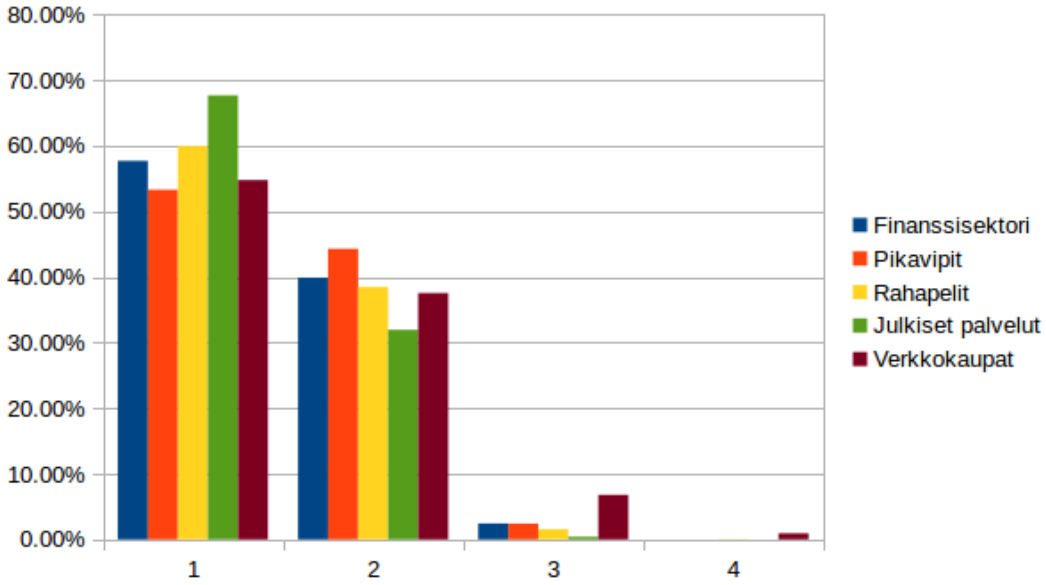
KUVIO 4: Sisäkkäistyystasojen jakauma JavaScript-tiedostoissa

jonkin verran eroja. Finanssisektorissa ja verkkokaupoissa taso 1 on suurin ryhmä. Muissa sensijaan taso 2 on suurin ja rahapeleissä taso 2 on huomattavan dominoiva. HTML-tiedostojen osalta kuviossa 5 näkyy tason 1 hallitseva osuus, johon vaikuttaa tapahtuma-attribuuttien määrä. Kuvio 6 on luotu muutoin samoista tiedoista, mutta siitä on poistettu tapahtuma-attribuuttien osuus. Näin saadaan kuvaaja, joka kertoo vain HTML-tiedostojen script-elementtien sisältämän JavaScript-koodin funktioiden sisäkkäisyydet. Näyttäisi siltä että ryhmien välillä ei ole juurikaan eroja, vaan hyvin yhtenäisesti noin kolmasosa anonyymeistä funktioparametreista on tasolla kaksi.

Funktioparametreittain laskettuna (F3) anonyymien funktioparametrien suhde kaikkiin funktioparametreihin noudattelee funktiokutsuittain laskettua vastaavaa arvoa (B3). Kaikissa ryhmissä F3:n arvo on hieman suurempi kuin B3:n. Aineistossa takaisinkutsullisella funktiokutsulla on yleisimmin vain yksi funktio-



KUVIO 5: Sisäkkäisyystasojen jakauma HTML-tiedostoissa



KUVIO 6: Sisäkkäistyystasojen jakauma HTML-tiedostoissa ilman tapahtuma-attribuutteja

parametri (H1-3). Hieman yllättäen maksimiarvot näistä (H3) löytyvät HTML-tiedostojen JavaScript-koodista. Kategoriat ovat tässä asiassa melko yhtenäisiä, mutta julkisen sektorin palveluissa HTML-tiedostoissa takaisinkutsullisten funktiokutsujen parametrina on korostetusti aina vain yksi funktio. Taulukon 11, D2-arvo näyttää myös, että tässä ryhmässä tapahtuma-attribuuttien osuus takaisinkutsullisista funktiokutsuista on huomattavasti suurempi kuin muissa. Tapahtuma-attribuuttityylillä asetettu funktiokutsu sisältääkin aina vain yhden funktioparametrin.

Takaisinkutsuvien funktiomäärittelyjen (G3) osalta ryhmät jakaantuvat kahteen leiriin. Finanssisektori ja julkinen sektori hyödyntävät aineiston mukaan takaisinkutsumenetelmää selkeästi vähemmän kuin muut ryhmät.

Pikavippisivustojen mittauksessa vain yhdelle sivustolle toteutettiin HTML-tiedostojen satunnaisotanta, muuten kaikki saadut sivut käsiteltiin ja mitattiin. JavaScript-tiedostoja oli sivustoilta saatavissa keskimäärin 19 ja mediaani tälle oli 11. Noin joka kolmas rivi pikavippisivustoilta tutkituista HTML-tiedostoista sisälsi JavaScript-koodia. Sivustoilta saatavien tiedostojen kohtuullisen määrän vuoksi tälle kategorialle oli mahdollista laskea myös JavaScript-koodin ja HTML-koodin suhde koko sivustoa ajatellen. 27 sivustoista mitattiin sellaisena kuin ne olivat verkosta saatavissa. Näistä laskettiin JavaScript-koodirivien osuus suhteessa kaikkiin koodiriveihin, jotka sisältävät myös HTML-koodit. Keskiarvoksi saatiin 55 %, mediaanin ollessa 50 %. Vastaavasti 32:sta rahapelisivustosta 25 sivustoa oli sellaisia, joista tutkittiin kaikki ladatut HTML ja JavaScript-tiedostot. Näiden sivustojen koodiriveistä keskimäärin 45 % oli JavaScriptiä. JavaScript on siten merkittävässä osassa ainakin tutkittujen pikavippi- ja rahapelisivustojen koodista koodirivien perusteella laskettuna.

JQuery-kirjasto on tässä aineistossa erittäin yleisesti käytetty. Koko aineistosta 89 % sivustoista hyödyntää jQuery-kirjastoa jollain tavalla. Rahapelit jäävät ainoana alle 80 %:n osuuden, ja toisaalta tutkituista verkkokauppasivustoista vain yksi ei hyödynnä jQuery-kirjastoa eli hyödyntämisprosentti on 97 %.

Noin kolmasosa sivustoista hyödyntää lupauksia ohjelmassaan. Ääripäinä ovat pikavipit, joista niitä löytyi vain 14 %:sta sivustoista ja toisaalta julkiset palvelut joista viitteitä lupausten käyttämiselle löytyi 62,5 %:sta sivustoista. Async.js -kirjastoa esiintyi tässä aineistossa hyvin vähän, vain 5,2 %:ssa sivustoista. Finanssisektorista ja julkisista palveluista sitä ei löytynyt ollenkaan.

5.3 Vertailu aiemman tutkimuksen tuloksiin

Alaluvussa vertaillaan saatuja tutkimustuloksia Gallaban ym. (2015) tutkimustuloksiin, joiden havainnot ovat seuraavien alalukujen otsikoita. Alalukujen tekstissä käsitellään heidän havaintojaan suhteessa tämän tutkimuksen tuloksiin. Vertailu tehdään vain JavaScript-tiedostojen sisältämän ohjelmakoodin mittaustulosten suhteen. Tällä tavalla saadaan ehkä parempi vertailukelpoisuus koska he eivät ole ulottaneet tutkimustaan HTML-tiedostojen sisältämään JavaScriptiin.

Havainto 1: Asiakaspään ohjelmakoodissa takaisinkutsuvien funktiomäärittelyjen osuus oli 4,5 %.
Mittaus on suoritettu tässä tutkimuksessa heidän GMB-komponentillaan myös tämän tutkimuksen aineistosta. Tutkimustulokset (taulukko 13, G1) ovat hyvin yhteneviä, kategorioiden keskiarvojen vaihdellessa 2,1 % ja 4,7 % välillä.

Havainto 2: Asiakaspään ohjelmakoodissa takaisinkutsullisia oli 9 % kaikista funktiokutsukohdista
Funktiokutsuihin liittyvän mittauksen tuloksia ei saatu GMB-komponentilla toteutettua luotettavasti ja oikein. Tätä tutkimusta varten luotiin oma mittausohjelma. Takaisinkutsullisia funktiokutsuja todettiin olevan eri kategorioissa 12,9 % ja 18,2 % välillä (taulukko 11, A1). Takaisinkutsullisia funktiokutsuja näyttäisi siten olevan verkkopalveluissa jonkin verran enemmän kuin avoimen lähdekoodin ohjelmakoodissa. On kuitenkin epävarmaa ovatko aiemman tutkimuksen tulokset luotettavia GMB-komponentin mahdollisen virheellisen toiminnan vuoksi.

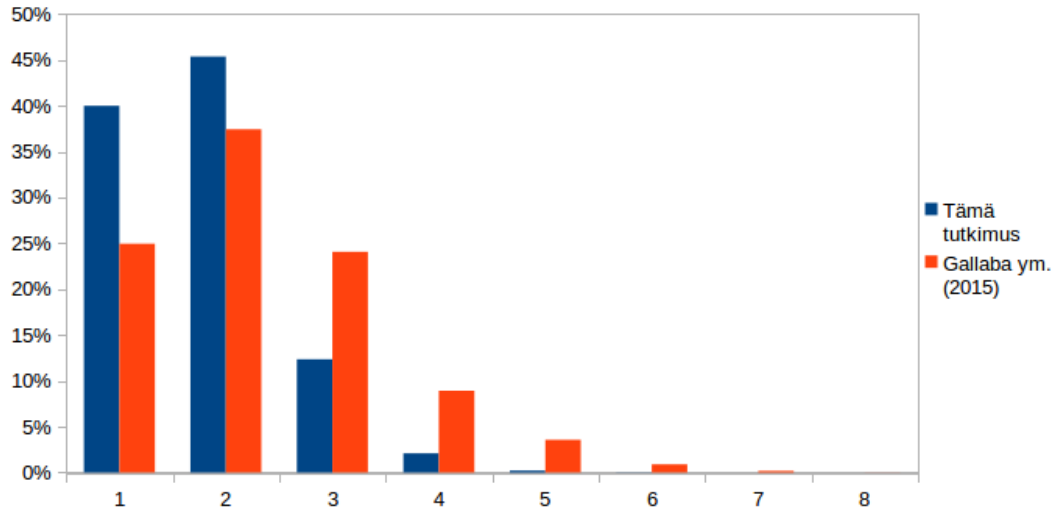
Havainto 3: Asiakaspään ohjelmakoodissa oli asynkronisia 72 % takaisinkutsullisista funktiokutsuista.
Tutkimuksen yhteydessä havaittiin, että GMB-komponentti mittasi asynkronisia funktiokutsuja hyvin heikolla tavalla. Asynkronisten funktiokutsujen tunnistamisessa hyödynnettiin vain yksinkertaista merkkijonohakua, eikä edes tarkastettu onko löydös takaisinkutsullinen funktiokutsu. Tuloksena oli hyvin paljon vääriä positiivisia. Havaintoa ei voi pitää missään määrin luotettavana tuloksena. Tässä tutkimuksessa ei saatu mitattua asynkronisia funktiokutsuja, vaikka ehdottomasti olisi haluttu tämä mittaustulos osaksi tutkimusta. Tavoite oli hyödyntää GMB-komponenttia, mutta tulosten osoituttua harhaisiksi päätettiin se jättää pois. Valitettavasti omaa toteutusta tälle laskennalle ei ehditty tekemään.

Havainto 4: Asiakaspään ohjelmakoodissa oli yli 43 %:ssa takaisinkutsullisista funktioista ainakin yksi anonyymi takaisinkutsufunktio.
Tutkimuksessa ei selvinnyt miten GMB-komponenteilla takaisinkutsulliset pitäisi laskea. Muutamasta vaihtoehdosta mikään ei tuottanut testiaineistolla oikeita tuloksia. On mahdollista että tämän laskeminen tällä välineellä olisi vaatinut ylimääräistä tietoa komponentin toiminnasta. Tässä tutkimuksessa nämä muuttujat mitattiin eri ohjelmakoodilla, mutta tulokset ovat kuitenkin tämän Gallaban ym. (2015) tutkimustulosten kanssa samansuuntaiset. Taulukon 11 B1-arvon keskiarvot vaihtelevat 37,1 %:n ja 55,3 %:n välillä.

Havainto 5: Takaisinkutsufunktioiden sisäkkäisyyden maksimi on 8.
Gallaban ym. (2015) tutkimuksessa sisäkkäisyyksien laskennassa ei ilmeisesti eroteltu asiakaspäätä ja palvelinpäätä. GMB-komponenteilla ei onnistuttu tuottamaan sisäkkäisyyksien laskentaa, vaan tätä tutkimusta varten kehitettiin oma mittausohjelma. Mittaus rajattiin koskemaan ainoastaan anonyymeja funktioparametreja ja niiden sisältämää sisäkkäisyyttä. Havaittiin että JavaScript-tiedostoista löytynyt anonyymien funktiokutsujen sisäkkäisyyden maksimi oli 6, kun

Gallaban ym. (2015) tutkimuksessa maksimi oli 8, joka mainitaan poikkeustapauksena.

Kuviossa 7 näkyy sekä JavaScript-tiedostojen sisältämän JavaScript-koodin sisäkkäisyyksien että Gallaban ym. (2015) tutkimuksen sisäkkäisyyksien jakauma. Havaitaan että molemmissa on näkyvissä selvästi piikki toisessa sisäkkäisyyskohdassa. Isona erona näissä on nähtävissä kolmas taso, joka poikkeaa merkittävästi. Gallabanlla se säilyy ensimmäisen tason kaltaisena, kun tässä tutkimuksessa kolmas taso on merkittävästi alhaisempi.



KUVIO 7: Sisäkkäistyystasojen jakauma tämän tutkimuksen JavaScript-tiedostoissa ja Gallaban aineistossa

Havainto 6: Viidesosa kaikista funktiomäärittelyistä noudatti virhekoodi ensin -menettelyä.
Tämän tutkimuksen osalta näitä ei mitattu. GMB-komponenteista ei löytynyt tällaista toiminnallisuutta, eikä omaa toteutusta tehty tämän tutkimuksen puitteissa.

Havainto 7: Async.js -kirjastoa käytti 56 % web-sovelluksista.
Havainnossa ei valitettavasti ilmeisesti eroteltu asiakaspäätä ja palvelinpäätä. Tässä tutkimuksessa havaittiin että async.js -kirjaston käyttö on hyvin vähäistä tutkimuksen saavutettavissa olevasta ohjelmakoodista (5,2 %:ssa kaikista sivustoista). Tutkimusten tulokset eroavat siten hyvin merkittävästi. Valitettavasti ei voi tietää, onko async.js-kirjaston suuri esiintyvyys pääosin palvelinpään ohjelmakoodissa, vai onko sitä myös paljon asiakaspään koodissa, jolloin ero avoimen lähdekoodin ja verkkopalveluiden välillä olisi ilmeinen.

Havainto 8: Lupauksia löytyi 27 % järjestelmistä, keskittyen asiakaspäähän.
Lupauksia mitattiin myös tässä tutkimuksessa hyvin yksinkertaisella merkkijonohaulla, jossa väärien positiivisten mahdollisuus on olemassa. Kategorioiden tulokset vaihtelivat 14 %:n ja 62 %:n välillä. Kaikista tutkimuksen sivustoista

lupauksia löytyi 32 %:sta, joka on yllättävän yhteneväinen tulos Gallaban ym. (2015) tuloksen kanssa.

6 POHDINTA JA JOHTOPÄÄTÖKSET

Tutkimuksella kartoitettiin ja tuotiin uutta tietoa suomalaisista verkkopalveluita sisältävistä verkkosivustoista. Havaittiin että suomalaisten verkkosivustojen sisältämän JavaScript-koodin sijoittelu ja ominaisuudet vaihtelevat verkkosivujen tarkoitusten mukaan. Ensimmäisessä alaluvussa pohditaan tulosten merkityksiä ja luodaan johtopäätöksiä. Toisessa alaluvussa käsitellään tutkimusmenetelmään ja mittausohjelmistoon liittyviä vahvuuksia ja heikkouksia. Kolmannessa alaluvussa käsitellään tutkimuksen aikana opittuja asioita ja seikkoja joita tehtäisiin nyt toisin.

6.1 Tulosten merkityksestä

Empiirisessä tutkimuksessa verkkopalvelut jaoteltiin viiteen ryhmään niiden tarkoituksen mukaan. Tämä jaottelu osoittautui onnistuneeksi, sillä ryhmien välille oli löydettävissä selviä eroja. Jo sivustojen latausvaiheessa havaittiin että julkisen sektorin palvelut -ryhmän sivustot latautuivat muihin ryhmiin nähden poikkeavasti. Suurin osa julkisen sektorin sivustojen latausyrityksistä päättyi liian vähäiseen tiedostomäärään. Yleensä tiedostoista oli mahdollista ladata vain yksi index.html tai ei sitäkään. Muissa ryhmissä tätä esiintyi joissain sivustoissa, mutta ei ollenkaan tässä laajuudessa. Olettaisin tämän johtuvan julkisiin palveluihin ehkä kohdistuvasta erityisen tietoturvan vaatimuksesta, jolloin varmuuden vuoksi on kielletty tai estetty sivustojen lataaminen. On myös mahdollista että julkisten palveluiden toteuttamisessa on hyödynnetty jollain tavoin samaa ympäristöä tai menetelmää, jolloin toimintamalli on kopioitunut suurimpaan osaan sivustoista.

Tapahtuma-attribuuttien hyödyntäminen erottaa pikavippiryhmän selvästi erilleen muista ryhmistä. Tapahtuma-attribuuttityyliä on hyödynnetty näissä sivustoissa selkeästi vähemmän kuin muissa kategorioissa. Epäilen tämän johtuvan siitä, että tämä on verrattain uusi toimiala, jolle verkkosivutkin on luotu vain muutamia vuosia sitten. Näin on voitu alusta asti toimia uudempien suositusten mukaan, eikä ole tarvinnut huolehtia vanhojen osien toimivuudesta. Verkkokaupparyhmässä erot ryhmän sisällä ovat suuria, toisissa tapahtuma-attribuut-

tien hyödyntäminen on hyvin vähäistä ja toisissa sivustoissa se on hyvinkin suurta. Uskoisin tämän olevan seurausta samasta ilmiöstä, että uusien verkkokauppojen toteutuksessa tapahtuma-attribuuttit eivät ehkä ole enää niin yleisiä kuin vanhemmilla sivustoilla.

Hieman yllättävästi pikavippikategoriassa on rahapelien kanssa korkeimmat osuudet JavaScriptiä HTML-tiedostoihin sisällytettynä (L2). Voisi olettaa, että JavaScript-koodi kirjoitettaisiin erillisiin tiedostoihin ylläpidon helpottamiseksi, mutta näin ei ole tehty. Voi kuitenkin olla, että järjestelmät joilla sivustojen julkaisemiset hoidetaan ovat sellaisia että tällä ei ylläpidon kannalta ole merkitystä. Sivuston suorituskyvyn kannalta valinnalla luulisi olevan merkitystä, olettaisin että mm. välimuistia olisi helpompi hyödyntää JavaScript-koodin ollessa erillisissä tiedostoissa.

JavaScriptin havaittiin olevan oletetusti merkittävässä asemassa verkkosivustoilla. Ohjelmakoodin sijainnin ja kielen mahdollistaman tapahtumankäsittelyn menetelmät ovat kehittyneet ajan myötä, kuitenkin siten että vanhemmat menetelmät ovat edelleen käyttökelpoisia vaikka eivät suotavia. Suomalaisilla verkkosivustoilla on nähtävissä näiden kaikkien menetelmien hyödyntäminen edelleen. Tutkimuksessa todettiin, että tapahtuma-attribuutit ovat hyvin yleisesti edelleen käytössä ja JavaScript-koodia löytyy rivimääräisesti enemmän HTML-tiedostoista kuin JavaScript-tiedostoista. Uusien toimintatapojen lisääminen kieleen on nähdäkseni siten vain tältä osin monimutkaistanut verkkosivustojen JavaScript-ympäristöä, vaikka tavoite on ollut tietenkin toinen. Suositeltujen toimintatapojen jalkautumisen hitaus johtuu ehkä ainakin osittain ohjelmoijien osaamisen puutteesta ja alan yleisestä ongelmasta: liiallisesta kiireestä. Tapahtuma-attribuuttienhan avulla pienet toiminnallisuudet on hyvin näppärä tehdä.

Tutkimuksen kirjallisuusosiossa tuotiin mielestäni kiistatta esille asynkronisia, sisäkkäisiä funktiokutsuja käyttävän menetelmän keskeisin ongelma: se tuottaa ohjelmakoodia, jota on vaikea ymmärtää. Jos ohjelmoija ei ymmärrä ohjelmaa, se tuottanee helposti virheitä tai ohjelma jopa toimii oikein vain sattumalta. Toinen yhtä paha ongelma on virheidenkäsittelyn puuttuminen. Ympäristö ei avusta tässä asiassa mitenkään ja mikäli ohjelmoija ei ole valpas, niin se menee helposti jollain tavoin väärin. Ohjelmointiympäristön pitäisi ohjata sellaiseen rakenteeseen, joka päin vastoin helpottaa ymmärtämistä ja vähentää ohjelmoijan tekemiä virheitä. Promise-malli eli lupausabstraktio on lupaava, joskaan ei täydellinen menetelmä asynkronisten funktiokutsujen sisäkkäisyyksien vähentämiseen. Ilmeisesti rajapintoja hyödyntävän ohjelmoijan ei tarvitse täysin hallita monimutkaista lupausabstraktiota, vaan sitä voi onnistuneesti käyttää vähemmälläkin ymmärryksellä. Sen sijaan lupausabstraktiota tarjoavien rajapintojen luojien tulisi huolellisesti tutustua Promise/A+ -spesifikaatioon, jotta lupausabstraktio tulisi tehtyä nykymallin mukaisesti oikein. JQuery-kirjaston vaillinainen lupaustoteutus on surullinen esimerkki heikosti toteutetusta rajapinnasta.

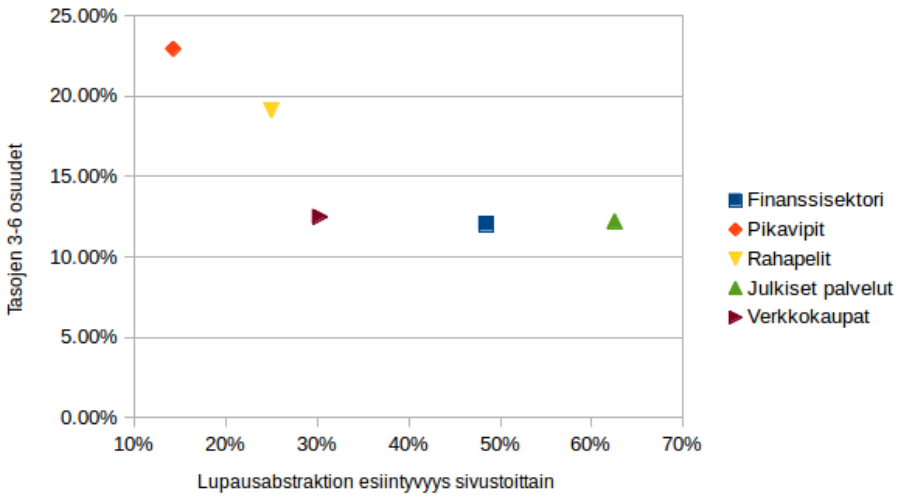
Anonyymitakaisinkutsullisten funktiokutsujen sisäkkäisyydet ovat tutkimusaineistossa keskimäärin kohtuullisia, mutta tutkimuksen aineistossa esiintyy myös joitakin hyvin syviä takaisinkutsuketjuja, jotka voivat olla jo kokonsa puolesta

vaikeasti ymmärrettäviä. Keskimäärin ketjujen sisäkkäisyys on kuitenkin melko hallittua, kun suurin osa anonyymeistä funktioparametreista löytyy tasoilta 1 tai 2. Ensimmäisellä tasolla ei edes vielä ole kyse varsinaisesti sisäkkäisyydestä. Loogisempaa olisikin ollut nimetä tämä taso nollaksi, mutta toive yhteneväisyydestä Gallaban ym. (2015) tutkimuksen kanssa ajoi nimeämään tasot heidän tutkimuksensa kanssa samalla tavalla.

Tutkimuksessa löytyi jopa tasolle kuusi ulottuvia ketjuja ja kaikissa kategorioissa vähintään 10 % anonyymeistä funktioparametreista löytyy tasoilta 3-6. Vertailun vuoksi mainittakoon, että takaisinkutsuhelvettiä havainnollistavassa listauksessa 3 ketjun pituus on neljä. Viitteitä lupausabstraktion hyödyntämiseen, eli Promise-objektiin liittyvän then-funktion käyttämiseen löytyi noin joka kolmannelta sivustolta. Näitä löytyi kuitenkin tyypillisesti vain hyvin pieniä määriä verrattuna kaikkiin takaisinkutsullisiin funktiokutsuihin (Liite 1, muuttuja 23). Lupausmallia ilmeisesti hyödynnetään verkkosivustoilla vasta vähän.

On valitettavaa, että asynkronisten funktiokutsujen mittaaminen ei tässä tutkimuksessa lopulta onnistunut, sillä lupausabstraktioiden vertaaminen niihin olisi kaikkia takaisinkutsullisia mielekkäämpää. Monitasoisesti sisäkkäiset funktiokutsut lienevät kuitenkin pääosin asynkronisia. Joka tapauksessa tuntuu epätodennäköiseltä, että takaisinkutsullisista funktiokutsuista asynkronisia olisi niin vähän, että lupausabstraktion esiintymät niitä olennaisesti korvaisivat.

Lupausabstraktio itsessään tuottaa tasojen yksi ja kaksi sisäkkäisyyksiä. Tätä havainnollistaa myös listaus 5, jossa lupausabstraktion kutsuja tarvitsee ensimmäisen tason takaisinkutsullisia funktiokutsuja, mutta abstraktion tarjoaja hyödyntää myös toisen tason sisäkkäisyyttä. Tasojen 3-6 sisäkkäisyyksiä näyttäisi olevan eniten niissä ryhmissä, joissa lupausabstraktiota löydettiin vähiten. Yhteyttä havainnollistetaan kuviossa 8, jossa tarkastellaan lupausabstraktion esiintyvyyttä ja sisäkkäisyystasojen 3-6 osuutta toisiinsa JavaScript-tiedostoissa sivustoryhmittäin.



KUVIO 8: Muuttujien riippuvuuksia

Huolimatta siitä että lupausabstraktio-löydösten määrät olivat vähäisiä, sivustoryhmittäin tarkastellen niiden suurempi määrä näyttäisi esiintyvän yhtäaikaa pienemmän sisäkkäisyyden kanssa. On mahdollista että lupausabstraktiolla olisi korvattukin juuri monimutkaisimpia toimintoja, jolloin selvittäisiin vähemmillä sisäkkäisyyksillä ja tavallisemmat toiminnot hoidettaisiin perinteisellä takaisinkutsumenetelmällä. On myös mahdollista että lupausabstraktion ja vähempien sisäkkäisyyksien yhteys selittyy muulla tavoin kuin suoraan Promise-objektien hyödyntämisen kautta. Selittävä tekijä voisi olla yleinen tietoisuus takaisinkutsuhelvetin ongelmasta ja pyrkimys välttämään suurten sisäkkäisyyksien muodostumista esimerkiksi toisenlaisin arkkitehtuurisin keinoin.

JQuery-kirjastolla näyttää olevan hyvin hallitseva ja vankka asema suomalaisilla verkkosivustoilla. Näin laajalti käytetyn kirjaston vaikutusvalta koko kielen kehitykseen ja kehittäjien mielipiteisiin on kiistaton. Esimerkiksi lupausabstraktion läpilyömistä on mahdollisesti haitannut jQuery-kirjaston sisältämä tulkinta, joka ei ole nykyisen standardin mukainen. Tutkimuksessa ei tutkittu sivustoille ulkoisesta lähteestä dynaamisesti hyödynnettävien kirjastojen osuutta, mutta oli havaittavissa että tätä tyyliä käytettiin paljon. Voisi siis olettaa, että näin laajalti käytetty kirjasto olisi kiinnostava kohde myös vihamielisille toiminnoille.

Voi olla että tulevaisuudessa suositeltava menetelmä asynkronisten toimintojen suorittamiseen on hyödyntää lupausabstraktiorajapintoja. Taaksepäin yhteensopivuuden vuoksi vanhoja rajapintoja ei noin vain voi kuitenkaan ottaa pois toiminnasta, siksi niitä vasten voi kehittää myös uutta toiminnallisuutta. Näin uuden ominaisuuden jalkautuminen voi olla hidasta. Epäilen että lupausabstraktion kanssa voi käydä sama ilmiö kuin tapahtuma-attribuuttien kanssa, eli että uusi ominaisuus käytännössä lisää kompleksisuutta ja toimii vanhan menetelmän rinnalla. Lupausabstraktiolla ja perinteisellä takaisinkutsumenetelmällä toteutettujen osien ohjelmointi poikkeavat merkittävästi toisistaan esimerkiksi virheidenkäsittelyn osalta. Näiden menetelmien sekakäyttö voikin helposti johtaa lisääntyneisiin inhimillisiin virheisiin.

6.2 Tutkimusmenetelmän vahvuudet ja heikkoudet

Tutkimus päätettiin toteuttaa ohjelmallisesti, jotta tutkimukseen sisällytettävän aineiston määrä voisi olla suuri ja manuaalisten työvaiheiden helposti aiheuttamat satunnaiset virheet voitaisiin minimoida. Tämä ratkaisu osoittautui erittäin onnistuneeksi. Automaattisten työvaiheiden rakentaminen vei aikaa joitakin viikkoja, mutta tämän jälkeen se oli käytettävissä rajattomasti ja väsymättömästi päivin ja öin. Alkuperäinen ajatus oli toteuttaa pieniä apuvälineitä GMB-komponenttien ympärille. Tästä lähtökohdasta ajauduttiin kuitenkin hyvin paljon laajemman sovelluksen ohjelmointiin. Muuttujien mittaaminen ei nimittäin onnistunut aivan suoraviivaisesti, sillä hyödynnettävien GMB-komponenttien toiminnassa ja laadussa oli aika paljon toivomisen varaa. Mittausohjelmistoa itse kehittämällä suurin osa puuttuvista tiedoista saatiin kuitenkin mitattua. Valitettavinta on asynkronisten funktiokutsujen mittaustuloksen erittäin suuri har-

haisuus, joka huomattiin vasta loppuvaiheessa vertailtaessa näitä tutkimustuloksia takaisinkutsullisten funktioiden määriin. Näin asynkronisten funktiokutsujen mittaustulokset oli jätettävä pois tämän tutkimuksen tuloksista.

Tutkimusmenetelmällä oli mahdollista tutkia laajoja tietomassoja kohtuullisen vähäisellä manuaalisella työllä ja tulosten automaattinen kirjaus havaintomatriisiin oli myös onnistunut valinta. Näin tutkimustuloksia ei tarvinnut käsitellä paljoakaan manuaalisesti.

Tutkimuksen raaka-aineisto on mahdollista saada käyttöön tutkimuksia vartan. Tähän aineistoon on kohdistettu ainoastaan tiedostojen nimeämiseen liittyviä toimia, jotka tehtiin tutkimuksessa hakuvaiheessa. Näin tutkimuksen toistettavuus on verrattain hyvä.

Tutkimuksen validius tarkoittaa tutkimuksen kykyä mitata sitä, mitä tutkimuksessa oli tarkoituskin mitata, niin ettei systemaattisia virheitä esiinny (Vilkka, 2007). Tutkimuksen validiteettia heikentävät koko mittausohjelmistossa esiintyvät mahdolliset virheet ja puutteet. Mittausohjelmistossa käytetään myös monia toisen osapuolen kehittämiä komponentteja. Näissä olevat mahdolliset virheet kumuloituvat vastaavasti myös tähän tutkimukseen. Mittausohjelmistoa on pyritty testaamaan tutkimuksen validiteetin parantamiseksi, mutta missään tapauksessa testauksesta ei ole voitu tehdä sillä tavalla kattavaa kuin esimerkiksi usein tuotantosovelluksissa edellytetään.

Mittausohjelmisto perustuu staattiseen, offline-analyysiin niistä tiedostoista joita sivustoilta on saatavissa. Sivuston ajonaikainen, dynaaminen tila voi poiketa staattisen analyysin tuloksista. Osa ohjelmakoodista voi esimerkiksi olla sellaista jota ei koskaan ajeta, mutta syystä tai toisesta se löytyy palvelimelta. Tutkimusta varten ladattu sivusto ei ehkä yleensä sisällä kaikkea sivuston toteuttamiseen tarvittavaa HTML- ja JavaScript-materiaalia. Tutkimuksessa kunnioitetaan robots.txt -ohjeistusta, joka voi estää joidenkin tiedostojen lataamisen. Pääasiassa sivustot kuitenkin hoitanevat salassapidettävien tiedostojen turvaamisen esimerkiksi autentikoinnin keinoin. Sivustojen sisältämien saavutettavien tiedostojen määrä siten vaihtelee sivustokohtaisesti, mutta saavutettujen tiedostojen osuutta kaikista tiedostoista ei tiedetä.

HTML-tiedostoista haetut JavaScript-osuudet tarkistettiin syntaksin osalta ennen lisäämistä. Virheellisiä ei talletettu, sillä myöhemmässä vaiheessa GMB-komponentti kaatuisi siihen, jolloin koko tiedoston kaikki tieto hävitettäisiin. Virheellistä koodia oli yllättävän paljon, mutta tätä ei mitattu tässä tutkimuksessa. Funktioiden määräkin saattoi jäädä todellisuutta vähäisemmäksi myös siitä syystä, että tiedostot olivat syntaksiltaan liian vääriä, jolloin mittausohjelma ohitti tällaiset tiedostot tai tiedostonosat.

JavaScript-kirjastojen osalta tutkimus ei tuota positiivista tulosta, mikäli sivuston tuottaja on hyödyntänyt kirjastoa, mutta ladattuaan kirjaston onkin nimennyt tämän tiedoston uudella tavalla. Tämä on täysin mahdollista, mutta ehkä hieman epätyypillistä.

Wget-operaation ominaisuudet vaikuttavat tutkimuksen saamiin tietoihin. Osa tiedostoista saattaa mm. jäädä tunnistamatta HTML tai JavaScript-tiedostoiksi.

Oli myös havaittavissa, että joskus menetelmällä sivustoilta tuli ladatuksi sama tiedosto useaan kertaan eri nimillä. Syytä tähän ei selvitetty eikä tuplatiedostoja poistettu.

6.3 Tutkimuksesta opittua

Mittausohjelmiston luominen kävi JavaScript- ja Node.js-ohjelmointiharjoituksesta. Nämä eivät olleet minulle kovinkaan tuttuja entuudestaan, mutta olin kiinnostunut oppimaan. Mittausohjelmisto kehitettiin siitä lähtökohdasta, että pääasiallisen mittauksen tekisi Gallaban ym. (2015) kehittämä ohjelmisto ja siihen rakennettaisiin pientä aputoiminnallisuutta ympärille. Ajatuksena oli luoda muutamia apuskriptejä, jotka vähentäisivät manuaalista työtä ja vähentäisivät satunnaisia virheitä. Sovelluksen kehittämisessä käytetyt ohjelmointikielet tulivat siten ikäänkuin perintönä tästä ajatuksesta: Gallaban ym. (2015) olivat myös hyödyntäneet bash-skriptausta ja Node.js-ympäristöä ohjelmassaan. Bash-skriptit tulivat luonnollisesti mukaan myös sen myötä että tutkimuksen sivustot päätettiin hakea wget-operaatiolla. Pienessä mittakaavassa tämä oli oikein hyvä tapa sillä bash-skriptit ovat hyvin näppäriä monissa operaatioissa. Ohjelmiston koon kasvaessa ja sen monimutkaistuessa tämä lähestymistapa ei ollut optimaalisin. Mikäli mittausohjelmisto kehitettäisiin alusta asti uudelleen, se tehtäisiin luultavasti lähes kokonaan JavaScriptiä käyttäen ja kokonaisuus tarkemmin suunnitellen. Työn loppupuolella olisinkin halunnut kirjoittaa suuren osan ohjelmasta uudelleen, mikä toisaalta kertoo oppimista tapahtuneen. Valitettavasti tähän ei kuitenkaan ollut mahdollisuutta. Mikäli tekisin vastaavaa työtä uudelleen, hyödyntäisin enemmän valmiita, suosittuja Node.js -moduuleita, joiden oikeellisuutta testaisin ensin. GMB-komponenttia en käyttäisi sen sisältämien lukuisien virheiden vuoksi.

Jälkikäteen ajateltuna olisin luultavasti saanut tehtyä työn nopeammin ja paremmin, mikäli olisin lähtenyt alusta asti tekemään mittausohjelmat itse. Tämä olisi toki vaatinut perehtymistä melko paljon, mutta samalla olisi vältytty muilta ongelmilta ja mitattavia muuttujia olisi voinut määritellä vapaammin. Nyt monia kiinnostavia muuttujia jäi laskematta, myös sellaisia joita on Gallaban ym. (2015) paperissa kuvattu, mutta analysointityökalusta ei saanut kohtuullisessa ajassa varmuutta siitä mitä luvut tarkoittivat tai miten mittaus olisi ollut tarkoitus tehdä. Asynkronisten funktioiden mittaaminen jäi tästä tutkimuksesta pois, koska sen havaittiin tuottavan niin paljon vääriä positiivisia ettei mittauksessa ollut mitään mieltä. Verrattuna takaisinkutsullisiin funktiokutsuihin asynkronisten kutsujen määrät saattoivat olla satoja, jopa tuhansia prosentteja! GMB-komponenttien asynkronisten funktiokutsujen mittaus perustui funktiokutsujen sisältämille merkkijonoille. Esimerkiksi mikä tahansa funktiokutsu joka sisälsi merkkijonon ”net” tuli lasketuksi mukaan asynkronisiin funktiokutsuihin. Komponentti ei mittaa sitä mitä sen on tarkoitus, eikä näin mitattuna Gallaban ym. (2015) tuloksissakaan ole mitään järkeä. Melko pienillä virheillä on mahdollista saada aikaan suurta harhaa mittaustuloksiin. Mittausohjelmiston toiminnallisuuteen ja

laatuun pitäisi kiinnittää erityistä huomiota. Tässäkin työssä saattoi huomata, että kokonaisuuteen nähden näille seikoille jäi aivan liian vähän aikaa ja energiaa.

Opin myös, että hyödynnettäessä muiden tekemiä komponentteja olisi ne testattava huolellisesti ennen hyödyntämispäätöksen tekemistä. Se että komponenteilla on tehty julkaistua ja palkittua tutkimusta ei riitä, vaan komponenttien oikeellisuus on testattava huolellisesti itse.

7 YHTEENVETO JA MAHDOLLINEN JATKOTUTKIMUS

Tutkimus käsitteli JavaScriptin takaisinkutsumenetelmän yleisyyttä verkkosivustoilla ja ongelmia joita menetelmä voi aiheuttaa. Työ jakautui kirjallisuusosioon ja empiiriseen osioon. Kirjallisuusosiossa tutkittiin takaisinkutsumenetelmään liitettyjä ongelmia sekä joitakin korvaavia menetelmiä. Kirjallisuusosio sijoittuu lukuihin kaksi ja kolme. Empiirisessä osiossa suoritettiin automaattinen ja ohjelmallinen tutkimus suomalaisten verkkopalveluita sisältävien sivustojen JavaScript-ohjelmakoodeista ja takaisinkutsumenetelmän yleisyydestä. Empiirinen tutkimus työtapoineen ja menetelmineen esiteltiin ja perusteltiin luvussa neljä. Tutkimuksen tulokset ja pohdinta sijoittuvat lukuihin viisi ja kuusi.

Kirjallisuustutkimus osoitti, että takaisinkutsumenetelmään on kirjallisuudessa liitetty monenlaisia ongelmia, jotka aiheuttavat helposti inhimillisiä virheitä. Esimerkiksi ohjelmia ja niiden sisäistä tilaa voi olla vaikea ymmärtää ja ohjelmien virheidenkäsittely joudutaan tekemään manuaalisesti. Korvaavista menetelmistä lupausabstraktiota eli JavaScriptin Promise-toiminnallisuutta käsiteltiin tarkemmin ja todettiin sen vähentävän joitakin löydetyistä ongelmista.

Empiirisessä osuudessa tutkittiin 36,5 miljoonaa riviä JavaScript-koodia. Tutkimus käsitti 134 suomalaista verkkosivustoa viidestä eri kategoriasta. Nämä ovat finanssisektori, pikavipit, rahapelit, julkinen sektori ja verkkokaupat. Sivustoilta mitattiin JavaScript-koodin takaisinkutsumenetelmän ja joidenkin sitä korvaavien menetelmien yleisyyttä, takaisinkutsullisten funktioiden sisäkkäisyyksiä sekä mm. tapahtuma-attribuuttien käyttöä. Mittaustulokset talletettiin havaintomatriiseihin (liite 1).

Tutkimusta varten kehitettiin räätälöity mittausohjelmisto, jonka avulla sivustojen hakeminen, mittaaminen ja tulosten kirjaaminen suoritettiin. Mittausohjelmiston avulla kaikki sivustot voitiin käsitellä samalla tavalla ja mahdollisimman vähäisellä manuaalisella työllä. Mittausohjelmistossa pyrittiin hyödyntämään Gallaban ym. (2015) kehittämiä mittauskomponentteja. Näiden sisältämien virheiden vuoksi suurin osa varsinaisesta mittauksestakin jouduttiin kuitenkin kehittämään uudelleen. Asynkronisten funktiokutsujen osalta mittausoh-

jelmisto toimi erityisen heikosti. Uutta toiminnallisuutta ei ehditty tältä osin tekemään ja siksi se jäi valitettavasti pois tästä tutkimuksesta.

Tutkimuksessa todettiin, että takaisinkutsumenetelmä on hyvin yleinen menetelmä ja tyypillisesti takaisinkutsullisten funktiokutsujen muodostama sisäkkäisyys on maltillista. Joka kymmenes takaisinkutsullinen funktiokutsu on kuitenkin vähintään sisäkkäisyystasolla kolme ja aineistossa esiintyi myös jopa tasolle kuusi ulottuvia ketjuja. Lupausabstraktiota käytettiin jossain määrin kolmasosassa sivustoja, mutta sen hyödyntämisen määrät olivat vielä vähäisiä. Tutkimuksessa esiintyi kuitenkin viitteitä siitä, että juuri monimutkaisimpia toimintoja olisi toteutettu lupausabstraktiota hyödyntämällä ja perustoiminnallisuus edelleen perinteisellä tavalla. Kirjastojen osalta tutkimuksessa havaittiin, että async.js-kirjaston hyödyntäminen oli vähäistä. JQuery-kirjasto oli sen sijaan odotetusti hyvin yleinen. Tapahtuma-attribuuttien hyödyntäminen oli edelleen yleistä, vaikka tätä menetelmää on suositeltu vältettäväksi. JavaScript-koodia löytyi myös rivimääräisesti laskettuna enemmän HTML-tiedostoista kuin varsinaisista JavaScript-tiedostoista.

Empiirisen tutkimuksen tuloksia verrattiin vuonna 2015 julkaistun tutkimuksen (Gallaban ym. 2015) tuloksiin. Tuon julkaistun tutkimuksen mittausohjelmisto oli tässä tutkimuksessa käytettävissä ja siinä valitettavasti havaittiin huomattavia virheitä. Osa julkaistun tutkimuksen mittaustuloksista jouduttiin siten nyt tulkitsemaan harhaisiksi. Vertailukelpoisten tulosten osalta empiirisen tutkimuksen tulokset olivat kuitenkin pääosin yhteneväisiä Gallaban ym. (2015) tutkimustulosten kanssa.

Mittausohjelmiston oikeellisuus ja laatu on tämänkin tutkimuksen tuloksiin merkittävästi vaikuttava tekijä. Työssä on pyritty huolellisuuteen, mutta valitettavasti kattava testaus ei ollut pro gradu -työn puitteissa mahdollista. Siten mittausohjelmiston erilaiset virheet voivat vähentää työn validiteettia.

Tutkimuksessa kerätyistä ja tutkielman liitteenä olevista havaintomatriiseista olisi mahdollista tehdä jatkotutkimusta. Esimerkiksi lupausabstraktion ja matalamman sisäkkäisyyden välistä yhteyttä olisi hyvä selvittää myös sivustoittain tarkasteltuna. Tätä tutkimusta varten tehtyjä välineitä ja tutkimuksessa kerättyä aineistoa on myös mahdollista saada käyttöön jatkotutkimuksia varten. Ne on saatavilla Jyväskylän yliopiston Dataverse-arkistossa (https://dvn.jyu.fi/dvn/). Kerätyn aineiston avulla tutkimus on mahdollista toistaa ja aineistosta voi tutkia muita kiinnostavia tietoja. Erityisesti asynkronisten funktiokutsujen mittaaminen olisi hyödyllistä. Tutkimuksessa kehitetyllä mittausohjelmistolla olisi mahdollista tehdä jatkotutkimusta muista kiinnostavista verkkosivustoista tai JavaScript-ohjelmista. Voisi olla kiinnostavaa mitata Gallaban ym. (2015) mittaamien avoimen lähdekoodin projektien ohjelmakoodeja myös tällä mittausohjelmalla. Voi olla että tulokset eroaisivat heidän tuloksistaan.

Tutkimus käsitteli vähän tutkittua aluetta, ja sen myötä heräsi myös monia muita kysymyksiä jatkotutkimuksia varten. JavaScriptin tiedetään sisältävän usein virheitä. Yllättävää sen sijaan oli, että erityisesti HTML-tiedostojen sisältämät JavaScript-koodit eivät aina olleet edes syntaktisesti oikein. Tätä ei tässä tutki-

muksessa varsinaisesti mitattu, mutta se tuli esille työn aikana. Olisikin mielenkiintoista tutkia lisää miten yleisiä syntaksivirheet ovat HTML-tiedostoihin sisällytetyssä JavaScriptissä ja ovatko ne mahdollisesti sivuston toimintaan vaikuttavia virheitä. On mahdollista että selaimien toiminnallisuus korjaa tällaisia virheitä, jolloin ne eivät todellisuudessa aiheuta ongelmia.

Muita jatkotutkimusaiheita voisi olla esimerkiksi robots.txt -tiedoston hyödyntämisen tutkiminen. Olisi mielenkiintoista tietää miten suurelti ja millä tavalla niitä käytetään sivustoilla ja kuinka suuri osuus sivustoista niillä pyritään piilottamaan sekä tehdäänkö sitä järkevästi. Myös tapahtuma-attribuuttien luomista mahdollisista ongelmista olisi mielenkiintoista saada tutkittua tietoa. Tuottavatko ne virheherkempää ohjelmakoodia koodin sirpaloitumisen, sulkeuman monimutkaisuuden ja joidenkin muiden syiden vuoksi.

Olisi myös mielenkiintoista tutkia verkkopalveluita sisältävien sivustojen JavaScript-koodia siltäkin osalta joka on autentikoinnin suojissa. Ohjelmakoodin vaikuttavuuden näkökulmasta todennäköisesti kaikkein mielenkiintoisimmat koodit ovat niitä jotka todella toteuttavat operatiivista verkkopalvelua. Lisäksi olisi kiinnostavaa tietää millaisia JavaScript-kirjastoja sivustot hyödyntävät koodissaan ja käytetäänkö turvallisuusuhkaksi todettua kirjaston liittämistä verkon yli paljon ja millaisista lähteistä.

LÄHTEET

Adya, A., Howell, J., Theimer, M., Bolosky, W. J., & Douceur, J. R. (2002). Cooperative task management without manual stack management. Teoksessa *USENIX Annual Technical Conference, General Track* (s. 289–302). USENIX Association.

Alimadadi, S., Sequeira, S., Mesbah, A., & Pattabiraman, K. (2014). Understanding JavaScript event-based interactions. Teoksessa *Proceedings of the 36th International Conference on Software Engineering* (s. 367–377). ACM.

Andrews, G. R. (1991). *Concurrent Programming: Principles and Practice*. Benjamin/Cummings Publishing Company.

Arpaci-Dusseau, R. H. & Arpaci-Dusseau, A. C. (2015). *Operating Systems: Three Easy Pieces*. Arpaci-Dusseau Books. Haettu 10.11.2015 osoitteesta http://pages.cs.wisc.edu/ remzi/OSTEP.

Azegami, T., Fukuda, H., & Leger, P. (2014). Towards a virtual block approach to tame asynchronous programming. Teoksessa *Proceedings of the 8th International Conference on Bioinspired Information and Communications Technologies* (s. 239–242). ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

Bainomugisha, E., Carreton, A. L., van Cutsem, T., Mostinckx, S., & De Meuter, W. (2013). A survey on reactive programming. *ACM Computing Surveys*, *45*(4), 1–34.

Bierman, G., Russo, C., Mainland, G., Meijer, E., & Torgersen, M. (2012). Pause'n'play: Formalizing asynchronous C sharp. Teoksessa *ECOOP 2012–Object-Oriented Programming* (s. 233–257). Springer.

Billes, M. (2015). Race-driven UI-level test generation for JavaScript-based web applications. Teoksessa *Companion Proceedings of the 2015 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity* (s. 81–82). New York, NY, USA: ACM.

Boduch, A. (2015). *JavaScript Concurrency*. Packt Publishing Ltd.

Brodu, E., Frénot, S., & Oblé, F. (2015). Toward automatic update from callbacks to promises. Teoksessa *Proceedings of the 1st Workshop on All-Web Real-Time Systems*. ACM.

Chaffer, J. & Swedberg, K. (2013). *Learning jQuery* (4. painos). Birmingham: Packt Publishing.

Denicola, D. (2012). *You're Missing the Point of Promises*. Haettu 4.5.2016 osoitteesta https://blog.domenic.me/youre-missing-the-point-of-promises/.

Dijkstra, E. W. (1968). Letters to the editor: go to statement considered harmful. *Communications of the ACM*, *11*(3), 147–148.

Dijkstra, E. W. (1976). *A Discipline of Programming*. Upper Saddle River, NJ, USA: Prentice Hall PTR.

ECMA (2015). *ECMAScript 2015 Language Specification*. ECMA International. Haettu 10.10.2015 osoitteesta http://www.ecma-international.org/ecma-262/6.0/index.html.

Edwards, J. (2009). Coherent reaction. Teoksessa *Proceedings of the 24th ACM SIGPLAN Conference Companion on Object Oriented Programming Systems, Languages and Applications* (s. 925–932). ACM.

Fard, A. M. & Mesbah, A. (2013). JSNOSE: Detecting JavaScript code smells. Teoksessa *Source Code Analysis and Manipulation (SCAM), 2013 IEEE 13th International Working Conference on* (s. 116–125). IEEE.

Flanagan, D. (2006). *JavaScript: the definitive guide*. O'Reilly Media, Inc.

Friedman, D. P., Haynes, C. T., & Kohlbecker, E. (1984). Programming with continuations. Teoksessa Pepper, P. (toim.), *Program Transformation and Programming Environments* (s. 263–274). Berlin Hedelberg: Springer.

Friedman, D. P. & Wise, D. S. (1976). *The Impact of Applicative Programming on Multiprocessing*. Technical report, Indiana University, Computer Science Department.

Gallaba, K., Mesbah, A., & Beschastnikh, I. (2015). Don't call us, we'll call you: Characterizing callbacks in JavaScript. Teoksessa *Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on* (s. 1–10). IEEE.

Garrett, J. J. (2005). *AJAX: A New Approach to Web Applications*. Technical report. Haettu 3.2.2016 osoitteesta https://courses.cs.washington.edu/courses/cse490h/07sp/readings/ajax_adaptive_path.pdf.

Gaud, F., Geneves, S., Lachaize, R., Lepers, B., Mottet, F., Muller, G., & Quéma, V. (2010). Efficient workstealing for multicore event-driven systems. Teoksessa *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on* (s. 516–525). IEEE.

Graziotin, D. & Abrahamsson, P. (2013). Making sense out of a jungle of javascript frameworks. Teoksessa Heidrich, J., Oivo, M., Jedlitschka, A., & Baldassarre, M. (toim.), *Product-Focused Software Process Improvement* (s. 334–337). Berlin, Heidelberg: Springer.

Harmes, R., Diaz, D., & Willison, S. (2008). *Pro JavaScript Design Patterns*. Berkeley CA: Apress.

Haverbeke, M. (2014). *Eloquent JavaScript: A Modern Introduction to Programming*. No Starch Press.

Herman, D. (2012). *Effective JavaScript: 68 Specific Ways to Harness the Power of JavaScript*. Addison-Wesley Professional.

Johnson, R. E. & Foote, B. (1988). Designing reusable classes. *Journal of Object-Oriented Programming*, *1*(2), 22–35.

jQuery (2016). *jQuery-library*. Haettu 8.2.2016 osoitteesta http://jquery.com.

Kambona, K., Boix, E. G., & De Meuter, W. (2013). An evaluation of reactive programming and promises for structuring collaborative web applications. Teoksessa *Proceedings of the 7th Workshop on Dynamic Languages and Applications* (s. 1–9). New York, NY, USA: ACM.

Kananen, J. (2015). *Opinnäytetyön kirjoittajan opas : näin kirjoitan opinnäytetyön tai pro gradun alusta loppuun*. Jyväskylä: Jyväskylän ammattikorkeakoulu.

Koskimies, K. & Mikkonen, T. (2005). *Ohjelmistoarkkitehtuurit*. Helsinki: Talentum.

Li, D., Mickens, J., Nath, S., & Ravindranath, L. (2015). Domino: Understanding wide-area, asynchronous event causality in web applications. Teoksessa *Proceedings of the Sixth ACM Symposium on Cloud Computing* (s. 182–188). New York, NY, USA: ACM.

Liskov, B. & Shrira, L. (1988). Promises: linguistic support for efficient asynchronous procedure calls in distributed systems. Teoksessa *Proceedings of the ACM SIGPLAN '88 Conference on Programming Languages Design and Implementation* (s. 260–267). New York, NY, USA: ACM.

McMahon, C. (2016). *Async.js-library*. Haettu 2.2.2016 osoitteesta https://github.com/caolan/async.

Mickens, J. (2014). Pivot: Fast, synchronous mashup isolation using generator chains. Teoksessa *Proceedings of the 2014 IEEE Symposium on Security and Privacy* (s. 261–275). Washington, DC, USA: IEEE Computer Society.

Mickens, J., Elson, J., & Howell, J. (2010). Mugshot: Deterministic capture and replay for javascript applications. Teoksessa *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation* (s. 159–174). Berkeley, CA: USENIX Association.

Mikkonen, T. & Taivalsaari, A. (2007). *Web Applications: Spaghetti Code for the 21st Century*. Technical report, Sun Microsystems, Inc.

Mogk, R. (2015). Concurrency control for multithreaded reactive programming. Teoksessa *Companion Proceedings of the 2015 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity* (s. 77–78). New York, NY, USA: ACM.

Mozilla (2014). *Event Attributes*. Haettu 12.5.2016 osoitteesta https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Event_attributes.

Mozilla (2015). *AJAX*. Haettu 15.6.2016 osoitteesta https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest/Synchronous_and_Asynchronous_Requests.

Myers, B. A. (1991). Separating application code from toolkits: eliminating the spaghetti of call-backs. Teoksessa *Proceedings of the 4th Annual ACM Symposium on User Interface Software and Technology* (s. 211–220). ACM.

Ocariza, F., Bajaj, K., Pattabiraman, K., & Mesbah, A. (2013). An empirical study of client-side JavaScript bugs. Teoksessa *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (s. 55–64). IEEE.

Ocariza, F., Pattabiraman, K., & Zorn, B. (2011). JavaScript errors in the wild: An empirical study. Teoksessa *Proceedings of the 2011 IEEE 22Nd International Symposium on Software Reliability Engineering* (s. 100–109). IEEE.

Ogden, M. (2015). *Callback hell*. Haettu 10.9.2015 osoitteesta http://callbackhell.com.

Otero, C. J. & Larsen, R. (2012). *Professional jQuery*. Indianapolis, IN: Wiley Pub., Inc.

Ousterhout, J. (1996). *Why threads are a bad idea (for most purposes)*. Haettu 10.10.2015 osoitteesta http://www-

lb.cs.umd.edu/class/fall2004/cmsc433/lectures/threadsBad.pdf.

Parker, D. (2015). *JavaScript with Promises*. Sebastopol, CA: O'Reilly Media, Inc.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, *15*(12), 1053–1058.

Petrov, B., Vechev, M., Sridharan, M., & Dolby, J. (2012). Race detection for web applications. Teoksessa *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation* (s. 251–262). New York, NY, USA: ACM.

Promises/A+ (2015). *Promise Specification*. Haettu 10.10.2015 osoitteesta https://promisesaplus.com.

Raychev, V., Vechev, M., & Sridharan, M. (2013). Effective race detection for event-driven programs. Teoksessa *Proceedings of OOPSLA'13, ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications* (s. 151–166). New York, NY, USA: ACM.

Reaktor (2016). *Reaktor Space*. Haettu 19.6.2016 osoitteesta https://reaktor.com/space.

Severance, C. (2012). JavaScript: Designing a language in 10 days. *Computer*, (2), 7–8.

Simpson, K. (2014). *You Don't Know JS: Scope & Closures*. O'Reilly Media, Inc.

Simpson, K. (2015). *You Don't Know JS: Async & Performance*. O'Reilly Media, Inc.

StackOverflow (2015). *Developer Survey*. Haettu 20.9.2015 osoitteesta http://stackoverflow.com/research/developer-survey-2015.

Syed, B. (2014). *Beginning Node.js*. Berkeley CA: Apress.

Szyperski, C., Gruntz, D., & Murer, S. (1998). *Component Software: Beyond Object-Oriented Programming*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.

Vilkka, H. (2007). *Tutki ja mittaa : määrällisen tutkimuksen perusteet*. Helsinki: Tammi.

von Behren, J. R., Condit, J., & Brewer, E. A. (2003). Why events are a bad idea (for high-concurrency servers). Teoksessa *HotOS* (s. 19–24).

W3C (2011). *Evolution of Events*. Haettu 19.6.2016 osoitteesta https://www.w3.org/wiki/Handling_events_with_JavaScript.

Welsh, M., Culler, D., & Brewer, E. (2001). SEDA: an architecture for well-conditioned, scalable internet services. *ACM SIGOPS Operating Systems Review*, *35*(5), 230–243.

Yue, C. & Wang, H. (2013). A measurement study of insecure JavaScript practices on the web. *ACM Transactions on the Web*, *7*(2), 1–39.

Zakas, N. C. (2009). *Professional JavaScript for Web Developers* (2. painos). Indianapolis: Wiley Pub.

Zeldovich, N., Yip, A., Dabek, F., Morris, R., Mazieres, D., & Kaashoek, M. F. (2003). Multiprocessor support for event-driven programs. Teoksessa *USENIX Annual Technical Conference, General Track* (s. 239–252). San Antonio, TX, USA: USENIX Association.

Zheng, Y., Bao, T., & Zhang, X. (2011). Statically locating web application bugs

caused by asynchronous calls. Teoksessa *Proceedings of the 20th international conference on World Wide Web* (s. 805–814). New York, NY, USA: ACM.

LIITE 1 HAVAINTOMATRIISIT

Finanssisektori: muuttujat

1	2	3	4	5	6	7	8	9	10	11	12	13
Sivusto	Js-tiedostot (lkm)	Html-tiedostot (lkm)	Html-tiedostot (lkm)	Js-tiedostot: Js loc	Html-tiedostot: Js loc	Html-tiedostot: Html loc	Js-tied: Funktio-kutsut	Html-tied: Funktio-kutsut	Js-tied: Takaisin-kutsulliset funktiokutsut	Html-tied: Takaisin-kutsulliset funktiokutsut	Js-tied: Anonyymit takaisinkutsut	Html-tied: Anonyymit takaisinkutsut
1	1	10367	1000	11169	21524	637744	6870	20187	1031	3779	413	3779
2	30	618	618	64893	102194	355524	30312	8900	2703	1296	1938	1296
3	3	94	94	1135	2258	12044	822	480	89	178	51	91
4	221	8377	1000	274448	170976	500486	117441	77866	7543	5207	5896	4187
5	7	683	683	21954	28005	439877	15190	9077	2356	58	1137	42
6	21	792	792	16325	50378	276830	6535	22230	735	1584	414	792
7	11	484	484	15145	36101	238174	6535	23706	1519	4719	522	4719
8	1	11853	1000	11169	11360	528142	6870	7811	1031	2113	413	2113
9	19	1477	1000	19540	155568	642976	8577	89041	1847	14225	496	13347
10	12	151306	1000	7740	15784	594853	3631	3968	337	10	248	10
11	4	533	533	37170	68562	313890	21057	18674	5554	1066	1634	1066
12	10	912	912	22758	21888	307115	10605	9120	1156	912	772	912
13	35	881	881	27168	444832	818064	14953	207660	1405	32842	814	32842
14	11	670	670	9188	80437	345319	2989	35330	765	6355	183	6355
15	42	1132	845	31050	73180	457085	18766	64682	3476	14222	1212	12170
16	16	1665	1000	19686	242532	654515	9902	90333	1337	4106	601	4064
17	2	42	42	1676	1428	4731	756	986	151	128	95	32
18	32	229	229	155418	115943	1434165	44583	10056	4593	458	2857	458
19	0	1831	1000	0	168263	1473846	0	93724	0	30153	0	29162
20	78	2064	1000	323176	591481	8904397	97046	80796	10450	5035	6203	5021
21	16	4269	1000	11352	207991	593508	6259	79409	1359	18328	522	18323
22	69	24649	1000	89042	519570	1179611	33910	258439	3723	64554	1790	64554
23	39	8497	1000	41801	39993	998006	20632	15080	2456	6731	1417	6728
24	6	752	662	6202	58150	415873	3528	40614	768	9101	291	7118
25	4	40477	1000	12467	14078	705053	7701	15005	1119	2044	491	2044
26	49	4385	1000	27176	518059	327272	10807	181618	966	16766	581	16766
27	11	158	158	10742	17299	74744	5862	4540	1306	395	428	326
28	17	50	50	6601	4235	19131	4390	2242	881	266	327	116
29	4	273	273	37001	40780	188120	20976	10730	5539	550	1629	550
30	1447	3852	1000	971804	470954	1274726	306998	137921	15824	15937	12422	15475
31	15	155	155	16910	30738	46539	8189	14999	664	3876	456	3876
32	14	1900	1000	11580	35911	1260600	6387	15169	1378	2837	472	2837
33	0	1107	1000	0	215377	1721307	0	127898	0	41921	0	40950

1	14	15	16	17	18	19	20	21	22	23	24	25
Sivusto	Html-tied: Tapahtuma-attribuutit	Js-tied: Kaikki takaisin-kutsu-parametrit	Html-tied: Kaikki takaisin-kutsu-parametrit	Js-tied: Kaikki anonyy funktioparame	Html-tied: Kaikki anonyy funktioparame	Js-tied: Takaisin-kutsuja-funktiomäär.	Html-tied: Takaisin-kutsuja-funktiomäär.	Js-tied: Funktio-määritykset	Html-tied: Funktio-määritykset	Kaikki tied: Promiset ".then(" lkm	Kaikki tied: Async.js	Kaikki tied: jQuery
1	1971	1126	3779	413	3779	40	0	1539	4683	1	0	0
2	60	2770	1296	1942	1296	329	14	6615	1948	11	0	9553
3	0	90	178	51	91	14	0	191	188	0	0	91
4	2623	7845	5207	5936	4187	824	8	23130	13875	12	0	1540
5	10	2537	58	1143	42	130	0	3935	1451	6	0	1
6	0	762	1584	415	792	70	792	2015	5553	0	0	6339
7	27	1714	4719	522	4719	36	0	1860	5741	0	0	484
8	157	1126	2113	413	2113	40	0	1539	3091	1	0	2
9	1094	2159	14260	496	13364	32	116	2296	23251	4	0	1145
10	0	341	10	249	10	21	2	965	1324	3	0	3942
11	0	6400	1066	1654	1066	254	0	6264	6398	88	0	533
12	912	1162	912	774	912	126	0	3037	1824	0	0	10034
13	4126	1469	34782	818	34782	110	1	3340	44394	0	0	10617
14	3330	867	8704	184	8704	15	0	819	11978	0	0	2210
15	2470	4030	14222	1221	12170	121	28	3500	17771	0	0	3240
16	1083	1393	4106	601	4064	53	948	2827	24547	0	0	114
17	0	157	128	95	32	8	0	304	257	0	0	32
18	0	4789	458	2857	458	196	0	13630	19007	0	0	0
19	29102	0	30153	0	29162	0	1011	0	32126	0	0	0
20	3013	10818	5041	6204	5021	460	7	29251	96805	4	0	1024
21	15341	1445	18328	522	18323	49	0	2045	25277	0	0	29
22	62457	4004	64554	1799	64554	263	3	7968	91688	9	0	2993
23	5023	2740	6731	1531	6728	151	4	5254	6827	10	0	36
24	3624	832	9101	292	7118	44	1385	1044	12406	0	0	1983
25	2018	1215	2044	491	2044	47	1	1701	2060	1	0	2
26	6661	989	16766	586	16766	94	0	2556	92864	3	0	8060
27	50	1450	395	428	326	71	0	1880	1243	0	0	159
28	0	966	266	333	116	41	2	1140	482	0	0	354
29	14	6381	550	1649	550	254	0	6234	3754	86	0	273
30	5538	15846	15937	12425	15475	6225	6	89910	79845	2	0	6878
31	2506	683	3876	464	3876	45	0	1697	5527	3	0	466
32	997	1516	2877	477	2877	61	0	1907	4888	0	0	2022
33	40905	0	41922	0	40950	0	971	0	43963	0	0	21

Finanssisektori: johdetut arvot

	A1	A2	A3	B1	B2	B3	C2	C3	D2	D3	E2	E3	F1	F2	F3
Sivusto	JS: tkf / fk	HTML: tkf / fk	JS ja HTML: tkf / fk (est)	JS: anon / tkf	HTML: anon / tkf	JS ja HTML: anon / tkf (est)	HTML: tap.attr / fk	JS ja HTML: tap.attr / fk (est)	HTML: tap.attr / tkf	JS ja HTML: tap.attr / tkf	HTML: tap.attr / anon	JS ja HTML: tap.attr / anon (est)	JS: kaikki anon / kaikki tkf	HTML: kaikki anon / kaikki tkf	JS ja HTML: kaikki anon / kaikki tkf (est)
1	15.01%	18.72%	18.60%	40.06%	100.00%	98.46%	9.76%	9.45%	52.16%	50.82%	52.16%	51.61%	36.68%	100.00%	98.23%
2	8.92%	14.56%	10.20%	71.70%	100.00%	80.87%	0.67%	0.15%	4.63%	1.50%	4.63%	1.86%	70.11%	100.00%	79.64%
3	10.83%	37.08%	20.51%	57.30%	51.12%	53.18%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	56.67%	51.12%	52.99%
4	6.42%	6.69%	6.65%	78.17%	80.41%	80.08%	3.37%	2.85%	50.37%	42.95%	62.65%	53.63%	75.67%	80.41%	79.69%
5	15.51%	0.64%	9.95%	48.26%	72.41%	48.84%	0.11%	0.04%	17.24%	0.41%	23.81%	0.85%	45.05%	72.41%	45.66%
6	11.25%	7.13%	8.06%	56.33%	50.00%	52.01%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	54.46%	50.00%	51.45%
7	23.24%	19.91%	20.63%	34.36%	100.00%	84.02%	0.11%	0.09%	0.57%	0.43%	0.57%	0.52%	30.46%	100.00%	81.47%
8	15.01%	27.05%	26.22%	40.06%	100.00%	97.63%	2.01%	1.87%	7.43%	7.14%	7.43%	7.31%	36.68%	100.00%	97.28%
9	21.53%	15.98%	16.32%	26.85%	93.83%	88.42%	1.23%	1.15%	7.69%	7.07%	8.20%	8.00%	22.97%	93.72%	87.14%
10	9.28%	0.25%	0.31%	73.59%	100.00%	95.19%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	73.02%	100.00%	95.04%
11	26.38%	5.71%	16.66%	29.42%	100.00%	40.79%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	25.84%	100.00%	36.43%
12	10.90%	10.00%	10.48%	66.78%	100.00%	81.43%	10.00%	4.62%	100.00%	44.10%	100.00%	54.16%	66.61%	100.00%	81.29%
13	9.40%	15.82%	15.38%	57.94%	100.00%	98.27%	1.99%	1.85%	12.56%	12.05%	12.56%	12.26%	55.68%	100.00%	98.20%
14	25.59%	17.99%	18.58%	23.92%	100.00%	91.83%	9.43%	8.69%	52.40%	46.77%	52.40%	50.93%	21.22%	100.00%	92.86%
15	18.52%	21.99%	21.37%	34.87%	85.57%	77.75%	3.82%	3.14%	17.37%	14.69%	20.30%	18.89%	30.30%	85.57%	75.92%
16	13.50%	4.55%	5.10%	44.95%	98.98%	90.14%	1.20%	1.12%	26.38%	22.06%	26.65%	24.47%	43.14%	98.98%	89.53%
17	19.97%	12.98%	16.02%	62.91%	25.00%	45.52%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	60.51%	25.00%	44.56%
18	10.30%	4.55%	9.24%	62.20%	100.00%	65.63%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	59.66%	100.00%	63.18%
19		32.17%	32.17%		96.71%	96.71%	31.05%	31.05%	96.51%	96.51%	99.79%	99.79%		96.71%	96.71%
20	10.77%	6.23%	7.90%	59.36%	99.72%	79.48%	3.73%	2.36%	59.84%	29.84%	60.01%	37.54%	57.35%	99.60%	78.06%
21	21.71%	23.08%	23.06%	38.41%	99.97%	98.92%	19.32%	18.97%	83.70%	82.27%	83.73%	83.17%	36.12%	99.97%	98.81%
22	10.98%	24.98%	24.90%	48.08%	100.00%	99.88%	24.17%	24.04%	96.75%	96.53%	96.75%	96.64%	44.93%	100.00%	99.86%
23	11.90%	44.64%	40.10%	57.70%	99.96%	98.22%	33.31%	28.69%	74.62%	71.55%	74.66%	72.85%	55.88%	99.96%	97.94%
24	21.77%	22.41%	22.36%	37.89%	78.21%	75.42%	8.92%	8.29%	39.82%	37.07%	50.91%	49.14%	35.10%	78.21%	75.00%
25	14.53%	13.62%	13.63%	43.88%	100.00%	99.25%	13.45%	13.28%	98.73%	97.41%	98.73%	98.15%	40.41%	100.00%	99.14%
26	8.94%	9.23%	9.23%	60.14%	100.00%	99.48%	3.67%	3.62%	39.73%	39.21%	39.73%	39.42%	59.25%	100.00%	99.46%
27	22.28%	8.70%	16.35%	32.77%	82.53%	44.33%	1.10%	0.48%	12.66%	2.94%	15.34%	6.63%	29.52%	82.53%	40.87%
28	20.07%	11.86%	17.29%	37.12%	43.61%	38.62%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	34.47%	43.61%	36.44%
29	26.41%	5.13%	19.20%	29.41%	100.00%	35.79%	0.13%	0.04%	2.55%	0.23%	2.55%	0.64%	25.84%	100.00%	31.73%
30	5.15%	11.56%	9.21%	78.50%	97.10%	93.29%	4.02%	2.54%	34.75%	27.63%	35.79%	29.62%	78.41%	97.10%	93.27%
31	8.11%	25.84%	19.58%	68.67%	100.00%	95.42%	16.71%	10.81%	64.65%	55.20%	64.65%	57.85%	67.94%	100.00%	95.20%
32	21.58%	18.70%	19.22%	34.25%	100.00%	86.61%	6.57%	5.38%	35.14%	27.99%	35.14%	32.31%	31.46%	100.00%	85.12%
33		32.78%	32.78%		97.68%	97.68%	31.98%	31.98%	97.58%	97.58%	99.89%	99.89%		97.68%	97.68%

	G1	G2	G3	H1	H2	H3
Sivusto	JS: tkfm / fm	HTML: tkfm / fm	JS ja HTML: tkfm / fm (est)	JS: kaikki tkf / tkf	HTML: kaikki tkf / tkf	JS ja HTML: kaikki tkf / tkf
1	2.60%	0.00%	0.08%	1.09	1.00	1.00
2	4.97%	0.72%	4.01%	1.02	1.00	1.02
3	7.33%	0.00%	3.69%	1.01	1.00	1.00
4	3.56%	0.06%	0.64%	1.04	1.00	1.01
5	3.30%	0.00%	2.41%	1.08	1.00	1.07
6	3.47%	14.26%	11.39%	1.04	1.00	1.01
7	1.94%	0.00%	0.47%	1.13	1.00	1.03
8	2.60%	0.00%	0.10%	1.09	1.00	1.00
9	1.39%	0.50%	0.55%	1.17	1.00	1.02
10	2.18%	0.15%	0.16%	1.01	1.00	1.00
11	4.05%	0.00%	2.01%	1.15	1.00	1.13
12	4.15%	0.00%	2.59%	1.01	1.00	1.00
13	3.29%	0.00%	0.23%	1.05	1.06	1.06
14	1.83%	0.00%	0.12%	1.13	1.37	1.34
15	3.46%	0.16%	0.58%	1.16	1.00	1.02
16	1.87%	3.86%	3.73%	1.04	1.00	1.01
17	2.63%	0.00%	1.43%	1.04	1.00	1.02
18	1.44%	0.00%	0.60%	1.04	1.00	1.04
19		3.15%	3.15%		1.00	1.00
20	1.57%	0.01%	0.21%	1.04	1.00	1.02
21	2.40%	0.00%	0.04%	1.06	1.00	1.00
22	3.30%	0.00%	0.01%	1.08	1.00	1.00
23	2.87%	0.06%	0.29%	1.12	1.00	1.00
24	4.21%	11.16%	10.68%	1.08	1.00	1.01
25	2.76%	0.05%	0.10%	1.09	1.00	1.00
26	3.68%	0.00%	0.02%	1.02	1.00	1.00
27	3.78%	0.00%	2.27%	1.11	1.00	1.08
28	3.60%	0.41%	2.65%	1.10	1.00	1.07
29	4.07%	0.00%	2.54%	1.15	1.00	1.14
30	6.92%	0.01%	1.57%	1.00	1.00	1.00
31	2.65%	0.00%	0.62%	1.03	1.00	1.00
32	3.20%	0.00%	0.54%	1.10	1.01	1.03
33		2.21%	2.21%		1.00	1.00

	I1	I2	J1	J2	K1	K2
	JS: Maksimi-sisäkkäis.	HTML: Maksimi-sisäkkäis.	JS: Sisäkkäis. keskiarvo	HTML: Sisäkkäis. keskiarvo	JS: Sisäkkäis. mediaani	HTML: Sisäkkäis. mediaani
	4	2	1.68	1.24	2	1
	4	1	1.78	1.00	2	1
	2	1	1.11	1.00	1	1
	5	2	1.79	1.00	2	1
	5	2	1.54	1.24	1	1
	4	1	1.47	1.00	1	1
	5	1	1.62	1.00	1	1
	4	2	1.68	1.46	2	1
	5	4	2.06	1.10	2	1
	4	3	1.80	2.00	2	2
	4		1.49		1	
	3	1	1.52	1.00	1	1
	4	3	1.48	1.62	1	2
	3	1	1.51	1.00	1	1
	6	2	1.77	1.66	1	2
	4	2	1.61	1.24	2	1
	3		2.27	0.00	2	0
	4	2	1.81	1.50	2	1
		3		1.00		1
	5	2	1.80	1.08	2	1
	5	3	2.34	1.00	2	1
	5	2	1.51	1.00	1	1
	4	2	1.46	1.00	1	1
	5	3	2.02	1.23	2	1
	4	2	1.75	1.00	2	1
	5	2	1.71	1.38	2	1
	5	2	1.84	1.21	2	1
	6	1	2.31	1.00	2	1
	4	1	1.48	1.00	1	1
	4	2	1.56	1.08	1	1
	5	3	1.77	1.16	2	1
	5	2	1.78	1.31	2	1
		3		1.00		1

	L2
	HTML: js loc / loc
	3.26%
	22.33%
	15.79%
	25.46%
	5.99%
	15.40%
	13.16%
	2.11%
	19.48%
	2.58%
	17.93%
	6.65%
	35.22%
	18.89%
	13.80%
	27.04%
	23.19%
	7.48%
	10.25%
	6.23%
	25.95%
	30.58%
	3.85%
	12.27%
	1.96%
	61.28%
	18.79%
	18.12%
	17.82%
	26.98%
	39.78%
	2.77%
	11.12%

Finanssisektori: sisäkkäisyydet

	JavaScript-tiedostojen sisäkkäisyystasot						HTML-tiedostojen sisäkkäisyystasot			
Sivusto	1	2	3	4	5	6	1	2	3	4
1	40.26%	52.27%	6.82%	0.65%	0.00%	0.00%	76.08%	23.92%	0.00%	0.00%
2	32.85%	57.77%	7.55%	1.83%	0.00%	0.00%	100.00%	0.00%	0.00%	0.00%
3	88.89%	11.11%	0.00%	0.00%	0.00%	0.00%	100.00%	0.00%	0.00%	0.00%
4	31.65%	58.44%	8.99%	0.82%	0.10%	0.00%	99.73%	0.27%	0.00%	0.00%
5	53.98%	40.63%	3.13%	1.85%	0.43%	0.00%	76.47%	23.53%	0.00%	0.00%
6	59.31%	35.11%	5.32%	0.27%	0.00%	0.00%	100.00%	0.00%	0.00%	0.00%
7	53.54%	32.96%	11.73%	1.55%	0.22%	0.00%	100.00%	0.00%	0.00%	0.00%
8	40.26%	52.27%	6.82%	0.65%	0.00%	0.00%	53.77%	46.23%	0.00%	0.00%
9	17.20%	61.56%	19.09%	1.88%	0.27%	0.00%	90.84%	8.42%	0.68%	0.06%
10	30.39%	60.78%	7.35%	1.47%	0.00%	0.00%	20.00%	60.00%	20.00%	0.00%
11	57.49%	37.72%	3.59%	1.20%	0.00%	0.00%				
12	53.36%	41.28%	5.37%	0.00%	0.00%	0.00%	100.00%	0.00%	0.00%	0.00%
13	57.21%	38.51%	3.80%	0.48%	0.00%	0.00%	44.94%	47.88%	7.18%	0.00%
14	54.40%	40.11%	5.49%	0.00%	0.00%	0.00%	100.00%	0.00%	0.00%	0.00%
15	50.36%	27.49%	18.33%	3.11%	0.53%	0.18%	34.47%	65.53%	0.00%	0.00%
16	45.38%	48.95%	5.25%	0.42%	0.00%	0.00%	76.39%	23.61%	0.00%	0.00%
17	5.41%	62.16%	32.43%	0.00%	0.00%	0.00%				
18	44.62%	31.35%	22.28%	1.75%	0.00%	0.00%	50.00%	50.00%	0.00%	0.00%
19							99.86%	0.07%	0.07%	0.00%
20	43.96%	33.64%	20.90%	1.47%	0.02%	0.00%	91.96%	8.04%	0.00%	0.00%
21	8.94%	53.38%	33.33%	3.62%	0.72%	0.00%	99.98%	0.01%	0.01%	0.00%
22	55.70%	39.17%	4.05%	0.91%	0.17%	0.00%	99.93%	0.07%	0.00%	0.00%
23	61.63%	31.47%	6.14%	0.77%	0.00%	0.00%	99.82%	0.18%	0.00%	0.00%
24	30.00%	41.54%	25.77%	2.31%	0.38%	0.00%	77.57%	21.45%	0.98%	0.00%
25	40.37%	46.70%	10.82%	2.11%	0.00%	0.00%	99.80%	0.20%	0.00%	0.00%
26	35.70%	58.23%	5.37%	0.52%	0.17%	0.00%	61.92%	38.08%	0.00%	0.00%
27	40.06%	38.86%	18.67%	2.11%	0.30%	0.00%	78.83%	21.17%	0.00%	0.00%
28	15.00%	45.33%	35.00%	3.33%	1.00%	0.33%	100.00%	0.00%	0.00%	0.00%
29	58.02%	37.04%	3.70%	1.23%	0.00%	0.00%	100.00%	0.00%	0.00%	0.00%
30	56.22%	35.86%	3.99%	3.93%	0.00%	0.00%	92.26%	7.74%	0.00%	0.00%
31	33.74%	57.39%	6.90%	1.72%	0.25%	0.00%	88.18%	7.66%	4.17%	0.00%
32	42.61%	39.35%	15.79%	2.01%	0.25%	0.00%	69.44%	30.56%	0.00%	0.00%
33							99.94%	0.06%	0.00%	0.00%

Pikavippiryhmä: muuttujat

1	2	3	4	5	6	7	8	9	10	11	12	13
Sivusto	Js-tiedostot (lkm)	Html-tiedostot (lkm)	Html-tiedostot (lkm)	Js-tiedostot: Js loc	Html-tiedostot: Js loc	Html-tiedostot: Html loc	Js-tied: Funktio-kutsut	Html-tied: Funktio-kutsut	Js-tied: Takaisin-kutsulliset funktiokutsut	Html-tied: Takaisin-kutsulliset funktiokutsut	Js-tied: Anonyymit takaisinkutsut	Html-tied: Anonyymit takaisinkutsut
1	4	10	10	4410	1133	2175	2401	624	379	52	183	52
2	42	55	55	25700	6723	27604	18600	3117	2602	317	1024	137
3	4	23	23	9595	1020	4092	7188	603	1395	71	517	71
4	33	50	50	43977	100808	15162	20666	19828	4782	1438	1241	1438
5	4	12	12	15123	0	3551	8227	0	1601	0	551	0
6	40	226	226	20208	38206	91238	13543	5137	4221	0	974	0
7	43	90	90	20038	8974	35288	13485	4033	2711	490	956	196
8	0	40	40	0	8056	11218	0	2648	0	681	0	205
9	1	49	49	58	7814	19099	27	3101	6	389	1	193
10	13	50	50	9582	70458	22475	5657	9476	1003	1223	404	1169
11	33	21	21	15056	2693	6468	7539	3072	1598	281	617	216
12	3	1351	1000	1772	212718	312766	703	119140	52	13043	40	12043
13	28	250	250	119597	50047	168107	66510	28223	15945	2739	2459	2041
14	2	18	18	7825	10260	3173	2981	2229	343	347	188	347
15	116	721	721	39019	46884	219318	20801	31315	4161	2389	1091	968
16	3	29	29	11655	28574	12278	9615	9487	1155	2372	786	1224
17	11	26	26	14707	1131	41581	11403	598	2203	13	648	13
18	2	59	59	5979	11410	22882	4166	5779	730	841	264	841
19	14	54	54	9544	7998	18588	7408	4566	1245	309	411	105
20	3	39	39	753	27640	12385	376	11980	41	3280	11	1668
21	13	90	90	13143	8764	28548	6737	3102	1243	280	314	28
22	8	11	11	5422	235	1488	2968	118	427	1	214	1
23	5	11	11	6456	694	1811	2753	11	728	0	141	0
24	25	82	82	11557	10548	41674	6448	7209	1239	1392	461	1308
25	1	9	9	18332	5958	6817	14426	2592	2943	729	834	360
26	17	29	29	3606	816	9744	1879	319	166	0	117	0
27	63	171	171	28374	12429	59751	20369	7964	3588	799	1212	382
28	11	301	301	5614	26897	347411	3296	7687	731	1047	226	952

1	14	15	16	17	18	19	20	21	22	23	24	25
Sivusto	Html-tied: Tapahtuma attribuutit	Js-tied: Kaikki taka kutsu-parametrit	Html-tied: Kaikki tak kutsu-parametrit	Js-tied: Kaikki anor funktiopara	Html-tied: Kaikki anonyy funktioparame	Js-tied: Takaisin-kutsuja-funktiomäär.	Html-tied: Takaisin-kutsuja-funktiomää	Js-tied: Funktio-määritykset	Html-tied: Funktio-määritykset	Kaikki tied: Promiset ".then(" lkm	Kaikki tied: Async.js	Kaikki tied: jQuery
1	22	391	52	183	52	17	0	743	150	0	0	20
2	0	2768	317	1028	137	106	46	3948	725	0	0	184
3	47	1443	71	519	71	95	0	1923	120	0	0	9
4	78	5410	1438	1241	1438	208	33	5434	1809	6	0	324
5	0	1700	0	551	0	79	0	2370	0	0	0	6
6	0	5077	0	989	0	176	0	4290	913	50	0	10864
7	0	2986	508	960	196	115	40	3663	827	0	0	1545
8	1	0	885	0	205	0	238	0	1332	0	0	1
9	141	6	389	1	193	3	49	8	1124	0	0	49
10	294	1066	1223	406	1169	57	225	1548	1661	0	0	535
11	0	1835	281	628	216	64	0	2024	400	0	0	249
12	0	53	13043	40	12043	9	17	178	27331	0	0	1002
13	220	18673	2739	2468	2041	364	0	17965	5296	27	18	250
14	169	354	347	188	347	39	0	816	497	0	0	38
15	163	4515	2389	1095	968	192	12	4881	5906	0	0	1598
16	0	1204	2652	801	1224	252	504	3127	3625	31	0	3
17	0	2441	13	648	13	22	0	1929	104	0	0	0
18	722	766	841	264	841	34	0	1143	1672	0	0	3
19	54	1326	309	411	105	39	0	1558	1041	0	0	206
20	5	49	3670	11	1668	4	702	131	4870	0	0	4
21	0	1395	280	314	28	46	0	1637	881	0	0	23
22	1	446	1	214	1	19	0	841	22	0	0	11
23	0	892	0	142	0	19	0	672	0	0	0	161
24	484	1363	1392	463	1308	68	296	1760	1724	0	0	901
25	0	3237	792	839	360	83	207	2993	1080	0	9	0
26	0	174	0	117	0	16	0	430	58	0	0	58
27	54	3905	799	1218	382	134	7	4396	1415	0	0	730
28	281	787	1062	226	952	27	52	959	1858	0	1	1503

Pikavippiryhmä: johdetut arvot ja sisäkkäisyydet

	A1	A2	A3	B1	B2	B3	C2	C3	D2	D3	E2	E3	F1	F2	F3
Sivusto	JS: tkf / fk	HTML: tkf / fk	JS ja HTML: tkf / fk (est)	JS: anon / tkf	HTML: anon / tkf	JS ja HTML: anon / tkf (est)	HTML: tap.attr / fk	JS ja HTML: tap.attr / fk (est)	HTML: tap.attr / tkf	Js ja HTML: tap.attr / tkf	HTML: tap.attr / anon	JS ja HTML: tap.attr / anon (est)	JS: kaikki anon / kaikki tkf	HTML: kaikki anon / kaikki tkf	JS ja HTML: kaikki anon / kaikki tkf (est)
1	22.18%	13.62%	16.19%	30.92%	90.93%	66.25%	3.66%	2.56%	26.84%	15.80%	29.52%	23.85%	28.72%	89.64%	63.71%
2	8.83%	0.00%	7.55%	70.48%		70.48%	0.00%	0.00%		0.00%		0.00%	67.24%		67.24%
3	21.20%	9.15%	17.71%	38.61%	76.87%	44.33%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	34.22%	76.87%	39.89%
4	7.40%	10.95%	10.93%	76.92%	92.33%	92.29%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	75.47%	92.33%	92.28%
5		25.72%	25.72%		30.10%	30.10%	0.04%	0.04%	0.15%	0.15%	0.49%	0.49%		23.16%	23.16%
6	15.79%	8.33%	14.25%	48.28%	100.00%	54.52%	3.53%	0.73%	42.31%	5.10%	42.31%	9.36%	46.80%	100.00%	53.05%
7	16.81%	6.77%	12.98%	33.01%	33.98%	33.20%	1.18%	0.45%	17.48%	3.47%	51.43%	10.47%	31.00%	33.98%	31.56%
8	22.22%	12.54%	12.63%	16.67%	49.61%	49.11%	4.55%	4.51%	36.25%	35.70%	73.06%	72.68%	16.67%	49.61%	49.11%
9	19.22%	19.31%	19.26%	37.21%	93.97%	67.24%	6.71%	3.54%	34.77%	18.40%	37.00%	27.36%	33.97%	93.97%	64.28%
10	23.97%	9.70%	19.72%	15.42%	74.52%	24.08%	0.78%	0.23%	8.03%	1.18%	10.78%	4.89%	13.22%	74.52%	21.06%
11	31.17%	0.00%	22.60%	23.08%		23.08%	0.00%	0.00%		0.00%		0.00%	19.48%		19.48%
12	18.45%	9.03%	15.48%	25.26%	10.00%	22.46%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	22.51%	10.00%	20.42%
13	10.90%	27.38%	26.88%	26.83%	50.85%	50.56%	0.04%	0.04%	0.15%	0.15%	0.30%	0.30%	22.45%	45.45%	45.15%
14	17.52%	14.55%	15.80%	36.16%	100.00%	70.34%	12.49%	7.26%	85.85%	45.96%	85.85%	65.34%	34.46%	100.00%	68.76%
15	20.00%	7.63%	12.57%	26.22%	40.52%	31.44%	0.52%	0.31%	6.82%	2.49%	16.84%	7.92%	24.25%	40.52%	29.88%
16	17.62%	10.03%	15.48%	33.78%	47.81%	36.33%	0.68%	0.19%	6.76%	1.23%	14.14%	3.39%	31.19%	47.81%	34.01%
17	26.44%	0.00%	26.34%	19.37%		19.37%	0.00%	0.00%		0.00%		0.00%	15.92%		15.92%
18	19.32%	2.17%	18.47%	29.41%	100.00%	29.83%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	26.55%	100.00%	26.94%
19	23.14%	7.25%	15.36%	25.95%	100.00%	43.07%	0.39%	0.19%	5.42%	1.25%	5.42%	2.91%	22.94%	100.00%	39.12%
20	13.99%	10.17%	13.44%	39.35%	43.22%	39.77%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	37.14%	43.22%	37.76%
21	19.41%	11.77%	18.82%	37.06%	100.00%	40.11%	7.79%	0.60%	66.20%	3.21%	66.20%	7.99%	35.97%	100.00%	38.97%
22	11.51%	15.57%	13.24%	54.81%	100.00%	77.54%	7.58%	3.24%	48.70%	24.49%	48.70%	31.59%	53.11%	100.00%	76.32%
23	17.73%	12.91%	14.71%	40.28%	95.58%	70.66%	3.10%	1.94%	24.04%	13.21%	25.15%	18.69%	38.09%	95.58%	68.81%
24	20.40%	28.13%	21.58%	28.34%	49.38%	32.52%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	25.92%	45.45%	29.76%
25	19.46%		19.46%	34.42%		34.42%		0.00%		0.00%		0.00%	32.41%		32.41%
26	20.10%	12.15%	18.27%	35.26%	40.00%	35.99%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	32.15%	38.58%	33.09%
27	14.39%	0.85%	13.87%	50.12%	100.00%	50.23%	0.85%	0.03%	100.00%	0.23%	100.00%	0.47%	47.98%	100.00%	48.10%
28	12.01%	25.00%	18.46%	68.05%	51.60%	56.99%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	66.53%	46.15%	52.52%

	G1	G2	G3	H1	H2	H3
Sivusto	JS: tkfm / fm	HTML: tkfm / fm	JS ja HTML: tkfm / fm (est)	JS: kaikki tkf / tkf	HTML: kaikki tkf / tkf	JS ja HTML: kaikki tkf / tkf
1	2.82%	2.80%	2.80%	1.08	1.01	1.04
2	3.72%	0.00%	3.28%	1.05		1.05
3	3.16%	0.00%	2.64%	1.15	1.00	1.13
4	5.06%	0.06%	0.09%	1.02	1.00	1.00
5		17.87%	17.87%		1.30	1.30
6	2.29%	0.00%	1.90%	1.03	1.00	1.03
7	2.50%	0.00%	1.50%	1.07	1.00	1.05
8	37.50%	4.36%	4.59%	1.00	1.00	1.00
9	3.86%	17.17%	10.45%	1.10	1.00	1.05
10	2.03%	0.00%	1.56%	1.17	1.00	1.15
11	4.10%	0.00%	3.38%	1.20		1.20
12	2.81%	0.00%	1.83%	1.12	1.00	1.10
13	3.05%	14.41%	14.12%	1.20	1.12	1.12
14	2.97%	0.00%	1.21%	1.05	1.00	1.02
15	3.93%	0.20%	1.89%	1.09	1.00	1.05
16	3.05%	0.49%	2.43%	1.09	1.00	1.07
17	2.83%		2.83%	1.23		1.23
18	1.14%	0.00%	1.08%	1.11	1.00	1.11
19	3.83%	1.82%	3.33%	1.13	1.00	1.10
20	2.68%	6.34%	3.25%	1.06	1.00	1.06
21	4.94%	0.00%	4.65%	1.03	1.00	1.03
22	4.78%	0.00%	2.97%	1.03	1.00	1.02
23	3.68%	13.55%	8.79%	1.06	1.00	1.03
24	2.77%	19.17%	7.12%	1.10	1.09	1.10
25	3.33%		3.33%	1.06		1.06
26	3.14%	4.84%	3.45%	1.10	1.04	1.09
27	2.26%	0.00%	2.20%	1.04	1.00	1.04
28	8.06%	13.90%	11.20%	1.04	1.12	1.09

	I1	I2	J1	J2	K1	K2
	JS: Maksimi-sisäkkäis.	HTML: Maksimi-sisäkkäis.	JS: Sisäkkäis. keskiarvo	HTML: Sisäkkäis. keskiarvo	JS: Sisäkkäis. mediaani	HTML: Sisäkkäis. mediaani
	5	3	2.29	1.14	2	1
	4		1.58		1	
	5	3	1.90	2.06	2	2
	4	2	2.60	1.26	2	1
		2		1.83		2
	4	2	1.53	1.06	1	1
	5		2.03	0.00	2	0
	1	1	1.00	1.00	1	1
	5	3	1.69	1.34	1	1
	5	1	1.96	1.00	2	1
	4		1.51		1	
	5	2	1.87	1.54	2	2
	1	3	1.00	2.02	1	2
	5	1	1.97	1.00	2	1
	5	3	2.20	1.32	2	1
	5	3	1.78	1.23	2	1
	3		1.69		2	
	5	1	2.08	1.00	2	1
	5	2	1.78	1.75	2	2
	4	2	1.50	1.50	1	1
	5		1.95		2	
	4	1	1.52	1.00	1	1
	5	3	1.77	1.60	2	2
		3		2.03		2
	5		2.22		2	
	5	2	1.89	1.14	2	1
	4	1	1.54	1.00	1	1
	6	3	2.67	1.99	2	2

	L2
	HTML: js loc / loc
	7.19%
	7.73%
	29.40%
	40.48%
	41.80%
	34.25%
	30.08%
	29.03%
	20.20%
	22.94%
	29.52%
	23.49%
	69.06%
	33.27%
	17.61%
	17.22%
	27.70%
	2.65%
	86.93%
	19.59%
	19.95%
	76.38%
	75.82%
	46.64%
	0.00%
	20.27%
	13.64%
	69.95%

	JavaScript-tiedostojen sisäkkäisyystasot					
Sivusto	1	2	3	4	5	6
1	11.46%	52.23%	32.48%	3.18%	0.64%	0.00%
2	52.53%	39.39%	6.06%	2.02%	0.00%	0.00%
3	36.19%	43.14%	16.40%	3.21%	1.07%	0.00%
4	10.64%	29.79%	48.94%	10.64%	0.00%	0.00%
5						
6	52.38%	42.86%	3.97%	0.79%	0.00%	0.00%
7	26.07%	47.86%	23.74%	1.95%	0.39%	0.00%
8	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%
9	52.42%	29.26%	16.03%	1.78%	0.51%	0.00%
10	35.89%	36.84%	23.44%	3.35%	0.48%	0.00%
11	53.76%	42.29%	3.58%	0.36%	0.00%	0.00%
12	40.61%	35.49%	20.82%	2.73%	0.34%	0.00%
13	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%
14	30.56%	44.91%	21.76%	2.31%	0.46%	0.00%
15	22.26%	40.12%	33.43%	3.69%	0.50%	0.00%
16	42.05%	43.89%	9.46%	3.42%	1.18%	0.00%
17	34.72%	61.11%	4.17%	0.00%	0.00%	0.00%
18	25.78%	44.89%	25.78%	3.11%	0.44%	0.00%
19	32.67%	58.23%	7.83%	1.18%	0.09%	0.00%
20	59.24%	32.48%	7.64%	0.64%	0.00%	0.00%
21	32.58%	43.44%	21.27%	2.26%	0.45%	0.00%
22	53.01%	42.62%	3.83%	0.55%	0.00%	0.00%
23	46.90%	32.15%	18.29%	2.06%	0.59%	0.00%
24						
25	18.53%	46.55%	29.74%	4.53%	0.65%	0.00%
26	39.14%	37.63%	19.07%	3.03%	1.14%	0.00%
27	52.79%	41.62%	4.57%	1.02%	0.00%	0.00%
28	0.27%	50.73%	33.38%	13.62%	1.74%	0.27%

	HTML-tiedostojen sisäkkäisyystasot			
	1	2	3	4
	87.18%	11.59%	1.24%	0.00%
	33.50%	26.60%	39.90%	0.00%
	73.72%	26.28%	0.00%	0.00%
	17.07%	82.93%	0.00%	0.00%
	93.75%	6.25%	0.00%	0.00%
	100.00%	0.00%	0.00%	0.00%
	100.00%	0.00%	0.00%	0.00%
	66.46%	33.38%	0.15%	0.00%
	100.00%	0.00%	0.00%	0.00%
	46.43%	53.57%	0.00%	0.00%
	3.01%	91.88%	5.10%	0.00%
	100.00%	0.00%	0.00%	0.00%
	75.41%	16.80%	7.79%	0.00%
	79.22%	18.18%	2.60%	0.00%
	100.00%	0.00%	0.00%	0.00%
	24.62%	75.38%	0.00%	0.00%
	50.00%	50.00%	0.00%	0.00%
	100.00%	0.00%	0.00%	0.00%
	100.00%	0.00%	0.00%	0.00%
	48.67%	42.94%	8.38%	0.00%
	2.70%	91.89%	5.41%	0.00%
	85.98%	14.02%	0.00%	0.00%
	100.00%	0.00%	0.00%	0.00%
	5.67%	89.36%	4.96%	0.00%

Rahapeliryhmä: muuttujat

1	2	3	4	5	6	7	8	9	10	11	12	13
Sivusto	Js-tiedostot (lkm)	Html-tiedostot (lkm)	Html-tiedostot (lkm)	Js-tiedostot: Js loc	Html-tiedostot: Js loc	Html-tiedostot: Html loc	Js-tied: Funktio-kutsut	Html-tied: Funktio-kutsut	Js-tied: Takaisin-kutsulliset funktiokutsut	Html-tied: Takaisin-kutsulliset funktiokutsut	Js-tied: Anonyymit takaisinkutsut	Html-tied: Anonyymit takaisinkutsut
1	0	658	658	0	467304	219304	0	239376	0	33716	0	22285
2	1	2083	1000	1585	1750562	3403262	0	140966	0	23319	0	22782
3	2	117	117	4032	6495	101205	2533	1783	478	254	193	20
4	8	493	493	20943	213900	389045	9957	23655	2541	2416	694	1434
5	1	683	683	1190	110712	174054	1148	62988	94	9576	59	9576
6	5	2	2	20261	0	348	14147	0	3123	0	1147	0
7	2	161	161	11043	211663	874935	5915	17315	597	3918	344	3651
8	66	4227	1000	2443	272879	432329	1293	88043	162	20176	160	19852
9	17	93	93	3169	14555	84235	1661	7609	210	821	208	821
10	24	5138	1000	10063	280698	803126	3623	168958	188	52541	147	52379
11	0	621	621	0	551074	339050	0	374617	0	51281	0	42191
12	5	10	10	32133	4263	428	18628	90	5152	0	1544	0
13	1	42	42	95	16297	17670	27	7195	6	1592	6	923
14	0	88	88	0	490273	619399	0	35465	0	10862	0	7254
15	0	2897	1000	0	181492	863841	0	26322	0	3033	0	3033
16	0	546	546	0	29382	214030	0	13028	0	1073	0	1073
17	14	101367	1000	11877	55029	280368	4295	37009	1198	7004	271	7004
18	2	172	172	512	143640	157365	166	83206	15	12567	14	12515
19	0	326	326	0	28314	170687	0	11361	0	751	0	101
20	0	136	136	0	56639	58815	0	21430	0	4968	0	2616
21	9	1245	1000	2669	357229	2452314	1659	138689	121	12416	87	9367
22	1	96	96	686	24827	42450	205	8177	11	2143	3	787
23	3	36	36	20166	33840	1656	10692	1656	1955	108	1101	36
24	7	436	436	11383	46136	104040	7076	13868	951	425	423	425
25	4	137	137	6581	33966	29960	4072	10762	427	2589	284	1219
26	4	211	211	5850	53556	60617	3531	21267	358	6602	211	3802
27	20	1423	1000	108601	20986	1422709	70712	7987	14633	5	3824	5
28	4	71	71	4429	14049	22065	2089	4560	294	732	121	620
29	3	511	511	12955	7154	327600	5206	3066	1377	0	395	0
30	39	68	68	167723	4084	22619	72883	1636	17314	10	8324	2
31	0	790	790	0	136576	369560	0	37218	0	3512	0	2700
32	28	168	168	26096	5562	98716	12206	3280	3555	770	1218	770

1	14	15	16	17	18	19	20	21	22	23	24	25
Sivusto	Html-tied: Tapahtuma attribuutit	Js-tied: Kaikki takaisin kutsu-parametrit	Html-tied: Kaikki takaisin kutsu-parametrit	Js-tied: Kaikki anonyy funktioparame	Html-tied: Kaikki anonyy funktioparame	Js-tied: Takaisin-kutsuja-funktiomäär.	Html-tied: Takaisin-kutsuja-funktiomäär.	Js-tied: Funktio-määritykset	Html-tied: Funktio-määritykset	Kaikki tied: Promiset ".then(" lkm	Kaikki tied: Async.js	Kaikki tied: jQuery
1	1920	0	37588	0	22285	0	4605	0	55412	0	0	674
2	3553	0	23324	0	22787	0	7	0	38494	1626	0	4
3	20	504	254	193	20	23	117	765	605	0	0	117
4	79	2917	2416	694	1434	114	0	3483	4878	0	0	0
5	8212	94	9576	59	9576	6	0	235	15467	0	0	684
6	0	3623	0	1191	0	506	0	5623	0	198	0	0
7	608	625	3918	353	3651	36	165	743	5142	0	0	322
8	10282	162	20176	160	19852	22	126	432	28240	0	0	120
9	0	210	838	208	838	16	76	441	1251	0	0	92
10	46105	193	52541	151	52379	22	58	770	59585	0	73	2426
11	3946	0	54839	0	42191	0	4205	0	77098	0	0	38
12	0	6123	0	1574	0	220	0	7850	10	217	0	0
13	298	6	1844	6	923	0	294	14	2473	0	0	254
14	2643	0	11742	0	7254	0	1584	0	14494	0	0	88
15	0	0	3033	0	3033	0	0	0	8254	0	0	0
16	0	0	1073	0	1073	0	0	0	2161	0	0	2176
17	1000	1406	7004	271	7004	27	0	1373	8004	0	0	5002
18	4804	16	13212	15	13160	2	342	32	17419	0	0	1
19	6	0	751	0	101	0	0	0	2382	0	0	5
20	1790	0	5824	0	2616	0	992	0	8208	0	0	136
21	81	121	12417	87	9368	8	1042	284	41256	0	0	4031
22	147	11	2731	3	787	6	672	47	4099	0	0	864
23	36	2114	108	1115	36	222	0	3720	324	1	1	0
24	3	1017	425	423	425	77	0	2183	1810	0	0	1689
25	397	435	3411	288	1219	23	1644	890	6151	0	0	554
26	2591	366	7802	215	3802	20	1400	776	10002	0	0	817
27	5	16389	5	3859	5	945	0	24115	2996	17	0	997
28	420	307	732	121	620	8	75	588	1319	3	0	280
29	0	1604	0	395	0	44	0	1521	511	0	0	1024
30	0	19419	10	8337	2	1196	0	22441	434	2	0	0
31	0	0	3652	0	2840	0	0	0	10163	0	0	0
32	558	4249	770	1247	770	355	0	4383	1122	207	0	176

Rahapeliryhmä: johdetut arvot

	A1	A2	A3	B1	B2	B3	C2	C3	D2	D3	E2	E3	F1	F2	F3
Sivusto	JS: tkf / fk	HTML: tkf / fk	JS ja HTML: tkf / fk (est)	JS: anon / tkf	HTML: anon / tkf	JS ja HTML: anon / tkf (est)	HTML: tap.attr / fk	JS ja HTML: tap.attr / fk (est)	HTML: tap.attr / tkf	Js ja HTML: tap.attr / tkf	HTML: tap.attr / anon	JS ja HTML: tap.attr / anon (est)	JS: kaikki anon / kaikki tkf	HTML: kaikki anon / kaikki tkf	JS ja HTML: kaikki anon / kaikki tkf (est)
1		14.08%	14.08%		66.10%	66.10%	0.80%	0.80%	5.69%	5.69%	8.62%	8.62%		59.29%	59.29%
2		16.54%	16.54%		97.70%	97.70%	2.52%	2.52%	15.24%	15.24%	15.60%	15.60%		97.70%	97.70%
3	18.87%	14.25%	16.96%	40.38%	7.87%	29.10%	1.12%	0.46%	7.87%	2.73%	100.00%	9.39%	38.29%	7.87%	28.10%
4	25.52%	10.21%	14.75%	27.31%	59.35%	42.93%	0.33%	0.24%	3.27%	1.59%	5.51%	3.71%	23.79%	59.35%	39.90%
5	8.19%	15.20%	15.08%	62.77%	100.00%	99.64%	13.04%	12.80%	85.76%	84.92%	85.76%	85.23%	62.77%	100.00%	99.64%
6	22.08%		22.08%	36.73%		36.73%		0.00%		0.00%		0.00%	32.87%		32.87%
7	10.09%	22.63%	19.44%	57.62%	93.19%	88.48%	3.51%	2.62%	15.52%	13.47%	16.65%	15.22%	56.48%	93.19%	88.14%
8	12.53%	22.92%	22.88%	98.77%	98.39%	98.39%	11.68%	11.64%	50.96%	50.86%	51.79%	51.69%	98.77%	98.39%	98.39%
9	12.64%	10.79%	11.12%	99.05%	100.00%	99.81%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	99.05%	100.00%	99.81%
10	5.19%	31.10%	30.99%	78.19%	99.69%	99.68%	27.29%	27.17%	87.75%	87.69%	88.02%	87.97%	78.24%	99.69%	99.68%
11		13.69%	13.69%		82.27%	82.27%	1.05%	1.05%	7.69%	7.69%	9.35%	9.35%		76.94%	76.94%
12	27.66%	0.00%	27.52%	29.97%		29.97%	0.00%	0.00%		0.00%		0.00%	25.71%		25.71%
13	22.22%	22.13%	22.13%	100.00%	57.98%	58.14%	4.14%	4.13%	18.72%	18.65%	32.29%	32.08%	100.00%	50.05%	50.22%
14		30.63%	30.63%		66.78%	66.78%	7.45%	7.45%	24.33%	24.33%	36.44%	36.44%		61.78%	61.78%
15		11.52%	11.52%		100.00%	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%		100.00%	100.00%
16		8.24%	8.24%		100.00%	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%		100.00%	100.00%
17	27.89%	18.93%	18.94%	22.62%	100.00%	99.87%	2.70%	2.70%	14.28%	14.25%	14.28%	14.27%	19.27%	100.00%	99.84%
18	9.04%	15.10%	15.09%	93.33%	99.59%	99.58%	5.77%	5.76%	38.23%	38.18%	38.39%	38.34%	93.75%	99.61%	99.60%
19		6.61%	6.61%		13.45%	13.45%	0.05%	0.05%	0.80%	0.80%	5.94%	5.94%		13.45%	13.45%
20		23.18%	23.18%		52.66%	52.66%	8.35%	8.35%	36.03%	36.03%	68.43%	68.43%		44.92%	44.92%
21	7.29%	8.95%	8.94%	71.90%	75.44%	75.42%	0.06%	0.06%	0.65%	0.65%	0.86%	0.86%	71.90%	75.44%	75.42%
22	5.37%	26.21%	25.70%	27.27%	36.72%	36.68%	1.80%	1.75%	6.86%	6.82%	18.68%	18.61%	27.27%	28.82%	28.81%
23	18.28%	6.52%	16.71%	56.32%	33.33%	55.11%	2.17%	0.29%	33.33%	1.75%	100.00%	3.17%	52.74%	33.33%	51.80%
24	13.44%	3.06%	6.57%	44.48%	100.00%	61.63%	0.02%	0.01%	0.71%	0.22%	0.71%	0.35%	41.59%	100.00%	58.81%
25	10.49%	24.06%	20.33%	66.51%	47.08%	49.83%	3.69%	2.68%	15.33%	13.16%	32.57%	26.41%	66.21%	35.74%	39.18%
26	10.14%	31.04%	28.07%	58.94%	57.59%	57.66%	12.18%	10.45%	39.25%	37.23%	68.15%	64.57%	58.74%	48.73%	49.18%
27	20.69%	0.06%	17.84%	26.13%	100.00%	26.17%	0.06%	0.01%	100.00%	0.05%	100.00%	0.19%	23.55%	100.00%	23.58%
28	14.07%	16.05%	15.43%	41.16%	84.70%	72.22%	9.21%	6.32%	57.38%	40.94%	67.74%	56.68%	39.41%	84.70%	71.32%
29	26.45%	0.00%	16.65%	28.69%		28.69%	0.00%	0.00%		0.00%		0.00%	24.63%		24.63%
30	23.76%	0.61%	23.25%	48.08%	20.00%	48.06%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	42.93%	20.00%	42.92%
31		9.44%	9.44%		76.88%	76.88%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%		77.77%	77.77%
32	29.13%	23.48%	27.93%	34.26%	100.00%	45.97%	17.01%	3.60%	72.47%	12.90%	72.47%	28.07%	29.35%	100.00%	40.19%

	G1	G2	G3	H1	H2	H3
Sivusto	JS: tkfm / fm	HTML: tkfm / fm	JS ja HTML: tkfm / fm (est)	JS: kaikki tkf / tkf	HTML: kaikki tkf / tkf	JS ja HTML: kaikki tkf / tkf
1		8.31%	8.31%		1.11	1.11
2		0.02%	0.02%		1.00	1.00
3	3.01%	19.34%	10.22%	1.05	1.00	1.04
4	3.27%	0.00%	1.36%	1.15	1.00	1.08
5	2.55%	0.00%	0.04%	1.00	1.00	1.00
6	9.00%		9.00%	1.16		1.16
7	4.85%	3.21%	3.42%	1.05	1.00	1.01
8	5.09%	0.45%	0.46%	1.00	1.00	1.00
9	3.63%	6.08%	5.44%	1.00	1.02	1.02
10	2.86%	0.10%	0.10%	1.03	1.00	1.00
11		5.45%	5.45%		1.07	1.07
12	2.80%	0.00%	2.80%	1.19		1.19
13	0.00%	11.89%	11.82%	1.00	1.16	1.16
14		10.93%	10.93%		1.08	1.08
15		0.00%	0.00%		1.00	1.00
16		0.00%	0.00%		1.00	1.00
17	1.97%	0.00%	0.00%	1.17	1.00	1.00
18	6.25%	1.96%	1.97%	1.07	1.05	1.05
19		0.00%	0.00%		1.00	1.00
20		12.09%	12.09%		1.17	1.17
21	2.82%	2.53%	2.53%	1.00	1.00	1.00
22	12.77%	16.39%	16.35%	1.00	1.27	1.27
23	5.97%	0.00%	5.49%	1.08	1.00	1.08
24	3.53%	0.00%	1.93%	1.07	1.00	1.05
25	2.58%	26.73%	23.68%	1.02	1.32	1.28
26	2.58%	14.00%	13.17%	1.02	1.18	1.17
27	3.92%	0.00%	3.33%	1.12	1.00	1.12
28	1.36%	5.69%	4.35%	1.04	1.00	1.01
29	2.89%	0.00%	2.17%	1.16		1.16
30	5.33%	0.00%	5.23%	1.12	1.00	1.12
31		0.00%	0.00%		1.04	1.04
32	8.10%	0.00%	6.45%	1.20	1.00	1.16

	I1	I2	J1	J2	K1	K2
	JS: Maksimi-sisäkkäis.	HTML: Maksimi-sisäkkäis.	JS: Sisäkkäis. keskiarvo	HTML: Sisäkkäis. keskiarvo	JS: Sisäkkäis. mediaani	HTML: Sisäkkäis. mediaani
		3		1.48		1
		3		1.21		1
	5	1	2.11	1.00	2	1
	5	2	2.02	1.01	2	1
	2	1	1.16	1.00	1	1
	6		2.66		2	
	4	2	1.32	1.13	1	1
	3	2	1.13	1.20	1	1
	3	3	2.17	1.49	2	1
	4	4	1.77	1.02	2	1
		2		1.34		1
	4		1.21		1	
	1	4	1.00	1.52	1	2
		3		1.56		2
		2		1.34		1
		1		1.00		1
	5	2	2.00	1.71	2	2
	2	3	1.27	1.38	1	1
		2		1.74		2
		2		1.26		1
	4	3	1.63	1.06	1	1
	1	2	1.00	1.62	1	2
	6	1	1.93	1.00	2	1
	5	1	1.85	1.00	2	1
	4	2	1.29	1.56	1	2
	4	2	1.28	1.26	1	1
	4	1	2.08	1.00	2	1
	5	2	1.61	1.00	2	1
	5		1.94		2	
	6	1	2.15	1.00	2	1
		2		1.14		1
	5	1	1.90	1.00	2	1

	L2
	HTML: js loc / loc
	68.06%
	33.97%
	6.03%
	35.48%
	38.88%
	0.00%
	19.48%
	38.69%
	14.73%
	25.90%
	61.91%
	90.88%
	47.98%
	44.18%
	17.36%
	12.07%
	16.41%
	47.72%
	14.23%
	49.06%
	12.71%
	36.90%
	95.33%
	30.72%
	53.13%
	46.91%
	1.45%
	38.90%
	2.14%
	15.29%
	26.98%
	5.33%

Rahapeliryhmä: sisäkkäisyydet

	JavaScript-tiedostojen sisäkkäisyystasot							HTML-tiedostojen sisäkkäisyystasot			
Sivusto	1	2	3	4	5	6		1	2	3	4
1								52.36%	46.85%	0.78%	0.00%
2								83.33%	12.65%	4.01%	0.00%
3	23.95%	44.91%	27.54%	2.99%	0.60%	0.00%		100.00%	0.00%	0.00%	0.00%
4	29.86%	41.89%	24.72%	3.05%	0.48%	0.00%		99.30%	0.70%	0.00%	0.00%
5	83.64%	16.36%	0.00%	0.00%	0.00%	0.00%		100.00%	0.00%	0.00%	0.00%
6	0.41%	42.54%	48.15%	8.21%	0.55%	0.14%					
7	75.78%	18.69%	3.11%	2.42%	0.00%	0.00%		86.71%	13.29%	0.00%	0.00%
8	91.62%	4.19%	4.19%	0.00%	0.00%	0.00%		79.52%	20.48%	0.00%	0.00%
9	0.00%	83.33%	16.67%	0.00%	0.00%	0.00%		60.12%	31.17%	8.71%	0.00%
10	40.83%	43.33%	14.17%	1.67%	0.00%	0.00%		97.83%	2.05%	0.06%	0.06%
11								66.42%	33.58%	0.00%	0.00%
12	81.53%	15.76%	2.46%	0.25%	0.00%	0.00%					
13	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%		48.81%	50.87%	0.22%	0.11%
14								47.54%	48.52%	3.93%	0.00%
15								66.30%	33.70%	0.00%	0.00%
16								100.00%	0.00%	0.00%	0.00%
17	27.24%	49.03%	20.23%	3.11%	0.39%	0.00%		28.60%	71.40%	0.00%	0.00%
18	73.33%	26.67%	0.00%	0.00%	0.00%	0.00%		64.20%	33.34%	2.45%	0.00%
19								25.74%	74.26%	0.00%	0.00%
20								73.85%	26.15%	0.00%	0.00%
21	50.00%	38.16%	10.53%	1.32%	0.00%	0.00%		94.55%	5.28%	0.17%	0.00%
22	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%		38.37%	61.63%	0.00%	0.00%
23	39.82%	33.94%	21.28%	4.05%	0.61%	0.30%		100.00%	0.00%	0.00%	0.00%
24	37.84%	40.79%	19.90%	1.23%	0.25%	0.00%		100.00%	0.00%	0.00%	0.00%
25	75.76%	20.78%	2.60%	0.87%	0.00%	0.00%		43.81%	56.19%	0.00%	0.00%
26	77.36%	18.87%	2.52%	1.26%	0.00%	0.00%		73.62%	26.38%	0.00%	0.00%
27	8.89%	75.78%	13.57%	1.76%	0.00%	0.00%		100.00%	0.00%	0.00%	0.00%
28	46.67%	48.57%	2.86%	0.95%	0.95%	0.00%		99.60%	0.40%	0.00%	0.00%
29	34.88%	39.53%	22.79%	2.33%	0.47%	0.00%					
30	3.08%	80.84%	14.40%	1.49%	0.16%	0.03%		100.00%	0.00%	0.00%	0.00%
31								86.19%	13.81%	0.00%	0.00%
32	37.84%	39.86%	17.21%	4.21%	0.88%	0.00%		100.00%	0.00%	0.00%	0.00%

Julkinen sektori: muuttujat, johdetut arvot ja sisäkkäisyydet

1	2	3	4	5	6	7	8	9	10	11	12	13
Sivusto	Js-tiedostot (lkm)	Html-tiedostot (lkm)	Html-tiedostot (lkm)	Js-tiedostot: Js loc	Html-tiedostot: Js loc	Html-tiedostot: Html loc	Js-tied: Funktio-kutsut	Html-tied: Funktio-kutsut	Js-tied: Takaisin-kutsulliset funktiokutsut	Html-tied: Takaisin-kutsulliset funktiokutsut	Js-tied: Anonyymit takaisinkutsut	Html-tied: Anonyymit takaisinkutsut
1	15	501	501	44055	45526	183998	18874	21367	1911	4046	806	3559
2	47	99	99	107779	50198	28178	37561	16644	4567	2110	2191	2002
3	12	47	47	109708	14269	11176	27522	1154	2608	92	1799	92
4	7	2	2	12092	42	278	7965	30	1496	12	424	12
5	7	1336	1000	13907	1719	118677	9645	1146	1622	573	504	573
6	9	1092	1000	35578	259455	589161	13127	103857	747	17012	227	10297
7	44	6109	1000	209357	301529	315495	63199	31563	6180	5103	3949	5103
8	11	275	275	27567	1937	54281	19476	1650	5046	275	1473	275

1	14	15	16	17	18	19	20	21	22	23	24	25
Sivusto	Html-tied: Tapahtuma attribuutit	Js-tied: Kaikki takaisin kutsu-parametrit	Html-tied: Kaikki takaisin kutsu-parametrit	Js-tied: Kaikki anonyy funktioparame	Html-tied: Kaikki anonyy funktioparame	Js-tied: Takaisin-kutsuja-funktiomäär.	Html-tied: Takaisin-kutsuja-funktiomä	Js-tied: Funktio-määritykset	Html-tied: Funktio-määritykset	Kaikki tied: Promiset ".then(" lkm	Kaikki tied: Async.js	Kaikki tied: jQuery
1	2567	2008	4046	813	3559	96	0	5386	5593	0	0	495
2	1	5001	2202	2193	2002	242	0	10922	9476	15	0	1015
3	2	2673	92	1799	92	88	0	9445	3287	3	0	94
4	8	1647	12	425	12	45	0	1962	12	0	0	4
5	573	1759	573	504	573	35	0	1957	573	0	0	2622
6	8411	781	17012	229	10297	34	939	3705	24622	6	0	1878
7	3016	6342	5103	3964	5103	294	0	19274	72537	7	0	2037
8	0	5901	275	1500	275	304	0	5523	275	63	0	0

	A1	A2	A3	B1	B2	B3	C2	C3	D2	D3	E2	E3	F1	F2	F3
Sivusto	JS: tkf / fk	HTML: tkf / fk	JS ja HTML: tkf / fk (est)	JS: anon / tkf	HTML: anon / tkf	JS ja HTML: anon / tkf (est)	HTML: tap.attr / fk	JS ja HTML tap.attr / fk (est)	HTML: tap.attr / tkf	Js ja HTML: tap.attr / tkf	HTML: tap.attr / anon	JS ja HTML: tap.attr / anon (est)	JS: kaikki anon / kaikki tkf	HTML: kaikki anon / kaikki tkf	JS ja HTML: kaikki anon / kaikki tkf (est)
1	10.13%	18.94%	14.80%	42.18%	87.96%	73.28%	12.01%	6.38%	63.45%	43.09%	72.13%	58.81%	40.49%	87.96%	72.22%
2	12.16%	12.68%	12.32%	47.97%	94.88%	62.80%	0.01%	0.00%	0.05%	0.01%	0.05%	0.02%	43.85%	90.92%	58.24%
3	9.48%	7.97%	9.42%	68.98%	100.00%	70.04%	0.17%	0.01%	2.17%	0.07%	2.17%	0.11%	67.30%	100.00%	68.39%
4	18.78%	40.00%	18.86%	28.34%	100.00%	28.91%	26.67%	0.10%	66.67%	0.53%	66.67%	1.83%	25.80%	100.00%	26.34%
5	16.82%	50.00%	21.36%	31.07%	100.00%	53.17%	50.00%	6.85%	100.00%	32.06%	100.00%	60.30%	28.65%	100.00%	50.29%
6	5.69%	16.38%	15.27%	30.39%	60.53%	59.36%	8.10%	7.26%	49.44%	47.53%	81.68%	80.07%	29.32%	60.53%	59.27%
7	9.78%	16.17%	14.59%	63.90%	100.00%	94.03%	9.56%	7.20%	59.10%	49.32%	59.10%	52.46%	62.50%	100.00%	93.66%
8	25.91%	16.67%	25.19%	29.19%	100.00%	32.85%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	25.42%	100.00%	28.74%

	G1	G2	G3	H1	H2	H3
Sivusto	JS: tkfm / fm	HTML: tkfm / fm	JS ja HTML: tkfm / fm (est)	JS: kaikki tkf / tkf	HTML: kaikki tkf / tkf	JS ja HTML: kaikki tkf / tkf
1	1.78%	0.00%	0.87%	1.05	1.00	1.02
2	2.22%	0.00%	1.19%	1.10	1.04	1.08
3	0.93%	0.00%	0.69%	1.02	1.00	1.02
4	2.29%	0.00%	2.28%	1.10	1.00	1.10
5	1.79%	0.00%	1.29%	1.08	1.00	1.06
6	0.92%	3.81%	3.46%	1.05	1.00	1.00
7	1.53%	0.00%	0.06%	1.03	1.00	1.00
8	5.50%	0.00%	5.24%	1.17	1.00	1.16

I1	I2	J1	J2	K1	K2
JS: Maksimi-sisäkkäis.	HTML: Maksimi-sisäkkäis.	JS: Sisäkkäis. keskiarvo	HTML: Sisäkkäis. keskiarvo	JS: Sisäkkäis. mediaani	HTML: Sisäkkäis. mediaani
4	1	1.53	1.00	1	1
5	3	1.78	1.66	2	2
5	2	1.65	1.49	2	1
5	1	2.12	1.00	2	1
5	1	2.02	1.00	2	1
5	1	1.60	1.00	2	1
4	2	1.64	1.20	2	1
5	1	1.63	1.00	1	1

L2
HTML: js loc / loc
19.83%
64.05%
56.08%
13.13%
1.43%
30.57%
48.87%
3.45%

	JavaScript-tiedostojen sisäkkäisyystasot					
Sivusto	1	2	3	4	5	6
1	52.05%	43.22%	4.10%	0.63%	0.00%	0.00%
2	38.29%	46.31%	14.22%	1.12%	0.05%	0.00%
3	42.11%	53.38%	3.01%	0.75%	0.75%	0.00%
4	25.43%	42.20%	27.75%	4.05%	0.58%	0.00%
5	28.04%	45.33%	23.83%	2.34%	0.47%	0.00%
6	46.73%	48.74%	3.02%	1.01%	0.50%	0.00%
7	41.72%	52.79%	5.31%	0.17%	0.00%	0.00%
8	51.32%	36.26%	10.76%	1.49%	0.17%	0.00%

	HTML-tiedostojen sisäkkäisyystasot			
	1	2	3	4
	100.00%	0.00%	0.00%	0.00%
	35.91%	62.64%	1.45%	0.00%
	51.09%	48.91%	0.00%	0.00%
	100.00%	0.00%	0.00%	0.00%
	100.00%	0.00%	0.00%	0.00%
	100.00%	0.00%	0.00%	0.00%
	79.67%	20.33%	0.00%	0.00%
	100.00%	0.00%	0.00%	0.00%

Verkkokaupparyhmä: muuttujat

1	2	3	4	5	6	7	8	9	10	11	12	13
Sivusto	Js-tiedostot (lkm)	Html-tiedostot (lkm)	Html-tiedostot (lkm)	Js-tiedostot: Js loc	Html-tiedostot: Js loc	Html-tiedostot: Html loc	Js-tied: Funktio-kutsut	Html-tied: Funktio-kutsut	Js-tied: Takaisin-kutsulliset funktiokutsut	Html-tied: Takaisin-kutsulliset funktiokutsut	Js-tied: Anonyymit takaisinkutsut	Html-tied: Anonyymit takaisinkutsut
1	25	31105	1000	34850	586528	2067882	16377	17425	2147	1725	924	1572
2	16	98	98	18673	21414	103732	10215	7918	983	1124	538	1116
3	1	8617	1000	10587	422290	2062009	6781	193443	1239	37490	519	34727
4	2	17251	1000	975	93626	1781737	721	31119	75	4803	68	4803
5	38	880	880	22505	53551	355827	13795	41009	2827	5828	750	3722
6	63	6954	1000	54010	1183072	1044206	33155	601467	4312	48552	1944	44664
7	19	409	409	7682	120378	177971	3688	33874	548	8976	277	2448
8	6	45897	1000	7229	2052552	4705445	176	592076	5	150816	5	140898
9	5	58719	1000	4484	1232059	2214981	1284	355421	122	247560	44	247560
10	20	3207	1000	21052	414197	1334764	11411	275050	1075	23180	750	21292
11	6	3685	1000	7790	297633	1346660	4801	79811	525	15952	358	5982
12	54	3016	1000	30828	817994	1217541	16663	356039	1323	95073	1002	53745
13	15	23720	1000	4053	34335	989092	2067	19566	153	2933	137	2933
14	12	57420	1000	147529	2054707	3879503	66215	1097696	13201	437407	3426	435374
15	6	18518	1000	8892	127022	1647904	3046	54296	223	20673	144	20673
16	57	17122	1000	25293	1060899	2024362	16500	507733	1231	161076	805	161056
17	1	257	257	10576	80857	312428	6781	41938	1238	7235	519	6489
18	7	1750	1000	5352	133178	1289688	3812	60239	681	24072	298	24072
19	7	104839	1000	41317	101081	336852	18003	61755	1031	8865	883	7865
20	7	1944	1000	25906	226763	1259787	6394	101899	641	6993	444	999
21	0	4994	1000	0	569753	1260829	0	41911	0	20887	0	18899
22	4	4619	1000	2055	42492	367851	676	15276	41	5147	39	4147
23	11	5	5	9238	806	1046	4182	365	582	23	244	22
24	19	25866	1000	6406	124543	376136	4382	39423	938	4920	354	1968
25	21	696	696	66461	534753	1557871	34571	203669	4398	18558	2277	17855
26	37	1232	1000	96633	203216	306605	56695	63505	11507	19540	3918	5862
27	6	2734	1000	22478	1186308	252240	8796	459632	1340	50131	588	42453
28	21	28612	1000	21382	506625	940654	11195	269216	833	74261	546	74140
29	5	15995	1000	9984	727109	956129	7031	616209	1272	44372	434	29372
30	15	44691	1000	7542	2617765	5134663	3652	1131521	486	378503	262	368458
31	14	3784	1000	12815	64519	2577197	8364	34598	1227	2377	451	1079
32	22	254	254	6039	9799	43634	4244	5945	918	580	346	145
33	36	1009	1000	66564	862054	2600433	37153	296879	4840	32487	2289	30273

1	14	15	16	17	18	19	20	21	22	23	24	25
Sivusto	Html-tied: Tapahtuma-attribuutit	Js-tied: Kaikki takaisin-kutsu-parametrit	Html-tied: Kaikki takai-kutsu-parametrit	Js-tied: Kaikki anonyy funktioparame	Html-tied: Kaikki anonyy funktioparame	Js-tied: Takaisin-kutsuja-funktiomäär.	Html-tied: Takaisin-kutsuja-funktiomäär.	Js-tied: Funktio-määritykset	Html-tied: Funktio-määritykset	Kaikki tied: Promiset ".then(" lkm	Kaikki tied: Async.js	Kaikki tied: jQuery
1	273	2303	1725	932	1572	137	126	4484	2554	0	0	6993
2	912	1017	1124	551	1116	90	0	2170	1856	3	0	212
3	284	1369	37490	520	34727	71	7402	1803	48687	0	0	1
4	0	76	4803	69	4803	3	592	134	11710	0	0	0
5	203	3092	5828	750	3722	96	1404	3302	7955	0	0	1441
6	581	4630	48552	1998	44664	394	2611	8495	115886	48	0	9453
7	0	572	11424	278	2448	26	2856	1049	17954	3	0	4489
8	106245	5	150816	5	140898	1	1545	28	180669	0	0	2014
9	238616	137	247560	44	247560	26	0	445	250012	0	0	1852
10	10461	1103	23677	760	21789	51	0	2079	40457	0	0	3996
11	0	549	21934	361	5982	27	11964	1148	42874	0	0	1000
12	4862	1373	105211	1010	53745	201	18161	3513	136569	6	0	12012
13	2152	155	2933	139	2933	11	0	367	4737	0	0	11000
14	422007	14421	437408	3470	435375	404	1156	18189	459410	0	0	13
15	16739	226	20673	146	20673	45	0	940	23603	0	0	1000
16	132037	1258	161076	813	161056	158	2115	4133	178502	0	0	15309
17	297	1367	7235	520	6489	71	1872	1803	10111	0	0	1
18	23074	723	24072	298	24072	52	0	994	26936	2	0	1996
19	3526	1042	8865	890	7865	127	0	3588	12909	0	0	3001
20	0	667	6993	450	999	30	0	1292	16983	0	0	1007
21	16819	0	21826	0	18899	0	2123	0	12938	0	0	40
22	4096	43	5147	41	4147	18	0	293	5147	0	0	1000
23	1	608	23	244	22	20	0	1268	43	6	0	15
24	0	1037	4920	356	1968	33	984	1140	9847	0	0	7566
25	1800	4550	18558	2300	17855	171	157	8776	29617	27	0	6948
26	0	12724	25402	3924	5862	427	6839	16853	36149	0	0	11141
27	12786	1496	50834	602	42453	142	13959	2520	92186	2	18122	2000
28	61690	863	74261	548	74140	44	633	2354	86468	0	0	2789
29	0	1366	50372	434	29372	62	7000	1595	76145	0	0	3001
30	314963	498	378503	262	368458	16	1000	929	386238	0	0	5036
31	1	1304	2377	456	1079	57	0	1816	5366	3	0	7140
32	0	1005	580	347	145	34	0	1118	1015	0	0	813
33	2157	5234	32487	2336	30273	428	5511	9516	68828	72	2530	6930

Verkkokaupparyhmä: johdetut arvot

	A1	A2	A3	B1	B2	B3	C2	C3	D2	D3	E2	E3	F1	F2	F3
Sivusto	JS: tkf / fk	HTML: tkf / fk	JS ja HTML: tkf / fk (est)	JS: anon / tkf	HTML: anon / tkf	JS ja HTML: anon / tkf (est)	HTML: tap.attr / fk	JS ja HTML: tap.attr / fk (est)	HTML: tap.attr / tkf	Js ja HTML: tap.attr / tkf	HTML: tap.attr / anon	JS ja HTML: tap.attr / anon (est)	JS: kaikki anon / kaikki tkf	HTML: kaikki anon / kaikki tkf	JS ja HTML: kaikki anon / kaikki tkf (est)
1	13.11%	9.90%	9.99%	43.04%	91.13%	89.28%	1.57%	1.52%	15.83%	15.22%	17.37%	17.04%	40.47%	91.13%	89.05%
2	9.62%	14.20%	11.62%	54.73%	99.29%	78.50%	11.52%	5.03%	81.14%	43.28%	81.72%	55.14%	54.18%	99.29%	77.86%
3	18.27%	19.38%	19.38%	41.89%	92.63%	92.44%	0.15%	0.15%	0.76%	0.75%	0.82%	0.82%	37.98%	92.63%	92.40%
4	10.40%	15.43%	15.43%	90.67%	100.00%	99.99%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	90.79%	100.00%	99.99%
5	20.49%	14.21%	15.79%	26.53%	63.86%	51.67%	0.50%	0.37%	3.48%	2.35%	5.45%	4.54%	24.26%	63.86%	50.13%
6	13.01%	8.07%	8.11%	45.08%	91.99%	91.40%	0.10%	0.10%	1.20%	1.18%	1.30%	1.29%	43.15%	91.99%	91.33%
7	14.86%	26.50%	25.36%	50.55%	27.27%	28.61%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	48.60%	21.43%	22.72%
8	2.84%	25.47%	25.47%	100.00%	93.42%	93.42%	17.94%	17.94%	70.45%	70.45%	75.41%	75.41%	100.00%	93.42%	93.42%
9	9.50%	69.65%	69.65%	36.07%	100.00%	100.00%	67.14%	67.13%	96.39%	96.39%	96.39%	96.39%	32.12%	100.00%	100.00%
10	9.42%	8.43%	8.44%	69.77%	91.86%	91.54%	3.80%	3.75%	45.13%	44.49%	49.13%	48.60%	68.90%	92.03%	91.69%
11	10.94%	19.99%	19.84%	68.19%	37.50%	37.77%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	65.76%	27.27%	27.53%
12	7.94%	26.70%	26.42%	75.74%	56.53%	56.62%	1.37%	1.34%	5.11%	5.09%	9.05%	8.99%	73.56%	51.08%	51.18%
13	7.40%	14.99%	14.96%	89.54%	100.00%	99.98%	11.00%	10.95%	73.37%	73.21%	73.37%	73.23%	89.68%	100.00%	99.98%
14	19.94%	39.85%	39.83%	25.95%	99.54%	99.50%	38.44%	38.40%	96.48%	96.43%	96.93%	96.92%	24.06%	99.54%	99.49%
15	7.32%	38.07%	37.98%	64.57%	100.00%	99.98%	30.83%	30.74%	80.97%	80.92%	80.97%	80.94%	64.60%	100.00%	99.98%
16	7.46%	31.72%	31.68%	65.39%	99.99%	99.97%	26.01%	25.96%	81.97%	81.94%	81.98%	81.96%	64.63%	99.99%	99.97%
17	18.26%	17.25%	17.39%	41.92%	89.69%	82.71%	0.71%	0.61%	4.11%	3.51%	4.58%	4.24%	38.04%	89.69%	81.48%
18	17.86%	39.96%	39.19%	43.76%	100.00%	99.11%	38.30%	36.97%	95.85%	94.33%	95.85%	95.18%	41.22%	100.00%	99.01%
19	5.73%	14.36%	14.33%	85.65%	88.72%	88.72%	5.71%	5.69%	39.77%	39.73%	44.83%	44.78%	85.41%	88.72%	88.72%
20	10.03%	6.86%	6.96%	69.27%	14.29%	16.76%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	67.47%	14.29%	16.77%
21		49.84%	49.84%		90.48%	90.48%	40.13%	40.13%	80.52%	80.52%	88.99%	88.99%		86.59%	86.59%
22	6.07%	33.69%	33.43%	95.12%	80.57%	80.60%	26.81%	26.56%	79.58%	79.44%	98.77%	98.57%	95.35%	80.57%	80.60%
23	13.92%	6.30%	13.31%	41.92%	95.65%	43.97%	0.27%	0.02%	4.35%	0.17%	4.55%	0.38%	40.13%	95.65%	42.16%
24	21.41%	12.48%	12.52%	37.74%	40.00%	39.98%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	34.33%	40.00%	39.95%
25	12.72%	9.11%	9.64%	51.77%	96.21%	87.70%	0.88%	0.76%	9.70%	7.84%	10.08%	8.94%	50.55%	96.21%	87.22%
26	20.30%	30.77%	26.37%	34.05%	30.00%	31.31%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	30.84%	23.08%	25.32%
27	15.23%	10.91%	10.94%	43.88%	84.68%	84.29%	2.78%	2.76%	25.51%	25.26%	30.12%	29.97%	40.24%	83.51%	83.05%
28	7.44%	27.58%	27.55%	65.55%	99.84%	99.82%	22.91%	22.88%	83.07%	83.04%	83.21%	83.19%	63.50%	99.84%	99.82%
29	18.09%	7.20%	7.21%	34.12%	66.19%	66.14%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	31.77%	58.31%	58.27%
30	13.31%	33.45%	33.45%	53.91%	97.35%	97.34%	27.84%	27.83%	83.21%	83.21%	85.48%	85.48%	52.61%	97.35%	97.34%
31	14.67%	6.87%	7.34%	36.76%	45.39%	44.36%	0.00%	0.00%	0.04%	0.04%	0.09%	0.08%	34.97%	45.39%	44.07%
32	21.63%	9.76%	14.70%	37.69%	25.00%	32.78%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	34.53%	25.00%	31.04%
33	13.03%	10.94%	11.17%	47.29%	93.18%	87.28%	0.73%	0.65%	6.64%	5.79%	7.13%	6.63%	44.63%	93.18%	86.50%

	G1	G2	G3	H1	H2	H3
Sivusto	JS: tkfm / fm	HTML: tkfm / fm	JS ja HTML: tkfm / fm (est)	JS: kaikki tkf / tkf	HTML: kaikki tkf / tkf	JS ja HTML: kaikki tkf / tkf
1	3.06%	4.93%	4.83%	1.07	1.00	1.00
2	4.15%	0.00%	2.24%	1.03	1.00	1.02
3	3.94%	15.20%	15.16%	1.10	1.00	1.00
4	2.24%	5.06%	5.05%	1.01	1.00	1.00
5	2.91%	17.65%	13.33%	1.09	1.00	1.03
6	4.64%	2.25%	2.28%	1.07	1.00	1.00
7	2.48%	15.91%	15.17%	1.04	1.27	1.26
8	3.57%	0.86%	0.86%	1.00	1.00	1.00
9	5.84%	0.00%	0.00%	1.12	1.00	1.00
10	2.45%	0.00%	0.04%	1.03	1.02	1.02
11	2.35%	27.91%	27.72%	1.05	1.38	1.37
12	5.72%	13.30%	13.23%	1.04	1.11	1.11
13	3.00%	0.00%	0.01%	1.01	1.00	1.00
14	2.22%	0.25%	0.25%	1.09	1.00	1.00
15	4.79%	0.00%	0.01%	1.01	1.00	1.00
16	3.82%	1.18%	1.19%	1.02	1.00	1.00
17	3.94%	18.51%	16.31%	1.10	1.00	1.02
18	5.23%	0.00%	0.11%	1.06	1.00	1.00
19	3.54%	0.00%	0.01%	1.01	1.00	1.00
20	2.32%	0.00%	0.09%	1.04	1.00	1.00
21		16.41%	16.41%		1.04	1.04
22	6.14%	0.00%	0.07%	1.05	1.00	1.00
23	1.58%	0.00%	1.53%	1.04	1.00	1.04
24	2.89%	9.99%	9.96%	1.11	1.00	1.00
25	1.95%	0.53%	0.85%	1.03	1.00	1.01
26	2.53%	18.92%	14.42%	1.11	1.30	1.24
27	5.63%	15.14%	15.05%	1.12	1.01	1.02
28	1.87%	0.73%	0.73%	1.04	1.00	1.00
29	3.89%	9.19%	9.19%	1.07	1.14	1.14
30	1.72%	0.26%	0.26%	1.02	1.00	1.00
31	3.14%	0.00%	0.26%	1.06	1.00	1.01
32	3.04%	0.00%	1.59%	1.09	1.00	1.06
33	4.50%	8.01%	7.58%	1.08	1.00	1.01

	I1	I2	J1	J2	K1	K2
Sivusto	JS: Maksimi-sisäkkäis.	HTML: Maksimi-sisäkkäis.	JS: Sisäkkäis. keskiarvo	HTML: Sisäkkäis. keskiarvo	JS: Sisäkkäis. mediaani	HTML: Sisäkkäis. mediaani
1	4	4	1.43	1.82	1	2
2	5	1	1.67	1.00	2	1
3	5	4	2.03	1.81	2	2
4	3	1	1.88	1.00	2	1
5	5	2	2.12	1.46	2	1
6	4	4	1.46	1.87	1	2
7	4	2	1.50	1.83	1	2
8	2	3	1.26	1.01	1	1
9	2	1	1.23	1.00	1	1
10	4	4	1.63	1.06	2	1
11	4	2	1.50	1.83	1	2
12	6	3	2.01	1.80	2	2
13	4	1	2.02	1.00	2	1
14	5	2	1.64	1.01	1	1
15	3	2	1.57	1.05	2	1
16	4	4	1.52	1.14	1	1
17	5	3	2.03	1.59	2	1
18	5	1	1.79	1.00	2	1
19	5	1	1.99	1.00	2	1
20	4	1	1.40	1.00	1	1
21	1	3	1.00	1.24	1	1
22	2	1	1.30	1.00	1	1
23	4	3	1.55	1.16	2	1
24	5	1	1.89	1.00	2	1
25	5	3	1.62	1.70	2	2
26	3	2	1.47	1.83	1	2
27	4	3	1.39	1.30	1	1
28	3	2	1.43	1.03	1	1
29	5	2	2.05	1.34	2	1
30	6	2	1.52	1.01	1	1
31	4	2	1.45	1.20	1	1
32	5		1.95		2	
33	6	1	1.61	1.00	1	1

	L2
Sivusto	HTML: js loc / loc
1	22.10%
2	17.11%
3	17.00%
4	4.99%
5	13.08%
6	53.12%
7	40.35%
8	30.37%
9	35.74%
10	23.68%
11	18.10%
12	40.19%
13	3.35%
14	34.62%
15	7.16%
16	34.39%
17	20.56%
18	9.36%
19	23.08%
20	15.25%
21	31.12%
22	10.36%
23	43.52%
24	24.87%
25	25.55%
26	39.86%
27	82.47%
28	35.01%
29	43.20%
30	33.77%
31	2.44%
32	18.34%
33	24.90%

Verkkokaupparyhmä: sisäkkäisyydet

	JavaScript-tiedostojen sisäkkäisyystasot							HTML-tiedostojen sisäkkäisyystasot			
Sivusto	1	2	3	4	5	6		1	2	3	4
1	63.07%	31.41%	5.04%	0.48%	0.00%	0.00%		32.08%	57.10%	7.82%	3.00%
2	39.49%	55.20%	4.25%	0.85%	0.21%	0.00%		100.00%	0.00%	0.00%	0.00%
3	36.08%	28.09%	33.17%	2.42%	0.24%	0.00%		40.30%	41.45%	15.09%	3.16%
4	29.69%	53.13%	17.19%	0.00%	0.00%	0.00%		100.00%	0.00%	0.00%	0.00%
5	24.63%	44.03%	26.87%	3.48%	1.00%	0.00%		53.63%	46.37%	0.00%	0.00%
6	58.80%	36.96%	3.87%	0.36%	0.00%	0.00%		33.96%	49.19%	12.48%	4.37%
7	53.33%	44.00%	2.00%	0.67%	0.00%	0.00%		16.67%	83.33%	0.00%	0.00%
8	74.19%	25.81%	0.00%	0.00%	0.00%	0.00%		99.08%	0.46%	0.46%	0.00%
9	77.03%	22.97%	0.00%	0.00%	0.00%	0.00%		100.00%	0.00%	0.00%	0.00%
10	44.41%	48.95%	6.12%	0.52%	0.00%	0.00%		94.44%	5.24%	0.28%	0.04%
11	55.46%	40.34%	3.36%	0.84%	0.00%	0.00%		16.67%	83.33%	0.00%	0.00%
12	21.49%	61.28%	13.46%	2.83%	0.71%	0.24%		23.72%	72.35%	3.93%	0.00%
13	16.67%	66.67%	14.49%	2.17%	0.00%	0.00%		100.00%	0.00%	0.00%	0.00%
14	52.74%	32.88%	13.01%	0.68%	0.68%	0.00%		99.39%	0.61%	0.00%	0.00%
15	48.03%	47.37%	4.61%	0.00%	0.00%	0.00%		95.29%	4.71%	0.00%	0.00%
16	54.85%	38.92%	5.40%	0.83%	0.00%	0.00%		89.94%	5.67%	4.36%	0.03%
17	36.08%	28.09%	33.17%	2.42%	0.24%	0.00%		51.86%	37.60%	10.54%	0.00%
18	44.81%	34.07%	18.89%	1.85%	0.37%	0.00%		100.00%	0.00%	0.00%	0.00%
19	35.69%	33.77%	26.71%	3.59%	0.24%	0.00%		100.00%	0.00%	0.00%	0.00%
20	67.60%	25.78%	5.92%	0.70%	0.00%	0.00%		100.00%	0.00%	0.00%	0.00%
21	100.00%	0.00%	0.00%	0.00%	0.00%	0.00%		78.59%	18.39%	3.02%	0.00%
22	69.57%	30.43%	0.00%	0.00%	0.00%	0.00%		100.00%	0.00%	0.00%	0.00%
23	48.83%	48.36%	1.88%	0.94%	0.00%	0.00%		89.47%	5.26%	5.26%	0.00%
24	36.30%	42.35%	17.79%	3.20%	0.36%	0.00%		100.00%	0.00%	0.00%	0.00%
25	47.51%	44.68%	6.53%	0.87%	0.41%	0.00%		40.29%	49.09%	10.63%	0.00%
26	60.49%	32.24%	7.27%	0.00%	0.00%	0.00%		16.67%	83.33%	0.00%	0.00%
27	65.54%	30.71%	3.37%	0.37%	0.00%	0.00%		75.94%	18.42%	5.64%	0.00%
28	60.26%	36.24%	3.49%	0.00%	0.00%	0.00%		96.60%	3.40%	0.00%	0.00%
29	16.80%	64.00%	16.53%	2.40%	0.27%	0.00%		66.37%	33.63%	0.00%	0.00%
30	60.94%	31.33%	4.72%	1.29%	1.29%	0.43%		99.20%	0.80%	0.00%	0.00%
31	60.99%	34.07%	4.40%	0.55%	0.00%	0.00%		80.15%	19.85%	0.00%	0.00%
32	31.87%	45.79%	18.68%	3.30%	0.37%	0.00%					
33	54.73%	31.38%	12.67%	0.97%	0.20%	0.05%		100.00%	0.00%	0.00%	0.00%