Tampereen yliopisto

Eero Lehtinen

TYPESCRIPTIN VAIKUTUKSET OHJELMAN LAATUUN

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintyö
Huhtikuu 2022

TIIVISTELMÄ

Eero Lehtinen: TypeScriptin vaikutukset ohjelman laatuun
Kandidaatintyö
Tampereen yliopisto
Tietotekniikan tutkinto-ohjelma
Huhtikuu 2022

Internet muuttuu vuosi vuodelta tärkeämmäksi, ja JavaScript on verkkosivuilla laajassa käytössä. JavaScript on dynaamisen ja heikon tyypityksen ohjelmointikieli, jonka ominaisuudet eivät kaikkia miellytä. JavaScriptiä korvaamaan on kehitetty ohjelmointikieli TypeScript, jossa on staattinen ja vahva tyypitys. Sen tarkoituksena on korjata JavaScriptin ongelmia. TypeScriptin erityisenä hyötynä ajatellaan olevan ohjelmistoprojektin skaalautuvuus suureen kokoon. Tässä työssä on tarkoitus selvittää, millaiset vaikutukset TypeScriptillä on ohjelman laatuun. Olennaisena kysymyksenä on, kumpi on parempi vaihtoehto tuleviin ohjelmistoprojekteihin.

Tämä työ on toteutettu kirjallisuuskatsauksena. Analysoitavana oli muun muassa kokeellisia tutkimuksia, joissa vertaillaan staattisen ja dynaamisen tyypityksen vaikutuksia virheiden määrään, tuotteliaisuuteen ja ohjelmoinnin nopeuteen. Nämä ovatkin olennaisia ohjelman laadun osa-alueita. Eräs tutkimus myös analysoi yleisen versiohallinta-alustan GitHubin koodimuutoksia, ja yrittää saada selville ohjelmointikielistä ja niiden ominaisuuksista johtuvia vaikutuksia ohjelman laatuun.

Kirjallisuuden perusteella voidaan tehdä johtopäätöksiä. TypeScript on useimmiten suositeltava vaihtoehto JavaScriptille. Suurin osa kokeellisista tutkimuksista osoittivat joko positiivista tai neutraalia vaikutusta staattisesti tyypitetyille kielille. Kolmessa tutkimuksessa kuitenkin huomattiin, että pienet ohjelmointitehtävät on todennäköisesti nopeampi kirjoittaa dynaamisesti tyypitetyllä kielellä. JavaScript siis sopii tällaiseen tapaukseen. Kahdessa tutkimuksessa, jotka käsittelivät todellisia projekteja tehtävien sijaan, huomattiin selkeitä vähennyksiä virheiden määrässä TypeScriptillä. Siis hyvin pienimuotoiseen projektiin tai prototyyppiin voi käyttää JavaScriptiä, mutta yhtään suurempaan projektiin TypeScript on järkevämpi vaihtoehto. Myös JavaScript-koodin muuttaminen TypeScriptiksi kesken projektin on mahdollista vähittäisen tyypityksen ansiosta.

Avainsanat: staattinen tyypitys, ohjelman laatu, TypeScript, JavaScript

SISÄLLYSLUETTELO

1. JOHDANTO

Internet on viime vuosikymmenien aikana noussut hyvin tärkeäksi osaksi jokaisen arkea digitalisaation myötä. Vanhemmat verkkosivustot saattavat vielä käyttää pelkkää staattista HTML-koodia tekstin jakamiseen, mutta vuonna 1995 Netscape Navigator 2 -selaimessa julkaistun JavaScriptin [8] myötä dynaamiset toiminnot tulivat mahdollisiksi. Nykyään monet verkkosivut ovat läpeensä dynaamisia ja niiden toiminnallisuudet ovat verrattavissa selaimen ulkopuolisiin sovelluksiin. JavaScriptiä käytetään myös selaimen ulkopuolisessa ohjelmoinnissa Node.js-moottorilla esimerkiksi palvelimien backendissä.

JavaScript on siis tullut erittäin suosituksi. Jopa kaksi kolmasosaa ammatikseen ohjelmoivista on käyttänyt JavaScriptiä laajamittaiseen ohjelmistokehitykseen viimeisen vuoden aikana [19]. Tämä luku on suurin kaikista kyselyssä olleista ohjelmointikielistä. JavaScript ei kuitenkaan kaikkia miellytä, koska sen tyypitys on heikkoa ja dynaamista. Esimerkiksi JavaScriptin vertailulausekkeiden implisiittiset tyyppimuunnokset aiheuttavat päänvaivaa jopa kokeneille ohjelmoijille.

Tähän ongelmaan onkin kehitetty ratkaisu: TypeScript. Se on rakennettu JavaScriptin päälle lisäten staattisen ja vahvan tyypityksen ohjelmoijan avuksi. TypeScript on päässyt myös suureen suosioon. Jopa yksi kolmasosaa ammatikseen ohjelmoivista käyttää TypeScriptiä [19]. Luku on viidenneksi suurin kyselyssä listatuista kielistä. Myös yli yhdeksän kymmenestä TypeScriptiä käyttäneestä käyttäisi sitä uudestaan [2].

TypeScript on siis pidetty, mutta sen konkreettiset käytännön hyödyt eivät kuitenkaan ole täysin itsestäänselviä. Ne voivat perustua yleisiin uskomuksiin, joita ei välttämättä ole todistettu tieteellisesti. Tämän työn tavoitteena onkin selvittää, millainen vaikutus TypeScriptillä on ohjelman laatuun verrattuna JavaScriptiin. Olennainen kysymys on, että kumpi on parempi vaihtoehto seuraavaan projektiin, jos tahtoo mahdollisimman virheetöntä koodia mahdollisimman nopeasti. Työssä käsitellään TypeScriptin hyötyjä verrattuna JavaScriptin lähinnä seuraavien vastakkainasettelujen kautta: vahva vastaan heikko tyypitys ja staattinen vastaan dynaaminen tyypitys. Työ on tehty kirjallisuuskatsauksena painottaen empiiristä tutkimusta.

Seuraavassa luvussa esittelen JavaScriptin ominaisuuksia ja kolmannessa TypeScriptin ominaisuuksia. Neljännessä luvussa käsittelen yleisiä näkemyksiä tyyppijärjestelmistä. Viidennessä luvussa tähdennän ohjelman laadun käsitettä. Kuudennessa luvussa esitte-

len tutkimusten tuloksia ja seitsemännessä pohdin niiden merkityksiä suhteutettuna tämän kandidaatintyön tavoitteisiin. Kahdeksannessa luvussa on yhteenveto.

2. JAVASCRIPT

JavaScript on ohjelmointikieli, joka julkaistiin Netscape Navigator 2 -selaimessa vuonna 1995 [8]. Sitä aikaisemmat sivustot olivat hyvin staattisia, mutta JavaScript mahdollisti dynaamiset ominaisuudet. Nykyään JavaScript on erittäin suuressa käytössä, ja sen avulla rakennetaan monimutkaisia käyttöliittymiä, jotka eivät olisi muuten mahdollisia.

JavaScript ei ole sama kuin Sun Microsystemsin kehittämä Java. JavaScriptin nimeämisessä tarkoituksena oli vain Javan olemassa olevan suosion hyödyntäminen. [8, 12] Hämmennystä lisäsi se, että myös Java-kieltä pystyi aikoinaan ajamaan selaimessa. Yksi ainoista näitä kieliä yhdistävistä asioista on se, että molemmilla on hieman samantyylinen koodisyntaksi.

JavaScript on moniparadigmainen, yksisäikeinen ja dynaaminen kieli. Se tukee olio-, imperatiivista ja funktionaalista ohjelmointia. [10] JavaScriptissä on dynaaminen ja heikko tyypitys, joista kerrotaan lisää luvussa 4. JavaScriptin viralliseen standardiin kuuluvia ominaisuuksia hallitsee nykyään Ecma international [21], jonka mukaan sen kehittämää kieltä kuuluisi kutsua ECMAScriptiksi, mutta yleiseen ymmärrettävyyteen vedoten sitä kutsutaan edelleen JavaScriptiksi.

JavaScript on tulkattu kieli. Internetselaimet sisältävät oman tulkkinsa, johon JavaScript-lähdekoodi syötetään tekstimuodossa. [8] Tällöin ohjelmoijan ei itse tarvitse käyttää kääntäjää. Selaimet voivat kuitenkin suorittaa ajonaikaista kääntämistä (just-in-time compilation), jossa ohjelma käännetään tehokkaampaan muotoon tulkkaamisen jälkeen. JavaScript-koodia voidaan ajaa nykyään myös selaimen ulkopuolella. Yleisin työkalu tämän saavuttamiseksi on Node.js [13]. Sitä käytetään esimerkiksi internetsivujen palvelimissa.

JavaScriptin suosio on selvää. W3Techsin [23] mukaan JavaScript on käytössä 97,9 %:ssa internetsivustojen frontendeistä. Vain 2,1 % sivustojen frontendeistä ei käytä mitään ohjelmointikieltä [23]. Lisäksi Stack Overflowin [19] vuoden 2021 kyselyn mukaan lähes 69 % ammatikseen ohjelmoivista on käyttänyt JavaScriptiä laajamittaiseen ohjelmistokehitykseen viimeksi kuluneen vuoden aikana. Tämä luku on suurin kaikista kyselyssä olleista ohjelmointikielistä. Backendin puolella käyttö on kuitenkin harvinaisempaa. W3Techsin [24] mukaan vain 1,8 % verkkosivujen palvelimista käyttää Node.js:ää, vaikka sen trendi onkin kasvava.

3. TYPESCRIPT

TypeScript on Microsoftin kehittämä ohjelmointikieli, joka laajentaa JavaScriptiä mahdollistaen staattisen ja vahvan tyypityksen käyttämisen [22]. Ilman TypeScriptiä JavaScriptin tyypitys on dynaamista ja heikkoa. Näistä tyyppijärjestelmistä kerrotaan lisää luvussa 4. Eräs TypeScriptin iskulauseista on suomeksi käännettynä ”JavaScript, joka skaalautuu” [22]. Skaalautumisen ajatellaan onnistuvan paremmin TypeScriptin tyyppijärjestelmällä.

TypeScriptin ominaisuudet on toteutettu sen kääntäjän avulla. TypeScriptin kääntäjä ei kuitenkaan toimi tavanomaisesti. Se ei muunna lähdekoodia konekieliseksi tiedostoksi, vaan se kääntää TypeScriptin lähdekoodin JavaScriptin lähdekoodiksi. [3] Tämän takia TypeScriptin kääntäjää sanotaankin transkääntäjäksi (transpiler). Näin mahdollistetaan yhteensopivuus internetselainten JavaScript-tulkkien kanssa.

TypeScript on JavaScriptin ylijoukko. Se tarkoittaa, että jokainen JavaScriptillä kirjoitettu koodipätkä on myös validia TypeScriptiä ja TypeScript tuo myös kieleen omia lisäyksiä [3]. Käytännössä tämä usein tarkoittaa sitä, että TypeScript-lähdekoodi on JavaScript-lähdekoodia, jossa muuttujiin ja funktioiden palautusarvoihin on merkitty tyypit tyyppiannotaatioilla. Esimerkki tästä näkyy ohjelmassa 3.1. Siinä funktion *summa* parametrit ja palautusarvot on merkitty numeroiksi. Myös *muuttuja1* on merkitty numeroksi ja *muuttuja2* merkkijonoksi. TypeScript kykenee myös päättelemään muuttujan tyypin [3]. Päättely onnistuu esimerkiksi silloin, kun muuttujaan asetetaan arvo samalla kun se määritellään. Tällaisia asetuksia tapahtuu ohjelman riveillä 12 ja 13. TypeScript siis silti tunnistaisi niiden tyypit oikein, vaikka niissä ei olisikaan tyyppiannotaatioita.

TypeScriptin toiminnallisuus perustuu siihen, että sen kääntäjä tunnistaa tyyppivirheet ja varoittaa niistä. Jos esimerkiksi yrittää tehdä vertailuoperaation tekstijonon ja numeron kesken, kääntäjä ilmoittaa, että tulos ei välttämättä ole odotusten mukainen. TypeScript tunnistaa tyyppivirheitä kääntämisen aikana, mutta ei kuitenkaan huolehdi ajon aikaisista tyypeistä, koska se pyyhkii tyyppi-informaation kääntäessään koodin JavaScriptiksi [3]. Ohjelmoija joutuu itse varmistamaan, että ulkoinen, esimerkiksi tietokannasta tuleva data on aina varmasti samaa tyyppiä kuin siihen merkitty tyyppiannotaatio.

TypeScriptin suosio on kasvanut JavaScriptin kanssa. StackOverflowin [19] vuonna 2021 pidetyn kyselyn mukaan TypeScriptiä käyttää 36 % ammatikseen ohjelmoivista vastaajista. Se on viidenneksi suurin kaikista kyselyssä olleista kielistä. Käytön taso on myös hie-

Ohjelma 3.1. *Esimerkki JavaScript-koodista ja sitä vastaavasta TypeScript-koodista*

```
1  // JavaScript
2  function summa(a, b) {
3      return a + b;
4  }
5  let muuttuja1 = 0;
6  let muuttuja2 = "asd";
7
8  // TypeScript
9  function summa(a: number, b: number): number {
10     return a + b;
11 }
12 let muuttuja1: number = 0;
13 let muuttuja2: string = "asd";
```

man yli puolet verrattuna JavaScriptin vastaavaan käytön tasoon. Lisäksi State of JS:n [2] kyselyn mukaan 79 % vastanneista on käyttänyt TypeScriptiä ja 94 % käyttäneistä käyttäisi sitä uudestaan. Suosiosta voisi päätellä, että TypeScriptin ominaisuudet tuovat ainakin jonkinlaista haluttua lisäarvoa ohjelmointiin.

4. TYYPPIJÄRJESTELMÄT

Suuri osa nykypäivän suosituista ohjelmointikielistä sisältää tyypityksen. Eri kielten toteutukset kuitenkin eroavat merkittävästi toisistaan. Eräät näkökulmat, joista asiaa voidaan käsitellä, ovat kielen tyyppien staattisuus ja tyyppien vahvuus.

4.1 Staattinen tyypitys

Bruce [1, s. 8–9] määrittelee staattisen tyypityksen siten, että jokaiseen lausekkeeseen asetetaan tyyppi jo kääntämisen aikana. Tämä mahdollistaa tyyppivirheiden tarkistamisen heti kääntämisen aikana. Brucen mukaan staattisessa tyypityksessä on useita hyötyjä:

- Se antaa aikaisemmin ja tarkemmin informaatiota virheistä.
- Sen kanssa ei tarvitse suorittaa ajon aikaisia tyyppitarkistuksia, jotka kasvattavat ohjelman kokoa ja hidastavat sitä.
- Se itsessään tuottaa dokumentaatiota.
- Se antaa kääntäjälle lisäinformaatiota optimointia varten. [1, s. 8–9]

Dynaaminen tyypitys tarkoittaa taas sitä, että tyypit tarkistetaan vasta ajon aikana [1, s. 8–9]. Myös Meijer ja Drayton [11] kertovat, että staattinen tyypitys on tehokas työkalu oikeellisen ohjelmoinnin kirjoittamiseen, mutta heidän mukaansa dynaaminenkin tyypitys on tarpeellista. Dynaamisen tyypityksen hyödyiksi ajatellaan esimerkiksi seuraavat:

- Se antaa joustavuutta, kaikkia toimivia ohjelmia ei voi määritellä staattisesti [11, 9].
- Se nopeuttaa prototyypitystä [11, 9].

Näiden määritelmien mukaisesti JavaScriptissä on dynaaminen tyypitys. TypeScript mahdollistaa staattisen tyypityksen käyttämisen, mutta ei pakota siihen. Käytännössä TypeScriptin staattinen tyypitys ilmenee kääntäjän tyyppitarkastuksista ja itse koodissa olevista tyyppiannotaatioista. Esimerkki koodista ilman annotaatioita ja niiden kanssa näkyy aiemmin esitetystä ohjelmasta 3.1.

4.2 Vahva tyypitys

Ray et al. [16] määrittelevät vahvan tyypityksen käsitteen tyyppihämmennys (type-confusion) avulla. Tyyppihämmennyksessä ohjelma yrittää lukea tyypiksi T1 määritellyn muistialueen tyypin T2 instanssina. Vahvan tyypityksen ohjelmointikielet tunnistavat tyyppihämmennykset joko ajon tai kääntämisen aikana ja ilmoittavat niistä. [16] Heikko tyypitys on taas vahvan vastakohta. Siinä jotkin tyyppihämmennykset hyväksytään. Tämän määritelmän mukaan TypeScriptissä on vahvempi tyypitys kuin JavaScriptissä.

JavaScriptissä heikko tyypitys ilmenee implisiittisellä tyyppien pakottamisella (implicit type coercion). Pradelin ja Senin [14] mukaan se on hyvin kiistanalainen ominaisuus. Esimerkiksi C-kieli muuttaa tasaluvun liukuluvuksi, jos operaatio sitä vaatii. JavaScript kuitenkin käyttää tyypin pakotuksia paljon laajemmin, mikä tekee sen tyyppijärjestelmästä erityisen heikon. JavaScriptissä tyyppien pakotusten tavoitteena on välttää ajonaikaisia poikkeuksia niin paljon kuin mahdollista. Toisaalta tyyppien pakotus aiheuttaa myös hämmentäviä interaktioita, joiden käyttäytymistä monet kokeneetkaan ohjelmoijat eivät ymmärrä. [14] Esimerkiksi lauseke

```
"0" == false
```
(4.1)

evaluoituu arvoksi `true`. Siinä JavaScript ei osaa verrata merkkijonoa ja totuusarvoa keskenään, joten ne yritetään muuttaa samaan muotoon. JavaScript tekee implisiittisen tyypin pakotuksen merkkijonolle ja tekee siitä numeron 0. Myös totuusarvo `false` pakotetaan numeroksi 0. Loppujen lopuksi siis verrataan nollaa toiseen nollaan, jolloin tulos on tosi.

TypeScriptin vahva tyypitys taas estää lausekkeen 4.1 kaltaiset vertailut. Se estää ohjelman kääntämisen ja kertoo virheviestissä, että merkkijonoja ja totuusarvoja ei kuulu vertailla keskenään. Ohjelmoija joutuu itse muuttamaan muuttujat oikeaan muotoon.

4.3 Vähittäinen tyypitys

Siek ja Taha [17] määrittelevät käsitteen vähittäinen tyypitys (gradual typing) siten, että se on tyyppijärjestelmä, joka antaa ohjelmoijan sekoittaa staattista ja dynaamista tyypitystä. Kun tyyppiannotaation ottaa käyttöön johonkin lausekkeeseen, kääntäjä varmistaa sen tyypin käytön oikeellisuuden staattisesti kääntämisen aikana. Taas jos ei käytä tyyppiannotaatiota lausekkeessa, sen tyyppi varmistetaan dynaamisesti ajon aikana.

TypeScript on vähittäisesti tyypitetty kieli [3]. TypeScriptin käyttöönoton helppoudessa on ollut aina valttina se, että tyyppejä voi lisätä vähitellen olemassa olevaan suureenkin JavaScript-projektiin.

TypeScriptissä oletustyyppinä on `any` [3]. TypeScript käyttää sitä, jos muuta tyyppiä ei ole merkitty tyyppiannotaatiolla tai sitä ei saa pääteltyä kontekstista. Tyyppi `any` voi nimensä

mukaisesti olla mitä tahansa tyyppiä staattisen tyyppitarkistuksen näkökulmasta, eli se toimii kuten tavallinen JavaScriptin arvo. Cherny [3] varoittaa käyttämästä `any`-tyyppiä, koska se poistaa tyyppitarkistuksen suoman turvan. TypeScriptissä hän suositteleekin käyttämään kääntäjän konfiguraatioasetusta *noImplicitAny*.

5. OHJELMAN LAADUN MITTAAMINEN

Ohjelman laatu on monimuotoinen ja subjektiivinen käsite. Yleisellä tasolla laadukas ohjelma on sellainen, joka täyttää sille asetetut vaatimukset. Vaatimuksiin useimmiten kuuluvat toiminnalliset ominaisuudet, eli ohjelman kyky suorittaa vaaditut asiat ilman virheitä. Myös ei-toiminnalliset asiat, kuten esimerkiksi ylläpidettävyys ja laajennettavuus, ovat tärkeitä laadun merkkejä. Myös kaikki nämä osa-alueet täytyy todellisessa maailmassa suhteuttaa ohjelmointityön nopeuteen, sillä aikaa on aina rajallisesti. Voidaan olettaa, että muiden muuttujien pysyessä samana suurempi ajankäyttö korreloi suuremman laadun kanssa.

Näiden laadun osa-alueiden mittaaminen empiirisissä kokeissa on usein ongelmallista. Ei-toiminnallisten asioiden mittaaminen olisi liian subjektiivista ja aikaa vievää. Toiminnallisten asioiden mittaaminen onkin hieman helpompaa. Tässä työssä analysoitavina olevissa tutkimuksissa useimmiten annettiinkin ohjelmointitehtäviä koehenkilöille, joista sitten mitattiin tuotettujen vaadittujen toiminnallisuuksien määrää ja siihen käytettyä aikaa. Usein toinen näistä muuttujista pidettiin vakiona, ja vain toista mitattiin. Nämä tutkimukset eivät myöskään luvanneet mittaavansa laatua yleensä, vain nopeutta tai toiminnallisuuksia.

Erilaista metodeja käytti Ray et al. [16] tutkimuksessaan, jossa koodin laatua yritettiin mitata suoremmin käyttämällä tekstianalyysia GitHubin projektien koodimuutoksiin. Niistä yritettiin laskea ja kategorisoida vain virheiden korjaukset. Sitten näitä vertailemalla pystyi saamaan selville, mitkä ohjelmointikielet aiheuttivat verrattain eniten ja vakavimpia virheitä.

Hanenberg et al. [6] tutki vaikeammin määriteltävissä olevaa ylläpidettävyyttä. Heidän kehittelemänsä koe sisälsi dokumentoimattoman koodin ymmärtämistä, tyyppivirheiden korjaamista ja semanttisten virheiden korjaamista. Mittauksen alaisena oli tehtäviin kulunut aika. Näiden tulosten ajateltiin korreloivan ylläpidettävyyden kanssa.

6. TYYPPIJÄRJESTELMIEN HYÖTYJEN JA HAITTOJEN TOTEUTUMINEN

Prechelt ja Tichy [15] tekivät yhden ensimmäisistä tutkimuksista, jossa selvitettiin staattisen tyypityksen vaikutuksia empiirisesti. Siihen on viitattu suuressa osassa myöhemmistä aihetta käsittelevistä tutkimuksista. Siinä annettiin 40 opiskelijalle kaksi tehtävää C-kielellä, joista toisessa oli käytössä kääntäjä staattisella tyyppien tarkistuksella, kun taas toisessa kääntäjässä tyyppejä ei tarkastettu. Koodi oli tarkoituksella epätriviaalia ja käytössä oli kirjasto monimutkaisella rajapinnalla. Tuloksena oli, että tyyppitarkistetun kääntäjän kanssa tehdyt ohjelmat sisälsivät vähemmän virheitä, virheet korjattiin nopeammin ja opiskelijat olivat tuottoisampia.

Tulevat tutkimukset tarkentivat tuloksia, ja esittivät jonkin verran jopa vastakkaisia tuloksia. Pienten tehtävien ohjelmointi dynaamisesti tyypitetyllä kielellä huomattiin staattista nopeammaksi Hanenbergin [5], Stuchlikin ja Hanenbergin [20] ja Mayerin et al. [9] tutkimuksissa. Suurista tehtävistä kaikki eivät tulleetkaan samoihin tuloksiin. Mayer et al. [9] huomasi staattisen tyypityksen nopeuttavan suurien tehtävien tekemistä. Stuchlik ja Hanenberg [20] ja Hanenberg [5] taas eivät nähneet nopeuttavaa vaikutusta. Näissä pienten ja suurten tehtävien vertailussa on myös rajoituksia. Suuriksi luonnehditut tehtävät eivät olleet todellisuudessa kovin suuria. Ne olivat maksimissaan muutaman tunnin mittaisia yhdelle ohjelmoijalle tarkoitettuja tehtäviä, jotka eivät ole ollenkaan suuria verrattuna yritysten vuosikymmeniä kestäviin ohjelmointiprojekteihin.

Paremmin suuria ohjelmointiprojekteja tutki Ray et al. [16]. Siinä tutkittiin eri ohjelmointikielten ja paradigmojen vaikutuksia koodin laatuun tarkastelemalla GitHubista 80 miljoonaa koodiriviä 729 projektista. Projekteista laskettiin virheiden korjaamiseen liittyvät koodimuutokset. Siinä yritettiin erottaa koodin laatu muista muuttujista kontrolloimalla tiimin kokoa, projektin kokoa ja projektin historiaa. Tuloksena oli, että vahva tyypitys on parempi kuin heikko ja staattinen tyypitys on parempi kuin dynaaminen tyypitys. Tutkimuksen mallin mukaan TypeScript tuotti kaikkein vähiten ohjelmointivirheitä 17:stä tutkinnan alla olleesta ohjelmointikielestä. Nämä tulokset johdattelevat siihen johtopäätökseen, että TypeScript aiheuttaa parempaa ohjelman laatua kuin JavaScript. Tutkimuksen mukaan kuitenkin esimerkiksi projektin koko ja ikä vaikuttavat virheiden määrään paljon enemmän kuin ohjelmointikieli.

Spizan ja Hanenbergin [18] kehittämässä kokeessa koehenkilöille kerrottiin rajapintojen tyypit ilman staattista tyyppien tarkastusta. He huomasivat, että tuntemattoman ohjelmointirajapinnan käyttämisessä pelkkä tyyppi-informaatio nopeuttaa ohjelmointia. He myös huomasivat, että jos tyyppi-informaatio on virheellistä, se hidastaa ohjelmointia hyvin merkittävästi. Tästä voi tehdä johtopäätöksen, että staattinen tyypitys on hyödyllistä, koska staattinen tyypitys pakottaa tyyppi-informaation lisäämisen, mikä nopeuttaa ohjelmointia. Lisäksi se tarkistaa, että kaikki tyypit ovat oikein, millä saadaan vältettyä virheellisistä tyypeistä johtuvat merkittävät hidastukset.

Kaikkein tarkimmin tämän työn aiheeseen liittyvä Gaon et al. julkaisu [4] tutki staattisen tyypityksen kykyä löytää virheitä JavaScript-koodista. Siinä etsittiin JavaScript-projektien versiohallintahistorioista korjattuja virheitä ja testattiin, olisivatko manuaaliset tyyppiannotaatiot löytäneet virheen itsestään. Tuloksena oli, että TypeScript ja toinen TypeScriptin kaltainen tyyppitarkistin Flow olisivat molemmat löytäneet 15 % virheistä. Tulokset olivat konservatiivisia, koska tutkinnan alla oleva koodi oli testattu ja mennyt koodiarvioinneista läpi.

Hanenbergin et al. [6] tutkimuksessa arvioitiin staattisen tyypityksen vaikutusta ylläpidettävyyteen. Siinä tultiin positiivisiin tulokseen staattisen tyypityksen hyödyllisyyden puolesta jatkaen samaa linjaa Precheltin ja Tichyn [15] kanssa. Siinä staattiset tyypit auttoivat koehenkilöitä käyttämään uusia luokkia nopeammin ja nopeuttivat tyyppivirheiden korjaamista. Semanttisten virheiden korjaamisessa staattisella ja dynaamisella tyypityksellä ei ollut eroja. He myös tutkivat nopeushyötyjen syitä. He arvelevat, että staattisen tyypityksen aiheuttama nopeusvaikutus voisi johtua siitä, että ohjelmoija ei joudu vaihtelemaan eri tiedostojen välillä yhtä paljoa kuin dynaamisen tyypityksen kanssa.

Harlin, Washizaki ja Fukazawa [7] osoittavat, että staattisella tyypityksellä ei ole vaikutusta toiminnallisuuksien tuottamisessa. Tehtävissä, jossa piti korjata virheitä, staattisella tyypityksellä oli nopeuttava vaikutus.

7. POHDINTAA

Rayn et al. [16] ja Gaon et al. [4] tutkimukset tarkastelivat tämän työn aihetta kaikkein lähimmin. Niissä oli käytössä TypeScript, ja tutkimuksen alla olivat todelliset oikeassa käytössä olevat ohjelmointiprojektit. Niiden tulokset olivat selvästi TypeScriptin käyttöönoton kannalla virheiden vähentämiseksi.

Loput tässä työssä käsitellyistä tutkimuksista [15, 5, 20, 9, 6, 18, 7] käsittelivät kokeita, joissa pieni määrä opiskelijoita (14–49 henkilöä) laitetaan ohjelmoimaan tehtäviä sekä dynaamisesti että staattisesti tyypitetyllä kielellä. Niissä oli tuloksena, että yleensä ottaen staattisella tyypityksellä on joko neutraali tai positiivinen vaikutus toiminnallisuuksien tuottamiseen ja ohjelmointinopeuteen. On myös hieman todisteita siitä, että pienet tehtävät onnistuvat nopeammin dynaamisen tyypityksen kielillä kuin staattisen.

Näiden tulosten perusteella voidaan päätellä, että yleinen oletus dynaamisen tyypityksen kielellä prototyypityksen nopeudesta on jokseenkin perusteltu. Isomman skaalan ohjelmoinnissa huomataan, että staattisen tyypityksen lupaukset virheiden vähentämisestä vaikuttavat tosilta.

Kun käytännössä tahtoo valita seuraavaan projektiinsa ohjelmointikielen JavaScriptin ja TypeScriptin välillä, sen voisi tehdä seuraavasti. Jos on tarkoitus tehdä pieni projekti tai prototyyppi, joka tulevaisuudessa voi kasvaa suuremmaksi, kannattaa alustavasti valita JavaScript. Tämä voi nopeuttaa ohjelmointia. Sitten tulevaisuudessa, jos projekti kasvaa useamman päivän mittaiseksi, voi vähittäisen tyypityksen avulla vaihtaa TypeScriptiin, joka on vähemmän virheherkkä. Tietysti esimerkiksi yrityksissä, joissa lähes jokaisen projektin voidaan olettaa olevan vuosien mittainen ja ison ryhmän työstettävänä, on järkevää valita heti TypeScript.

Tämän työn tutkimusmenetelmissä on rajoituksena se, että kaikkia aiheeseen liittyviä tutkimuksia ei löydetty tietokantahauilla. On myös mahdollista, että tutkimusten tuloksia on ymmärretty väärin tai niistä on vedetty liian laajoja johtopäätöksiä. Itse analysoitavana olevassa kirjallisuudessakin on omat metodilliset heikkouksensa. Empiirisissä kokeissa [15, 5, 20, 9, 6, 18, 7] oli hyvin pienet otantakoot (14–49 henkilöä) ja tehtävät olivat hyvin pienimuotoisia. Tehtävät olivat myös itse pitäjien suunnittelemia, eli ne voivat vahingossa vinoutuneita osoittamaan tiettyä tulosta. Esimerkiksi jotkin tehtävät voivat olla helpompia kuin toiset, ja joissakin tehtävissä tyypit voivat olla monimutkaisempia kuin toisissa. Olisi

hyvä, jos saisimme jatkotutkimusta suuremmilla otantakoilla ja suuremmilla ohjelmointitehtävillä.

8. YHTEENVETO

Tässä työssä selvitin kirjallisuuskatsauksena, millaisia vaikutuksia TypeScriptillä on ohjelman laatuun verrattuna JavaScriptiin. Tämä on merkittävä aihe, koska mielestäni ohjelmointikielen valinta tulisi tehdä informoidusti mielummin kuin vetoamalla yleisiin uskomuksiin. Tässä työssä keskitytään internetsivustojen ohjelmointiin, jossa JavaScript ja TypeScript ovat yleisimmät kielet. TypeScript on suurilta osin samanlainen kuin JavaScript, mutta TypeScriptiin on lisätty staattinen tyypitys tyyppiannotaatioiden avulla. Staattinen tyypitys tarkoittaa sitä, että jokaiseen lausekkeeseen asetetaan tyyppi jo kääntämisen aikana. TypeScript myös käyttää tätä tyyppi-informaatiota tyyppivirheiden löytämiseen. TypeScript on myös vahvasti tyypitetty, eli se välttää huomaamattomia tyyppimuunnoksia. TypeScriptin vähittäinen tyypitys myös mahdollistaa osittaisen dynaamisen tyypityksen hyödyntämisen.

TypeScript on suositeltava vaihtoehto JavaScriptille lähes jokaiselle ohjelmointiprojektille, sillä todellisia suurehkoja ohjelmointiprojekteja analysoivat tutkimukset [16, 4] ovat osoittaneet sen vähentävän virheiden määrää. On kuitenkin myös kokeellisia tutkimuksia [5, 20, 9], jotka osoittavat staattisen tyypityksen hidastavan ohjelmointia merkittävästi hyvin pienissä projekteissa. Muuten kokeelliset tutkimukset [15, 6, 18, 7] ovat löytäneet staattisen tyypityksen olevan joko neutraalia tai positiivista ohjelman laadulle. Lyhyesti sanottuna JavaScript voi olla sopiva pienimuotoiselle ohjelmalle tai prototyypille ja TypeScript kaikelle muulle. Pienimuotoisen JavaScript-kielisen ohjelman kasvaessa se on helppo myös muuttaa TypeScriptiksi vähittäisen tyypityksen ja yleisen yhteensopivuuden ansiosta.

LÄHTEET

[1] Kim B Bruce. *Foundations of object-oriented languages: types and semantics*. MIT press, 2002.

[2] *Build Tools*. The State of JS 2021. URL: https://2021.stateofjs.com/en-US/libraries/build-tools/ (viitattu 20. 02. 2022).

[3] Boris Cherny. *Programming TypeScript: Making Your JavaScript Applications Scale*. Sebastopol: O'Reilly Media, Incorporated, 2019.

[4] Zheng Gao, Christian Bird ja Earl T. Barr. "To Type or Not to Type: Quantifying Detectable Bugs in JavaScript". Teoksessa: *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 2017, s. 758–769. DOI: 10.1109/ICSE.2017.75.

[5] Stefan Hanenberg. "An Experiment about Static and Dynamic Type Systems: Doubts about the Positive Impact of Static Type Systems on Development Time". *SIGPLAN Not.* 45.10 (lokakuu 2010), s. 22–35. DOI: 10.1145/1932682.1869462.

[6] Stefan Hanenberg et al. "An empirical study on the impact of static typing on software maintainability". *Empirical Software Engineering* 19.5 (lokakuu 2014), s. 1335–1382. DOI: 10.1007/s10664-013-9289-1.

[7] Ismail Rizky Harlin, Hironori Washizaki ja Yoshiaki Fukazawa. "Impact of Using a Static-Type System in Computer Programming". Teoksessa: *2017 IEEE 18th International Symposium on High Assurance Systems Engineering (HASE)*. 2017, s. 116–119. DOI: 10.1109/HASE.2017.17.

[8] Jeremy Keith. "A Brief History of JavaScript". *DOM Scripting: Web Design with JavaScript and the Document Object Model* (2005), s. 3–10.

[9] Clemens Mayer et al. "An Empirical Study of the Influence of Static Type Systems on the Usability of Undocumented Software". *SIGPLAN Not.* 47.10 (lokakuu 2012), s. 683–702. DOI: 10.1145/2398857.2384666.

[10] *MDN Web Docs: JavaScript*. Mozilla Foundation. URL: https://developer.mozilla.org/en-US/docs/Web/JavaScript (viitattu 12. 03. 2022).

[11] Erik Meijer ja Peter Drayton. "Static Typing Where Possible, Dynamic Typing When Needed: The End of the Cold War Between Programming Languages" (tammikuu 2004).

[12] Tom Negrino. *JavaScript*. 8th ed. Visual quickstart guide JavaScript. Peachpit Press, 2012.

[13] *Node.js*. OpenJS Foundation. URL: https://nodejs.org/ (viitattu 12. 03. 2022).

[14] Michael Pradel ja Koushik Sen. ”The Good, the Bad, and the Ugly: An Empirical Study of Implicit Type Conversions in JavaScript”. Teoksessa: *29th European Conference on Object-Oriented Programming (ECOOP 2015)*. Toim. John Tang Boyland. Vol. 37. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2015, s. 519–541. DOI: 10.4230/LIPIcs.ECOOP.2015.519.

[15] L. Prechelt ja W.F. Tichy. ”A controlled experiment to assess the benefits of procedure argument type checking”. *IEEE Transactions on Software Engineering* 24.4 (1998), s. 302–312. DOI: 10.1109/32.677186.

[16] Baishakhi Ray et al. ”A Large Scale Study of Programming Languages and Code Quality in Github”. Teoksessa: *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2014. Hong Kong, China: Association for Computing Machinery, 2014, s. 155–165. DOI: 10.1145/2635868.2635922.

[17] Jeremy Siek ja Walid Taha. ”Gradual Typing for Objects”. Teoksessa: *ECOOP 2007 – Object-Oriented Programming*. Toim. Erik Ernst. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, s. 2–27.

[18] Samuel Spiza ja Stefan Hanenberg. ”Type Names without Static Type Checking Already Improve the Usability of APIs (as Long as the Type Names Are Correct): An Empirical Study”. Teoksessa: *Proceedings of the 13th International Conference on Modularity*. MODULARITY ’14. Lugano, Switzerland: Association for Computing Machinery, 2014, s. 99–108. DOI: 10.1145/2577080.2577098.

[19] *Stack Overflow Developer Survey 2021: Most Popular Technologies*. Stack Overflow. URL: https://insights.stackoverflow.com/survey/2021#technology-most-popular-technologies (viitattu 20. 02. 2022).

[20] Andreas Stuchlik ja Stefan Hanenberg. ”Static vs. Dynamic Type Systems: An Empirical Study about the Relationship between Type Casts and Development Time”. 47.2 (lokakuu 2011), s. 97–106. DOI: 10.1145/2168696.2047861.

[21] *TC39*. Ecma International. URL: https://www.ecma-international.org/technical-committees/tc39/ (viitattu 20. 02. 2022).

[22] *TypeScript: Why does TypeScript exist?* Microsoft. URL: https://www.typescriptlang.org/why-create-typescript (viitattu 15. 03. 2022).

[23] *Usage statistics of client-side programming languages for websites*. Q-Success. URL: https://w3techs.com/technologies/overview/client_side_language (viitattu 12. 03. 2022).

[24] *Usage statistics of Node.js*. Q-Success. URL: https://w3techs.com/technologies/details/ws-nodejs (viitattu 12. 03. 2022).