Tampereen yliopisto

Jaakko Nikkilä

JAVASCRIPTISTÄ SIIRTYMINEN TYPESCRIPTIIN

Kandidaatintutkielma
Informaatioteknologian ja viestinnän tiedekunta
Huhtikuu 2023

TIIVISTELMÄ

Jaakko Nikkilä: JavaScriptistä siirtyminen TypeScriptiin
Kandidaatintutkielma
Tampereen yliopisto
Tietotekniikan tutkinto-ohjelma
Huhtikuu 2023

Ohjelmointikieli TypeScriptin suosion kasvaessa ammattilaisten keskuudessa moni aloitteleva ohjelmoija, joka on käyttänyt JavaScriptiä, pohtii TypeScriptiin siirtymistä. Tässä työssä tarkastellaan, mitkä seikat tukevat ohjelmoijan siirtymistä JavaScriptistä TypeScriptiin ja mitkä eivät. Työ on tehty kirjallisuuskatsauksena ja lähteinä on käytetty lähinnä aiheesta kirjoitettua kirjallisuutta, mutta myös dokumentaatioita ja tilastoja on käytetty.

Työssä käydään läpi ohjelmointikielien taustaa, tyyppiannotaatioita, tyyppitiedostoja, luokkia ja rajapintoja. Nämä ovat tärkeimmät asiat, jotka ohjelmoijan tulee huomioida siirtyessään TypeScriptiin JavaScriptistä. TypeScriptissä on asteittainen tyyppijärjestelmä JavaScriptin dynaamisen tyypityksen sijaan. TypeScriptiin siirtyessä tulee myös vastaan tyyppitiedostot (eng. declaration file), niitä tarvitaan, kun käytetään ensimmäisen tai kolmannen osapuolen paketteja TypeScriptillä. TypeScript esittelee myös olio-ohjelmoinnin tärkeät ominaisuudet luokat ja rajapinnat.

Asteittainen tyyppijärjestelmä tuo ohjelmoijalle vapautta siirtyä TypeScriptiin vaivatta. Se mahdollistaa staattisen tyypityksen, mutta myös ”any”-luokan avulla koodia pystyy kirjoittamaan dynaamisesti tyypitettynä. Monien empiiristen tutkimuksien mukaan staattisen tyypityksen on näytetty parantavan ohjelman laatua, koodin ylläpidettävyyttä ja ymmärrettävyyttä ja myös mahdollisesti ohjelmoijan tuottavuutta. Tyyppitiedostojen osalta ohjelmoija joutuu tekemään päätöksen, jos paketti ei tue valmiiksi TypeScriptiä. Ohjelmoija voi valita kirjoittaako tyyppitiedoston itse vai käyttääkö esimerkiksi GitHubista löytyvää DefinitelyTyped-tyyppitiedostokirjastoa. Osan koodista voi myös kirjoittaa JavaScriptinä, jos tyyppitiedostot tuovat liikaa ongelmia. Luokat ja rajapinnat tuovat ohjelmoijalle uusia työkaluja, joilla koodista saa ymmärrettävämpää, uudelleenkäytettävämpää ja monipuolisempaa. Näitä uusia ominaisuuksia ovat muun muassa näkyvyysmääreet (eng. access modifiers), abstraktit luokat ja rajapinnat, rakentajien ylikuormitus ja vain luku -ominaisuus.

TypeScriptiin siirtymisen JavaScriptistä on tarkoitettu olevan sulava, ja työssä löydettiin tekijöitä, jotka edistävät tätä tavoitetta. Tyyppitiedostot saattavat luoda vaikeuksia ohjelmoijalle, mutta kuitenkin staattisen tyypityksen, luokkien ja rajapintojen tuomat edut ohjelman ylläpidettävyydestä ja laadusta ovat suurempia. Asteittainen tyyppijärjestelmä ja staattisen tyypityksen tuoma mahdollinen tuottavuuden paraneminen tekevät siirtymästä vaivattomampaa ja aikansa arvoista etenkin suuremmissa ohjelmissa.

Avainsanat: TypeScript, JavaScript, tyypitys, ohjelman laatu, tyyppitiedosto, asteittainen tyyppijärjestelmä

SISÄLLYSLUETTELO

1. JOHDANTO

Verkkosivusto Stack Overflow'n kyselyn mukaan aloittelevista koodareista lähes 60 % käyttää JavaScriptiä, kun taas TypeScriptiä käyttää vain noin 15 % aloittelijoista. Ammattilaiskoodareilla TypeScriptin osuus on jo 40 %. TypeScript on myös neljänneksi rakastetuin ohjelmointikieli kyselyn mukaan. (*Stack Overflow Developer Survey, 2022*) JavaScriptiä voidaan siis pitää aloittelevien koodareiden ja pienten harrasteprojektien suosimana, mutta siirryttäessä isompiin ohjelmiin TypeScriptin suosio kasvaa merkittävästi.

TypeScript on suosituin ohjelmointikieli, joka kääntyy JavaScriptiksi. Modernit JavaScript-ohjelmat käyttävät paljon koodia, ja koska JavaScript on modulaarinen kieli, sovellus koostuu usein ensimmäisen ja kolmannen osapuolen paketeista. Koska ohjelma kääntyy JavaScriptiksi koodi voi olla yhdistelmä JavaScriptiä ja TypeScriptiä. Skaalautuvuus tarkoittaa enemmän koodia, enemmän koodareita ja enemmän laitteita. Mitä isompi yritys, sitä isompi projekti ja enemmän koodia. Kun koodia muutetaan niin koodi voi mennä rikki, jos ei ole täysin selvää mitä on tekemässä. Jos dokumentointi on huonoa, niin TypeScriptin vahva tyypitys auttaa asiassa. Suurin osa ajoajan virheistä JavaScriptissä on tyyppivirheitä ja TypeScriptin tehtävä oli eliminoida tyyppivirheet. (Black, 2020)

Tämän työn tavoitteena on selvittää, mitkä seikat tukevat ohjelmoijan siirtymistä JavaScriptistä TypeScriptiin ja mitkä eivät. Työssä käsitellään kirjallisuuskatsauksena TypeScriptin uusia ominaisuuksia ja mahdollisia haittoja ja hyötyjä, mitä siirtymisestä tulee. Hakuja on tehty Andor-järjestelmällä käyttäen eri hakusanoja ja -lauseita, muun muassa TypeScript, JavaScript, static typing ja software maintainability. Olen suodattanut hakuja myös rajaamalla hakuja aineistotyyppien mukaan. Aineistotyypeistä olen pois suodattanut kirjat ja videot, mutta jos ei ole löytynyt osuvia lähteitä olen myös ottanut kirjat mukaan. Lähteistä olen pyrkinyt ottamaan mahdollisimman uusia lähteitä. Muita lähteitä ovat myös erilaiset dokumentaatiot ja tilastot.

Seuraavassa luvussa esittelen hieman ohjelmointikielien taustaa. Kolmannessa luvussa käsitellään tyyppiannotaatioita. Neljännessä luvussa käyn läpi tyyppitiedostojen (eng. declaration file) haasteita. Viidennessä luvussa perehdytään luokkiin ja rajapintoihin ja niiden eroihin JavaScriptin ja TypeScriptin välillä. Kuudennessa luvussa on yhteenveto.

2. OHJELMOINTIKIELIEN TAUSTAA

Ohjelmointikieliä tulee uusia ja vanhoja kehitetään jatkuvasti. Ohjelmointikielen valinnalla on merkitystä, koska se voi vaikuttaa kehitysprosessiin, lopputuotteeseen sekä projektin tai ohjelman pitkän aikavälin ylläpitoon ja skaalautuvuuteen. On tärkeää harkita huolellisesti projektin vaatimuksia ja valita kieli, joka soveltuu hyvin tehtävään. Proseduraalinen ohjelmointi on perinteinen lähestymistapa, joka käyttää sarjaa komentoja tai toimintoja, joita kutsutaan peräkkäin. Tämä tapahtuu usein imperatiivisilla kielillä, joissa käytetään tilamuuttujia. (Dobhal, 2021) Proseduraalisessa ohjelmoinnissa tyyppimuunnoksia käytetään usein tietojen muuttamiseen ja muokkaamiseen suoraan tilamuuttujien kautta.

Funktionaalinen ohjelmointi sen sijaan pyrkii rajoittamaan muuttujien käyttöä ja korostaa funktionaalisten ohjelmointikonseptien käyttöä, kuten puhtaita funktioita ja korkeamman asteen funktioita. Funktionaalisessa ohjelmoinnissa tyyppimuunnoksia käytetään yleensä uusien tietotyyppien luomiseen ja yhdistämiseen, jotka ovat yleensä muuttumattomia. Tyyppimuunnokset voivat olla erityisen tärkeitä funktionaalisessa ohjelmoinnissa, koska funktionaalisten kielten tyyppijärjestelmät ovat yleensä vahvoja ja staattisia. (Dobhal, 2021) Tämä tarkoittaa sitä, että tyyppimuunnokset on määriteltävä tarkasti ja että tyyppimuunnosten käyttöön liittyy vähemmän epävarmuutta kuin proseduraalisessa ohjelmoinnissa.

Artikkelin Ray et al. (2017) laajassa tutkimuksessa tutkittiin eri ohjelmointikieliä ja koodin laatua GitHubissa. Tutkimuksen data implikoi, että funktionaaliset kielet ovat parempia kuin proseduraaliset kielet. Tämä viittaa siihen, että implisiittisen tyyppimuunnoksen kieltäminen on parempi kuin sen salliminen ja staattinen tyypitys on parempi kuin dynaaminen tyypitys. Seuraavaksi esittelen lyhyesti JavaScriptiä ja TypeScriptiä.

2.1 JavaScript

JavaScript on ajonaikaisen kääntämisen ohjelmointikieli. W3Techsin (2023) tilastojen mukaan JavaScriptiä käytetään 98,4 %:ssa nettisivuista selainpuolella. JavaScriptillä on kuitenkin muitakin käyttökohteita kuin selaimen selainpuolen-ohjelmointi, kuten palvelinpuolen-ohjelmointi Node.js:ssä, Apache CouchDB:ssä sekä muissa selainympäristön ulkopuolisissa ympäristöissä, kuten Adobe Acrobatissa. (*JavaScript | MDN*, 2023)

JavaScript on dynaamisesti tyypitetty ohjelmointikieli, mikä tarkoittaa, että muuttujien tyypit määräytyvät suorituksen aikana sen sijaan, että ne määriteltäisiin koodin kirjoitusvaiheessa. JavaScriptin dynaaminen tyypitys mahdollistaa joustavan ohjelmoinnin, koska

ohjelmoijan ei tarvitse huolehtia muuttujien tyypeistä tai niiden muuntamisesta suorituksen aikana. Muuttujat voivat myös muuttaa tyyppiään suorituksen aikana, mikä mahdollistaa esimerkiksi erilaisten operaatioiden suorittamisen samalla muuttujalla. JavaScriptin dynaamisuuden takia se onkin suosittu aloittelevien koodarien keskuudessa.

Kun käytetään JavaScriptiä, kolmannen osapuolen paketteja käytetään yleisesti. Näiden pakettien avulla kehittäjät voivat välttää pyörän keksimistä uudelleen ja löytää parempia ratkaisuja ongelmiin. Stack Overflow'n kyselyn mukaan kaikkien vastaajien suosituin kehittäjätyökalu on npm (Node Package Manager) (*Stack Overflow Developer Survey, 2022*). Npm on paketinhallintaohjelma, joka auttaa kehittäjiä hallitsemaan, jakamaan ja asentamaan koodipaketteja. Tämä ohjelma tulee asennettuna Node.js:n kanssa. (*What Is Npm*, 2023)

2.2 TypeScript

TypeScript on ohjelmointikieli, joka kääntyy JavaScriptiksi. TypeScriptin on luonut Microsoft, mutta Google on myös auttanut sen kehittämisessä (Black, 2020). TypeScript tuo mukanaan monia uusia ominaisuuksia JavaScriptiin verrattuna. TypeScript tarjoaa muun muassa luokkia, rajapintoja ja tyyppiannotaatioiden lisäämisen. TypeScriptiin siirtyminen JavaScriptistä on tarkoitettu olevan sulava, koska samat JavaScript kirjoitustavat ja periaatteet pätevät myös TypeScriptissä. (Bierman et al., 2014)

Yksi merkittävimmistä eroista JavaScriptiin verrattuna on TypeScriptin staattinen ja vahva tyypitys. Staattinen tyypitys tarkoittaa, että muuttujien ja funktioiden tyypit on määriteltävä ennen suoritusta, kun taas vahva tyypitys tarkoittaa, että tyyppejä tarkistetaan tiukasti eikä niitä muuteta implisiittisesti. TypeScriptissä voi kuitenkin olla myös dynaamisesti tyypitettyä koodia "any"-luokan avulla. Usein väitetään, että staattinen tyypitys parantaa sovelluksen ylläpitoa.

Myös TypeScriptissä käytetään kolmannen osapuolen paketteja yleisesti. Npm-työkalua käytetään TypeScriptissä hallitsemaan kolmannen osapuolen paketteja. Samoja JavaScript-kirjastoja voidaan käyttää myös TypeScriptissä, jos kirjastolle on määritelty tyyppitiedosto, missä kirjaston tyypitetyt rajapinnat kuvataan.

3. TYYPPIANNOTAATIOT

Yksi merkittävimmistä uusista ominaisuuksista TypeScriptissä on tyyppiannotaatioiden lisääminen. Staattinen tyypitys on tehokas työkalu, joka auttaa ohjelmoijaa ilmaisemaan oletuksiaan ongelmasta, jota he yrittävät ratkaista. Se antaa ohjelmoijalle mahdollisuuden kirjoittaa tiiviimpää koodia ja välttämään virheitä. Meijer (2004) kirjoittaa konferenssiartikkelissaan, että sen sijaan että hokisimme dynaamisesti ja staattisesti tyypiteltyjen kielten eroja, meidän olisi pyrittävä staattisen ja dynaamisen kielen rauhanomaiseen integrointiin samassa kielessä. Staattinen tyypitys silloin, kun se on mahdollista, dynaaminen tyypitys silloin, kun se on tarpeen. TypeScriptissä onkin juuri tämänlaista toimintaa tukeva asteittainen tyyppijärjestelmä.

3.1 Asteittainen tyyppijärjestelmä

Asteittainen tyyppijärjestelmä on tyyppijärjestelmä, jossa joillekin muuttujille ja lausekkeille voidaan antaa tyypit, ja tyypityksen oikeellisuus tarkistetaan kääntämisen yhteydessä (staattinen tyypitys), kun taas osa ohjelmasta voidaan jättää tyypittämättä ja mahdolliset tyyppivirheet raportoidaan ajon aikana (dynaaminen tyypitys). Asteittainen tyypitys antaa ohjelmistokehittäjille mahdollisuuden valita kumman tyyppiparadigman kokee sopivaksi yhden kielen sisällä.

TypeScript on esimerkki asteittaisesta tyyppijärjestelmästä. TypeScriptissä on tarkoitus käyttää staattista tyypitystä, mutta osa ohjelmasta voi olla dynaamisesti tyypitetty. Asteittainen tyypitys saavutetaan TypeScriptissä ”any”-luokalla. (Bierman et al., 2014) TypeScript suorittaa tyyppitarkistuksen kääntövaiheessa, kun taas JavaScript suorittaa sen ajonaikana. Tämä on yksi syy siihen, miksi JavaScriptillä ei ole parasta mainetta laadukkaiden ohjelmistojen tuottamisessa. JavaScriptin aloittelijoille suunnattu helppokäyttöisyys ja dynaaminen luonne ilman kääntäjää saavat jotkut ihmiset olettamaan, että JavaScriptin käyttö johtaa usein ohjelmiston huonoon laatuun. (Bogner & Merkel, 2022)

Asteittainen tyyppijärjestelmä mahdollistaa omalta osaltaan sulavamman siirtymisen TypeScriptiin JavaScriptistä. Ohjelmoija voi rauhassa alkaa käyttämään TypeScriptiä ilman, että joutuu staattisesti tyypittämään kaiken koodin. Myös kokonaisen ohjelman tai osan ohjelmasta, joka on kirjoitettu JavaScriptillä voi muuttaa vaivatta TypeScriptiksi muuttamatta kaikkea koodia.

3.2 Vaikutus ohjelman laatuun

Bognerin ja Merkelin (2022) konferenssijulkaisussa tullaan tulokseen, että TypeScript parantaa koodin laatua ja ymmärrettävyyttä, mutta yllättäen bugiherkkyys ja bugien ratkomiseen kuluva aika olivat molemmat pienempiä JavaScript-projekteissa, mikä on vastoin staattisen tyypityksen oletettuja etuja tällä alalla. Toki siitä ei seuraa, että JavaScriptin käyttö aiheuttaisi vähemmän bugeja kuin TypeScriptin käyttö. Julkaisussa osoitettiin vain, että JavaScript ja TypeScript-sovellusten suorassa vertailussa ja vastoin yleistä uskomusta TypeScript ei ollut merkittävästi yhteydessä vähäisempään virheiden määrään.

Hanenberg Stefanin et al. (2013) artikkelissa päästiin tulokseen, että staattinen tyypitys paransi sovelluksien ylläpitoa dokumentoimattoman koodin ymmärtämisen ja tyyppivirheiden korjaamisen suhteen. Ohjelmaa on helpompi ylläpitää ja päivittää kun ohjelma on staattisesti tyypitetty. Ohjelmoijan ei tarvitse muistaa tai katsoa eri tiedostoista päivittäessään koodia.

Koska JavaScript on dynaamisesti tyypitetty kieli, siinä on myös mahdollista tehdä automaattisia pakotettuja tyyppimuunnoksia. Näin ollen voisi olettaa, että niistä koituu virheitä. Pradel ja Koushik (2015) huomasivat konferenssiartikkelissaan, että useimmat pakkomuutokset ovat todennäköisesti kuitenkin vaarattomia (98,85 %).

TypeScriptin asteittainen tyyppijärjestelmä siis vaikuttaa positiivisesti ohjelman laatuun, monen eri empiirisen tutkimuksen mukaan. Aloitteleva TypeScript koodari voi kuitenkin helposti nojautua käyttämään ”any”-tyyppiä. Bognerin ja Merkelin (2022) konferenssijulkaisussa huomattiin myös, että ”any”-tyypin käytön vähentäminen näyttäisi olevan hyödyllistä ohjelman laadun suhteen.

3.3 Vaikutus ohjelmoijan tuottavuuteen

Viimeaikaiset empiiriset tutkimukset ovat näyttäneet, että staattinen ja dynaaminen tyypitys vaikuttaa positiivisesti ohjelmoijan tuottavuuteen. Moderneissa integroiduissa ohjelmointiympäristöissä (IDE) kuten Microsoft Visual Studiossa on monia ominaisuuksia, jotka auttavat ohjelmoijia olemaan tehokkaampia. Staattinen tyypitys mahdollistaa myös tehokkaamman IDE-tuen koodin navigointiin, automaattiseen täydentämiseen ja refaktorointiin (Kristensen & Møller 2017). Fischerin ja Hanenbergin (2016) artikkelin empiirisessä tutkimuksessa päädytään tulokseen, että TypeScriptin ja JavaScriptin välillä mitattiin suuri ja merkittävä ero tuottavuudessa, mutta koodin täydentämisen osalta voitiin osoittaa vain lähes merkitsevä, pieni vaikutus. On mahdollista, että tuloksiin vaikuttaa

se, että tyyppijärjestelmän vaikutus peittää koodin täydentämisen positiivisen vaikutuksen. Vaikka tämä väite on mahdollinen se merkitsisi silti sitä, että koodin täydentämisen vaikutus on pienempi kuin tyyppijärjestelmän vaikutus.

Hanenberg Stefanin et al. (2013) artikkelissa myös huomattiin, että sovelluskehittäjillä, jotka käyttivät dynaamista tyypitystä, oli tapana katsoa eri tiedostoja useammin tehdessään ohjelmointitehtäviä. Ohjelmoijat myös tekivät vähemmän koeajoja käyttäessään vahvaa tyypitystä. Tämä on myös merkki siitä, että vahvalla tyypityksellä voi potentiaalisesti olla vaikutus ohjelmoijan nopeuteen. Myös keskittyminen voi olla monella ohjelmoijalla helpompaa, kun ei tarvitse katsoa eri tiedostoja niin useasti.

4. TYYPPITIEDOSTO

Kristensen ja Møller (2017) kertovat konferenssiartikkelissaan, että voidakseen käyttää olemassa olevia JavaScript kirjastoja TypeScriptillä, täytyy tällaisten kirjastojen tyypitetyt rajapinnat kuvata erillisissä tyyppitiedostoissa. Näitä tiedostoja joudutaan kirjoittamaan ja ylläpitämään manuaaliseksi ja tämän takia niissä saattaa olla virheitä. Tämä voi luoda koodissa vääriä virheitä eli tilanteita, joissa kääntäjä ilmoittaa väärin käytetystä metodista tai muusta, vaikka virhe onkin tyyppitiedostossa. Manuaalinen ylläpito vaikeuttaa myös ylläpitämistä ja korjaamista.

Yleisin tapaus, jossa tyyppitiedostojen toimintaa on opeteltava on se, että käyttää npm-pakettia joissa ei ole tyyppejä (Documentation - Introduction, 2023). Ohjelmoijan on siis syytä tutustua tyyppitiedostoihin tai kirjoittaa ne osat koodia JavaScriptillä, joihin ei löydy TypeScriptiä tukevaa pakettia. Yksi apu tilanteeseen on kuitenkin DefinitelyTyped.

DefinitelyTyped on julkinen GitHub-tietovarasto tyyppitiedostoille. Tietovarastoa täyttävät ja ylläpitävät vapaaehtoiset manuaalisesti. Eroavaisuudet tyyppitiedoston ja JavaScript-toteutuksen välillä johtaa virheelliseen palautteeseen IDE:stä ja siten myös virheelliseen JavaScript-kirjaston käyttöön. (Cristiani & Thiemann, 2021) Joshua Hoeflich et al. (2022) kertovat artikkelissaan työkalusta Scotty, joka havaitsee DefinitelyTyped-tietovarastossa olevien tyyppien ja koodin väliset virheet. Scotty käsitteli 26 % DefinitelyTypedin 8006 paketista, joista 61 % koodi oli saatavilla ja jotka läpäisivät Scottyn testin. Ehkäpä ei ole yllättävää, että testien suorittaminen paljasti monia virheitä DefinitelyTypedissä. Vielä yllättävämpää on, että harjoitus johti sataan hyväksyttyyn pull request -pyyntöön, jotka korjaavat DefinitelyTypedin virheitä, mikä osoittaa tämän lähestymistavan arvon DefinitelyTypedin pitkän aikavälin ylläpidon kannalta.

Jos ei ole aikaa tai osaamista kirjoittaa itse manuaalisesti tyyppitiedostoja DefinitelyTyped on hyvä vaihtoehto. DefinitelyTypediä kehitetään ja parannellaan koko ajan, mutta se ei ole kuitenkaan täydellinen, joten tämä kannattaa pitää mielessä tehdessä valintaa.

TypeScriptiin siirryttäessä ohjelmoijalla on siis paljon vapauksia ja vaihtoehtoja tyyppitiedostojen ongelmien suhteen. Omaa harkintakykyä käyttäen on kuitenkin melko helppoa valita projektiin tai ohjelmaan sopivat vaihtoehdot. Suosituimmat TypeScript-kirjastot ovat kuitenkin lähtökohtaisesti hyvin määriteltyjä, ja niitä käyttäessä ei tarvitse huolestua tyyppitiedostoista.

5. LUOKAT JA RAJAPINNAT

TypeScriptissä luokat ja rajapinnat ovat olio-ohjelmoinnin tärkeitä ominaisuuksia, jotka auttavat järjestämään koodia, edistävät uudelleenkäytettävyyttä ja ylläpidettävyyttä sekä takaavat tyyppiturvallisuuden. Vaikka JavaScriptissä on myös luokkia, TypeScriptin luokat ja rajapinnat tarjoavat lisäominaisuuksia ja -etuja, kuten näkyvyysmääreet (eng. access modifiers), abstrakteja luokkia ja rajapintoja, rakentajien ylikuormituksia ja vain luku -ominaisuus. Tässä luvussa tarkastellaan TypeScriptin luokkien ja rajapintojen eroja JavaScriptin vastaaviin luokkiin verrattuna sekä sitä, miten niitä voidaan hyödyntää vankemman ja ylläpidettävämmän koodin kirjoittamiseen.

5.1 Luokat

Olio-ohjelmoinnissa luokat ovat tärkeä työkalu. Luokkien avulla organisoi koodia ja ne ovat tapa, jolla ajatella koodia. Luokat toimivat kapseloinnin ensisijaisena yksikkönä. (Cherny, 2019) Luokat lisättiin JavaScriptiin ES6 versioon vuonna 2015 (*JavaScript ES6,* 2023). JavaScript-luokat tukevat myös joitakin TypeScript-luokkien ominaisuuksia kuten periytymistä ja ylikirjoittamista.

Koska TypeScript on staattisesti tyypitetty kieli, voi luokilla olla ominaisuuksille ja metodeille eksplisiittiset tyypit, jotka voidaan tarkistaa kääntämisen yhteydessä. JavaScriptissä mahdolliset tyyppivirheet voidaan havaita vasta ajonaikana, jos erillisiä työkaluja ei ole käytössä. TypeScriptin luokat myös mahdollistavat näkyvyysmääreen muokkaamisen. Julkiset, yksityiset ja suojatut näkyvyysmääreet kontrolloivat ominaisuuksien ja metodien näkyvyyttä. (Cherny, 2019) Tämä auttaa varmistamaan kapseloinnin ja edistämään hyvää oliosuuntautunutta suunnittelua. JavaScriptissä ei ole näitä käyttöoikeusmuuttujia.

TypeScriptin luokilla voi olla rakentajien ylikuormituksia. TypeScript luokilla voi siis olla useampi kuin yksi rakentaja, joten jokainen eri rakentaja voi suorittaa eri tehtäviä ja niillä voi olla eri parametrejä. Tämä avaa ohjelmoijalle lisää vapautta ja tekee luokista monipuolisempia. TypeScript-luokissa voi asettaa myös vain luku avainsanan ominaisuuteen, jolloin sitä ei voida käyttää rakentajan ulkopuolella (*Documentation - Introduction*, 2023). Tämä auttaa varmistamaan muuttumattomuuden ja välttämään odottamattomat muutokset tietoihin.

TypeScript tukee myös abstrakteja luokkia ja rajapintoja, joiden avulla voidaan luoda yhteisiä rajapintoja tai käyttäytymistä, jotka voidaan jakaa luokkien kesken. Tämä edistää koodin uudelleenkäyttöä ja yksinkertaistaa ylläpitoa.

5.2 Rajapinnat

Rajapinnat mahdollistavat objektin tai funktion muodon ja tyypin, varmistaen että koodia käytettään oikeanlaisesti ja minimoiden virheet kääntämisaikana. Koodin organisoinnista tulee myös helpompaa, koska rajapinnat tarjoavat tavan määritellä sopimus, jota luokan tai funktion on noudatettava. Tämä edistää koodin organisointia ja ylläpidettävyyttä luomalla selkeän rajan komponenttien välille. Rajapinnat myös mahdollistavat paremman koodin uudelleenkäyttämisen, mikä tekee koodista selkeämpää.

Rajapinnat ovat vain TypeScriptin ominaisuus, joka on olemassa vain kääntämisaikana eikä tuota koodia, kun sovellus käännetään JavaScriptiksi (Cherny, 2019). Rajapinnat ovat tapa nimetä tyyppi niin, ettei sitä tarvitse määritellä rivissä. Rajapintoja on hyvä käyttää vastaavissa tilanteissa:

- Ominaisuuksien erityisrakenteen validointi
- Kun käytetään olioita parametreina
- Kun funktio palauttaa olioita
- Polymorfismissa

Ominaisuuksien eritysrakennetta validoidessa voidaan sisältöä tarkastella rajapinnan kautta. Rajapinta voi määrittää tiettyjä vaatimuksia elementeille, kuten tietyn tyypin tai määrän. Tämä voi auttaa varmistamaan, että elementit ovat oikeanlaisia. Kun olioita käytetään parametreina, rajapinta voi määrittää mitä tietoja olion tulee sisältää, jotta sitä voidaan käyttää parametrina. Samankaltainen tilanne on, kun funktio palauttaa olioita. Rajapinta auttaa varmistamaan, että funktio palauttaa oikeanlaista tietoja.

Polymorfismi on ohjelmoinnin käsite, joka viittaa siihen, että saman niminen funktio tai metodi voi toimia eri tavoin riippuen siitä, millainen olio sitä käyttää. Tämä tarkoittaa sitä, että ohjelmoija voi kirjoittaa yleisen toiminnon, joka toimii kaikille objekteille, mutta jokainen objekti voi toteuttaa kyseisen toiminnon omalla ainutlaatuisella tavallaan. Rajapinnat formalisoivat polymorfismin. Rajapintojen avulla voidaan määritellä polymorfismin deklaratiivisella tavalla, joka ei liity toteutukseen. Kaksi elementtiä on polymorfisia käyttäytymismallien suhteen, jos ne toteuttavat samat rajapinnat. Rajapinnat ovat keino, jolla voi todentaa, valvoa ja ilmaista polymorfismia. (*Polymorphism and Interfaces*, 2023)

6. YHTEENVETO

Tässä työssä selvitettiin kirjallisuuskatsauksena, mitkä seikat tukevat ohjelmoijan siirtymistä JavaScriptistä TypeScriptiin ja mitkä eivät. Tämä on ajankohtainen aihe, koska JavaScript on yksi suosituimmista ohjelmointikielistä aloittelevien koodaajien keskuudessa. TypeScript on taas suosittu ammattilaisten keskuudessa. JavaScriptistä TypeScriptiin siirtymisen on tarkoitus olla sulava.

TypeScript on hyvin samankaltainen kuin JavaScript. TypeScript on ohjelmointikieli, joka kääntyy JavaScriptiksi. Tässä työssä käsiteltiin ohjelmointikielien taustaa, tyyppiannotaatioita, tyyppitiedostoja, luokkia ja rajapintoja. Nämä seikat ovat keskeisimmässä osassa siirtymistä, sillä TypeScriptin koodin kirjoitustavat ovat kuitenkin lähes samanlaiset kuin JavaScriptissä.

Merkittävin ero JavaScriptiin TypeScriptissä on tyyppiannotaatioiden lisääminen ja asteittainen tyyppijärjestelmä. Asteittainen tyyppijärjestelmä mahdollistaa staattisesti ja dynaamisesti kirjoitettavan koodin yhden kielen sisällä. Usean eri empiirisen tutkimuksen mukaan staattisen tyypityksen on näytetty vaikuttavan positiivisesti ohjelman laatuun ja ylläpidettävyyteen. On myös näytteitä siitä, että staattinen tyypitys parantaisi ohjelmoijan tuottavuutta. Staattinen tyypitys tekee koodista virheettömämpää ja auttaa ohjelmoijaa ymmärtämään koodia paremmin, vaikka se olisi dokumentoimatonta. Staattinen tyypitys mahdollistaa myös vahvemman IDE:n tuen koodin navigointiin, automaattiseen täydentämiseen ja refaktorointiin.

TypeScriptissä voi käyttää JavaScript-kirjastoja, jos kirjastojen tyypitetyt rajapinnat on kuvattu erillisissä tyyppitiedostoissa. Tyyppitiedostot täytyy kuitenkin kirjoittaa ja ylläpitää manuaalisesti, mikä voi aiheuttaa virheitä ja vaikeuttaa ylläpitoa. DefinitelyTyped on julkinen GitHub-tietovarasto tyyppitiedostoille, mutta se ei ole täydellinen ja saattaa sisältää virheitä. Tyyppitiedostot voivat siis tuottaa ohjelmoijalle ongelmia, mutta suosituimmat TypeScript-kirjastot ovat kuitenkin yleensä hyvin määriteltyjä, joten niitä käyttäessä ei tarvitse huolehtia tyyppitiedostoista. Ohjelmoija voi myös siirtyä käyttämään tietyssä osassa ohjelmaa JavaScriptiä, jos tyyppitiedostoista koituu liikaa ongelmia.

TypeScriptissä uusina ominaisuuksina on rajapinnat ja uudistetut luokat. Vaikka JavaScriptissä on myös luokkia, TypeScriptin luokat ja rajapinnat tarjoavat lisäominaisuuksia ja -etuja, kuten näkyvyysmääreet, abstrakteja luokkia ja rajapintoja, rakentajien ylikuormituksia ja vain luku -ominaisuuksia. Näkyvyysmääreet auttavat varmistamaan kap-

seloinnin ja edistämään hyvää oliosuuntautunutta suunnittelua. Rakentajien ylikuormitukset avaavat ohjelmoijalle lisää vapautta ja tekee luokista monipuolisempia. TypeScript-luokissa voi asettaa myös vain luku avainsanan ominaisuuteen, jolloin sitä ei voida käyttää rakentajan ulkopuolella. Tämä auttaa varmistamaan muuttumattomuuden ja välttämään odottamattomat muutokset tietoihin. Rajapinnat varmistavat, että koodia käytettään oikeanlaisesti. Rajapinnat tarjoavat tavan määritellä sopimus, jota luokan tai funktion on noudatettava. Tämä edistää koodin organisointia ja ylläpidettävyyttä luomalla selkeän rajan komponenttien välille. Rajapinnat myös mahdollistavat paremman koodin uudelleenkäyttämisen.

TypeScriptiin siirtymisen JavaScriptistä on tarkoitettu olevan sulava, ja työssä löydettiin tekijöitä, jotka edistävät tätä tavoitetta. Tyyppitiedostot saattavat luoda vaikeuksia ohjelmoijalle, mutta kuitenkin staattisen tyypityksen, luokkien ja rajapintojen tuomat edut ohjelman ylläpidettävyydestä ja laadusta ovat suurempia. Asteittainen tyyppijärjestelmä mahdollistaa siirtymisen vaiheittain ja sulavasti. Myös staattisen tyypityksen tuoma mahdollinen tuottavuuden paraneminen tekevät siirtymisestä aikansa arvoista. Kaikki nämä edut korostuvat, mitä suurempi ohjelma tai projekti on. JavaScript voi edelleen olla parempi vaihtoehto pieniin projekteihin ja ohjelmiin, koska sillä on nopeampaa kirjoittaa lyhyitä pätkiä koodia sen dynaamisuuden vuoksi. Toki ohjelman kasvaessa suuremmaksi, sen pystyy muuttamaan jälkeenpäin vaiheittain TypeScriptiksi esimerkiksi tiedosto kerrallaan.

Tuloksia osittain rajoittavat saatavilla olevan aineiston vähyys. Erityisesti TypeScriptin luokista ja rajapinnoista ei juuri löytynyt lähteitä, joissa suoraan olisi tämän työn näkökulmasta asiaa käyty läpi. Yleinen kirjallisuus aiheesta kuitenkin vahvistaa väitteitä. Työn tuloksista voidaan saada vakuuttavampia jatkotutkimuksilla. Haastattelut tai tilastojen kerääminen ohjelmoijilta olisi hyvä keino, jolla tuottaa lisää tuloksia.

LÄHTEET

Bierman, G., Abadi, M., & Torgersen, M. (2014). Understanding TypeScript. Teoksessa R. Jones (Toim.), *ECOOP 2014 – Object-Oriented Programming* (ss. 257–281). Springer. https://doi.org/10.1007/978-3-662-44202-9_11

Black, N. (2020). Boris Cherny on TypeScript. *IEEE Software*, *37*(2), 98–100. https://doi.org/10.1109/MS.2019.2958155

Bogner, J., & Merkel, M. (2022). To Type or Not to Type? A Systematic Comparison of the Software Quality of JavaScript and TypeScript Applications on GitHub. *2022 IEEE/ACM 19th International Conference on Mining Software Repositories (MSR)*, 658–669. https://doi.org/10.1145/3524842.3528454

Cherny, B. (2019). *Programming TypeScript*. O'Reilly Media, Inc. https://learning.oreilly.com/library/view/programming-typescript/9781492037644/

Cristiani, F., & Thiemann, P. (2021). Generation of TypeScript declaration files from JavaScript code. *Proceedings of the 18th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes*, 97–112. https://doi.org/10.1145/3475738.3480941

Dobhal, R. (2021, helmikuuta 26). *Programming paradigm | programming paradigms*. https://codinghero.ai/2-basic-types-of-programming-paradigms-available-to-programmers/

Documentation—Introduction. Noudettu 22. helmikuuta 2023, osoitteesta https://www.typescriptlang.org/docs/handbook/declaration-files/introduction.html

Fischer, L., & Hanenberg, S. (2015). An empirical investigation of the effects of type systems and code completion on API usability using TypeScript and JavaScript in MS visual studio. *ACM SIGPLAN Notices*, *51*(2), 154–167. https://doi.org/10.1145/2936313.2816720

Hanenberg, S., Kleinschmager, S., Robbes, R., Tanter, É., & Stefik, A. (2014). An empirical study on the impact of static typing on software maintainability. *Empirical Software Engineering*, *19*(5), 1335–1382. https://doi.org/10.1007/s10664-013-9289-1

Hoeflich, J., Findler, R. B., & Serrano, M. (2022). Highly illogical, Kirk: Spotting type mismatches in the large despite broken contracts, unsound types, and too many linters. *Proceedings of the ACM on Programming Languages*, *6*(OOPSLA2), 142:479-142:504. https://doi.org/10.1145/3563305

JavaScript | MDN. (2023, helmikuuta 20). https://developer.mozilla.org/en-US/docs/Web/JavaScript

JavaScript ES6. Noudettu 6. maaliskuuta 2023, osoitteesta https://www.w3schools.com/js/js_es6.asp#mark_class

Kristensen, E. K., & Møller, A. (2017). Inference and Evolution of TypeScript Declaration Files. Teoksessa M. Huisman & J. Rubin (Toim.), *Fundamental Approaches to Software Engineering* (ss. 99–115). Springer. https://doi.org/10.1007/978-3-662-54494-5_6

Meijer, E., & Drayton, P. (2004, tammikuuta 1). *Static Typing Where Possible, Dynamic Typing When Needed: The End of the Cold War Between Programming Languages*.

Polymorphism and Interfaces. Noudettu 12. huhtikuuta 2023, osoitteesta https://www.cs.utexas.edu/~mitra/csSummer2013/cs312/lectures/interfaces.html

Pradel, M., & Sen, K. (2015). The Good, the Bad, and the Ugly: An Empirical Study of Implicit Type Conversions in JavaScript. Teoksessa J. T. Boyland (Toim.), *29th European Conference on Object-Oriented Programming (ECOOP 2015)* (Vsk. 37, ss. 519–541). Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. https://doi.org/10.4230/LIPIcs.ECOOP.2015.519

Ray, B., Posnett, D., Devanbu, P., & Filkov, V. (2017). A large-scale study of programming languages and code quality in GitHub. *Communications of the ACM*, *60*(10), 91–100. https://doi.org/10.1145/3126905

Stack Overflow Developer Survey 2022. Stack Overflow. Noudettu 22. helmikuuta 2023, osoitteesta https://survey.stackoverflow.co/2022/?utm_source=social-share&utm_medium=social&utm_campaign=dev-survey-2022

Usage Statistics of Client-side Programming Languages for Websites, March 2023. Noudettu 23. maaliskuuta 2023, osoitteesta https://w3techs.com/technologies/overview/client_side_language

What is npm. Noudettu 11. huhtikuuta 2023, osoitteesta https://www.w3schools.com/whatis/whatis_npm.asp