Samuli Mäkelä

Staattisen ja dynaamisen tyyppisysteemin vaikutus ohjelman kehitykseen

Tietotekniikan kandidaatintutkielma

15. toukokuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Samuli Mäkelä

Yhteystiedot: samjooma@student.jyu.fi

Ohjaaja: Timo Tiihonen

Työn nimi: Staattisen ja dynaamisen tyyppisysteemin vaikutus ohjelman kehitykseen

Title in English: Impact of static and dynamic type systems for software development

Työ: Kandidaatintutkielma

Sivumäärä: 13+0

Tiivistelmä: Yksi tärkeimmistä ohjelmointikielen osista on sen tyyppisysteemi, ja siitä, että minkälainen tyyppisysteemi on paras väitellään paljon. Tässä tutkielmassa käsitellään staattisen ja dynaamisen tyyppisysteemin vaikutusta ohjelmointinopeuteen ja ohjelmavirheiden tunnistamiseen. Tutkimuksissa löydettiin, että dynaamisilla kielillä ohjelmointi on nopeampaa vain pienempiä tai yksinkertaisempia ohjelmia tehdessä, ja muuten staattiset kielet ovat nopeampia tai yhtä nopeita kun dynaamiset kielet. Lisäksi löydettiin, että tyyppivirheitä saa staattisilla kielillä korjattua enemmän kuin dynaamisilla, mutta semanttisten virheiden korjaamisessa ei ole eroa.

Avainsanat: tyyppisysteemit, ohjelmointikielet, ohjelmankehitys

Abstract: On of the most important parts of a programming language is its type system, and there is much debate about which kind of type system is the best. This thesis examines the impact of static and dynamic type systems on programming speed and detection of programming errors. Studies found that programming with dynamic languages is only faster when making smaller or simpler programs, and otherwise static languages are faster or as fast as dynamic languages. It was also found that static languages can fix more type errors than dynamic languages, but for fixing semantic errors there is no difference.

Keywords: type systems, programming languages, software development

Sisällys

1 Johdanto

Yksi isoimmista eroista ohjelmointikielten välillä on niiden tyyppisysteemi, ja siitä, että minkälainen tyyppisysteemi on paras on paljon eriäviä mielipiteitä. Usein väitetään dynaamisilla kielillä ohjelmoinnin olevan nopeampaa kuin staattisilla ("Dynamically Typed Languages" 2008). Staattisten kielten taas väitetään olevan parempi ohjelmavirheiden (eli bugien) korjaamisessa (Bruce 2002, luku 1.3). Motivaationa tämän tutkielman tekemiseen oli se, että tyyppisysteemin valinnalle yleensä ei anneta tieteellisisä perusteluja.

Tässä tutkielmassa vertaillaan kirjallisuuskatsauksen muodossa staattisen ja dynaamisen tyyppisysteemin vaikutusta ohjelmien kehittämiseen. Ohjelmankehitystä käsitellään vertailemalla eroja ohjelmointinopeudessa ja ohjelmavirheiden tunnistamisessa.

Luvussa 2 ensin määritellään mitä tyyppisysteemit ovat. Ohjelmointinopeutta tarkastellaan luvussa 3 vertailemalla tutkimuskokeita, joissa mitattiin ohjelmointiaikaa eri tyyppisysteemeillä. Ohjelmavirheitä käsitellään luvussa 4 tarkastelemalla tutkimuksia, jotka analysoivat ohjelmavirheiden esiintymistä GitHubissa olevissa projekteissa eri tyyppisysteemiä käyttävillä kielillä. Lopuksi luvussa 5 tehdään yhteenveto tutkielmasta.

2 Tyyppisysteemit

Tyyppiteoria on laaja matematiikan ja logiikan ala, mutta ohjelmointikieliä käsitellessä tyyppisysteemillä tarkoitetaan paljon suppeampaa asiaa. On vaikeaa määritellä tyyppisysteemi niin, että määritelmä sisältää ohjelmointikielten suunnittelijoiden epämuodollisen tavan käyttää termiä ja, että määritelmä on tarpeeksi tarkka ollakseen hyödyllinen. Yksi mahdollinen määritelmä tyyppisysteemille on, että se on syntaktinen tapa luokitella lausekkeita sen mukaan minkälaisia arvoja ne laskevat, millä todistetaan ohjelman olevan käyttäytymättä tietynlaisilla tavoilla. (Pierce 2002, luku 1.1)

Käytännössä yksi tyyppisysteemin tärkeimmistä ominaisuuksista on varmistaa, ettei ohjelmassa tapahdu järjettömiä toimintoja, kuten kokonaisluvun jakamista merkkijonolla. Tyyppisysteemit voidaan luokitella staattisiin ja dynaamisiin systeemeihin. Dynaamisessa tyyppisysteemissä ohjelman tyyppitarkastus tapahtuu ohjelman ajon aikana, ja staattisessa tyyppisysteemissä tyyppitarkastus tapahtuu ennen ohjelman ajoa, yleensä ohjelmaa käännettäessä. (Bruce 2002, luku 1.2)

Sen lisäksi, että tyyppisysteemi voi olla staattinen tai dynaaminen, voi se olla myös vahva tai heikko. Vahvan ja heikon tyyppisysteemin ero on se, että voiko ohjelmassa tapahtua automaattisia tyyppimuunnoksia. Vahvasti tyypitetyssä kielessä tyyppimuunnokset tapahtuvat vain jos ohjelmoija erikseen niin käskee. Heikosti tyypitetyssä kielessä muuttujan tyyppi voi muuntua toiseksi automaattisesti, esimerkiksi merkkijono voisi muuntua automaattisesti kokonaisluvuksi.

3 Tyyppisysteemin vaikutus ohjelmoinnin nopeuteen

Yleinen argumentti dynaamisten kielten puolesta on se, että niillä ohjelmointi on nopeampaa (“Dynamically Typed Languages” 2008). Tyyppisysteemien ohjelmointinopeuteen liittyen on tehty tutkimuskokeita, joissa mitattiin paljon aikaa kului ohjelmointitehtävien tekemiseen staattisella ja dynaamisella kielellä. Tässä luvussa käsitellään näiden tutkimusten tuloksia.

3.1 Ohjelmointinopeus

Joidenkin tutkimusten mukaan ohjelmointiaika eri tyyppisysteemeillä riippuu siitä minkälaista ohjelmointitehtävää tehtiin. Stuchlik ja Hanenberg (2011) sanoivat tutkimuksessaan, että ohjelmointiajassa ei huomattu eroa tyyppisysteemien välillä, kun tehtiin ohjelmointitehtäviä, jotka staattisella kielellä vaativat paljon tyyppimuunnoksia. Dynaaminen kieli oli nopeampi vain tehtävissä, jotka heidän mukaansa olivat triviaaleja. Mayer ym. (2012) väittivät, että dynaamisilla kielillä on nopeampaa ohjelmoida helpompia ohjelmointitehtäviä. He kuitenkin myönsivät, että on vaikea määritellä mitä ovat helpot ja vaikeat ohjelmointitehtävät. Hanenberg (2010) ei löytänyt tutkimuksessaan ohjelmointitehtävän suorittamisen kokonaisajassa eroa staattisen ja dynaamisen kielen välillä. Tutkimuksessa kuitenkin huomattiin tehtävän pienemmän osa-alueen suorittamisen olevan nopeampaa dynaamisella kielellä, ja staattisen kielen saavan kirittyä aikaa koko tehtävän suorituksessa.

On myös tutkimuksia, joissa dynaamisella systeemillä ei esiintynyt ollenkaan etua ohjelmointinopeudelle. Tutkijoiden Fischer ja Hanenberg (2015) sekä Hanenberg ym. (2014) tekemien tutkimusten mukaan staattisella tyyppisysteemillä ohjelmointitehtävien suorittaminen oli nopeampaa. Hanenberg ym. (2014) tekemän tutkimuksen mukaan staattisella kielellä oli aina nopeampaa tai yhtä nopeaa ohjelmoida kuin dynaamisella kielellä.

3.2 Ohjelmoinnin nopeuteen vaikuttavat tekijät

Edellä käsiteltiin tyyppisysteemien vaikutusta yleisesti ohjelmoinnin nopeuteen, mutta siihen voi vaikuttaa monia eri tekijöitä. Yksi mahdollinen tekijä on ohjelmoitaessa käytettävän

API:n dokumentaatio. Staattisilla kielillä väitetään olevan se etu, että niiden dokumentaatio on parempaa, koska koodin tyyppinimet automaattisesti selventävät koodin käyttötarkoitusta. Seuraavaksi käsitellään tutkimuksia tähän liittyen.

Hanenberg ym. (2014) vertasivat eri tyyppisysteemien nopeutta, kun ohjelmoija käyttää tuntematonta ja dokumentoimatonta API:tä. Tutkimuksen mukaan staattinen tyyppisysteemi oli nopeampi. Endrikat ym. (2014) sanoivat, että ohjelmointi on nopeampaa molemmilla tyyppisysteemeillä kun API on hyvin dokumentoitu, mutta staattinen systeemi hyötyy siitä enemmän kuin dynaaminen. Molemmissa näistä tutkimuksissa väitettiin staattisen systeemin edun johtuvan siitä, että sillä on helpompaa ymmärtää koodin toimintaa ja käyttötapaa.

Spiza ja Hanenberg (2014) löysivät tutkimuksessaan, että tyyppinimien esittäminen API:ssä olisi hyödyllistä myös dynaamisissa kielissä. Tutkimuksen mukaan ohjelmointiin kului vähemmän aikaa, kun API:n dokumentaatioon oli lisätty tyyppinimet näkyviin, vaikkei kielessä ollutkaan staattista tyyppitarkastusta. He huomauttivat kuitenkin, että ilman tyyppitarkastusta tyyppinimet voivat olla vääriä, mikä hidastaa ohjelmointia.

Edellisten tutkimusten perusteella, ja kuten Uesbeck (2019, luku 3.5) sanoo väitöskirjassaan, voidaan päätellä staattisen tyyppisysteemin ohjelmointinopeuden edun johtuvan siitä, että koodin ymmärtäminen on helpompaa koodissa esiintyvien tyyppinimien ansiosta. Tätä väitettä tukee myös Mayer ym. (2012) tutkimuksessaan. He uskovat, että staattisen tyyppisysteemin etu monimutkaisemmissa ohjelmointitehtävissä johtuu siitä, että staattinen systeemi kannustaa ohjelmoijaa lukemaan luokkien määritelmiä. Dynaamisessa systeemissä koodin lukeminen on vähemmän systemaattista, mikä johtaa suurempaan koodin ja eri tiedostojen selaamisen määrään.

Yksi asia, joka myös voi vaikuttaa ohjelmoinnin nopeuteen on IDE:n automaattinen koodintäydennys. Fischer ja Hanenberg (2015) vertasivat tutkimuksessaan kuinka paljon IDE:n koodintäydennys vaikuttaa ohjelmointinopeuteen verrattuna tyyppisysteemin vaikutukseen. Tutkimuksen johdannossa he pohdiskelevat, että koodintäydennys voisi pienentää ohjelmointinopeuden eroa staattisten ja dynaamisten kielten välillä, koska se vähentää tarvetta API:n dokumentaation selaamiselle. Tämä ei kuitenkaan vaikuta pitävän paikkaansa, sillä tutkimuksen mukaan koodintäydennyksen vaikutus ohjelmointinopeuteen oli hyvin pieni.

3.3 Yhteenveto

Tässä osiossa tarkasteltujen tutkimusten perusteella dynaamisesti tyypitetyillä ohjelmointikielillä ohjelmointi on nopeampaa vain tietyissä tapauksissa. Monissa tutkimuksissa esiintyi etu staattisella systeemillä eikä dynaamisella. Silloinkin kun dynaamisella systeemillä oli etu, niin se oli vain yksinkertaisemmissa tai pienemmissä ohjelmointitehtävissä.

Lisäksi huomattiin, että API:den dokumentaation olemassaolo on yksi tekijä ohjelmoinnin nopeudelle, ja se hyödyttää staattisia kieliä enemmän kuin dynaamisia. Staattisen tyyppisysteemin etu vaikuttaisi johtuvan siitä, että koodissa esiintyvien tyyppinimien ansiosta koodin ymmärtäminen on helpompaa. Tarkasteltiin myös IDE:n koodintäydennyksen vaikutusta ohjelmointinopeuteen, mutta sen vaikutus on pieni.

4 Ohjelmavirheiden tunnistaminen ja korjaaminen

Staattisen tyyppisysteemin väitetään olevan parempi ohjelmavirheiden löytämisessä ja korjaamisessa (Bruce 2002, luku 1.3). Dynaamisella systeemillä monet virheet täytyisi huomata itse ohjelmaa tehdessä tai vasta sen aiheuttaessa virheilmoituksen ohjelmaa ajettaessa. Tässä luvussa tutkitaan sitä, että kuinka suuri etu staattisella tyyppisysteemillä oikeasti on ohjelmavirheiden tunnistamisessa ja korjaamisessa.

Gao, Bird ja Barr (2017) etsivät dynaamisesti tyypitetyn JavaScript kielen GitHub projekteista committeja, jotka sisälsivät ohjelmavirheiden korjauksia, ja kokeilivat kuinka paljon virheistä olisi automaattisesti löydetty staattisen tyyppitarkastajan avulla. Tyyppitarkastaja tunnisti 15 % ohjelmavirheistä. Khan ym. (2022) tekivät samanlaisen tutkimuksen Python kielellä, jossa he myös tunnistivat staattisella tyyppitarkastajalla 15 % ohjelmavirheistä. Näiden tutkimusten perusteella nähdään, että dynaamisella tyyppisysteemillä esiintyy enemmän ohjelmavirheitä kuin staattisella.

Tutkijat Ray ym. (2017) analysoivat tutkimuksessaan GitHub projekteja ja vertasivat eri kielissä esiintyvien ohjelmavirheiden määrää. Heidän mukaansa staattisilla kielillä esiintyy vähemmän ohjelmavirheitä kuin dynaamisella. Tämä tutkimus ei välttämättä pidä paikkaansa, sillä Berger ym. (2019) tekemän jäljennöstutkimuksen mukaan alkuperäinen tutkimus on harhaanjohtava. Alkuperäisessä tutkimuksessa löydettiin yhdelletoista kielelle assosiaatio ohjelmavirheiden kanssa, mutta jäljennöstutkimuksessa vain neljälle, ja niilläkin se oli erittäin pieni.

Harlin, Washizaki ja Fukazawa (2017) tutkivat ohjelmavirheiden korjaamista eri tyyppisysteemeillä. Tutkimuksen ohjelmointitehtävissä staattisella tyyppisysteemillä sai korjattua enemmän ohjelmavirheitä kuin dynaamisella. Staattisella systeemillä on siis etu tyyppivirheiden korjaamisessa, mutta onko sillä vaikutusta muunlaisten virheiden korjaamiseen? Hanenberg ym. (2014) kokeilivat tutkimuksesaan miten tyyppisysteemit eroavat tyyppivirheiden ja semanttisten virheiden korjaamisessa. Semanttisella virheellä tarkoitetaan virhettä ohjelman logiikassa. Tutkimuksen mukaan staattisella tyyppisysteemillä on etu tyyppivirheiden korjaamisessa, mutta ei semanttisten virheiden korjaamisessa.

5 Yhteenveto

Tässä tutkielmassa vertailtiin ohjelmointikielten staattisen ja dynaamisen tyyppisysteemin vaikutusta ohjelmankehitykseen. Tämä tehtiin kirjallisuuskatsauksena eri tyyppisysteemien ohjelmointinopeutta vertailevista tutkimuksista ja ohjelmavirheiden tunnistamista ja korjaamista vertailevista tutkimuksista. Tutkimukset ohjelmointinopeuteen liittyen olivat kontrolloituja kokeita, joissa verrattiin ohjelmointitehtävien tekemiseen kulunutta aikaa eri tyyppisysteemiä käyttävillä ohjelmointikielillä. Tutkimukset ohjelmavirheistä olivat pääasiassa toteutettu analysoimalla GitHub projekteissa esiintyviä ohjelmavirheitä eri tyyppisysteemeillä.

Ohjelmointinopeutta tarkasteltaessa löydettiin, että dynaamisella tyyppisysteemillä on nopeampaa ohjelmoida pienempiä tai yksinkertaisempia ohjelmia, mutta muuten ohjelmointi on nopeampaa tai yhtä nopeaa staattisella systeemillä. Huomattiin myös, että ohjelmointi on nopeampaa jos ohjelmoitaessa käytettävillä API:llä on olemassa dokumentaatiota, ja staattinen systeemi hyötyy dokumentaatiosta enemmän kuin dynaaminen. Syy staattisen tyyppisysteemin ohjelmointinopeuden edulle väitetään olevan se, että koodissa esiintyvien tyyppinimien ansiosta koodin ymmärtäminen on helpompaa, mikä vähentää tarvetta koodin selailulle.

Ohjelmavirheitä käsiteltäessä huomattiin, että staattisella tyyppisysteemillä saa tunnistettua enemmän ohjelmavirheitä kuin dynaamisella systeemillä. Dynaamisilla kielillä tehdyistä projekteista oltaisiin voitu tunnistaa niissä esiintyvistä ohjelmavirheistä 15 % automaattisesti, jos oltaisiin käytetty staattista tyyppitarkastajaa. Staattinen tyyppisysteemi ei kuitenkaan auta semanttisten virheiden korjaamisessa, vaan pelkästään tyyppivirheiden.

Lähteet

Berger, Emery D., Celeste Hollenbeck, Petr Maj, Olga Vitek ja Jan Vitek. 2019. “On the Impact of Programming Languages on Code Quality: A Reproduction Study”. *ACM Trans. Program. Lang. Syst.* (New York, NY, USA) 41, numero 4 (lokakuu). ISSN: 0164-0925. https://doi.org/10.1145/3340571.

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

Endrikat, Stefan, Stefan Hanenberg, Romain Robbes ja Andreas Stefik. 2014. “How Do API Documentation and Static Typing Affect API Usability?” Teoksessa *Proceedings of the 36th International Conference on Software Engineering,* 632–642. ICSE 2014. Hyderabad, India: Association for Computing Machinery. ISBN: 9781450327565. https://doi.org/10.1145/2568225.2568299.

Fischer, Lars ja Stefan Hanenberg. 2015. “An Empirical Investigation of the Effects of Type Systems and Code Completion on API Usability Using TypeScript and JavaScript in MS Visual Studio”. *SIGPLAN Not.* (New York, NY, USA) 51, numero 2 (lokakuu): 154–167. ISSN: 0362-1340. https://doi.org/10.1145/2936313.2816720.

Gao, Zheng, Christian Bird ja Earl T. Barr. 2017. “To Type or Not to Type: Quantifying Detectable Bugs in JavaScript”. Teoksessa *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE),* 758–769. https://doi.org/10.1109/ICSE.2017.75.

Hanenberg, Stefan. 2010. “An Experiment about Static and Dynamic Type Systems: Doubts about the Positive Impact of Static Type Systems on Development Time”. *SIGPLAN Not.* (New York, NY, USA) 45, numero 10 (lokakuu): 22–35. ISSN: 0362-1340. https://doi.org/10.1145/1932682.1869462.

Hanenberg, Stefan, Sebastian Kleinschmager, Romain Robbes, Éric Tanter ja Andreas Stefik. 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.

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

“Dynamically Typed Languages”. 2008. *IEEE Software* 25 (2): 7–10. https://doi.org/10.1109/MS.2008.35.

Khan, Faizan, Boqi Chen, Daniel Varro ja Shane McIntosh. 2022. “An Empirical Study of Type-Related Defects in Python Projects”. *IEEE Transactions on Software Engineering* 48 (8): 3145–3158. https://doi.org/10.1109/TSE.2021.3082068.

Mayer, Clemens, Stefan Hanenberg, Romain Robbes, Éric Tanter ja Andreas Stefik. 2012. “An Empirical Study of the Influence of Static Type Systems on the Usability of Undocumented Software”. *SIGPLAN Not.* (New York, NY, USA) 47, numero 10 (lokakuu): 683–702. ISSN: 0362-1340. https://doi.org/10.1145/2398857.2384666.

Pierce, Benjamin C. 2002. *Types and programming languages.* MIT press.

Ray, Baishakhi, Daryl Posnett, Premkumar Devanbu ja Vladimir Filkov. 2017. “A Large-Scale Study of Programming Languages and Code Quality in GitHub”. *Commun. ACM* (New York, NY, USA) 60, numero 10 (syyskuu): 91–100. ISSN: 0001-0782. https://doi.org/10.1145/3126905.

Spiza, Samuel ja Stefan Hanenberg. 2014. “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,* 99–108. MODULARITY ’14. Lugano, Switzerland: Association for Computing Machinery. ISBN: 9781450327725. https://doi.org/10.1145/2577080.2577098.

Stuchlik, Andreas ja Stefan Hanenberg. 2011. “Static vs. Dynamic Type Systems: An Empirical Study about the Relationship between Type Casts and Development Time”. *SIGPLAN Not.* (New York, NY, USA) 47, numero 2 (lokakuu): 97–106. ISSN: 0362-1340. https://doi.org/10.1145/2168696.2047861.

Uesbeck, Phillip Merlin. 2019. “On the Human Factors Impact of Polyglot Programming on Programmer Productivity”. Tohtorinväitöskirja, University of Nevada, Las Vegas.