Samu Peltonen

Ohjelmointikielet sovelluskehityksessä 2020-luvulla -systemaattinen kirjallisuuskatsaus

Tietotekniikan pro gradu -tutkielma

2. joulukuuta 2021

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Samu Peltonen

Yhteystiedot: samu.m.peltonen@student.jyu.fi

Ohjaaja: Ville Tirronen, Vesa Lappalainen

Työn nimi: Ohjelmointikielet sovelluskehityksessä 2020-luvulla -systemaattinen kirjallisuuskatsaus

Title in English: Programming languages in application development in 2020s -a systematic literature review

Työ: Pro gradu -tutkielma

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Sivumäärä: 69+15

Tiivistelmä: Mitä kieliä sovelluskehityksessä käytetään nyt, mitä kannattaa opetella lähitulevaisuutta silmällä pitäen ja miksi? Nämä kysymykset toistuvat eri muodoissa vuodesta toiseen. Tässä tutkimuksessa pyritään selvittämään alkaneella vuosikymmenellä käytettäviä ohjelmointikieliä sovelluskehityksessä ja syitä kielien menestymiselle. Vastaukset saavutetaan purkamalla ja tarkastelemalla ohjelmointikielten elinkaaren vaiheita, selvittämällä tilastoista käyttö- ja hakumäärien kehitystä, keräämällä ammattilaisten lausuntoja, tutkimalla kielten historiaa, sekä systemaattisen kirjallisuuskatsauksen keinoin selvittämällä tuoreimpia sovelluskehityksen ilmiöitä. Tutkimus tarjoaa samalla myös tiiviin katsauksen suosituimpiin ohjelmointikieliin. Tuloksista voidaan havaita, että tilastoissa näkyvä kehitys jatkunee samankaltaisena myös lähitulevaisuudessa ja esimerkiksi kielien ympärille muodostuneiden yhteisöjen ja kirjoitetun koodin määrän vaikutus käyttömäärältään laskussa olevissa kielissä on avainasemassa kielien käytettynä pysymisessä.

Avainsanat: ohjelmointikielet, sovelluskehitys, gradu, systemaattinen kirjallisuuskatsaus

Abstract: What are the languages being used in the application development currently, what is worth learning for the near future and why? These questions recur in different forms one year after another. This study aims to find out the languages that will be used during this

decade in application development, and the reasons behind the popularity of those languages. Results will be acquired by deconstructing and examining different stages in a programming language's lifespan, studying search and usage rates from statistics, gathering reports from professionals, studying history of programming languages and researching newest trends in application development by means of systematic literature review. Thesis also provides a compact overview of the most popular programming languages. Results show that trajectories shown in the statistics won't probably change in the near future and for example the communities and the amount of written code are very significant factors in keeping declining languages still relevant and used.

Keywords: programming languages, application development, Masters Theses, systematic literature review

Kuviot

Sisältö

1 Johdanto

Yksittäisen ohjelmointikielen suosiota sovelluskehityksessä voidaan mitata monella tavalla, kuten hakukonehakujen, kehittäjien määrän, avoimen lähdekoodin latausten tai avoimien työpaikkojen määriä mittaamalla, eikä ole yleistä määritelmää siitä minkä rajan ylittävä kieli on merkittävä tai suosittu ohjelmointikieli. Ohjelmointikieliä voidaan vertailla keskenään eri näkökulmista. Joillakin osa-alueilla voidaan suoraan osoittaa toisen ohjelmointikielen olevan parempi kuin toisen. Toisilla osa-alueilla paremmuus on tulkinnanvaraisempaa, riippuu käyttäjästä, tai on mahdotonta määrittää.

Paremmuus jollakin osa-alueella ei kuitenkaan välttämättä näy kielen käyttömäärässä toiseen kieleen verrattuna ja osa-alueita on paljon (Nanz ja Furia 2015), mikä tekee aiheen tutkimisesta mielenkiintoista. Tämän tutkimuksen tarkoituksena on selvittää, mitkä näistä asioista todellisuudessa vaikuttavat ohjelmointikielien käyttömäärään ja suosioon sovelluskehityksessä, sekä selvitetään alkaneen vuosikymmenen vallitsevia ohjelmointikieliä eri sovelluskehityksen osa-alueilla. Tutkimuksessa pyritään tutkimaan siis sitä, mitä kieliä lähitulevaisuudessa käytetään ja miksi, sen sijaan että erilaisia ominaisuuksia vertailemalla pyrittäisiin itse osoittamaan paras kieli eri tarpeisiin. Osittain nämä kaksi aluetta saattavat sivuta toisiaan, sillä ei ole täyttä varmuutta siitä mitkä ohjelmointikielten ominaisuuksista tulevat vaikuttamaan käyttöönottoon johtaviin päätöksiin. Lisäksi tutkimuksessa kartoitetaan miltä ohjelmointikielien kenttä näyttää nyt ja lähimenneisyydessä, jotta voidaan ymmärtää paremmin tulevaisuutta.

Tämä tutkimus on rajattu sovelluskehitykseen, mikä itsessään ei ole tarkasti määritelty käsite, mutta jolla tästä tutkimuksesta rajataan laajasta ohjelmistokehityksen käsitteestä joitakin osa-alueita. Sovelluskehitys on rajauksesta huolimatta varsin laaja alue, minkä takia aihetta tulee jakaa pienempiin osiin. Tässä tutkimuksessa aihetta jaetaan ohjelmointikielten elinkaaren vaiheiden mukaan ja systemaattisen kirjallisuuskatsauksen osuudessa ilmiöitä kuvaavien kategorioiden avulla, riittävän rakenteisuuden saavuttamiseksi.

Tutkimuksesta saatavilla tuloksilla oppilaitokset voivat havaita nopeammin tarpeet uusien kielien lisäämiseen kurssivalikoimaan ja yritykset voivat havaita kannattavia uudistamisen

kohteita. Lisäksi tutkielma kokonaisuudessaan tarjoaa esimerkiksi aloitteleville tietotekniikan opiskelijoille tiiviin, mutta riittävän kattavan katsauksen erilaisiin ohjelmointikieliin, niiden historiaan ja niiden järkevään vertailuun, sekä kieliin liittyviin ajankohtaisiin ilmiöihin. Tutkimus on tärkeää myös siksi, että suuri osa olemassaolevasta tutkimuksesta perustuu pelkästään eri metriikoista johdettujen käyttömäärien graafien seuraamiseen ja tilastotieteen keinoin tulevaisuuden ennustamiseen. Näiden lisäksi tutkimus on ajankohtaista, eikä tälle ajanjaksolle vaikuta olevan muuta laajaa ja tutkimuksia kokoavaa kirjallisuutta.

Tutkimuksen päämäärät saavutetaan selvittämällä ohjelmointikielien käytön syitä tutkimuskirjallisuudesta ja kehittäjien lausunnoista, kartoittamalla tämänhetkistä tilannetta erilaisista tilastoista ja kuvaajista, sekä tutkimalla ohjelmointikielten lähitulevaisuutta koskevia tieteellisiä julkaisuja systemaattisen kirjallisuuskatsauksen keinoin. Lisäksi pyritään ottamaan huomioon uudet ja syntyvät sovelluskehityksen alueet ja niissä käytettävät ohjelmointikielet.

Tutkielman luvussa 2 selvitetään aiemman tutkimuskirjallisuuden kautta ohjelmointikielten suosioon vaikuttavia asioita kielien eri vaiheissa. Luvussa 3 kartoitetaan tämän hetken ja lähimenneisyyden ohjelmointikieliä tilastoja ja ammattilaisten lausuntoja tutkimalla. Luvussa 4 käydään läpi suoritettu systemaattinen kirjallisuuskatsaus. Luvussa 5 käydään läpi systemaattisesta kirjallisuuskatsauksesta saadut tulokset. Luvussa 6 vedetään tutkimuksen tulosten ja muun kerätyn aineiston pohjalta johtopäätöksiä ohjelmointikielten lähitulevaisuudesta ja pohditaan tutkimukseen liittyviä muuttujia.

2 Ohjelmointikielien suosio ja sen mittaaminen aiemmissa tutkimuksissa

Tässä luvussa selvitämme ohjelmointikielten suosion mittaamiseen liittyviä seikkoja ja kielten ryhmittelyä vertailun saavuttamiseksi, sekä suosioon vaikuttavia asioita eri ryhmien sisällä.

2.1 Mittausteknisiä kysymyksiä

Ensimmäinen ratkaistava asia ohjelmointikielten keskinäistä asemaa ja suosiota vertaillessa on määrittää, mitä metriikoita voidaan käyttää suosion mittaamiseen, sekä missä vaiheessa ohjelmointikieli ylittää jonkin rajan, jonka jälkeen sitä voidaan kutsua suosituksi. Erilaiset käyttömäärät ja hakukoneiden hakumäärät ovat keskenään vertailtavissa ja usein helposti kerättäviä metriikoita, joilla saadaan selville ohjelmointikielien suosiota. Kun tilastoja saadaan useasta lähteestä, voidaan olla varmempia tulosten objektiivisuudesta, joten tässä tutkimuksessa pyritään yhdistelemään riittävän laajasti erilaisia tilastoja eri toimijoilta. Näiden tilastojen laaja saatavuus on siis avainasemassa siinä, että saadaan mahdollisimman kokonaisvaltainen ja puolueeton kuva tarkasteltavasta aiheesta, esimerkiksi yksittäisiin asiantuntijalausuntoihin verrattuna.

Toinen ratkaistava ongelma on lajitella ohjelmointikielet jonkin periaatteen mukaan pienempiin, vertailukelpoisiin ryhmiin. Lajittelun tulee palvella mahdollisimman hyvin alkuperäisen ongelman, eli lähitulevaisuuden sovelluskehityksen ohjelmointikielten selvittämisen ratkaisemista. Lajittelu mahdollistaa samankaltaisuuksien havaitsemista ja johtopäätösten vetämistä ryhmän sisällä. On myös toivottavaa että lajittelulla saavutetaan mahdollisimman vähän päällekkäisyyttä, eli ohjelmointikielet eivät kuuluisi useaan ryhmään. Lisäksi mahdollisimman yksiselitteiset määritelmät lokeroille selkeyttävät jakamista. Esimerkiksi ohjelmointiparadigmojen käyttäminen lajittelussa on hankalaa, sillä paradigmoilla ei ole tarkkoja määritelmiä ja nykyisin moniparadigmaiset ohjelmointikielet ovat melko yleisiä (Nanz ja Furia 2015).

Tämän tutkimuksen luvussa kolme ohjelmointikielien ja niiden trendien kartoittamisessa on päädytty jaottelemaan kielet niiden tämänhetkisen aseman mukaan. Pääpiirteittäin kielet on jaettu nouseviin, laskeviin ja paljon käytettyihin monikäyttöisiin ohjelmointikieliin. Jaottelu on suoritettu useasta seuraavassa luvussa mainitusta lähteestä kerättyjen tilastojen perusteella. Tähän valintaan on päädytty ryhmien välisen päällekkäisyyden vähäisyyden ja jakamisessa käytettävien tilastojen laajan saatavuuden takia. Lisäksi muodostuneiden ryhmien sisällä kielet ovat samankaltaisia ja hyvin vertailtavissa keskenään. Chatley, Donaldson ja Mycroft (2019) ovat omassa tutkimuksessaan suorittaneet pääpiirteittäin samanlaisen jaon. Lisää tuloksista ja metodologiasta on kirjoitettu luvussa 3.

2.2 Ohjelmointikielten suosio ja sen muuttuminen

Kun yritetään selvittää mikä tekee sovelluskehityksen ohjelmointikielestä suositun, voidaan kysymystä purkaa pienempiin osiin sen tutkimisen helpottamiseksi. Puramme seuraavaksi ohjelmointikielen suosion sen elinkaaren vaiheisiin, joita lähdemme avaamaan aiemman tutkimuskirjallisuuden kautta, jotta saamme yksityiskohtaisemman käsityksen siitä, mitkä osatekijät suosioon vaikuttavat eri vaiheissa.

2.2.1 Miksi uusia ohjelmointikieliä luodaan ja miksi ne nousevat suosituksi?

Ohjelmointikielien syntyminen ei välttämättä aina ole pitkään suunniteltu prosessi (esimerkkinä esoteeriset ohjelmointikielet, jotka saattavat olla huumorilla kehitettyjä Turing-täydellisyyteen pyrkiviä kieliä), mutta kuten myöhemmin luvussa kolme havaitaan, menestyvät ohjelmointikielet näyttävät syntyvän johonkin käytännön tarpeeseen. Riippuen siitä mitä ohjelmointikielen menestyksellä tarkoitetaan, saattaa kestää vuosikymmeniä että ohjelmointikieli saavuttaa menestyneen kielen aseman, jonka myös havaitsemme myöhemmin kolmannen luvun tilastoista. Tämän lisäksi Chakraborty, Shahriyar ja Iqbal (2019) tutkimuksessa selvitetään, että kielien ympärille syntyvien yhteisöjen kasvunopeus johtuu ainakin yhteisön aktiivisuudesta, sekä kielen edeltäjän (kieli josta on selkeä siirtymä uuteen kieleen) yhteisön koosta.

2.2.2 Miksi ohjelmointikieli pysyy suosittuna?

Jotta ohjelmointikieli pitää senhetkisen asemansa ja käyttäjämääränsä, sovelluskehittäjien tulee jatkaa kielen käyttämistä, eikä korvata sitä kokonaan toisella kielellä. Tämä luo kielten välille jonkinasteisen kilpailuasetelman, mutta koska kehittäjien preferensseihin vaikuttavat muutkin kuin helposti vertailtavat ominaisuudet, ei vertailu ole täysin yksiselitteistä. Chatley, Donaldson ja Mycroft (2019) tutkimuksessa selvitetään laajasti tekijöitä, jotka pitävät ohjelmointikielen elossa ja suosittuna:

Vanhoja ylläpidettäviä ohjelmistoja on päivä päivältä enemmän. Jollei niitä kirjoiteta uusiksi tai jollei niihin kirjoiteta rajapintoja muihin kieliin, pitää mahdollisia muutoksia ja päivityksiä tehdä niissä käytetyillä ohjelmointikielillä. Tämä pitää kielen käytettynä, vaikka sille olisikin olemassa parempia vaihtoehtoja.

Ohjelmointikielen oma yhteisö tukee ohjelmointikieltä eri tavoin. Mitä enemmän kielestä käydään keskustelua, sitä paremmin puutteita ja tarpeita saadaan esille, minkä jälkeen kieltä voidaan rakentaa eteenpäin. Lisäksi yhteisön tuki auttaa apua tarvitsevia kehittäjiä kieleen liittyvien kysymyksien kanssa.

Käytön aloittamisen helppous on tärkeää varsinkin kun kyseessä on noviisi kehittäjä. Helppous kuitenkin palvelee myös kokenempia kehittäjiä, joiden kynnys vaihtaa kieltä tarpeen vaatiessa on oletettavasti matalampi. Aloittamisen helppouteen kuuluu kielen ymmärrettävyys, tarjolla oleva tuki kuten tutoriaalit ja dokumentaatio, sekä aika mikä vaaditaan siihen että tarvittavat ohjelmistot on asennettu.

Tuttuus tai kodikkuus tehostaa ja helpottaa kehittäjälle uuden kielen ja kehitysympäristön käyttöönottoa. Jos kielen rakenteet toimivat jostakin muualta tutulla tavalla, sekä kieleen mahdollisesti liitetty kehitysympäristö on intuitiivinen ja tuttu, kehittäjän ei tarvitse kuluttaa niiden tutkimiseen ja opetteluun aikaa ja mentaalisia resursseja, vaan voi aloittaa heti tuottavan työskentelyn. Tämä osatekijä näkyy myöhemmin varsinkin luvussa 3.7, jossa yksi nousevilla kielillä esiintyvä yleinen piirre on samankaltaisuus ja helppo siirtymä jostakin jo suositusta kielestä.

Olemassaolevat kirjastot houkuttelevat kehittäjiä kielen käyttäjiksi tehostamalla kehitystä,

kun yleisiin ongelmiin ja tarpeisiin tarjotaan valmiita, uudelleenkäytettäviä ratkaisuja. Mikäli kielellä ei ole tarvittavia kirjastoja helpottamaan sovellusten implementointia ja kehittäjä joutuu tekemään kaikkeen ratkaisut alusta asti, voi katse kääntyä jonkin toisen vaihtoehdon suuntaan. Kirjastojen lisäksi kieleen itseensä liittymättömät saatavilla olevat työkalut tehostavat kehitystä ja mahdollisesti paikkaavat kielen puutteita.

Ohjelmakoodin muutoksien tukeminen ja helpottaminen on osa-alue, joka hyvin toteutettuna vetää kehittäjiä puoleensa. Jos ohjelmakoodiin muutosten tekeminen on kielen puolesta suoraviivaista, siihen voidaan käyttää automaattista refaktorointia suorittavia työkaluja. Nämä toimivat paremmin staattisesti tyypitetyissä kielissä. Myös yksittäisen pienen muutoksen tekemiseen ja sen testaamiseen kuluva aika on tärkeässä roolissa. Esimerkiksi dynaamisesti tyypitetyn kielen tuottaminen voi olla nopeampaa, mutta sen testaaminen saattaa viedä enemmän aikaa.

Kielen suosio riippuu myös olemassaolevien kehittäjien määrästä. Yritys käyttää projekteissansa niitä kieliä, joiden osaajia on tarjolla, jolloin kyseisen kielen projektien määrä kasvaa, jolloin kieli on merkittävämmässä asemassa ja houkuttelee lisää kehittäjiä tutustumaan siihen. Tällä positiivisella kierteellä on helposti kuviteltavissa myös vastinpari, jossa kieltä osaavien kehittäjien vähyys on osallisena kielen suosion jatkuvaan heikkenemiseen.

Vaikka kaikkien sovellusten ei pidä nykyisin olla optimoituja suorituskyvyn suhteen, löytyy edelleen sovelluksia joissa kielen tehokkuus on kriittisessä asemassa, jolloin matalamman tason kieliä käytetään varsinkin lopullisen sovelluksen tai sen muistia tai suoritustehoa vaativan osan kirjoittamiseen.

2.2.3 Mitkä tekijät vauhdittavat kielen muuttumista suosion säilyttämiseksi?

Uudet saavutukset teknologiassa voivat avata kielelle uusia mahdollisuuksia. Mikäli kieli mahdollistaa näiden uusien teknologioiden käyttämisen, varsinkin ensimmäisten joukossa, asema voi pysyä samana tai vankistua huomattavasti (Chatley, Donaldson ja Mycroft 2019).

Kilpailu pakottaa kielen muuttumaan. Mikäli samalla kohdealueella vaikuttava kieli tekee jonkin asian tavalla, joka on kehittäjien mielestä parempi kuin kilpailijalla, voi se johtaa suurempaan määrään kehittäjiä. Tämä helposti pakottaa toisen kielen muuttamaan tämän

ominaisuuden omasta kielestään samanlaiseksi tai vielä paremmin toteutetuksi. Dynamiikka kielten välillä tässä tapauksessa voi olla kilpailullinen, tai ideoiden ja ominaisuuksien jakaminen voi vaikuttaa enemmän yhteistyöltä (Chatley, Donaldson ja Mycroft 2019).

Yrityksien ja yhteisöjen tarpeet tietyille ominaisuuksille saattavat saada aikaan sen, että tarvittavat ominaisuudet implementoidaan kieliin. Ominaisuudet voidaan lisätä kieliin suoraan, tai kieliin voidaan luoda laajennuksia joko kieltä kehittävän joukon tai ulkopuolisen ryhmän toimesta. Ulkopuolelta tulleet laajennukset voidaan jälkeenpäin ottaa osaksi kielen ydinominaisuuksia (Chatley, Donaldson ja Mycroft 2019).

Käänteisesti ohjelmointikielen käyttömäärä laskee mikäli edellä mainittuja ominaisuuksia ei implementoida, tai ominaisuudet eivät enää ole kilpailukykyisiä muihin vaihtoehtoihin nähden. Lisäksi tietoturvaan liittyvät ongelmat kielen rakenteissa voivat suuresti vähentää kielen houkuttelevuutta, joten muutostarpeiden ilmetessä niiden prioriteetti on korkea (Chatley, Donaldson ja Mycroft 2019). Varsinkin ei-hallitut kielet ovat kärsineet tietoturvaongelmista erilaisten mahdollisten muistiin kohdistuvien virheiden tai hyökkäysten takia, mikä lisää hallittujen kielien tai hallitun koodin vetovoimaa. Chatley, Donaldson ja Mycroft (2019) haluaa jakaa häviävät kielet kahteen osaan: niihin joilla ei ole uusia projekteja, mutta on vielä olemassa ylläpidettäviä projekteja, sekä niihin joilla ei ole olemassa edes ylläpidettäviä projekteja.

3 Ohjelmointikielet

Tässä luvussa selvitetään eri ohjelmointikielien suosiota tilastoista ja pyritään etsimään syitä käyttö- ja hakumäärille ja niiden suuntauksille. Syitä pyritään etsimään erilaisilla menetelmillä ja niistä aiheista, jotka soveltuvat kunkin ryhmän kieliin parhaiten.

Kun aletaan selvittämään alkaneen vuosikymmenen ohjelmointikieliä, sen lisäksi että selvitetään mitkä ohjelmointikielet ovat tällä hetkellä suosittuja, tulee tilastoja kartoittaa myös järkevältä ajanjaksolta menneisyydestä, jotta vertailu ja johtopäätösten tekeminen on mahdollista. Jo tarpeeksi pitkältä aikaväliltä kerättyjen tilastojen tulkitseminen voi antaa hyvän kokonaiskuvan kielien suhteista, vaikka se ei kerrokkaan syitä niiden suhteiden takana.

Kun on selvitetty mitä kieliä on käytetty ja käytetään tällä hetkellä, voidaan tilastojen perusteella selvittää miten kauan kielillä kestää niiden julkaisun jälkeen nousta huippuunsa käyttömäärän osalta. Lisäksi on hyödyllistä selvittää millainen on kielen, joka täyttää yhden tai useamman kohdan edellisen luvun kriteereistä käyttömäärän laskemiselle, tyypillinen käyttömäärän tai muun relevanssin vähenemisen nopeus ja tapa. Näiden tietojen avulla voidaan asettaa teoreettisia rajoitteita sille, miten nykyiset käyttömäärät voivat maksimissan muuttua kymmenen vuoden ajanjakson aikana.

3.1 Ohjelmointikielien tilastoja ja suuntauksia

Riittävän kattavien ja tuoreiden tilastojen saamiseksi pelkät vertaisarvioidut akateemiset julkaisut ja niiden tilastot havaittiin tutkielmaa kirjoittaessa heikoksi vaihtoehdoksi, koska niiden tilastot eivät käsittele tarpeeksi monia ohjelmointikieliä tai ne eivät ole tarpeeksi uusia. Verkossa on useita toimijoita, jotka päivittävät eri metriikoilla mitattuja tilastoja vähintään kerran vuodessa, suuressa osassa tapauksista useammin kuin kerran vuodessa. Nämä tilastot eivät välttämättä sisällä syitä, analyysiä tai johtopäätöksiä, joten niistä ei voi suoraan tehdä tulkintoja, mutta yhdistettynä niitä voidaan käyttää riittävän kattavan kartoituksen laatimiseen ja yleisten trendien havaitsemiseen.

Liitteessä 1 on listattuna 20 suosituinta ohjelmointikieltä neljästä eri lähteestä eri metriikoilla

mitattuna. Listaukset ovat vuosilta 2019 ja 2020. Kolmessa listauksessa on myös merkittynä prosentuaalinen muutos edeltävään vuoteen.

Kuvassa 1 on kuvattu TIOBE:n (“TIOBE Index” 2021) mittaama pitkän ajan kehitys suurimpien kielien osalta. TIOBE:n “Community index” perustuu osittain manuaalisesti varmistettujen hakukonetulosten määrään. Hakukoneina käytetään 25:tä korkeimmalle arvioitua hakukonetta. Hakukoneissa käytetty hakulauseke on `+"<language> programming"`.

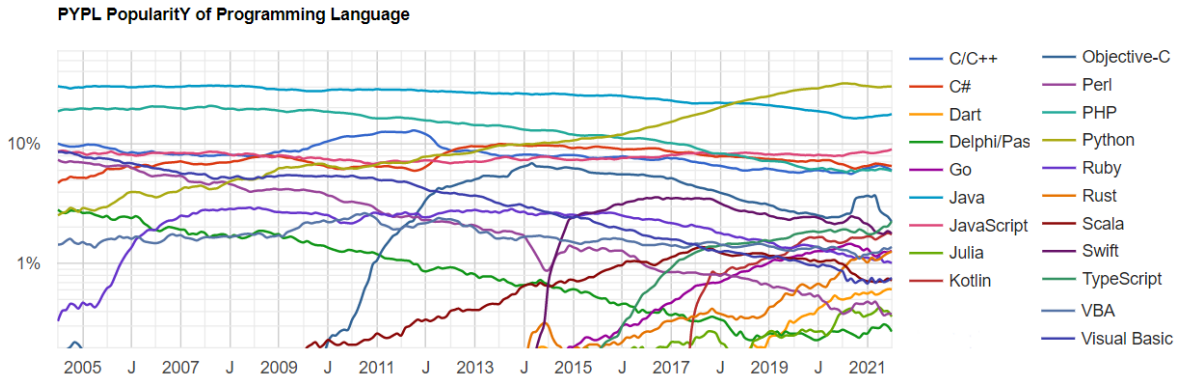
Very Long Term History

To see the bigger picture, please find below the positions of the top 10 programming languages of many years back. Please note that these are *average* positions for a period of 12 months.

Programming Language	2020	2015	2010	2005	2000	1995	1990	1985
Java	1	2	1	2	3	-	-	-
C	2	1	2	1	1	2	1	1
Python	3	7	6	6	23	21	-	-
C++	4	4	4	3	2	1	3	9
C#	5	5	5	7	9	-	-	-
JavaScript	6	8	8	8	6	-	-	-
PHP	7	6	3	4	25	-	-	-
SQL	8	-	-	96	-	-	-	-
Swift	9	181	-	-	-	-	-	-
Ruby	10	11	9	23	31	-	-	-
Objective-C	12	3	15	37	-	-	-	-
Lisp	27	21	14	13	8	5	6	2
Fortran	29	28	21	14	16	4	2	12
Ada	34	26	24	15	15	6	7	3
Pascal	235	13	12	53	13	3	8	5

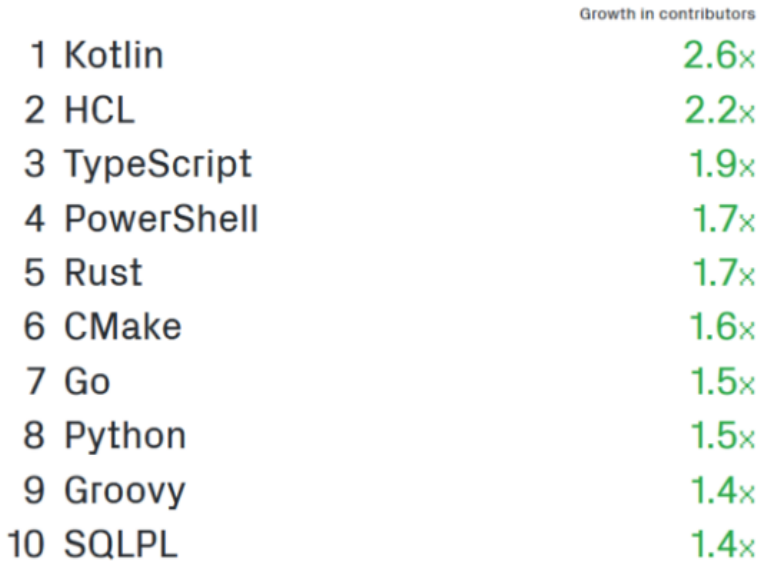
Kuva 1: TIOBE pitkän aikavälin tilastot

Kuvassa 2 on PYPL:n (“PYPL PopularitY of Programming Language” 2021) mittaama ohjelmointikielien suosio 15 vuoden ajalta. PYPL:n graafit perustuvat Googlen hakukoneen ohjelmointikielten tutoriaaleihin liittyvien hakujen määrään, saatavien tulosten määrän sijaan. Tutoriaali on sisällytetty hakuihin useiden muuttujien poistamiseksi. Myöhemmissä luvuissa tästä graafista eriytetään kuvaajia selkeämmän kuvan saamiseksi.



Kuva 2: PYPL ohjelmointikielten suosio

Kuvassa 3 on kuvattu eniten vuosien 2017 ja 2018 välillä kasvaneet kielet GitHubissa. Laskelmat ovat osa “State of Octoverse 2018” (2021) raporttia. Mittaus on suoritettu laskemalla kielen kirjoittamiseen osallistuneiden kehittäjien kokonaismäärien kasvua. Koska GitHub määrittää projekteissa käytetyt kielet oletusarvoisesti Linguist-kirjaston avulla automaattisesti, tilastoissa voi esiintyä epätarkkuuksia virheellisten tulkintojen takia.



Kuva 3: State of octoverse 2018 ohjelmoijien kasvun määrä

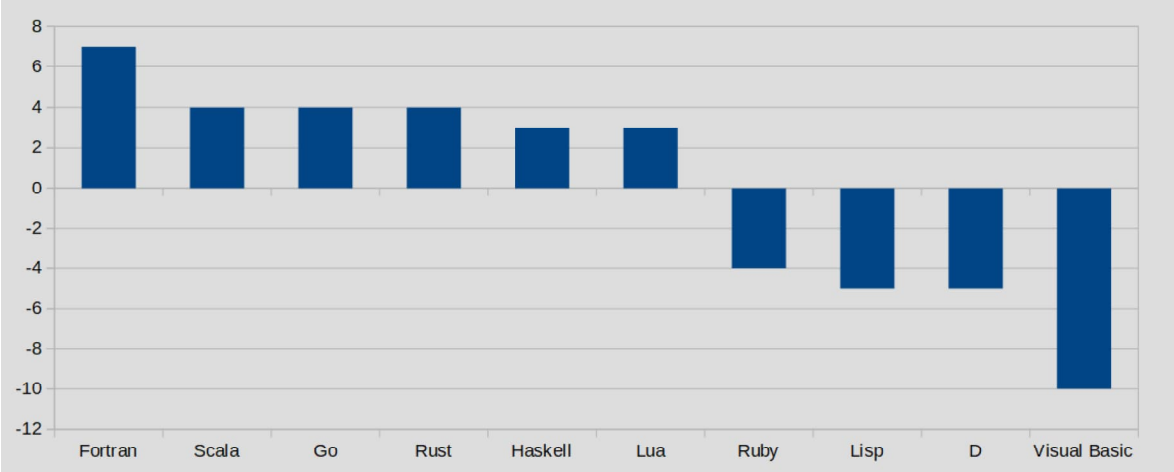
Kuvassa 4 on edellä mainitun Octoversen nopeimmin nousevat kielet vuodelta 2019 (“State of Octoverse 2019” 2021).

01	Dart	532%
02	Rust	235%
03	HCL	213%
04	Kotlin	182%
05	TypeScript	161%
06	PowerShell	154%
07	Apex	154%
08	Python	151%
09	Assembly	149%
10	Go	147%

* Top 10 growing languages, ranked by number of contributors to repositories with detectable primary programming languages between October 1, 2018 and September 30, 2019. All of these repositories had at least 10K contributors in 2018.

Kuva 4: State of octoverse 2019 eniten kasvaneet kielet

Kuvassa 5 on havainnollistettu IEEE Spectrum ranking -tilaston ohjelmointikielien sijoitusten suurimmat muutokset vuosien 2015 ja 2018 välillä. Graafiin on sisällytetty kielet, jotka ovat olleet tällä ajanjaksolla kolmenkymmenen parhaiten pisteytetyn kielen joukossa. Tässä tilastossa huomioidaan ohjelmointikielien akateemiset julkaisut, sosiaalisen median julkaisut, kieliin liittyvät uutiset, kielien avoimet työpaikat, avoimen lähdekoodin projektit ja hakukoneiden hakumäärät (“Interactive: The Top Programming Languages 2018” 2021).



Kuva 5: IEEE Spectrum ranking sijoitusten muutokset 2015-2018

Seuraavassa alaluvussa on tiiviisti summattu tämän hetken näkyvimmät suuntaukset näiden tilastojen pohjalta.

3.2 Ohjelmointikielet tällä hetkellä

Kaikilla listoilla monikäyttöiset ja modernit, mutta kohtalaisen pitkään olemassa olleet kielet Python, Java, C# ja C++ ovat melko korkeilla sijoituksilla. Myös C on kaikissa mittauksissa relevantti.

Web-sovelluskielistä laajalle levinnyt JavaScript on kaikissa tilastoissa korkealla ja PHP:n sijoitus on edelleen relevantti. Osasta löytyy myös JavaScriptin tyypitetty versio TypeScript.

Uudemmista ja vähemmän käytetyistä monikäyttöisistä kielistä Go esiintyy monessa mittauksessa. Myös Scala, Rust, Dart ja Julia ovat nousseet omilla vahvuuksillaan jonkinasteiseen suosioon.

Sovellusspesifeistä kielistä Applen alustoille sovelluksia kehitettävät Objective-C ja Swift esiintyvät tilastoissa. Swift on edeltävän uudempi vaihtoehto.

Pienemmistä ja pääasiassa vain web-sovelluskehitykseen käytettävistä kielistä oli listattuna Ruby. Mobiilikehitykseen käytetyistä kielistä esiintyi Kotlin. Vanhemmista kielistä esiin nousivat Visual Basic ja Perl.

3.3 Havaittavat trendit

Tilastoista on havaittavissa useita kehityssuuntia, jotka toistuvat tilastoista toiseen. Kokonaiskäyttömäärän prosenttiyksiköillä mitattuna merkittävin muutos viimeisen viiden vuoden aikana on Pythonin edelleen jatkuva nousu. Python on noussut kaikkien yleiskäyttöisten ohjelmointikielten yläpuolelle lähes kaikissa metriikoisssa ja on tällä hetkellä suurin yleiskäyttöinen ohjelmointikieli. Vastaavasti tilastoissa muut paljon ja monessa kohteessa käytetyt kielet C++, Java, C# ja C ovat laskeneet useita prosentteja, mutta ovat edelleen selkeästi eriytettävissä loppujen kielien käyttömääristä.

2010 vuoden jälkeen luoduista kielistä Kotlin, TypeScript, Rust, Julia ja Dart ovat voimakkaassa, osin eksponentiaalisessa kasvussa, joskin Rustin hakukoneiden hakumäärät ovat jo osin laskeneet. Suosio ja käyttömäärät ovat kuitenkin vielä kaukana suurista ohjelmointikielistä. Näiden lisäksi Swift on 2014 tapahtuneen julkaisunsa jälkeen kasvanut huomattavasti ja samaan aikaan tämän edeltäjän Objective-C:n käyttömäärät ovat laskeneet samassa suhteessa.

Ruby on monella metriikalla mitattuna relevantti, mutta kuvaajilla sen suunta saattaa olla kääntymässä alaspäin. Käyttö- ja hakumäärissä jo useita vuosia laskussa ovat olleet Visual Basic, Perl, PHP ja Object Pascalin dialekti Delphi. Hieman yli 10 vuotta sitten julkaistuista kielistä Go:n ja Scala:n suosio on edelleen hienoisessa nousussa. Object Pascal itse esiintyy enää vain osassa tilastoja, mutta varsinkin ”Tiobe long term” -tilastossa sen sijoitus vuosien 2015 ja 2020 välillä on pudonnut rajusti.

Seuraavissa alaluvuissa suoritamme katsauksen eri asemassa olevien kielien historiaan ja kehitykseen ja pyrimme selvittämään tilastoista havaittavien kehityssuuntien osatekijöitä. Eri ryhmissä oleville kielille haetaan osasyitä niiltä alueilta, joiden voidaan kuvitella olevan relevantteja suuntauksen kannalta.

3.4 Suuren käyttömäärän kielet

Ensimmäiseksi selvitämme kuuden suositun kielen osalta niiden lähtökohdat, minkä suunnitteluvalintojen ympärille kielet ovat rakentuneet, mitkä ominaisuudet ja tekijät ovat tuo-

neet kielelle käyttäjiä, sekä mistä valinnoista kieltä kritisoidaan. Tällä saamme paremman käsityksen siitä, mitä tekijöitä menestyvän kielen kehittämiseen liittyy.

3.4.1 C

C:n on suunnitellut Dennis Ritchie toimiessaan Bell Labsilla. Kieli sai alkunsa vuonna 1972, kun Ritchie pyrki parantelemaan B-kieltä joidenkin sen puutteiden takia. Tarkoituksena oli kehittää kieli järjestelmäkehitystä varten silloin uuteen Unix-käyttöjärjestelmään. Kieli adoptoitiin samantien Unixin toiseen versioon. Ensimmäinen virallinen standardi kielestä tuotettiin vasta vuonna 1990 (Ritchie 1993).

C:llä ei ole erikseen kirjattua suunnittelufilosofiaa, mutta *Rationale for International Standard Programming Languages C* (2003):ssa mainitaan olemassaolevan koodin tärkeys ja arvostus, C:n käytön monimuotoisuus, hiljaisten muutoksien välttäminen ja standardien merkitys. Näiden lisäksi esitellään käsite ”spirit of C”, joka koostuu muunmuassa seuraavista käskyistä:

- Luota ohjelmoijaan
- Älä estä ohjelmoijaa tekemästä tilanteeseen tarvittavia asioita
- Pidä kieli pienenä ja yksinkertaisena
- Tarjoa vain yksi tapa suorittaa tietty operaatio
- Pyri nopeuteen, vaikka se heikentäisi joustavuutta

Osa C:n menestyksestä on seurausta Unixin valtaisasta menestyksestä ja C:n leviämisestä sitä kautta. Kääntäen myös Unixin menestys on osittain C:n ominaisuuksien ansiota. Myös C:n yhtenäisyys on ollut suosion kannalta tärkeää, eikä C ole juurikaan haarautunut erilaisiin dialekteihin. Lisäksi C on yksinkertainen ja pieni kieli jota voidaan kääntää pienillä ja yksinkertaisilla kääntäjillä, joilla koon ja tehokkuuden kannalta optimoitujen ohjelmien tuottaminen on nopeaa (Ritchie 1993). *Rationale for International Standard Programming Languages C* (2003) taas esittelee pääasiallisiksi menestymisen syiksi kilpailijioiden vuosikymmenten aukot migraatio- ja integraatiotuessa. Tämän lisäksi C:hen on onnistuttu luomaan ominaisuuksia, jotka mahdollistavat järjestelmän rinnakkaisten osien yhteistoiminnan erittäin hyvin.

C on yksinkertainen ja kompakti kieli ja suurin osa asioista jotka ovat joidenkin käyttäjien mielestä puutteita tai vikoja, voidaan perustella olevan suunnitteluvalintoja. Esimerkiksi olio-ohjelmointiin liitettävien ominaisuuksien, poikkeustenhallinnan ja datan piilottamiseen liittyvien ominaisuuksien puuttuminen ovat kaikki osa C:n olemusta (Ritchie 1993). Ritchie (1993) mukaan eniten kritiikkiä ovat saaneet taulukoiden ja kohdistimien suhde, sekä tapa jolla esittelyn syntaksi muistuttaa lausekkeiden syntaksia. Lisäksi monien mahdollisten haavoittuvuuksien takia C vaatii kehittäjältä paljon, jotta ohjelmistoista saadaan turvallisia ja erilaisia aukkoja ei jätetä.

3.4.2 C++

C++:n suunnitteli ja kehitti Bjarne Stroustrup AT&T Bell Laboratories -laitoksella 1980-luvun lopulla. Kieli käytti alussa nimeä ”C with Classes” ja nimi C++ tuli käyttöön vuonna 1985, kun kieli julkaistiin kaupallisesti saatavaksi. Stroustrup pyrki kielellään yhdistämään Simulan suunnittelun ja organisationaaliset vahvuudet, sekä C:n tarjoamat mahdollisuudet järjestelmien ohjelmoimiseen (Stroustrup 1998). Stroustrup on itse tiivistänyt alkuperäisen kielen olevan vastaus kysymykseen, ”Miten ohjata laitteistoa suoraan, kuitenkin samalla tukien tehokasta korkean tason abstraktiota?” (Stroustrup 2020). Alussa kieli tuki perinteistä järjestelmäohjelmointia ja data-abstraktiota, alkeellisen olio-ohjelmoinnin tuki lisättiin vuonna 1983. Kielen suosion ja monien erillään kehitettävien implementaatioiden takia virallisen standardin kehittäminen aloitettiin vuonna 1990 ANSI:n ja myöhemmin ISO:n toimesta ja kansainvälinen standardi kielelle valmistui vuonna 1998 (Stroustrup 1998).

Edellä mainittujen lähtökohtien lisäksi Stroustrup on myöhemmin laatinut joitakin sääntöjä joita C++:n tulisi aina noudattaa ja tavoitteita joita kohti kielen tulisi pyrkiä. Ensimmäiseksi tavoitteeksi hän on tiiviissä muodossa summannut sen, että kielen tulisi tehdä ammattilaisten ohjelmoinnista miellyttävämpää. Toinen tavoite on se, että C++ on yleiskäyttöinen ohjelmointikieli, joka on parempi kuin C ja tukee datan abstraktiota, oliosuuntautunutta ohjelmointia ja geneeristä ohjelmointia (Stroustrup 1998).

Lyhyet versiot säännöistä Stroustrup on jakanut neljään luokkaan: yleiset säännöt, suunnittelua tukevat säännöt, kielen tekniset säännöt, sekä matalan tason ohjelmointia tukevat säännöt

(Stroustrup 1998). Yleisten sääntöjen luonne vaihtelee konkreettisuudeltaan ja selkeydeltään ja ne ovat seuraavat:

- C++:n evoluutio täytyy tapahtua oikeiden ongelmien seurauksena
- C++ on kieli, ei kokonainen järjestelmä
- Älä pyri turhaan täydellisyyteen
- C++:n tulee olla hyödyllinen juuri nyt
- Kielen jokaisella ominaisuudella tulee olla riittävän ilmeinen implementaatiotapa
- Tarjoa aina väylä siirtymälle
- Tarjoa kattava tuki kaikille tuetuille kehittämistavoille
- Älä yritä pakottaa ihmisiä

Suunnittelua tukevat säännöt:

- Tue järkeviä suunnittelukäsitteitä
- Tarjoa mahdollisuudet organisoinnille
- Sano mitä tarkoitat
- Kaikkien ominaisuuksien tulee olla käytettävissä
- On tärkeämpää sallia hyödyllinen ominaisuus kuin pyrkiä estämään jokainen väärinkäyttö
- Tue ohjelmiston kokoamista erillään kehitetyistä osista

Kielen tekniset säännöt:

- Ei epäsuoraa staattisen tyypityksen rikkomista
- Tue käyttäjän määrittelemiä tyyppejä yhtä hyvin kuin sisäänrakennettuja tyyppejä
- Lokaalius on hyvä asia
- Vältä järjestysriippuvuuksia (”order dependencies”)
- Valitse ominaisuuden helpommin opetettava variantti
- Syntaksilla on väliä ja siihen liittyvät ongelmat ovat monimutkaisia
- Esikääntäjän käyttäminen tulisi lopettaa

Matalan tason ohjelmoinnin säännöt:

- Käytä perinteisiä, yksinkertaisia linkkereitä
- Ei turhia yhteensopimattomuuksia C:n kanssa
- Älä jätä tarvetta C++:aa matalamman tason kielelle assembleria lukuunottamatta
- Resursseja ei tule kuluttaa ominaisuuksiin jotka eivät ole käytössä (zero-overhead principle)
- Tarvittaessa tarjoa mahdollisuus manuaaliseen säätämiseen

Tärkein syy C++:n menestymiselle Stroustrupin mielestä on se, että se tukee sellaisten sovellusten kehittämistä, joiden täytyy käyttää laitteistoa tehokkaasti ja hallita merkittävää kompleksisuuden määrää. Jos teholla ei ole väliä, löytyy paljon muita valittavia kieliä ja jos pitää kehittää vain pieni ja tehokas ohjelmiston osa, niin C tai assembler ovat vaihtoehtoja. C++ toteuttaa nämä molemmat alueet ja se on riittänyt kielen suosioon (Stroustrup 2020). Vuosituhannen alun teknisiin uudistuksiin lukeutuvat muistimalli, tyyppiturvallinen samanaikaisuuden tukeminen, tyyppien päättely, useiden ominaisuuksien yksinkertaistaminen, move-semantiikka, kääntämisen aikainen ohjelmointi, geneerinen ohjelmointi ja metaohjelmointi, joista kaksi viimeistä ovat vielä kehityksen alla (Stroustrup 2020).

WG21 komitea, joka päättää C++ standardeista, on kasvanut rajusti ja kaikki yli 350:sta hengestä ovat hyvin harvoin täysin samaa mieltä päätettävistä asioista, jolloin kielen uudistaminen on kankeaa (Stroustrup 2020). Tämän lisäksi C++:n ikä tuo mukanaan joitakin ongelmia. Osa kielen rakenteista vaikeuttaa työkalujen kehitystä, kielen opetus on osin vanhentunutta, kielelle ei ole standardoitua version- tai paketinhallintajärjestelmää ja kieli ja peruskirjasto käyttävät vielä ASCII-järjestelmää. Näiden lisäksi kieleen on kertynyt paljon turhia ominaisuuksia ja toteutuksia, joiden muuttaminen on hankalaa kaikkien olemassaolevien C++ ohjelmien ja niiden riippuvuuksien takia. Stroustrup itse on sitä mieltä että suurimpaan osaan näistä ongelmista on kehitteillä jonkinlaisia vastauksia (Stroustrup 2020).

3.4.3 C#

C# kehitettiin vuosituhannen vaihteessa Microsoftilla Anders Hejlsbergin, Scott Wiltamuthin ja Peter Golden johdolla osana .NET-kehystä. .NET on Microsoftin kehittämä ohjelmistokehitykseen tarkoitettu avoimen lähdekoodin ohjelmistokomponenttikirjasto, joka koostuu pää-

asiassa luokkakirjastoista ja ajonaikaisesta ympäristöstä. Vuonna 2001 Microsoftin, Hewlett-Packardin ja Intelin toimesta aluille laitettu ensimmäinen standardi julkaistiin saman vuoden lopulla ECMA:n (European Computer Manufacturers Association, nykyisin Ecma International, on yksi alan voittoa tavoittelemattomista standardeja laativista järjestöistä) toimesta. Kielen merkittävimmät ominaisuudet olivat luokat, tietueet, rajapinnat, tapahtumat, ominaisuudet, delegaatit, operaattorit, lausekkeet, lauseet ja atribuutit (*Csharp Language Specification* 2002).

C#:n kehittyessä sille asetettiin seuraavat suunnittelutavoiteet (*Csharp Language Specification* 2002):

- C#:n on tarkoitus olla yksinkertainen, moderni, yleiskäyttöinen ja oliosuuntautunut ohjelmointikieli.
- Kielen tulee tukea ohjelmistokehityksen periaatteita tarjoamalla vahvaa tyyppien tarkistusta, taulukoiden rajojen tarkistusta, alustamattomien muuttujien käyttämisen havaitsemista ja automaattista roskienkeruuta. Ohjelmiston joustavuus, toimintavarmuus ja ohjelmoijan työskentelyn tehokkuus ovat tärkeitä asioita.
- Kieli on tarkoitettu käytettäväksi sellaisten ohjelmistokomponenttien kehittämisessä joita voidään käyttää hajautetuissa järjestelmissä.
- Ohjelmakoodin siirrettävyys on tärkeää, mutta niin on myös ohjelmoijien siirrettävyys, varsinkin niiden joille C ja C++ ovat tuttuja.
- Kansainvälistämisen tukeminen on erittäin tärkeää.
- C#:n on tarkoitus olla käytettävissä kaiken kokoisissa sulautetuissa tai muualla ylläpidetyissä järjestelmissä.
- Vaikka C#-sovelluksien on tarkoitus olla muistin käytön ja prosessointitehon tarpeiden suhteen säästeliäitä, kielen ei ole tarkoitus kilpailla näillä osa-alueella C:n tai assemblyn kanssa.

Edellä mainittu kiinteä kuuluminen .NET-kehykseen on melko ilmeinen osa C#:n menestystä ja Microsoft kehittäjänä on suuri taho, jolta voi odottaa riittävästi tukea. Varsinkin graafisia käyttöliittymiä sisältävien työpöytäsovellusten kehitys mielletään nopeaksi valmiiden osien ja GUI-editorien takia.

Yleisesti kritiikkinä C#:lle on mainittu kielen ja sen ympäristön suuri koko, sekä ympäristön käytöstä mahdollisesti koituvat suuret kustannukset. Aiemmin kieltä on kritisoitu alkuperäisen .NET-kehyksen takia osittaisesta Windows-riippuvuudesta, mutta nykyinen .NET (aiemmalta nimeltään .NET Core) on järjestelmäriippumaton. Lisäksi .NET:n ja CLR:n (Common Language Runtime) lataamisen takia ohjelmien ajaminen on hitaampaa kuin C++:ssa tai C:ssä.

3.4.4 Java

Javan ensimmäinen julkinen versio julkaistiin James Goslingin ja hänen tiiminsä toimesta vuonna 1995 Sun Microsystemsin Solaris-käyttöjärjestelmään (Evans 2015). Alkunsa kieli sai tutkimusprojektista jossa päämääränä oli kehittää edistyksellistä ohjelmistoa erilaisille tietoverkkolaitteille ja sulautetuille järjestelmille (Gosling ja McGilton 1995). Päämääränä oli kehittää pieni, vakaa, siirrettävä, yleiskäyttöinen ja reaaliajassa toimiva alusta. Koska kehityksessä aluksi käytetyn C++:n kanssa syntyneiden ongelmien määrä kasvoi liian suureksi, päätti kehitystiimi kehittää oman kielen ottaen vaikutteita muun muassa Eiffelistä, SmallTalkista, Objective C:stä ja Cedarista/Mesasta (Gosling ja McGilton 1995).

Gosling on itse kuvannut Javan olemusta sanoin ”Java on työväen kieli. Se ei ole väitöskirjamateriaalia, vaan kieli asioiden hoitamiseen” (Evans 2015). Javalle on alun perin asetettu seuraavat korkean tason suunnittelun tavoitteet (Evans 2015):

- Tarjoa alusta oliosuuntautuneiden sovellusten koodin yksinkertaiseen ajamiseen.
- Siirrä kehittäjien kognitiivista taakkaa kielen vastuulle.
- Poista C:n tai C++:n turvallisuusaukkoja niin paljon kuin mahdollista.
- Salli järjestelmäriippumatonta koodin ajamista.

Muissa Javan suunnittelun tavoitteissa näkyvät yksinkertaisuuden, tuttuuden ja tehokkuuden korostaminen (Gosling ja McGilton 1995).

Osa kielen menestyksestä voidaan olettaa kuuluvan ajonaikaiselle Java-ympäristölle, jolla ohjelmia voi ajaa useilla erilaisilla tietokoneilla ja käyttöjärjestelmillä. Myös kielen oliosuuntautuneisuus on erottanut sen esimerkiksi kilpailijastaan C++:sta. Näiden lisäksi Java

oli oletuskieli Android-sovellusten kirjoittamisessa kun Android julkaistiin vuonna 2008.

Javaa on kritisoitu siitä että se voi olla ympäristönsä takia hitaampi ajaa ja vaatii enemmän muistia kuin osa vaihtoehtoisista kielistä. “The Computer Language Benchmarks Game” (2021) on suosittu vuosituhannen vaihteessa alkunsa saanut projekti, joka pyrkii vertailemaan käytetyimpien ohjelmointikielien suoritustehoa yksinkertaisia algoritmeja ajamalla. Projektin tämän hetken tilastoista ilmenee, että vaihtoehtoisista kielistä C++ vaatii useimmissa allgoritmeissa huomattavasti vähemmän muistia ja on nopeampi ajaa, kun taas C# on keskimäärin nopeampi ajaa ja vaatii samankaltaisia määriä muistia. Pythoniin verrattaessa erot erilaisten algoritmien välillä ovat suuria, mutta useassa niistä suoritukseen kuluva aika on Pythonilla moninkertainen Javaan verrattuna.

Javaa on myös yleisesti kritisoitu sillä tuotettujen Swing-sovellusten oletusarvoisesta ulkonäöstä, mutta subjektiivista väitettä tukemaan ei löydy esimerkiksi mielipidemittauksia tai muita tilastoja.

3.4.5 JavaScript

JavaScript kehitettiin vuonna 1995 Netscapella Brendan Eichin toimesta, kun Netscape tarvitsi skriptikielen web-selaimeensa Navigatoriin dynaamisen sisällön mahdollistamiseksi. Ensimmäinen Netscapen vaihtoehto tämän tarpeen täyttämiseen oli yhteistyö Sun Microsystemsin kanssa ja Javan käyttäminen skriptikielenä. Toinen vaihtoehto oli kehittää oma kieli. Eich kehitti alunperin Schemeen perustuvan Mocha-nimisen prototyypin kymmenessä päivässä, josta sittemmin kehittyi JavaScript. Lopulta Netscape ja Sun sopivat että Javaa aletaan ajamaan suurempiin ja ammattimaisempiin web-projekteihin, kun taas JavaScript esiteltäisiin asiakaspuolen komponentteihin soveltuvana pienenä skriptikielenä (Wirfs-Brock ja Eich 2020). Tämän jälkeen JavaScript on kehittynyt kiinteäksi osaksi dynaamisia HTML-sivuja, myös tietokanta- ja palvelinpuolella (Rauschmayer 2014).

Rauschmayer (2014) jakaa JavaScriptin luonteen kuuteen piirteeseen: dynaaminen, dynaamisesti tyypitetty, funktionaalinen ja oliosuuntautunut, ”kaatuu äänettä”, otetaan käyttöön lähdekoodina ja on osa web-alustaa. JavaScript on dynaaminen, sillä olioiden ominaisuuksia voidaan vapaasti lisätä ja poistaa luomisen jälkeen, eikä olioiden luomiseen tarvita erillisiä

konstruktoreita. Äänettä kaatuminen (”fails silently”) johtuu siitä että JavaScriptillä ei ollut pitkään aikaan poikkeuksien käsittelyä. Rauschmayer (2014) sisältää myös Brendan Eichin listaamia edukseen erottuvia kielen ominaisuuksia, joita ovat ensiluokkaiset funktiot, sulkeumat, prototyypit sekä taulu-ja olioliteraalit.

Osa JavaScriptin kritiikistä liittyy sen keskeisiin piirteisiin ja ominaisuuksiin, kuten sen dynaamiseen tyypitykseen tai “pakotussääntöihin” (coercion rules), jotka liittyvät erikoisiin implisiittisiin tyyppimuunnoksiin. Osa taas kritisoi sen kiinteää yhteyttä HTML kanssa tai sen mahdollistamaa tietoturva-aukkojen hyödyntämistä (Wirfs-Brock ja Eich 2020). JavaScriptissä on myös huomattavia määriä muita erikoisuuksia, kuten kokonaislukujen puute tai taulujen joustavuus, joka johtaa tauluissa ilmeneviin aukkoihin (Rauschmayer 2014).

3.4.6 Python

Python on kehitetty 1990-luvun alussa Stichting Mathematisch Centrum -tutkimuslaitoksella Guido van Rossumin toimesta ABC-kielen seuraajaksi (Rossum ja Drake 2002). Rossum piti ABC:sta, mutta siihen oli mahdotonta tehdä laajennoksia eräiden puutteiden korjaamiseksi. Lisäksi Rossum tarvitsi erillisen kielen työtään varten Amoeba-käyttöjärjestelmän parissa. Tähän tarpeeseen hän kehitti skriptikielen, joka muistutti syntaksiltaan ABC-ohjelmointikieltä ja jolla pääsi käsiksi Amoeban järjestelmäkutsuihin (*Python 3.9.6 documentation* 2021). Tämän jälkeen Python on kehittynyt korkean tason yleiskäyttöiseksi ja dynaamisesti tyypitetyksi ohjelmointikieleksi, joka tukee useampaa ohjelmointiparadigmaa, kuten oliosuuntautunutta, proseduraalista ja funktionaalista ohjelmointia (*Python 3.9.6 documentation* 2021). Tämän lisäksi Python on suunniteltu monelta osin laajennettavaksi kieleksi, jolloin ydin voidaan pitää pienempänä ja yhteiskäyttö muiden kielien kanssa on mutkatonta (*Python 3.9.6 documentation* 2021).

Pythonille ei vaikuta olevan kattavaa kritiikkiä tutkimuskirjallisuudessa tai laajoissa kyselytutkimuksissa, mutta jotkin seikat toistuvat yleisissä keskusteluissa. Ensimmäinen kritisoitu suunnitteluvalinta on dynaaminen tyypitys, joka voi lisätä tyyppivirheiden määrää, hidastaa ajamista ja vaatii tietokoneelta enemmän muistia. Myös ajonaikainen kääntäminen lisää muiden ajonaikaisten virheiden määrää, mikä lisää testaamiseen kulutettavaa aikaa. Pytho-

nin muistin kulutuksesta ja roskienkeruusta sanotaan että Python kuluttaa paljon muistia ja kehityksen aikana kulutusta on seurattava erittäin tarkasti.

Pythonin näkyvyysalueita on myös kritisoitu ja ne ovat käyneet läpi useita muutoksia dynaamisesta staattiseen. Näkyvyysalueiden väliset interaktiot ovat saaneet kritiikkiä sisemmästä alueesta ulompaan tehtävien muutosten hankaluudesta. Syntaksista taas kritisoidaan sisennyksen käyttöä lohkojen erottelussa, joka suurissa tiedostoissa tai pitkillä koodiriveillä voi johtaa hankalaan hahmottamiseen tai sisennykseen liittyvien virheiden pitkään kestävään etsimiseen.

Myös Pythonin versioita kritisoidaan, koska Python 3 ei ole täysin yhteensopiva Python 2:n kanssa, jolloin kehittäjät joutuvat ylläpitämään ohjelmakoodia useissa versioissa, mikäli eivät halua manuaalisesti päivittää kaikkea Python 2 koodia Python 3:een. *Python 3.9.6 documentation* (2021) mukaan Python 2.7 koodi on vielä muunnettavissa toimimaan Python 3 koodin kanssa kohtalaisen vähällä vaivalla, mutta sitä aiempien versioiden koodin muuntaminen on hankalaa. Osa pienemmistä julkaisuista, kuten Python 3.5 ja 3.7 eivät myöskään ole keskenään täysin yhteensopivia ja sovellukset voivat vaatia porttaamisen uuteen versioon.

Pythonin suosittuja ominaisuuksia ja menestyksen tekijöitä on kartoitettu tarkemmin seuraavassa alaluvussa.

3.5 Syyt Pythonin menestykselle

Pythonin menestys on tilastojen mukaan suurin ilmiö lähimenneisyydessä viimeisimmän kymmenen vuoden ajalta, joten tarkastelemme sitä laajemmin ja useammasta näkökulmasta kuin muita kieliä. Pyrkiessämme löytämään syitä Pythonin menestykselle, sen sijaan että aloitettaisiin tutkiminen kielen ominaisuuksista, keskitytään enemmän siihen miksi oikeat kehittäjät ja yritykset ovat valinneet Pythonin. Näin saadaan mahdollisesti paremmin selville niitä syitä, jotka ovat todellisuudessa vaikuttaneet kielen nousuun.

3.5.1 Yritysten kommentit Pythonin sivuilta

Pythonin omilla web-sivuilla (“Quotes about Python” 2021) on kerätty eri yritysten henkilöstöjen kommentteja Pythonin käyttökohteista, sen toimimisesta yrityksen käytössä, sekä ennenkaikkea siitä miksi Python on valittu tai miten se eroaa muista kielistä. Pythonin omilta sivuilta kerätty tieto on luonnollisesti puolueellista ja voidaan olettaa että sivustolle valitut kirjoitukset pyrkivät esittämään Pythonia hyvässä valossa, mutta koska tiedetään jo ennakkoon Pythonin menestys, niin on järkevää keskittyä juuri niihin asioihin mitkä ovat mahdollistaneet kyseisen aseman. Kommentteja on listattu 12:sta yrityksestä 14:lta ammattilaiselta. Liitteessä 2 on listattu kaikki sovelluskehitykseen jollain tavalla liittyvät termit joita kommenteissa on esiintynyt.

Syitä Pythonin käytölle oli lyhyissä kommenteissa yhteensä 31 kappaletta. Osassa syitä oli useita osatekijöitä, osa syistä on keskenään samoja ja osa syistä voidaan luokitella littyvän useampaan kuin yhteen alueeseen. Syitä voidaan ryhmitellä seuraaviin alueisiin: suorituskyky, tuotannon tehokkuus, skaalautuvuus, monikäyttöisyys, yksinkertaisuus ja kirjastot, joista selkeimmin esiin nousevat tuotannon tehokkuus ja monikäyttöisyys

Lyhyiden kommenttien lisäksi koettiin paremman kokonaiskuvan saavuttamiseksi tarpeelliseksi kerätä myös pidempiä lausuntoja Pythonin käytöstä organisaatioissa ohjelmisto- ja sovelluskehityksessä. Kattavin kokoelma löydettiin myös Pythonin omilta websivuilta (“Python Success Stories” 2021). Näissä artikkeleissa kuvataan yrityksien toimintoja ja sitä, miten Pythonia niissä käytetään ja miten se erottuu muista käytetyistä kielistä. Kertomuksista on valittu kymmenen parhaiten sovelluskehitykseen liittyvää tekstiä, joista on kerätty syitä sille miksi Python on parempi kuin kilpailijansa. Kaikki kerätyt syyt on listattu liitteessä 3. Seuraavissa alaluvuissa on avattu sekä lyhyemmistä lausunnoista löytyneitä alueita, että niihin liittyviä yksityiskohtia pidemmistä teksteistä.

3.5.2 Tuotannon tehokkuus Pythonilla

Kaikkein useimmin ja selkeimmin Pythonia käyttävien yritysten kommenteissa esiintyi se, että Pythonia käytettäessä saadaan erittäin nopeasti tuotettua ohjelmistoja tai sen osia alusta ylläpitovaiheeseen asti. Tämän lisäksi on mainittu, että ylläpito vaatii vähän resursseja.

Kommenteissa myös painotetaan että Pythonilla kehittämällä tiimien koot tai yksittäisten kehittäjien määrä voidaan pitää pienenä. Erään yrityksen kommentissa jopa käytetään vertausta, että Pythonilla ohjelmoiminen on 10 kertaa tehokkaampaa kuin Javalla ja sata kertaa tehokkaampaa kuin C:llä. Toinen yritys taas kuvaa koodiin käsiksi pääsemisen, buildauksen ja testaamisen kestävän vain päiviä, sen sijaan että nämä toimenpiteet kestäisivät kuukausia muilla vaihtoehdoilla. Tarkempana yksityiskohtana yksi yritys on maininnut automaattisen roskankeruun joka nopeuttaa kehitystä C++ verrattuna.

Myös pidemmissä kertomuksissa tuotannon tehokkuus oli huomattavalla marginaalilla selkein teema. Siihen on jollain tavalla viitattu yhdeksässä kertomuksessa kymmenestä. Näissä teksteissä tuotannnon tehokkuutta perusteltiin edellisten kohtien lisäksi ohjelmoinnin selkeydellä, helppoudella ja ilmaisuvoimaisilla korkean tason rakenteilla.

3.5.3 Pythonin monikäyttöisyys

Toinen teema mikä jätetyistä kommenteista käy ilmi, on Pythonin joustavuus ja soveltuvuus moniin erilaisiin tilanteisiin. Tämä tulkinnanvarainen kuvaus pitää allaan monta ohjelmointikielen ominaisuutta, myös edellä käsitelty tuotannon tehokkuus voidaan jollain tavalla nähdä liittyvän joustavuuteen. Kommenteista suurin esille nouseva joustavuuden alue on skaalautuvuus. Kielen sanotaan soveltuvan sekä pienten että suurten sovelluskehitystiimien käyttöön pieniin sekä suuriin projekteihin. Yhdessä kommentissa on erikseen mainittu että ohjelmakoodin ylläpitoon tarvitaan vain vähän henkilöstöä. Toisessa kommentissa eritellään että Pythonilla toteutetussa pelissä on mahdollisuudet yli 50 000:lle samanaikaiselle pelaajalle. Kolmannessa kommentissa kerrotaan projektin kasvavien koodimäärien hallitsemisen olevan yksinkertaista. Skaalautuvuuden lisäksi kaksi kommenttia kehuu kielen monia käyttökohteita, sekä sitä miten Python laajentaa ohjelmien mahdollisuuksia ja nitoo niitä yhteen. Lisäksi on maininta Pythonin järjestelmäriippumattomuudesta. Joustavuuteen voidaan myös katsoa kuuluvaksi kolme kommenttia, jotka käsittelevät Pythonin skriptausmahdollisuuksia, sekä korkean tason kielen ominaisuuksia.

3.5.4 Pythonin muut ominaisuudet

Osassa kommenteista huomioitiin myös Pythonin tehokkuus, tai pääasiassa se että Python-ohjelmakoodin ajaminen on riittävän nopeaa. Edellä mainittiin esimerkki 50 000:sta samanaikaisesta pelaajasta, joka osaltaan kertoo myös siitä, että ohjelmien ajamisen pitää toimia riittävän nopeasti. Muita konkreettisia esimerkkejä yritysten edustajilla ei Pythonin tehokkuudesta ole, vaikka kielen tehokkuus on selkeästi mainittu neljässä kommentissa, joten kommentteihin voi suhtautua varauksella.

Pythonin selkeys ja yksinkertaisuus mainitaan useassa kohdassa. Nämä kuvaukset voidaan jakaa kahteen kategoriaan. Osa kuvauksista liittyy Pythonin syntaksin kuvaamiseen ytimekkääksi, helppolukuiseksi ja nopeasti opittavaksi. Lopuissa kuvataan kielen kieleen liittyvien käytännön toimien, kuten refaktoroinnin ja ylläpitämisen olevan hyvin selkeitä ja helppoja tehtäviä.

Pythonin kirjastot mainitaan kahdessa eri yhteydessä. Ensimmäisessä Pythonin peruskirjastoa kehutaan tuottavuutta tehostavaksi, kun taas toisessa mainitaan laaja web-sovelluskehitykseen liittyvien kirjastojen määrä.

3.5.5 Selittäviä tekijöitä osa-alueiden suosiolle

Pythonin suosituille osa-alueille on mahdollista löytää paljon erilaisia selityksiä ja lisäksi osaa niistä voidaan selittää toisella ylemmän tason kategorialla; esimerkiksi lienee selvää, että Pythonilla saavutettavan tuotannon tehokkuuden yksi osatekijä on sen yksinkertaisuus, varsinkin kokemattomien kehittäjien käsissä. Haluttujen konseptien siirtäminen päästä editoriin syntaktisesti oikeelliseksi rakenteiksi nopeutuu, kun logiikka ja syntaksi on selkeää. Seuraavassa pyritään etsimään käyttökokemuksissa esiintyviin väitteisiin perusteluja. Mahdollisuuksien mukaan perusteluja verrataan kilpailijoiden ratkaisuihin ja tutkitaan, onko kielien välillä selkeitä eroja lähestymistavoissa tai toteutuksissa.

Monessa artikkelissa esiintyvä tuotantoa tehostava tekijä on kielen yksinkertaisuus. Yksinkertaisuus ja sen tuomat edut ilmenevät teksteissä muutamissa muodoissa, joiden voidaan ajatella kaikkien vaikuttavan ohjelmakoodin kirjoittamisen ja ylläpidon nopeuteen. Yksinkertaisuuden mittaaminen ja vertaaminen muihin vaihtoehtoisiin kieliin on hankalaa asian

subjektiivisen luonteen takia, mutta sen tekijöitä voidaan silti yrittää vertailla lausunnoista esille nousevien seikkojen kautta. Syntaksin selkeys, ytimekkyys ja ilmaisuvoimaisuus toistuvat teksteissä. Ohjelmakoodin selkeys tehostanee oman ja vertaisen tuottaman koodin tarkastelun nopeutta tuotanto- sekä ylläpitovaiheessa, erityisesti jos koodia kirjoittaa tai tulkitse joku muu kuin päätoiminen ohjelmoija. Ytimekkyys ja ilmaisuvoimaisuus voivat myös nopeuttaa koodin kirjoittamista ja lukemista, koska rakenteita kirjoitetaan tiiviillä, mutta kuitenkin helposti ymmärrettävällä kielellä; kommenteissa on mainittu useaan kertaan syntaksin muistuttavan helposti tulkittavaa pseudokoodia. Yksinkertaistavana tekijänä on myös mainittu pelkkien sisennyksien käyttäminen koodilohkojen merkintään, mitä muissa moderneissa suosituissa kielissä ei yksinään käytetä.

Yksinkertaisuus tukee tuotannon tehostamisen lisäksi myös joustavuutta, kun oppilaat ja itsenäisesti ohjelmointia opiskelevat aloittavat ohjelmoinnin opiskelun yhä todennäköisemmin Pythonista. Itsenäisesti opiskellessa halutaan usein opiskella suosittua kieltä, jossa valmiiden ohjelmien tuottamisen kynnys on mahdollisimman pieni, eli yksinkertaisuudella on suuri rooli. Joustavuuden lisäksi riittävän yksinkertaisuuden voidaan nähdä palvelevan myös oppilaitoksia, jotka haluavat opettaa riitävän yksinkertaisia kieliä, jotta ohjelmointilogiikan oppiminen on riittävän nopeaa.

Joustavuutta lisää myös Pythonin yhteentoimivuus muiden kielien kanssa. Muissa kielissä on myös menetelmiä joilla saavutetaan laajaa yhteentoimivuutta, mutta vaikuttaa siltä että kehittäjät näkevät Pythonin helppokäyttöisenä vaihtoehtona; Bissyande ym. (2013) tutkimuksessa Python on JavaScriptin ja Rubyn jälkeen kolmanneksi yleisin kieli monikielisissä projekteissa. Useammalla artikkelien yrityksistä oli ennakkoon arveluksia Pythonin suorituskyvystä, tai kehityksen aikana tuli vahvistus siitä että osia ohjelmistosta pitää kehittää jollakin matalamman tason kielellä suorituskyvyn parantamiseksi, mikä myös onnistui helposti. Pythonin luoja Guido van Rossum selittää esimerkiksi Rossum (1998) tekstissä, että on Python kehitetty ja kehitetään vartavasten käytettäväksi muiden kielien kanssa, varsinkin suorituskykyä vaativissa tilanteissa ja muiden kielien kirjastojen ja komponentteja hyödynnettäessä. Pythonin käytettävyys muiden kielien kanssa näkyy selkeästi myös Python-wikissä, jonne on kerättynä pitkä lista resursseja muiden kielien integroimiseen liittyen (“Python integration wiki” 2021). Muilla ohjelmointikielillä ei selvityksen perusteella näytä olevan vastaavanlai-

sia yhteisöresursseja.

Yritysten lausunnoissa kattaviksi ja yksinkertaisiksi kuvatut kirjastot voivat olla osasyy kaikille havaituille tekijöille. Mikäli kaikkiin uusiin tarpeisiin löytyy nopeasti yksinkertainen mutta luotettava ja tehokas kirjasto, voi tuotannon nopeus kasvaa erittäin paljon. Kirjastojen käyttö voi myös yksinkertaistaa työtehtäviä, joten jos monimutkaisiin ongelmiin löytyy helposti vastaus valmiista kirjastosta, voi kokemattomampikin ohjelmoija pystyä toteuttamaan vaaditun ohjelmiston ominaisuuden. Vastaavasti skaalautuvuuden kannalta kirjastojen olemassaolo mahdollistaa pienemmiltäkin tiimeiltä ohjelmistoja, joiden tekeminen alusta asti olisi ollut saatavilla resursseilla mahdotonta. Kirjastojen osalta ohjelmointikielien vertailu on kuitenkin hankalaa. Kun vertaillaan suurimpia kieliä keskenään, kirjastojen kattavuuden vertailu ei välttämättä ole järkevää, sillä jokaisella kielellä on laajat peruskirjastot sekä määrättömästi kolmannen osapuolen kirjastoja. Tämä tarkoittaa siis sitä, että erot kielten välillä liittyvät käytettävyyteen ja selkeyteen, josta ei vaikuta olevan olemassaolevaa kattavaa tutkimusta. Kuitenkin jo aiemmin mainittu yhteentoimivuus muiden kielien kanssa takaa sen, että Pythonilla myös muiden kielien kirjastojen käyttäminen on yksinkertaista.

Yhteenvetona Pythonin suosittujen osa-alueiden tekijöiden selvittämiselle voidaan sanoa, että eniten esille tulevat ominaisuudet ovat niitä, jotka ovat subjektiivisia ja joita on hankala vertailla muiden kielien ratkaisuihin. Tämänlaisiin osakysymyksiin vastaamiseen kysely- tai haastattelututkimuksen järjestäminen voisi olla tehokkain tapa saada laadukkaita vastauksia, mutta siihen eivät ajalliset resurssit tämän tutkimuksen osalta riitä. Suurimmaiset tekijät jatkuvasti kasvavalle suosiolle vaikuttavat olevan kielen ominaisuuksien ja monikäyttöisyyden yksinkertaisuus ja selkeys, jotka taas edelleen vaikuttavat moneen muuhun kehuttuun piirteeseen.

3.6 Vähenevän käyttömäärän kielet

Seuraavaksi käymme läpi vähenevän käyttömäärän kieliä. Lähtökohtana tämän kategorian kielissä on se, että käyttäjiä on jo ollut olemassa huomattava määrä, jonka jälkeen niiden määrä on vähentynyt. Jos tehdään oletus että käyttäjä ei lopeta ohjelmointia kokonaan sen jälkeen kun lopettaa kyseenomaisen kielen käytön, tarkoittaa tämä sitä että käyttäjä siirtyy

uuteen kieleen, tai keskittyy enemmän toiseen, jo itsellään käytössä olevaan kieleen. Ilman syytä kielestä pois vaihtaminen ei liene merkittävän laaja ilmiö, koska siinä kehittäjä joutuu investoimaan erilaisia resursseja, kuten kielen opetteluun kuluvan ajan ja mahdollisesti uuden tuotantoympäristön asentamisen ja opettelemisen.

Tässä osiossa pyritään etsimään jokaisen tähän kategoriaan sijoitetun kielen osalta yksittäisiä tekijöitä, jotka ovat saattaneet vaikuttaa laskevaan käyttömäärään. Samalla etsitään myös tekijöitä, jotka osaltaan tulevat vielä tulevaisuudessakin pitämään kieltä käytettynä. Tämä osio on suurelta osin luonteeltaan spekulatiivinen, koska ilmiöt ovat monitulkintaisia, täysin varmoja syy-seuraus -suhteita on vaikea muodostaa ja kielestä riippuen tutkimuskirjallisuutta aiheesta ei välttämättä ole. Tämän takia syiden selvittämisessä joudutaan tyytymään kirjoittaneen omaan päättelyyn ja tulkintaan, joka perustuu omaan ammattitaitoon ja internetissä laajasti esiintyviin perusteltuihin mielipiteisiin.

3.6.1 Visual Basic (.NET)

Visual Basic .NET (jatkossa vain Visual Basic) on alkuperäisen Visual Basicin 6.0 version seuraaja, joka on julkaistu vuonna 2002 nimensä mukaisesti .NET ympäristöä varten. Visual Basicin pääasiallinen käyttökohde on työpöytäsovellusten yksinkertainen ja nopea toteuttaminen (“Visual Basic documentation” 2021).

Visual Basic muuttui kielen elinkaaren aikana alun ”harrastelijakielestä” lähemmäksi C#:ia erilaisten päivitysten myötä, kun taas samoihin aikoihin C# sai sovelluskehitystä helpottavia päivityksiä, jolloin siitä tuli Visual Basicin tavoin helppokäyttöinen kieli. Tämä johti siihen että molemmat kielet kilpailivat samoista käyttäjistä. ”Harrastajakielen” asemaan markkinoille on kehitetty myös paljon uusia kieliä, joiden kehitys ja tuki on ollut aktiivista. Sen sijaan Visual Basicin jatkokehitys on lopetettu lähes kokonaan (Torgersen 2017).

Visual Basicin jatkokehityksen loppuminen ei kuitenkaan tarkoita että kaikki sen tuki .NET kehyksessä olisi loppunut, vaan kielellä voi myös jatkossa luoda joillakin versioilla sovelluksia uusimmilla työkaluilla. Visual Basic for Applicationsilla (VBA) on myös kirjoitettu historian saatossa Microsoft Office -tuotepaketin työkaluihin paljon makroja jotka ovat toiminnassa ja joita päivitetään edelleen. Tämä tarkoittaa sitä että kielelle löytyy edelleen

käyttöä, mikä näkyy myös kielien hakumääriä vertailevissa tilastoissa.

3.6.2 Perl

Perl on vuonna 1987 kehitetty monikäyttöinen ja ilmaisuvoimainen ohjelmointikieli, joka on alunperin kehitetty skriptikieleksi erilaisiin tekstinkäsittelyn tarpeisiin. Perlin versio 6 on irtaantunut omaksi kielekseen ja se tunnetaan nykyisin nimellä Raku (“Perl website” 2021).

Perlin kahtia jakautuminen ja kehityksen hidastuminen on ollut ainakin osittain syyllinen siihen, että kielen käytön houkuttelevuus on vähentynyt. Perl 6 on ollut kehitteillä jo vuodesta 2000 ja ensimmäinen virallinen kokonainen julkaisu tapahtui vasta 2015 lopulla. Perl 5 jatkokehitys on jatkunut vuosien mittaan, mutta seuraava merkittävä päivitys versioon Perl 7 tapahtuu mahdollisesti vasta vuoden 2021 aikana, yli 20 vuotta version 5 julkaisun jälkeen.

Toinen Perlin houkuttelevuutta heikentävä seikka on melko yleinen konsensus siitä, että avoimissa resursseissa liikkeellä oleva Perl-koodi on usein laadultaan heikkoa. Tämä johtuu osittain siitä, että Perl ei ole turvallinen kieli kirjoittaa, vaan kehittäjä voi toteuttaa syntaktisesti toimivia ohjelmia käyttäen erittäin huonoja käytänteitä. Myös koodin luettavuuden kärsiminen mainitaan huonojen käytänteiden noudattamisen seurauksena.

Vaikkei Perlin ja nykyisen Rakun kehitys ole ollut nopeaa, sitä kuitenkin tapahtuu eikä Perlin tuki ole loppumassa. Lisäksi Perlille on edelleen omat käyttökohteensa varsinkin tekstinkäsittelyssä skriptikäytössä ja sillä on melko harras käyttäjäkunta. Tämän takia ei ole syytä odottaa että lähitulevaisuudessa Perl katoaisi kokonaan ohjelmointikielien joukosta.

3.6.3 Delphi

Delphi on vuonna 1995 julkaistu ohjelmistotuote johon kuuluu kehitysympäristö sekä ohjelmointikieli, joka on Object Pascalin dialekti. Delphin pääasiallinen käyttötarkoitus on tehokas ohjelmistotuotanto monille eri alustoille (“Delphi product website” 2021).

Syitä kielen käytön ja hakumäärien vähenemiselle löytyy joitakin. Työpaikat joissa kirjoitetaan Delphillä ja haetaan Delphi-osaajia ovat vähentyneet. Myös Delphin opetus sekä tuoreiden oppimateriaalien määrä on vähäinen. Kuten tutkielmassa on jo aiemmin mainittu, tämä

riittää ylläpitämään kielen käyttömäärän vähenemisen kehää.

Yksi tekijä ohjelmointikieliin liitettyjen ympäristöjen käyttö- ja myyntimäärissä saattaa olla ohjelmistojen ilmaisversiot, joilla kehittäjät saavat käyttää tuotantoympäristöä riittävästi ennen yksityistä tai yrityksen ostopäätöstä. Käyttömääriltään suuremmat ympäristöt ovat tarjonneet näitä vaihtoehtoja jo pitkään, kun taas Delphi toi ensimmäisen oman ilmaisversionsa saataville vasta vuonna 2018. Spekuloinnin tasolle jää, että Delphin tämänhetkinen käyttäjien määrä voisi olla suurempi, mikäli muutos olisi tapahtunut aiemmin. Lisäksi ilmaisia, varsinkin yksityiskäyttöön tarkoitettuja Delphin kaltaisia ympäristöjä, kuten nykyinen Lazarus, on ollut saatavilla jo pitkään.

Delphi on edellä mainituista asioista huolimatta tehokas kieli ja ympäristö kehittää sovelluksia yrityskäyttöön, jotka ovat tehokkuudeltaan erittäin kilpailukykyisiä. Delphiä kehitetään edelleen ja päivitykset ovat melko säännöllisiä. Lisäksi Delphillä on vannoutunut käyttäjäkunta, sekä kielen ja ympäristön oppimista luonnehditaan helpoksi.

3.6.4 PHP

PHP on vuonna 1995 julkaistu yleiskäyttöinen skriptikieli, joka pohjautuu kokoelmaan C:llä kirjoitettuja CGI-skriptejä. Sitä käytetään yleensä dynaamisten web-sivujen luomisessa, PHP-skriptit palvelinpuolelle sijoittaen (“PHP website” 2021).

Osasyy PHP:n suosion laskemiseen ovat lukuisat tietoturvariskit, jotka vaivaavat kielen mainetta. Esimerkiksi vielä vuoden 2002 aikana käyttäjät pystyivät sijoittamaan muuttujia sivujen PHP-skripteihin, jos asetukset oli jätetty oletusarvoihin. Nykyisin tilanne on hieman samankaltainen esimerkiksi aiemmin mainitun Perlin kanssa, eli kieli mahdollistaa huonoja ja turvattomia käytänteitä, joka vaatii kehittäjältä tarkkuutta.

Toinen syy on PHP:n versioiden välinen yhteensopimattomuus ja versioiden sisällä funktioiden suunnittelun ja yhteneväisyyden puutteet. Esimerkiksi samoja asioita tekeviä funktioita on useita ja käytettävien parametrien järjestys vaihtelee funktioiden välillä.

Yksi syy muiden web-kehityksessä käytettävien kielien suosioon ovat uudet kirjastot ja kehykset, joiden osaajia haetaan työmarkkinoilla. PHP:lla vastaavaa ilmiötä ei ole näkyvissä

ja suurin osa kielen työilmoituksista listaa PHP:n osaamisen vain yksittäisenä kriteerinä tai plussana muiden joukossa.

Vaikka PHP:n kilpailu on vaikeaa uusien web-kehitykseen erikoistuneiden kielien ja kirjastojen kanssa, edelleen erittäin monien web-sivujen toiminnallisuus on toteutettu PHP:llä. Joidenkin arvioiden mukaan jopa 4/5 kaikista kartoitettavista web-sivuista käyttää ainakin osittain PHP:tä. Kieli saa myös uusia päivityksiä, jotka korjaavat puutteita jatkuvasti, esimerkiksi kielen suoritusteho on saanut useita päivityksiä ja uusimmat versiot ovat lisänneet kielen tehokkuutta huomattavasti. Kielen laajasta käytöstä ja pitkästä historiasta johtuen myös yhteisön koko on suuri. Kaiken kaikkiaan vaikka historialliset stigmat vaivaavat kielen mainetta, sen käyttäminen jatkunee pitkälle tulevaisuuteen.

3.7 Nousevat kielet

Seuraavaksi käydään läpi käyttö- ja hakumäärältään kasvussa olevat, melko uudet kielet ja pyritään löytämään yhteisiä tekijöitä sille, miksi ne ovat nousseet käytetyiksi. Kielistä on pyritty selvittämään kehittäjä, yksittäisiä suurimpia syitä kielen kehittämiselle, eniten vaikuttaneet kielet ja tärkeimmät eroavaisuudet niihin, sekä pääasialliset käyttökohteet. Nämä tiedot on kirjattu tiiviissä muodossa havaintojen tekemisen ja vertailun yksinkertaistamiseksi.

3.7.1 Kotlin

Kotlin on JetBrainsin kehittämä avoimen lähdekoodin ohjelmointikieli, jonka 1.0-julkaisu tapahtui vuonna 2016. Se toimii Java-virtuaalikoneella, mutta tarvittaessa kääntyy muun muassa myös JavaScriptiksi ja natiiviksi koodiksi. Kotlin on kehitetty uudeksi yleiskäyttöiseksi kieleksi, joka tarjoaa joitakin parannuksia erityisesti Javaan verrattuna, säilyttäen kuitenkin yhteensopivuuden Javalla kirjoitettuun koodiin (Breslav 2016).

Kotlinin suunnittelussa on painotettu pragmaattisuuden lisäksi yhteentoimivuutta, turvallisuutta, selkeyttä ja työkalujen tukemista. Kielessä on yhdistetty funktionaalisen ohjelmoinnin sekä skriptikielen ominaisuuksia oliosuuntautuneeseen kieleen. Kotlinia kuvataan Javaa ilmaisuvoimaisemmaksi kieleksi ja siinä on parannettu tyhjäarvoihin liittyvää turvallisuutta tukemalla ”non-nullable” tyyppejä kielessä (Breslav 2016).

Viime vuosina Android-kehitys on ollut suuri kohde Kotlinin käytölle. Vuodesta 2017 Google on pitänyt Kotlinia yhtenä tärkeimmistä kielistä Android-kehityksessä ja on tukenut sen toimintaa, kun taas 2019 Google ilmoitti Kotlinin olevan ensisijainen kieli Android-kehitykseen (Grin 2020).

3.7.2 TypeScript

TypeScript on kehitetty Microsoftin toimesta ja ensimmäinen avoin versio siitä julkaistiin vuonna 2012. Pääasiallinen syy TypeScriptin kehittämiselle oli saada JavaScript-kehitykseen staattisen tyypityksen tuomia ominaisuuksia. TypeScriptin tavoite ei ole olla kokonaan uusi ohjelmointikieli, vaan tukea ja laajentaa JavaScript-kehityksen mahdollisuuksia. TypeScript on tehty käytettäväksi JavaScriptin tukena tai tilalla kohteissa joissa tyypityksestä on hyötyä, kuten suuremmissa projekteissa joissa helpommasta refaktoroinnista ja virheiden paikannuksesta on erittäin paljon hyötyä kehityksen ja ylläpidettävyyden kannalta (Bierman, Abadi ja Torgersen 2014).

TypeScript kääntyy Javascriptiksi ja on kielenä JavaScriptin kaltainen, mutta lisää syntaksin tyyppien käytölle. Lisäksi JavaScriptin päälle on lisätty muita kielen rakenteita, kuten luokat, moduulit ja lambda-lausekkeet. Tyypityksen vapaaehtoisuuden takia kaikki JavaScript ohjelmat ovat suoraan myös toimivia TypeScript ohjelmia (Bierman, Abadi ja Torgersen 2014).

3.7.3 Rust

Rustin on alunperin suunnitellut Graydon Hoare, myöhemmin projektia on kehitetty Mozillan jäsenten sekä muiden aktiivien toimesta. Kielen 1.0-julkaisu tapahtui vuonna 2015. Rust on monikäyttöinen kieli, jonka pyrkimyksenä on korvata vanhempia matalan tason kieliä. Kielen tavoitteena ovat mahdollisimman hyvä samanaikaisuus, turvallisuus ja muistin hallinta, varsinkin C++:aan verrattuna. Se pyrkii yhdistämään matalan tason suorituskyvyn hallitsemisen yhteen korkean tason helppouden ja turvallisuuden yksinkertaisuuden kanssa (The Rust Core Team 2015).

Syntaktisesti Rust on hyvin samankaltainen C++:n kanssa, mutta muistin käsittelyyn on luotu

automaattisen roskienkeruun sijaan kääntämisen aikana määriteltävä ”omistajuuden järjestelmä” muistin käytön tehostamiseksi. Rustin kääntäjän ”elinaika-tarkastaja” (borrow checker) taas vähentää muistin käytön ongelmia jos muistia käyttää useita säikeitä samaan aikaan. Näiden lisäksi Rust pyrkii siihen, että abstraktiot eivät maksa ylimääräistä laskentatehoa (Matsakis ja Klock 2014).

3.7.4 Dart

Dart on kehitetty Googlella alunperin Lars Bakin ja Kasper Lundin toimesta. Prorjekti esiteltiin vuonna 2011 ja 1.0-versio julkaistiin 2013. Dart on kehitetty tuotantoa tehostavaksi vaihtoehdoksi JavaScriptille web-kehitykseen, myöhemmin sitä on voitu käyttää enenevissä määrin myös muiden alustojen ja kohteiden, kuten palvelinratkaisujen, mobiiliaplikaatioiden ja komentorivisovellusten luomiseen (Bak 2011). Googlen omien tarpeiden lisäksi Dart on suunnattu eri tasoisille kehittäjille monialustakehitykseen. Googlen omista ohjelmistoista Google Ads, AdSense, AdMob, ja Google Assistant käyttävät Dartia (“Dart FAQ” 2021).

Ensimmäisen version jälkeen painopiste on muuttunut ja Flutter-kehyksen avulla Dart-sovellusten monialustakehitys on muuttunut ja tehostunut merkittävästi. Dartin suunnittelussa on pyritty luomaan rakenteinen, mutta joustava kieli web-kehitykseen, tuttu ja luonnollinen kieli oppia, sekä kaikille alustoille tehokas vaihtoehto. Vaikutteita Dart on ottanut muista ALGOL-perheen kielistä ja se muistuttaa niitä syntaksiltaan. Dart kääntyy natiivin kielen lisäksi myös JavaScriptiksi (“Dart overview” 2021).

3.7.5 Go

Kielen ovat suunnitelleet Robert Griesemer, Rob Pike ja Ken Thompson Googlella, ensimmäinen julkinen versio julkaistiin vuonna 2009. Go:n kehittäjät halusivat helpottaa ohjelmistojen kehitystä, parantaa tuottavuutta ja lisätä kieleen moderneja ominaisuuksia, kuten monien prosessorien samanaikaista hyödyntämistä. Googlen työntekijöiden aiemmin käyttämissä kielissä oli aina valittava tehokkaan kääntämisen, tehokkaan ajamisen ja ohjelmoinnin helppouden väliltä (“golang.org documentation” 2021).

Go pyrkii yhdistämään tulkittavan ja dynaamisesti tyypitetyn kielen tuotannon helppouden

ja staattisesti tyypitetyn kielen turvallisuuden, samalla yhdistäen siihen ohjelmien buildaamisen tehokkuuden. Syntaksiltaan Go on C:n kaltainen kieli, johon on tehty joitakin muutoksia ilmaisuvoimaisuuden ja luettavuuden yhdistämiseksi. Lisäksi siihen on lisätty moderneja ominaisuuksia kuten automaattinen roskienkeruu ja muistiturvallisuutta parantavia tekijöitä (“golang.org documentation” 2021).

Go on käytössä Googlen omissa toiminnoissa merkittävissä määrin. Esimerkiksi latauspalvelintoiminnot ovat kokonaan toteutettu Go:lla. Googlen ulkopuolella sitä käytetään varsinkin pilvilaskennassa. Säiliöityjen sovellusten ajamisessa käytettävät Docker ja Kubernetes on molemmat kirjoitettu Go:lla (“golang.org documentation” 2021).

3.7.6 Scala

Kielen kehittivät Martin Odersky ja École Polytechnique Fédérale de Lausanne -tutkimuslaitoksen tutkijat, ensimmäinen julkinen versio kielestä tuli saataville vuonna 2004. Syinä Scalan kehittämiselle Odersky ym. (2006) artikkelissa mainitaan pyrkimys korjata Javan puutteita ja lisätä funktionaalisia ominaisuuksia oliosuuntautuneeseen kieleen. Scalassa jokainen arvo on olio ja funktiot ovat arvoja, jolloin se täyttää molempien paradigmojen määritelmät. Funktionaalisia vaikutteita lukuunottamatta kieli on Javan tyylinen, Javan kanssa suurelta osin yhteentoimiva ja kääntyy Javan tavukoodiksi. Scalan luokat ja oliot voivat esimerkiksi periytyä Javan luokista ja implementoida Javan rajapintoja (Odersky ym. 2006).

Yhteentoimivuus Javan kanssa tarkoittaa että Scalaa voidaan käyttää Javan paikalla tai yhdessä sen kanssa, olemassaolevia Javan kirjastoja hyödyntäen. Useat suuryritykset ovat alkaneet käyttää Scalaa sovelluksissaan vuoden 2006 parannuksien jälkeen. Esimerkiksi Twitter kirjoitti uusiksi yhden osajärjestelmänsä Scalalla vuonna 2008, jonka jälkeen Scalalla on kirjoitettu merkittävä osuus alustan ydinohjelmistoista. Scalaa käytetään muun muassa myös LinkedIn:ssä, Kloutissa, Foursquarella ja Intelillä, joissa sillä on toteutettu osia ydinohjelmistoihin ja web-analytiikkaan (Odersky ja Rompf 2014) .

3.7.7 Julia

Julian kehittivät Jeff Bezanson, Stefan Karpinski, Viral B. Shah ja Alan Edelman MIT:llä, ensimmäinen julkaisu tapahtui 2012. Julian kehittäjät halusivat omia käyttötarkoituksiaan, kuten tieteellistä laskentaa, koneoppimista ja tiedonlouhintaa, sekä yleisiä ohjelmointitehtäviä varten kaikki hyödyllisimmät ominaisuudet useista kielistä. Tämä tarkoitti avoimen lähdekoodin kieltä vapaalla lisenssillä, C:n nopeutta, Lisp-tyylisiä makroja ja lähdekoodin muodostumista kielen omista tietorakenteista, Matlabin tyylistä selkeää matemaattista notaatioa ja Pythonin yleisiin ohjelmointitehtäviin soveltuvuutta, sekä monia muita ominaisuuksia ja piirteitä. Kielen vakaiden julkaisujen jälkeen suuret yritykset vaikuttavat käyttävän Juliaa juuri tieteellistä ja teknistä laskentaa vaativissa sovelluksissa (Bezanson ym. 2012a).

Juliaa ei ole kehitetty minkään kielen pohjalta tai sen kehittäjät eivät ole maininneet täysin suoraan ottaneensa suuria määriä teknisiä vaikutteita muista kielistä; inspiraatiota on otettu edellä mainittujen korkean tason tavoitteiden ja teknisten yksityiskohtien muodossa. Ehkä eniten Juliassa näkyvät yhteydet Lisp-dialekteihin, joihin on viitattu Bezanson ym. (2012b) artikkelissa useaan kertaan. Modernien dynaamisten kielien tekniikoiden avulla Julia saavuttaa samalla staattisesti käännettävän kielen tehokkuuden ja Pythonin, LISP:in ja Rubyn kaltaisen tuotannon tehokkuuden, sekä interaktiivisen ja dynaamsen käyttäytymisen. Julian kehittäjät näkevät kielen suosion perustuvan siihen, että kieli ratkaisee tuotannon tehokkuuden, ilmaisuvoiman ja kielen tehokkuuden välisen valinnan ennennäkemättömän hyvin (Bezanson ym. 2012b).

3.8 Kielen korvaaminen

Esiteltyjen kolmen kategorian lisäksi eriytetään myös neljäs kategoria, johon kuuluvat kielet joiden suhdanteet selittyvät ainakin osittain sillä, että olemassaoleva kieli on päätetty korvata uudella kielellä.

3.8.1 Swift ja Objective-C

Objective-C on yleiskäyttöinen oliosuuntautunut ohjelmointikieli, jonka pohjana toimii C ja jonka oliomalli pohjautuu Smalltalkiin. Kieli on kehitetty vuonna 1984 ja yrityskauppojen

myötä vuonna 1996 se päätyi Applen omistukseen. Esimerkiksi Mac OS X -käyttöjärjestelmä on suurelta osin toteutettu Objective-C:llä. Vuoteen 2014 asti se oli ainoa Applen tukema ohjelmointikieli (DeVoe 2011).

Swift on Applen itsensä kehittämä kieli, joka on julkaistu vuonna 2014. Kieli kehitettiin epäsuoraksi korvaajaksi Objective-C:lle, jota ei ole historiansa aikana uudistettu merkittävästi. Swift on Objective-C:hen verrattuna modernein standardein helpompilukuinen, siinä on mahdollistettu monialustakehitystä ja helpotettu muistinhallintaa. Näiden lisäksi Swift ei rakennu C:n päälle, joten kielessä ei ole muun muassa erillisiä otsikkotiedostoja tai C:n kaltaisia osoittimia, joista esimerkiksi jälkimmäinen nopeuttaa vianetsintää ja helpottaa tietoturvan kehittämistä (Garcia ym. 2015).

Sitä mukaa kun Swift on kypsynyt vakaammaksi ja monipuolisemmaksi kieleksi ja kerännyt yhteisöä, vähenevät kehittäjien syyt Objective-C:n käyttämiselle Applen alustojen sovellusten kehittämisessä. Objective-C:n yhteisö, kolmannen osapuolten kirjastojen valikoima, ylläpidettävät ohjelmistot ja yhteentoimivuus esimerkiksi C:n ja C++:n kanssa pitävät kieltä käytettynä vielä lähitulevaisuudessa, mutta siirtymä on selkeä. Kirjoitushetkellä Apple ei ole antanut lausuntoa tuen loppumisesta.

3.9 Käyttömäärien suuntausten syitä

Tässä alaluvussa kokoamme edellisissä alaluvuissa esiintyneet erilaiset syyt kielten suosion ja käyttömäärien laskemiselle ja kasvamiselle ja lyhyesti pohdimme niihin liittyviä asioita.

3.9.1 Syyt kielien käytön kasvamisen takana

Edellisistä luvuista havaitaan, että kaikilla nousevilla kielillä yhteinen tekijä on niiden takana oleva suuri yritys tai tutkimuslaitos, kuten esimerkiksi Google, Microsoft ja École Polytechnique Fédérale de Lausanne. Yhteistä on myös kielen paremmuus joillakin osa-alueilla johonkin vertailtavaan kieleen nähden, kuten Kotlinin Android-kehitykseen tekemät parannukset Javaan verrattuna ja TypeScriptin staattisen tyypityksen tuomat mahdollisuudet JavaScriptiin verrattuna. Kolmas yhteinen tekijä on jokin konkreettinen tarve kielelle. Mozillan tarve modernille matalan tason kielelle ja edellä mainittu tyyppien tarve JavaScriptiin ovat

melko selkeitä esimerkkejä tästä. Näiden lisäksi yhteistä on selkeä ja helppo siirtymä vertailtavasta kielestä, kuten samankaltainen syntaksi tai koodin yhteentoimivuus, kuten Scalan tapauksessa.

Kolme jälkimmäistä yhteneväisyyttä liittyvät toisiinsa melko kiinteästi. Ensin jossakin ympäristössä jollakin käyttäjäryhmällä on käytössä jokin kieli, jossa on ainakin yksi piirre, ominaisuus tai puute josta käyttävät tahot eivät pidä, mistä syntyy tarve uudelle kielelle. Tämän jälkeen kehitetään kieli joka on käytetyllä kohdealueella halutuilta osa-alueilta parempi kuin edellinen, minkä jälkeen siihen voidaan siirtyä luonnollisen siirtymän kautta, varsinkin jos kielestä on tehty yhteensopiva aiempaan kieleen, mikä poistaa vaihtamisesta syntyvää taakkaa. Kielen takana olevan tutkimuslaitoksen tai yrityksen koko voidaan myös usein liittää näihin kolmeen tekijään, koska pienemmillä toimijoilla ei välttämättä ole valmiuksia kokonaan uuden kielen luomiselle omia tarpeita varten. Rust on hyvä esimerkki kuvatusta skenaariosta. Mozillan tukema Graydon Hoare halusi matalan tason ohjelmointikielen, joka korjaa C++:n puutteita kuten muistinhallintaa ja kielen turvallisuutta. Ohjelmoijien siirtymä kehitettyyn Rustiin helpottuu sillä, että se on syntaktisesti samankaltainen C++:n kanssa. Lyhyesti voidaan siis todeta, että menestyviä kieliä ei kehitetä ilman yksityiskohtaista tarvetta ja riittävää määrää resursseja.

3.9.2 Syyt kielien käytön vähenemisen takana

Kielen käytön vähenemisessä on edellä olevissa luvuissa esiintynyt viittä eri tekijää: epäsuotuisa kilpailuasetelma, tuen tai kehityksen puute, negatiivinen stigma, koulutuksen tai työpaikkojen puute, sekä kielen puutteet. Epäsuotuisa kilpailuasetelma on syntynyt kun kielen uusi tai olemassaoleva kilpailija tekee joitakin asioita paremmin, tai kun kohdealueella toimivat kaksi tai useampia kieliä jossakin ajanjaksossa muuttuvat lähemmäksi toisiaan, mahdollisesti kohdealueella vallitsevien tarpeiden mukaan. Tuen ja kehityksen loppumiselle voi olla sitä kehittävältä yritykseltä tai yhteisöltä monia syitä, mutta on myös mahdollista että se on seurausta siitä, että kielen käyttömäärät ovat jo valmiiksi vähenemässä ja kohdealueella on parempia vaihtoehtoja tarjolla. Esimerkiksi Visual Basic ja C# muuttuivat samankaltaisiksi kieliksi, joka todennäköisesti oli yksi Visual Basicin käyttömääriä laskeva tekijä. Tämän jälkeen Microsoft on keskittynyt enemmän C#:n kehittämiseen, joka entisestään vähentää

Visual Basicin käyttäjien määrää.

Kielen puutteet ja kieleen liittyvät stigmat taas liittyvät toisiinsa. Sen lisäksi että puutteet tekevät joidenkin toteutusten tekemisestä mahdotonta tai tarpeettoman vaikeaa, voivat ne mahdollistaa suuressa mittakaavassa huonoja käytänteitä sovellusten toteuttamisessa. Tämä taas johtaa siihen että kieli saa stigmoja sekä kielessä olevien rakenteellisten puutteiden, että sillä toteutettujen puutteellisten ja viallisten sovellusten takia. Aiemmin esitellyistä kielistä nämä kuvaukset pätevät ainakin PHP:n ja Perlin osalta.

Kuten aiemmin luvussa 2 on mainittu, kielen käyttö muun muassa työelämässä voidaan Chatley, Donaldson ja Mycroft (2019) mukaan jakaa ylläpidettävien projektien päivittämiseen ja uusien projektien kehittämiseen. Tutkituista laskevan käyttömäärän kategorian kielistä havaitaan, että varsinkin uusien projektien määrä on vähäinen, jolloin kielen käyttömäärän laskun nopeus riippuu vahvasti siitä, paljonko ylläpidettäviä projekteja kielellä on kirjoitettu. Koska kaikilla tutkituista kielistä on suuria määriä vuosien aikana kirjoitettua koodia, kielien käyttämisen määrä ei tule loppumaan äkillisesti tai lyhyellä aikavälillä, vaikka joihinkin tarpeisiin voisi olla tällä hetkellä tarjolla parempia kieliä.

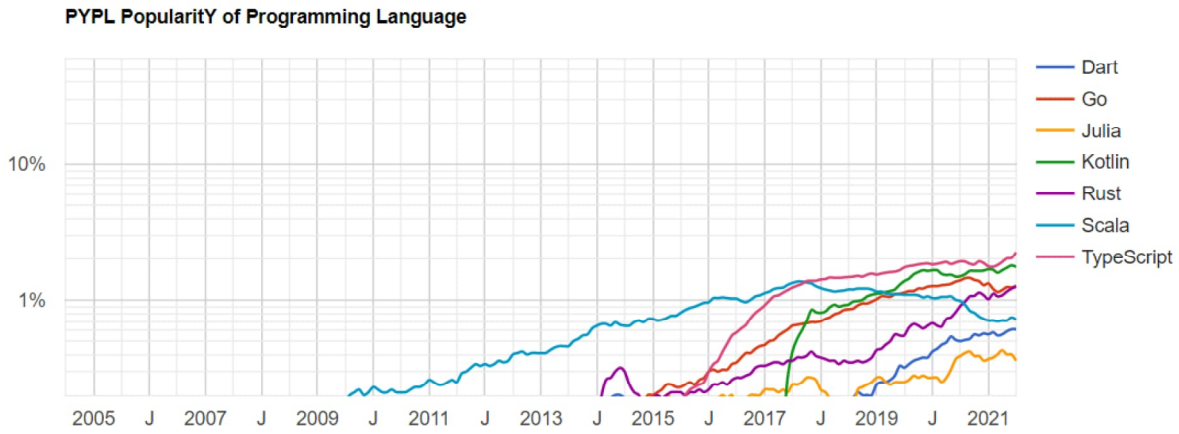
3.10 Kielien suosion kehitys ajan suhteen

“PYPL PopularitY of Programming Language” (2021) vertailee ohjelmointikielien suosiota mittaamalla Googlen hakukoneen ohjelmointikielien tutoriaaleihin kohdistuvien hakujen määrää. Sisällyttämällä “tutorial” hakuihin ohjelmointikielen lisäksi, karsitaan paljon mahdollisia vääriä tuloksia jos kielellä on useita merkityksiä (Java on saari). Yleistä tarkenninta kuten “language” tai “programming” ei myöskään käytetä, koska riippuu paljon kielen nimestä käyttävätkö ihmiset niitä hauissaan (C:tä haetaan paljon tarkentavien sanojen kanssa, kun taas PHP:ta ei). Indeksi päivitetään kerran kuukaudessa, joten muutosten havaitseminen suosion suunnassa on selkeää.

Kun määritämme yksittäiselle kielelle tyypillistä kasvunopeutta ja aikaa jona kieli saavuttaa käyttömääränsä huipun, sekä tyypillistä nopeutta jolla kielen käyttömäärät vähenevät sen jälkeen kun lasku edeltävissä luvuissa mainittujen syiden takia alkaa, tulee ottaa huomioon joitakin seikkoja. Kasvunopeuden mittaaminen käyttäjiä tai kieleen liittyviä hakuja yksikköi-

nä käyttäen eri vuosikymmeniltä menneisyydestä on ongelmallista, koska kehittäjien määrä on kasvanut nykyhetkeä kohti lähestyttäessä, jolloin luvut eivät ole vertailukelpoisia. Toisaalta kasvunopeuden mittaaminen käyttäjien hakujen kokonaismäärästä lasketuilla prosenttiyksiköillä ei ole täysin vertailukelpoinen eri ajanjaksojen välillä ohjelmointikielien jatkuvasti kasvaneen määrän takia. Mittayksikkönä PYPL:ssa on prosenttiosuus kaikista hauista, joten vertailtavien kielien julkaisuajankohdat eivät saa olla liian kaukana toisistaan edellä selvitetyn syyn vuoksi. Indeksi ulottuu vuoteen 2005 asti, joka riittää kohtalaisten uusien ja keskenään vertailukelpoisten kielien tutkimiseen niiden koko olemassaolon ajalta.

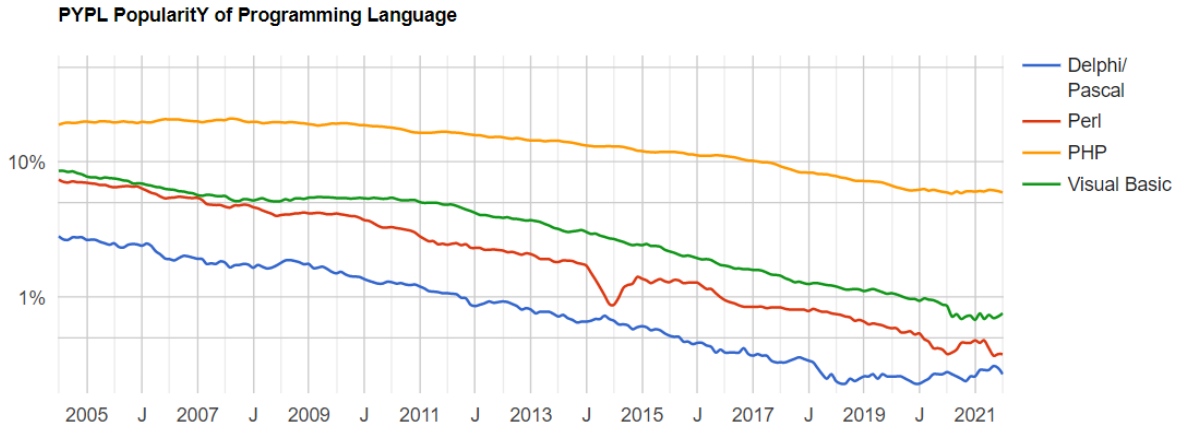
Toiseksi joudutaan määrittelemään se, missä vaiheessa kielen kasvun voidaan tulkita loppuneen ja missä vaiheessa haku -ja käyttömäärien vähenemistä kieli voidaan luokitella vähenevän käyttömäärän kieleksi. Vaikka mitattavat metriikat olisivatkin kääntyneet laskuun, ei voida olla varmoja siitä, etteikö trendi muuttuisi lähitulevaisuudessa takaisin kasvavaksi, jolloin kuvaajalla näkyisi vain vähäinen pudotus keskellä jatkuvaa kasvua. Sama pätee käänteisesti myös laskeviin kieliin. Tutkimalla PYPL-kuvaajia voidaan tehdä karkea oletus, että mikäli kielen hakumäärissä on kahden vuoden ajan jatkuvaa laskua, uutta nousuvaihetta ei tule, koska tälläistä ilmiötä ei ole tapahtunut millään tilastoiduista 28:sta kielestä tilastoidulla ajanjaksolla. Monien vuosien tasaisen tai hieman aaltoilevan kehityksen jälkeinen nousuvaihe vaikuttaa olevan myös hyvin epätodennäköinen muttei mahdoton, kuten C#:n “PYPL PopularitY of Programming Language” (2021) -kuvaajalta voidaan havaita.



Kuva 6: PYPL nousevat kielet

Aiemmin tässä luvussa nouseviin kieliin luokiteltujen kielien suosion kasvamista ei täysin

voida mitata, koska nimensä mukaisesti niiden haku- ja käyttömäärät ovat vielä nousussa, tai eivät ole aloittaneet vielä riittävää laskemista. Tämä näkyy myös PYPL kuvaajalla (kuva 6) kaikkien muiden kielien paitsi Scalan osalta, jolla pelkkien hakujen määrä on laskenut tasaisesti jo vuoden 2017 lopulta. Tästä ryhmästä ei juurikaan voida tehdä muita yleistettäviä havaintoja tai johtopäätöksiä, muuta kuin se että tämän kokoluokan ohjelmointikielillä kasvamisvaiheessa kestää vähintään kaksi vuotta. Logaritmisella asteikolla tutkittuna vähenevän käyttömäärien kielien relevanssin laskun nopeudesta taas voidaan tehdä havainto, että kaikilla neljällä aiemmin mainitulla kielellä lasku on melko lineaarista. (kuva 7)



Kuva 7: PYPL laskevat kielet

3.11 Teoriaosuuden yhteenveto

Tähän mennessä olemme siis selvittäneet ohjelmointikielen elinkaarista ja tämänhetkisistä suhteista sauraavia asioita.

Ohjelmointikieliä syntyy jatkuvasti, mutta kasvavat ja menestyvät ohjelmointikielet kehitetään johonkin tiettyyn tarpeeseen. Tämän lisäksi niiden taustalla ovat suuret organisaatiot tai tutkimuslaitokset. Uusien suosittujen kielien kasvuvaiheessa kestää ainakin 2 vuotta. Kohtalaisen uusia kasvuvaiheessa olevia kieliä ovat Kotlin, TypeScript, Rust, Dart, Go, Julia ja Scala.

Tekijöitä jotka pitävät ohjelmointikielen käytettynä ovat yhteisön koko ja kehittäjien määrä, kielen uusille kehittäjille aloittamisen helppous ja tuttuus verrattuna muihin kieliin, saatavilla olevat kirjastot, kielen tehokkuus, sekä ylläpidettävän koodin määrä ja muutoksien teon

helpottaminen tähän koodiin. Kehittäjien lausunnoista havaitaan että yksinkertaisuus ja tuotannon tehokkuus ovat keskeisessä roolissa. Selkein historiasta tehtävä havainto on se, että kielen suunnitteluperiaatteet ja tarve johon kieli kehitetään määrittävät suurelta osin kielen vahvuudet ja myös siihen kohdistuvan kritiikin. Tämän hetken käytetyimpiä kieliä ovat Python, Java, C#, C++, C ja JavaScript.

Ohjelmointikielen suosio ja käyttömäärä vaikuttavat laskevan kielen puutteiden, epäsuotuisan kilpailuasetelman, kielen tuen tai kehityksen puutteen ja negatiivisien stigmojen takia, sekä uusien ja ylläpidettävien projektien puuttumisen vuoksi. Lähihistoriaa tutkimalla voidaan havaita, että on olemassa myös tekijöitä jotka hidastavat käyttömäärien laskua, kuten vankka yhteisö, riittävät päivitykset, käytössä olevan koodin määrä ja kielen kilpailukykyisyys tietyllä alueella tai tietyssä tehtävässä. Visual Basic, Perl, Delphi ja PHP eivät ole syyttä saavuttaneet suosiota aiemmin, ja vaikka käyttömäärät ovatkin laskussa, menee kauan että näitä kieliä ei nähdä enää ollenkaan. Varsinkin PHP:n käyttöaste web-sivuilla on edelleen äärimmäisen korkea.

Seuraavaksi siirrymme kokoamaan systemaattisella kirjallisuuskatsauksella mahdollisimman kattavasti tuoretta tutkimustietoa sovelluskehityksen lähitulevaisuuden kielistä ja niihin liittyvistä suuntauksista, jonka jälkeen yhdistämme näillä eri menetelmillä kerätyt tiedot lopullisiksi johtopäätöksiksi.

4 Tutkimuksen toteutus

Tutkimuksella pyritään hakemaan vastausta tutkimuskysymyksen: Miten ohjelmointikielten keskinäinen asema muuttuu 2020-luvulla. Tässä luvussa kuvataan tutkimusmenetelmän valinta ja tutkimuksen läpiviennin vaiheet. Luvussa 4.1 kuvataan syyt käytetyn tutkimusmenetelmän valitsemiseen ja käytetyt tutkimusmenetelmän ohjesäännöt. Luvussa 4.2 kuvataan tutkimuksen läpiviennin vaiheet.

4.1 Tutkimusmenetelmä

Tutkimusmetodiksi valikoitui systemaattinen kirjallisuuskatsaus. Tutkittavaa aineistoa on laajasti, suureksi osaksi tutkimuskirjallisuuden muodossa ja tavoitteena on esittää monipuolinen ja puolueeton näkemys kohdealueesta. Ennen kaikkea halutaan tiivistelmä tietyn aihepiirin aiempien tutkimuksien olennaisesta sisällöstä tietyltä ajanjaksolta, sen sijaan että esimerkiksi tutkittaisiin sitä, mitä kaikkea tähän mennessä on tutkittu ja kenen toimesta. Tämä täsmää Salminen (2011) kuvaukseen systemaattisen kirjallisuuskatsauksen luonteesta sekä tavoitteista. Salminen (2011) määritellyssä systemaattisen kirjallisuuskatsauksen yleiskuvauksessa painotetaan myös keskustelun kartoitusta ja ”tieteellisten tulosten kannalta mielenkiintoisien ja tärkeiden tutkimuksien” seulomista, mikä sopii uusien ilmiöiden ja niiden vaikutuksien selvittämiseen.

Systemaattinen kirjallisuuskatsaus toteutetaan noudattaen pääpiirteittäin Kitchenham ja Charters (2007) ohjeistusta, jossa myös kuvataan muutoksia joita yksittäinen opiskelija voi tutkimukseensa soveltaa, verrattuna esimerkiksi kokonaisten tutkimusryhmien suorittamiin tutkimuksiin. Tutkimuksen rakenne ohjeistusta seuraten kuvataan seuraavassa alaluvussa.

4.2 Tutkimuksen suoritus

Aineisto kerätään syöttämällä tutkimuskysymyksestä määritettyjä hakulauseita kolmeen paljon käytettyyn tietokantaan, joissa on mahdollisuus rajata hakuja tiivistelmän ja otsikon perusteella. Nämä tietokannat ovat “IEEE Xplore digital library” (2021), “ScienceDirect”

(2021) ja “ACM Digital Library” (2021).

Aiheen laajuuden takia on hankalaa muodostaa tarkkoja hakutermejä, jotka löytäisivät kaikki mahdolliset kiinnostavat artikkelit. Tämän vuoksi artikkelien toisena pääasiallisena rajauksena toimii julkaisuvuosi. Käytettävä rajaus on vuonna 2016 ja sen jälkeen julkaistujen artikkelien sisällyttäminen. Tämä viiden vuoden rajaus on osittain mielivaltainen, mutta sillä oletetaan saatavan riittävän ajankohtaista ja päivitettyä tietoa. Käytettävä hakulause kaikissa tietokannoissa on ”programming language” ja se kohdistetaan artikkelin otsikkoon ja tiivistelmään. Yhdessä näiden rajauksien kanssa voidaan olettaa että relevantteja artikkeleita ei ohiteta ja artikkelien määrä pysyy käsiteltävän kokoisena. Lisäksi tutkimuksen suorittajan kielitaidon takia sisällytetään tutkimukseen vain englannin- ja suomenkieliset artikkelit. Tiivistettynä tutkimusartikkeleiden haku suoritetaan seuraavilla määrityksillä:

1. Haettavat tietokannat: IEEEXplore, ScienceDirect ja ACM-DL
2. Hakulauseke otsikkoon ja tiivistelmään: ”programming language”
3. Haettavien artikkeleiden julkaisuvuosi: 2016 - 2021
4. Artikkelien kieli: englanti ja suomi

Tämän jälkeen suoritetaan ensimmäinen seula, jossa haetut artikkelit seulotaan otsikon ja tiivistelmän perusteella. Artikkeli tallennetaan jos se täyttää kaikki seuraavat kriteerit:

1. Artikkelin otsikko tai tiivistelmä liittyy jollakin tavalla ohjelmointikielien lähitulevaisuuteen.
2. Artikkeli ei liity kiinteästi pedagogiikkaan.
3. Artikkelin kirjoittaja ei ole artikkelissa käsiteltävän ohjelmointikielen kehittäjä tai muu selkeästi kehittäjään liitettävä taho.

Toisessa seulassa nämä artikkelit luetaan läpi ja niistä säilytetään ne, jotka toteuttavat seuraavan ehdon:

1. Tutkimus sisältää tietoa, jota voi käyttää kysymykseen “Miten ohjelmointikielten keskinäinen asema muuttuu 2020-luvulla?” vastaamiseen.

Näistä artikkeleista poimitaan kaikki relevantit tulokset ja ne kirjataan tiiviissä muodossa

lomakkeelle, jolle kirjataan myös artikkelin nimi, tulosten poiminnan päivämäärä, artikkelin kirjoittajat ja julkaisu.

Tutkimukseen kirjataan tiivistetyt versiot artikkelien havainnoista ja ne lajitellaan kategorioihin havaintojen tyypin mukaan. Tutkimuksen suorittaja luo nämä kategoriat artikkelien yhtäläisyyksien perusteella. Analyysivaiheessa pyritään yhdistelemään useamman saman kategorian tai aihepiirin artikkelin tuloksia, jotta saadaan muodostettua paremmin kokonaiskuvia ilmiöistä ja selvittämään miten artikkelit yhdessä vastaavat tutkimuskysymykseen. Aiheen laajan luonteen vuoksi analyysi on melko vapaamuotoista, eikä tarkkoja ohjeita ole.

5 Tulokset ja analysointi

Tässä luvussa käydään läpi systemaattisen kirjallisuuskatsauksen tilastoja, tuloksia ja niiden ryhmäyttämistä. Tämän jälkeen saatuja tuloksia tutkitaan ja niistä pyritään löytämään trendejä tai tekemään havaintoja joita useampi tutkimus tukee.

5.1 Tulokset

Tietokannoista hakeminen luetelluilla kriteereillä tuotti 4922 artikkelia. Ensimmäisen seulan jälkeen tämä määrä väheni 52:een. Valtaosa hakujen tuloksista oli tutkimuskysymykseen liittymättömiä artikkeleita, jotka liittyivät muun muassa ohjelmistojen, tekniikoiden ja erilaisten innovaatioiden esittelyyn, teknisiin dokumentaatioihin tai vertailuihin. Ohjelmointikielet itsessään ovat näissä artikkeleissa sivuroolissa tai niitä käsitellään vain maininnan tasolla. Merkittävä osuus artikkeleista, joissa ohjelmointikielet ja niiden tulevaisuus olivat olennaisessa osassa, karsiutui ensimmäisessä seulassa siksi, että ne liittyivät vahvasti pedagogiikkaan, tai ne olivat kielien kehittäjien suorittamia kuvauksia omasta uudesta ohjelmointikielestään.

Toisessa seulassa näistä artikkelista 18 sisälsi tutkimustuloksia, joita voidaan vaihtelevalla tasolla käyttää tutkimuskysymykseen vastaamisessa. Tulokset kokonaisuudessaan ovat liitteessä 4. Tuloksien tyypit voidaan löyhästi jakaa neljään muodostettuun eri kategoriaan, jotka esitellään seuraavaksi.

Kaikki ensimmäiseen kategoriaan sijoitettujen artikkelien tulokset sisältävät jonkin implikaation mahdollisesta kielen käytön määrän kehityssuunnasta. Tähän kategoriaan liittyviä tuloksia löytyi kolmesta artikkelista. Tiivistelmät tuloksista ovat seuraavat:

1. Kyselytutkimuksen mukaan Bangladeshissa Pythonia opetetaan kouluissa ensisijaisena ohjelmointikielenä 8,2%:lle opiskelijoista ja kaupungin Pythonia osaavista yli puolet on oppinut kieltä jostakin muualta. Mikäli oppilaitokset alkavat vastata kysyntään, voidaan olettaa että Pythonin käytön määrä kasvaa vielä entisestään (Javed ym. 2019).
2. Javaa ja Kotlinia sisältävissä, kriteerit täyttävissä 1232:ssa Android-projektissa siir-

tyminen Javan ja Kotlinin välillä oli lähes yksisuuntaista Kotlinin eduksi ja Kotlinia enemmän sisältävät projektit saavuttivat paremman sijoituksen erilaisilla suosiometriikoilla mitattuna (Coppola, Ardito ja Torchiano 2019).

3. Swiftin kypsyminen on vuoden nopeampaa kuin Go:n, kun tutkitaan Stack Overflowissa olevaa yhteisöä. Tämä tarkoittaa että Swiftiin liittyviin kysymyksiin saa laadukkaampia vastauksia vuoden Go:ta nopeammin, mikä voi entisestään muuttaa yhteisöjen kokoja. Tutkimuksessa havaitaan että uudet kielet hyötyvät edeltäjiensä yhteisöistä (Chakraborty, Shahriyar ja Iqbal 2019).

Toiseen kategoriaan sijoitettujen artikkelien tulokset sisältävät yhden tai useamman vaatimuksen kielen menestymiselle. Artikkeleista kolmessa oli tähän kategoriaan liittyviä tuloksia:

1. Juliaa tutkimalla ja vertaamalla sitä C:n ja Pythonin ominaisuuksiin ja suosion kehitykseen, havaitaan että kielen kasvamiselle ja hallitsevassa asemassa pysymiselle olennaisinta on lojaalin käyttäjäkunnan ylläpitäminen. Suurelta osin tämä saavutetaan toimittamalla laadukasta dokumentaatiota ja ohjeistusta käyttäjille. Toinen olennainen osa on riittävän tehokkuuden saavuttaminen verrattuna käytetyimpiin kieliin (Cabutto ym. 2018).
2. C:n historiaa, käyttämistä ja ominaisuuksia analysoimalla voidaan todeta, että C:n menestymisen syyt ovat suorituskyvyn sijaan kilpailijoiden vuosikymmenien migraatio- ja integraatiotuen aukot, sekä C:n ilmaisuvoima (Kell 2017).
3. Vaikka kehittäjät vaikuttavat omaksuvan Swiftin nopeasti, suuri osa Stack Overflowin kysymyksistä liittyy yksinkertaisiin listoihin ja datatyyppeihin. Tämän lisäksi keskustellaan työkalujen bugeista ja virheviestien hyödyttömyydestä. Haastateltavien henkilöiden mielestä Swift vaatii näillä osa-alueilla parannuksia menestyksen saavuttamiseksi (Rebouças ym. 2016).

Kolmanteen kategoriaan voidaan sijoittaa viiden artikkelin tuloksia ja niitä yhdistää tulevaisuuden mahdollisen sovelluskehityksen kehityssuunnan kuvaaminen.

1. Monikielisten järjestelmien ongelmia ja niiden ratkaisuja kartoittavassa tutkimuksessa todetaan, että useita ohjelmointikieliä käyttävien järjestelmien määrä tulee jatkamaan

korkeana koodin uudelleenkäytettävyyden ja muiden kielien kirjastojen hyödyntämisen tuoman tuotannon tehokkuuden takia. Osittain tämä suuntaus kuitenkin vaikeuttaa sovelluskehitystä lisääntyvän monimutkaisuuden vuoksi (Abidi, Khomh ja Gueheneuc 2019).

2. Käytännön toteutuksella ja vertailemalla sitä vaihtoehtoisiin toteutustapoihin todistettiin, että tehokkaisiin tietokannanhallintajärjestelmiin voidaan käyttää myös korkean tason kieliä tehokkaan tuotannon saavuttamiseksi. Avainasemassa tässä on koodin automaattinen kääntäminen matalamman tason kielelle ennen ajamista (Shaikhha, Klonatos ja Koch 2018).
3. Vertailtaessa Python-ohjelmoijien ja Java- tai C++ -ohjelmoijien kirjoittaman koodin laadukkuutta GitHubin Python projekteissa, huomataan että Javan ja C++:n osaaminen auttaa olio-ohjelmointiin liittyvien rakenteiden kirjoittamisessa ja koodista tulee yleisesti ottaen laadukkaampaa, mikä korostaa useamman erilaisen kielen opiskelun hyödyllisyyttä esimerkiksi pelkän Pythonin sijaan (Horschig, Mattis ja Hirschfeld 2018).
4. Smaragdakis on tutkimuksessaan kielten ominaisuuksia, ohjelmistokehityksen tarpeita ja toimintaperiaatteita tutkimalla spekuloinut, mitä suuria muutoksia ohjelmointikielissä tulee tapahtumaan. Seuraavan paradigman ohjelmointikielet tarvitsevat hänen mukaansa erittäin suuren mullistuksen abstraktion tasossa, jotta ne voivat saavuttaa suuria parannuksia tuottavuudessa. Tämä muutos tulee vaikuttamaan merkittävästi kääntäjän rooliin, kehittäjän mentaalimalliin ja sovelluskehityksen käytänteisiin (Smaragdakis 2019).
5. Tutkimuksessa on kehitetty koneoppimisjärjestelmä, joka pyrkii muuttamaan englannin kielellä annettuja komentoja syntaktisesti oikeellisiksi ohjelmointikielen komennoiksi ja samalla parantamaan järjestelmän tarkkuutta. Syntaksittoman ohjelmoinnin järjestelmä saadaan toimimaan jo yli 80% tarkkuudella, minkä lisäksi järjestelmä päivittyy ja paranee jokaisella iteraatiolla. Mikäli kehitys jatkuu vastaavanlaisena, perinteisten ohjelmointikielten käyttöä voidaan vähentää ja tuottavuutta voidaan lisätä (Trivedi ym. 2019).

Viimeiseen kategoriaan kuuluvien artikkeleiden tuloksilla yhtenevänä piirteenä on niiden liittyminen ohjelmointikielten vertailuun. Tämä on tutkimuskysymykseen vastaamisen kannalta kategorioista spekulatiivisin, sillä tämänkaltaisen vertailun tulokset eivät välttämättä

näy kielien suosiossa tai käyttömäärissä. Tuloksia löytyi kuudesta artikkelista.

1. Käytännön testiasetelmassa suoritetussa, Javan ja Kotlinin Android-sovelluskehitystä vertailevassa tutkimuksessa havaittiin, että Javan käyttäminen kannattaa mobiilikehityksessä, mikäli APK-tiedoston koko tai kääntämisen ja buildaamiseen kuluvat ajat ovat prioriteettilistan kärjessä. Kotlinin käyttämisellä taas saavutetaan vähemmän bugeja, ilmaisuvoimaisempaa ohjelmakoodia ja nopeampia sovellusten kehitysaikoja (Putranto ym. 2020).
2. Bugien korjaamisen erityispiirteitä selvittäneessä tutkimuksessa on analysoitu GitHubin tarjoamia tilastoja 600:sta suositusta avoimen lähdekoodin projektista. Valitut projektit jakautuvat tasan kymmenen suosituimman ohjelmointikielen kesken. Bugien korjaaminen vie vähiten aikaa Javassa, kun taas Rubyssä aikaa kuluu eniten. Staattisesti tyypitetyissä projekteissa korjauksiin kuluu enemmän rivejä ja ne vaativat muutoksia useampiin tiedostoihin kuin dynaamisesti tyypitetyt projektit (Zhang ym. 2019).
3. Samanaikaisuuteen liittyviä ominaisuuksia analysoivassa tutkimuksessa on vertailtu Javan ja Go:n suoritustehoa muutamassa eri kategoriassa. Go on Javaa tehokkaampi kieli varsinkin sovelluksiin, jotka tarvitsevat jatkuvasti uusia säikeitä laskentaan, kun taas Java on tehokkaampi rinnakkaislaskennassa, jossa säikeitä ei tarvitse luoda jatkuvasti, sekä tiedostojen koot ovat pienemmät. Toisaalta Javalla voi saavuttaa monet Go:n eduista käyttämällä erilaisia säie-kirjastoja (Abhinav ym. 2020).
4. Suoritetussa tutkimuksessa on mitattu käytettävän ohjelmointikielen merkitystä ohjelman vikaherkkyyteen ja viansietokykyyn, kun testiohjelmiin on pakotettu ei-kriittisiä virheitä. Tulokset osoittavat että käännettävissä kielissä ajonaikaiset virheet aiheuttavat harvemmin vikatiloja tai ohjelman kaatumisen. Ei-käännettävät kielet taas tuottavat vähemmän hiljaista datan korruptiota (Cerveira ym. 2018).
5. Kaksiosaisessa tutkimuksessa pyrittiin selvittämään staattisen ja dynaamisen tyyppijärjestelmän käyttämisen eroja. Tutkimuksen ensimmäisessä osassa koehenkilöt kehittivät määritellyn ohjelman alusta asti ja toisessa osassa tavoitteena oli korjata valmiista ohjelmasta löytyviä virheitä. Tutkimuksessa käytetyt kielet olivat C# ja PHP. Virheiden korjaamiseen liittyvissä tehtävissä staattisesti tyypitetty kieli erottuu edukseen. Ohjelmia koodattaessa alusta asti ei tuotannon tehokkuudessa ole eroa dynaamisesti tai staattisesti tyypitetyn kielen välillä (Harlin, Washizaki ja Fukazawa 2017).

6. Android-kehityksen tehtäviä tarkastelevassa tutkimuksessa vertailtiin Javaa ja Kotlinia kehityksen tehokkuuden ja helppouden, sekä koodin ylläpidettävyyden näkökulmista. Tutkimuksessa 108 maisteriopiskelijaa suoritti laadittuja käytännön kehitystehtäviä pienryhmissä. Javan ja Kotlinin välillä ei havaittu eroa ylläpidettävyydessä. Kotlinin havaittiin olevan ilmaisuvoimaisempi ja helpompi kirjoittaa. Javan IDE:n tuen kerrottiin olevan parempi kuin Kotlinilla (Ardito ym. 2020).

5.2 Tuloksien analysointi

Ensimmäinen yleisen tason havainto on tulosten jakautuminen laajalle alueelle. Saadut tulokset voidaan jakaa esiteltyihin neljään kategoriaan, eli niillä on joitakin yhteneväisyyksiä, mutta kategorioiden sisällä harvat tulokset koskevat samaa aihetta tai ovat keskenään vertailtavia. Joitakin huomioita tuloksista voidaan kuitenkin tehdä.

5.2.1 Kotlin ja Java

Javan ja Kotlinin vertailu esiintyy kolmessa tutkimuskatsauksen artikkeleista. Vertailuissa tutkitaan suurimmaksi osaksi Android-alustan sovelluskehitystä. Android on tällä hetkellä suosituin mobiilialusta, vuonna 2019 sen markkinaosuus oli 74,85% (Coppola, Ardito ja Torchiano 2019).

Coppola, Ardito ja Torchiano (2019) tutkivat F-droid pakettivaraston avoimen lähdekoodin projekteja, niiden kulkeutumista sisältöpalveluihin ja käyttäjien kirjoittamia arvosteluja sisältöpalveluissa. Noin viidesosassa 2017 jälkeen päivitetyistä projekteista esiintyi Kotlinia, niistä 2/3:ssa Kotlin oli käytetyin kieli. Tutkituista projekteista joissa jossakin vaiheessa oli esiintynyt sekä Kotlinia että Javaa, 30%:ssa ei tutkimushetkellä esiintynyt enää ollenkaan Javaa. Tutkittujen projektien tai valmiiden tuotteiden arviointimetriikoissa ei ollut juurikaan merkittäviä havaintoja, Kotlinilla kirjoitetut projektit olivat Githubissa saaneet keskimääräisesti hieman paremmat arvostelut.

Putranto ym. (2020) rakensivat testiolosuhteissa Javalla ja Kotlinilla kaksi ulkopuoliseen palvelimeen yhteydessä olevaa käytännönläheistä sovellusta. Luotujen sovelluksien Java-versio kääntyy huomattavasti nopeammin kuin Kotlinilla kirjoitettu ja siitä muodostettu APK-tie-

dosto on pienempi. Kotlinilla kirjoitettujen rivien määrä oli hieman pienempi ja luokkien määrä merkittävästi pienempi. Kehitysympäristöjä ja ekosysteemejä vertailtaessa erot ovat vähäisiä laajan yhteentoimivuuden takia. Ainoat erot havaittiin monialustakehityksessä, jossa Kotlinilla on käytettävänä Javan Gluon Mobilen tarjoamaa kehystä kehittyneempi, kypsempi ja ilmainen KMM-kehys. Kielien rakenteita vertailtaessa havaittiin runsaasti eroavaisuuksia, joista suurin osa on tutkijoiden mielestä toteutettu paremmin Kotlinilla.

Ardito ym. (2020) tutkivat 27:ää neljän hengen ryhmää, jotka tekivät korjasivat puutteita ja tekivät ylläpitoon liittyviä tehtäviä kahteen ohjelmaan, joista toinen oli kirjoitettu Javalla ja toinen Kotlinilla. Tuloksista näkyy että Java-ohjelmat vaativat kolme kertaa enemmän koodirivejä kuin vastaavat Kotlinilla kirjoitetut ja jonkin verran enemmän luokkia. Ylläpitotehtävien yksinkertaisuudessa, korjattavien vikojen löytämisessä ja korjaamiseen käytetyssä ajassa ei havaittu eroja. Tutkimuksen ohjelmointiosuuden jälkeen toteutettiin kysely, jossa tutkittiin yleisien teknisien kompastuskivien vaikutusta osallistujien kehitykseen. Ainoa nollahypoteesin kumonnut osa-alue oli havaittu ”NullPointerException” -poikkeuksien määrä, joita Java-kehityksessä oli esiintynyt huomattavasti enemmän.

Tutkimuksista voidaan siis huomata että kyseessä ovat kaksi samankaltaista kieltä. Java on vanhempi kieli, johon on ollut aikaa kehittyä laaja yhteisö, työmarkkinaosuus, työkaluja ja kirjastoja. Kotlinin kehittäjät ovat luoneet vahvan siirtymäkohdan kieleensä tekemällä siitä mahdollisimman yhteensopivan Javan kanssa, jolloin vaihtaminen on mahdollisimman kitkatonta. Lisäksi Kotliniin on voitu alusta asti korjata Javan puutteita tai epäsuosittuja ominaisuuksia ja lisätä kielen rakenteita joita Javassa ei ole. Nämä näkyvät varsinkin Kotlinin tuottavuudessa ja ilmaisuvoimaisuudessa. Jo luvussa kolme havaitsimme kehittämisen helppouden ja tuotannon tehokkuuden olevan äärimmäisen olennaisessa osassa kielen menestymiselle. Ominaisuuksien lisäksi Coppola, Ardito ja Torchiano (2019) esittelevät Javasta Kotlinin suuntaan olevaa muuttoliikettä. Näiden seikkojen perusteella voidaan olettaa Kotlinin kasvun jatkuvan, kuten luvussa kolme olevissa tilastoissa on esitetty. Huomioitavaa joissakin tutkimustuloksissa on kuitenkin se, että ne tarkastelevat suurelta osin avoimen lähdekoodin projekteja tai varta vasten tutkimuskäyttöön luotuja sovelluksia. Coppola, Ardito ja Torchiano (2019) ja Ardito ym. (2020) tutkimuksissa mainitaan, että saadut tulokset eivät välttämättä täysin päde yritysmaailman realiteetteihin ja työtapoihin.

5.2.2 Ohjelmointikielten käytön monipuolisuus

Toinen havaittava aihe on kielten monipuolisen käytön seuraukset. Artikkelit käsittelevät ohjelmointikielen osaamisen vaikutusta toisessa kielessä, matalan tason projektien kirjoittamista korkean tason kielellä, sekä monikielisten järjestelmien hyötyjä ja haasteita.

Shaikhha, Klonatos ja Koch (2018) tutkimuksessa kehitetty Legobase on kyselymoottori (query engine), joka on toteutettu korkean tason ohjelmointikielellä Scalalla. Korkean tason toteutuksen tehokkuus on mahdollistettu generatiivisella ohjelmoinnilla (“generative programming”, jonkinasteista ohjelmakoodin automaattista luomista kääntämisen tai ajon aikana). Tässä tapauksessa LegoBase suorittaa lähdekoodista lähdekoodiin kääntämisen ja optimoi tietokantajärjestelmän koodin kääntämällä Scala koodin C koodiksi. Suoritettujen testien tulokset osoittavat, että kehitetty arkkitehtuuri optimoituna voittaa tehokkuudessa kaupalliset muistitietokannat ja kyselyjen kääntäjät. Legobase on myös kooltaan vain muutamia satoja koodiriveja pitkä. Negatiivisena puolena sen käyttämisessä on muistin käytön lisääntyminen verrattuna vaihtoehtoisiin tekniikoihin.

Abidi, Khomh ja Gueheneuc (2019) mukaan monikielisillä järjestelmillä voidaan hyödyntää eri kielien vahvuuksia, vähentää kustannuksia mikäli voidaan käyttää toisella kielellä valmiiksi kirjoitettuja osia, tai mahdollistaa legacy-koodin käyttöä. Monien kielien käyttäminen yhdessä tuo kuitenkin mukanaan haasteita ja tutkimus on löytänyt kuusi toisistaan eriytettävää ongelmien tyyppiä, joihin kehittäjien tulisi kiinnittää huomiota. Nämä ovat liiallinen kielten välinen kommunikointi, sirpaleisuus, toisistaan erilliset keskittymät jotka tekevät osin samoja asioita, migraatiosta syntyvät kieliin liittyvät ongelmat, hyödytön usean kielen käyttäminen, sekä kielen ja paradigman välinen yhteensopimattomuus.

Horschig, Mattis ja Hirschfeld (2018) tutkimuksessa on käytetty laajaa datajoukkoa Githubin avoimen lähdekoodin projektien kehityksestä. Tästä datamassasta on haettu niihin Java- ja C++ -ohjelmoijiin liittyviä tilastoja, jotka käyttävät kehityksessään osittain myös Pythonia. Heidän tekemiään virheitä ja niiden määrää Python-kehityksessä on verrattu päätoimisten Python-ohjelmoijien vastaaviin tilastoihin. Tuloksissa havaitaan että C++:n ja Javan osaaminen auttaa niille kielille ominaisten käytänteiden ja rakenteiden kanssa myös Pythonissa, mutta Java- ja C++ -ohjelmoijilla on enemmän virheitä Pythonille ominaisten käytänteiden

kanssa. Python-ohjelmoijiin verrattattuna Java-ohjelmoijat ryhmäyttävät importit paremmin, eivätkä käytä liikaa tai liian vähän julkisia metodeja pythonia kirjoittaessaan, mutta jättävät paljon useammin joitakin return-lauseita pois, kun funktiota käytetään ehtolauseessa. Myös C++ -ohjelmoijat ryhmäyttävät importit paremmin, jonka lisäksi he käyttävät välilyöntejä paremmin. Puolipisteitä C++ -ohjelmoijat lisäävät virheellisesti rivien loppuun huomattavan paljon.

Artikkeleista havaitaan useampia eri näkökulmia joista voidaan analysoida yhdessä ja erikseen. Legobasen tehokkuus osoittaa, että kieliä voi käyttää monin tavoin ja sellaisissakin tehtävissä joihin niitä ei välttämättä miellettäisi sopiviksi. Toisaalta nousee kysymys siitä, että voiko kaiken kirjoittaa tulevaisuudessa korkean tason kielellä ja tarpeen vaatiessa automaattisesti kääntää sen lähemmäksi konekieltä. Tämä taas entisestään vähentäisi monikielisten järjestelmien houkuttelevuutta. Abidi, Khomh ja Gueheneuc (2019) tutkimuksessa käy kuitenkin ilmi että suuri osa monikielisten järjestelmien hyödyistä liittyy vanhojen, mahdollisesti eri tekniikoilla ja eri tiimien toteuttamisen osien yhdistelyyn, joissa olisi mahdotonta koordinoida yhteisiä korkean tason tekniikoita, kun kehitys ajoittuu pitkälle menneisyyteen ja tulevaisuuteen. Kirjoitettavan koodin laadun näkökulmasta taas näyttää siltä, että useampien erilaisten ohjelmointikielten osaaminen opettaa paremmin erilaisia hyviä käytänteitä ja parantaa ymmärrystä erilaisista konsepteista, mikä taas parantaa kirjoitettavan koodin laatua.

5.2.3 Muut teemat

Luonnollisella kielellä kirjoitettava ohjelmointi näkyy esiintyvän eri tavoin. Koneoppimisella jalostettu datamassa ja niihin kytketyt tulkit kykenevät muuttamaan luonnollista kieltä ohjelmakoodiksi jatkuvasti kasvavalla virheettömyyden todennäköisyydellä. Mainitut tehokkaat tulkit ja korkean tason kielien sopiminen myös tehokkuutta vaativiin sovelluksiin mahdollistavat myös osaltaan tätä kehityssuuntaa. Näiden lisäksi tuloksissa oletetaan että seuraavan ohjelmointiparadigman on oltava luonteeltaan tuottavuutta ja työskentelyä mullistava, mikä sopii narratiiviin.

Tuloksien määrästä voidaan havaita lähitulevaisuuden ohjelmointikieliin kohdistuvan tuoreen tutkimuskirjallisuuden vähäisyys, kun verrataan saatujen tulosten määrää kaikkien tar-

kasteltujen artikkelien määrään. Kaikkiaan vain 18 artikkelia 4922:sta viimeisen viiden vuoden ajalta julkaistuista haulla saadusta artikkelista käsitteli aihetta jollakin tasolla. Tämä voi puoltaa sitä, että aihetta on vaikea tutkia tarkoilla tieteellisillä menetelmillä.

Tuloksista havaitaan myös se, että pienemmistä kielistä ei tehdä tai ei ole saatavilla kolmannen osapuolen tutkimustietoa. Ensimmäisiltä osapuolilta, eli kielen kehittäneeltä tai siihen kiinteästi liittyvältä taholta, tutkimustietoa on saatavilla useiden pienempien kielien osalta, mutta kuten tämän tutkimuksen sisäänottokriteereissä on määritelty, nämä karsiutuvat lopullisten artikkelien joukosta. Koska tutkimuksen tutkimuskysymys käsittelee alkaneen vuosikymmenen ohjelmointikieliä, niin pienimmät kielet voivat siinä ajassa jo kypsyä merkittävään asemaan, kuten luvun 3 graafeista käy ilmi. Tämän takia vaikkakin pienimpien kielien puuttuminen tutkimuskirjallisuudesta on odotettavissa, se ei ole optimaalista tämän tutkimuksen kattavuudelle.

6 Yhteenveto ja pohdinta

Tässsä luvussa pyritään yhdistelemään systemaattisen kirjallisuuskatsauksen tuloksista nousseita ilmiöitä ja aiemmissa luvuissa selvitettyjä kokonaisuuksia ja muodostamaan vastauksia tutkimuskysymykseen. Lisäksi pohditaan tutkimukseen liittyviä seikkoja jotka kannattaa ottaa huomioon.

6.1 Pohdinta ja tutkimuksen rajoitteet

Tutkimuksella saavutettiin kuvaus ohjelmointikielien suosion kehityksen vaiheista ja niihin liittyvistä muuttujista, selvitys käytetyimpien ja nousevien ohjelmointikielien historiasta ja nykyhetkestä, sekä katsaus tämän hetken uusimpiin tutkittaviin aiheisiin tutkimuskysymykseen liittyen. Ohjelmointikieliä tutkittaessa on pyritty arvioimaan niiden käyttömäärien kehitystä lähitulevaisuudessa. Suuri osa vaikuttavista tekijöistä on kuitenkin vaikeasti mitattavissa ja vertailtavissa ja osa on täysin mielipidekysymyksiä, kun taas tuore tutkimuskirjallisuus aiheesta on melko hajanaista, joten tutkimuksen luonne on melko spekuloiva.

Systemaattista kirjallisuuskatsausta suorittaessa ohitettiin melko suuri määrä tutkimuksia joista voisi olla apua tulevaisuuden selvittämisessä, koska nämä tutkimukset liittyivät kiinteästi pedagogiikkaan tai olivat ohjelmointikielen kehittäjien tutkimuksia omasta kielestään. Päätös jättää kielten opetukseen ja tarkempiin oppimismekanismeihin liittyvät tutkimukset ja käsittely sivuun tehtiin jo tutkielman alkuvaiheilla, koska kirjoittaneella ei ole riittävää osaamista tältä osa-alueelta ja tutkielman laajuutta piti rajoittaa. Ohjelmointikielien kehittäjien omat artikkelit taas päätettiin jättää objektiivisuuden takia systemaattisen kirjallisuuskatsauksen ulkopuolelle. Jälkikäteen mietittynä nämä artikkelit oltaisiin voitu jättää karsimatta, kunhan kirjoittajan asema oltaisiin otettu huomioon tuloksia tarkasteltaessa. Näin oltaisiin saatu useampia näkökulmia aiheeseen ja enemmän artikkeleita tutkittavaksi.

Toinen rajoite tutkimuksessa on systemaattisen kirjallisuuskatsauksen artikkelien rajaus viiteen vuoteen. Toisaalta tämä tarkoittaa sitä, että saadaan otos kattavasta määrästä kaikkein tuoreimpia tutkimuksia tarkasteltavaksi ja ajantasainen tilanne niistä alueista joiden artikkelit liittyvät tutkimuskysymykseen. Toisaalta taas viisi vuotta on lyhyt aika ohjelmointikielen

kehittymisessä ja tutkimalla artikkeleita pidemmältä aikaväliltä voitaisiin tutkia tuoreiden ilmiöiden lisäksi myös kielien kehitystä. Lisäksi pidemmän aikavälin tutkiminen tuottaa suuremman määrän artikkeleita jotka läpäisevät kaikki seulat, siinä missä tällä viiden vuoden aikavälillä tutkittuna seulojen läpäisseiden artikkelien määrä oli melko vähäinen. Systemaattinen kirjallisuuskatsaus kymmenen vuoden ajalta käytetyllä hakulauseella olisi kuitenkin tuonut tuhansia artikkeleita lisää tarkasteltavaksi ja vaatinut todennäköisesti vähintään kaksinkertaisen ajan. Tämä on osasyy siihen miksi yksittäisen henkilön toteuttamat systemaattiset kirjallisuuskatsaukset ovat hieman hankalia ja vaativat joitakin kompromisseja.

6.2 Yhteenveto

Systemaattisessa kirjallisuuskatseuksessa selvitetyt kohdat voivat kaikki osaltaan vaikuttaa ohjelmointikielien käyttömäärien kehitykseen ja analyysillä selvitetyt johtopäätökset, joita tukevat useamman tutkimuksen tulokset ja tilastot, ovat vielä todennäköisemmin osana lähitulevaisuuden toteutumisessa. Kuitenkin koska kyseessä on ilmiö johon vaikuttavat lukemattomat asiat, yksittäisten tai muutaman tutkimuksen puoltaman ilmiön vaikutus kokonaiskuvaan voi olla vaihteleva. Tämän takia tutkimukseen haluttiin sisällyttää myös ammattilaisten kommentteja sekä yksityiskohtia kielien elinkaarten vaiheista paremman kokonaiskuvan saamiseksi.

Kaiken kaikkiaan vaikuttaa siltä että ominaisuudet kuten kielen päivitysten määrä, turvallisuus ja osa elämää helpottavista ominaisuuksista vaikuttavat kielen käyttömäärään merkittävästi vain siinä tapauksessa jos niitä on laiminlyöty, tai jos ne ovat merkittävästi heikommat kuin kilpailijoilla. Kielen menestymiseen vaaditaan jokin konkreettinen tarve, suuri kehittävä taho, sekä vaikeammin määriteltäviä ja vertailtavia ominaisuuksia kuten helppokäyttöisyyttä, yksinkertaisuutta ja tuotannon tehokkuutta. Tarve johon kieli kehitetään määrittää sen, millä osa-alueilla kielen pitää olla vaihtoehtoisia kieliä parempi. Esimerkiksi kielen ei välttämättä tarvitse olla tehokas tai kooltaan optimoitu jos se ei yritä kilpailla matalan tason kielien kanssa.

Nyt kehitettävä uusi kieli ehtii vielä 2020-luvun aikana nousta merkittäväänkin asemaan, mikäli siitä löytyy edellä mainitut ominaisuudet ja puitteet. Kieli voidaan kehittää syystä, joka

ei ole vallankumouksellinen vaan täyttää jonkin yksinkertaisen tarpeen, mutta kieli voi myös edustaa täysin uutta paradigmaa ja yrittää mullistaa sovelluskehityksen parissa työskentelyn, kuten edellä mainitussa Smaragdakis (2019) tutkimusartikkelissa on selvitetty.

Luvun kolme tilastoissa ja graafeissa esiteltyjen kielien haku ja käyttömäärien suhteissa ei muun aineiston perusteella välttämättä lähivuosina tule tapahtumaan suuria muutoksia. Python jatkanee vielä hienoista kasvua kuitenkaan siirtymättä monopoliasemaan millään sovelluskehityksen alueella, kun taas muut suurista kielistä pysyvät samoissa käyttömäärissä tai laskevat hieman. Olemassaolevan ja ylläpidettävän koodin suuren määrän takia muutokset ovat hyvin maltillisia. C:n suorat käyttömäärät voivat laskea enemmän, koska C:n matalan tason kielenä saamat hyödyt saattavat jäädä modernien kääntäjien, tulkkien ja kirjastojen varjoon. C:n esiintyvyys itsessään ei välttämättä vähene, jos korkeamman tason kieliä käännetään siihen ennen ajamista. Näiden lisäksi on myös mahdollista, että Kotlin vie käyttäjiä Javalta Android-kehittämisen helppouden, tehokkuuden ja kitkattoman siirtymän takia.

Lähteet

Abhinav, P Y, Avakash Bhat, Christina Terese Joseph ja K Chandrasekaran. 2020. “Concurrency Analysis of Go and Java”. Teoksessa *2020 5th International Conference on Computing, Communication and Security (ICCCS),* 1–6. doi:10.1109/ICCCS49678.2020.9277498.

Abidi, Mouna, Foutse Khomh ja Yann-Gael Gueheneuc. 2019. “Anti-Patterns for Multi-Language Systems”. Teoksessa *Proceedings of the 24th European Conference on Pattern Languages of Programs.* EuroPLop ’19. Association for Computing Machinery. ISBN: 9781450362061. doi:10.1145/3361149.3364227. https://doi.org/10.1145/3361149.3364227.

“ACM Digital Library”. 2021. Viitattu 13. heinäkuuta. https://dl.acm.org/.

Ardito, Luca, Riccardo Coppola, Giovanni Malnati ja Marco Torchiano. 2020. “Effectiveness of Kotlin vs. Java in android app development tasks”. *Information and Software Technology* 127. ISSN: 0950-5849. doi:https://doi.org/10.1016/j.infsof.2020.106374. https://www.sciencedirect.com/science/article/pii/S0950584920301439.

Bak, Lars. 2011. “Dart: a language for structured web programming”. Viitattu 12. heinäkuuta 2021. http://googlecode.blogspot.com/2011/10/dart-language-for-structured-web.html.

Bezanson, Jeff, Stefan Karpinski, Viral Shah ja Alan Edelman. 2012a. “Why We Created Julia”. Viitattu 13. heinäkuuta 2021. https://julialang.org/blog/2012/02/why-we-created-julia/.

Bezanson, Jeff, Stefan Karpinski, Viral B. Shah ja A. Edelman. 2012b. “Julia: A Fast Dynamic Language for Technical Computing”. *ArXiv* abs/1209.5145.

Bierman, Gavin, Martin Abadi ja Mads Torgersen. 2014. “Understanding TypeScript”. Springer-Verlag. ISBN: 9783662442012. doi:10.1007/978-3-662-44202-9_11. https://doi.org/10.1007/978-3-662-44202-9_11.

Bissyande, Tegawende F., Ferdian Thung, D. Lo, Lingxiao Jiang ja L. Reveillere. 2013. “Popularity, Interoperability, and Impact of Programming Languages in 100,000 Open Source Projects”. *2013 IEEE 37th Annual Computer Software and Applications Conference:* 303–312. doi:10.1109/COMPSAC.2013.55.

Breslav, Andrey. 2016. “Kotlin 1.0 Released: Pragmatic Language for JVM and Android”. Viitattu 12. heinäkuuta 2021. https://blog.jetbrains.com/kotlin/2016/02/kotlin-1-0-released-pragmatic-language-for-jvm-and-android/.

Cabutto, Tyler A., Sean P. Heeney, Shaun V. Ault, Guifen Mao ja Jin Wang. 2018. “An Overview of the Julia Programming Language”. Teoksessa *Proceedings of the 2018 International Conference on Computing and Big Data,* 87–91. ICCBD ’18. Association for Computing Machinery. ISBN: 9781450365406. doi:10.1145/3277104.3277119. https://doi.org/10.1145/3277104.3277119.

Cerveira, Frederico, Alcides Fonseca, Raul Barbosa ja Henrique Madeira. 2018. “Evaluating the Inherent Sensitivity of Programming Languages to Soft Errors”. Teoksessa *2018 14th European Dependable Computing Conference (EDCC),* 65–72. doi:10.1109/EDCC.2018.00021.

Chakraborty, Partha, Rifat Shahriyar ja Anindya Iqbal. 2019. “Empirical Analysis of the Growth and Challenges of New Programming Languages”. *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)* 1:191–196. doi:10.1109/COMPSAC.2019.00034.

Chatley, R., A. Donaldson ja A. Mycroft. 2019. “The Next 7000 Programming Languages”. Teoksessa *Computing and Software Science.* doi:10.1007/978-3-319-91908-9_15.

Coppola, Riccardo, Luca Ardito ja Marco Torchiano. 2019. “Characterizing the Transition to Kotlin of Android Apps: A Study on F-Droid, Play Store, and GitHub”, 8–14. WAMA 2019. Association for Computing Machinery. ISBN: 9781450368582. doi:10.1145/3340496.3342759. https://doi.org/10.1145/3340496.3342759.

Csharp Language Specification. 2002. Tekninen raportti. http://www.open-std.org/JTC1/SC22/WG14/www/docs/C99RationaleV5.10.pdf.

"Dart FAQ". 2021. Viitattu 13. heinäkuuta. https://dart.dev/faq.

"Dart overview". 2021. Viitattu 13. heinäkuuta. https://dart.dev/overview.

"Delphi product website". 2021. Viitattu 16. heinäkuuta. https://www.embarcadero.com/products/delphi.

DeVoe, J. 2011. *Objective-C.* Wiley Publishing, inc. ISBN: 978-0-470-47922-3.

Evans, Benjamin J. 2015. *Java: The Legend.* O'Reilly Media, Inc. ISBN: 9781491934661.

Garcia, Cristian Gonzalez, Jordan Pascual Espada, B. C. P. Garcia-Bustelo ja J. M. C. Lovelle. 2015. "Swift vs. Objective-C: A New Programming Language". *Int. J. Interact. Multim. Artif. Intell.* 3. doi:0.9781/ijimai.2015.3310.

"GitHut 2.0". 2021. Viitattu 27. heinäkuuta. https://madnight.github.io/githut/.

"golang.org documentation". 2021. Viitattu 12. heinäkuuta. https://golang.org/doc/.

Gosling, James, ja H. McGilton. 1995. "The java language environment: a white paper".

Grin, Tatiana. 2020. *Kotlin programming language media kit.* Viitattu 12. heinäkuuta 2021. https://kotlinlang.org/assets/kotlin-media-kit.pdf.

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. doi:10.1109/HASE.2017.17.

Horschig, Siegfried, Toni Mattis ja Robert Hirschfeld. 2018. "Do Java Programmers Write Better Python? Studying off-Language Code Quality on GitHub". Teoksessa *Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming,* 127–134. Programming'18 Companion. Association for Computing Machinery. ISBN: 9781450355131. doi:10.1145/3191697.3214341. https://doi.org/10.1145/3191697.3214341.

"IEEE Xplore digital library". 2021. Viitattu 13. heinäkuuta. https://ieeexplore.ieee.org.

"Interactive: The Top Programming Languages 2018". 2021. Viitattu 13. heinäkuuta. https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2018.

Javed, Aaquib, Monika Zaman, M. Monir Uddin ja Tasnova Nusrat. 2019. "An Analysis on Python Programming Language Demand and Its Recent Trend in Bangladesh". ICCPR '19. Association for Computing Machinery. ISBN: 9781450376570. doi:10.1145/3373509.3373540. https://doi.org/10.1145/3373509.3373540.

Kell, Stephen. 2017. "Some Were Meant for C: The Endurance of an Unmanageable Language". Teoksessa *Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software,* 229–245. Onward! 2017. Association for Computing Machinery. ISBN: 9781450355308. doi:10.1145/3133850.3133867. https://doi.org/10.1145/3133850.3133867.

Kitchenham, Barbara Ann, ja Stuart Charters. 2007. *Guidelines for performing Systematic Literature Reviews in Software Engineering.* Tekninen raportti EBSE 2007-001. https://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf.

Matsakis, Nicholas D., ja Felix S. Klock. 2014. "The Rust Language". *Ada Lett.* (New York, NY, USA) 34 (3). ISSN: 1094-3641. doi:10.1145/2692956.2663188. https://doi.org/10.1145/2692956.2663188.

Nanz, Sebastian, ja Carlo A. Furia. 2015. "A Comparative Study of Programming Languages in Rosetta Code". Teoksessa *ICSE '15: Proceedings of the 37th International Conference on Software Engineering,* 778–788. doi:10.1109/ICSE.2015.90.

Odersky, Martin, P. Altherr, Vincent Cremet, B. Emir, Sean McDirmid, Stephane Micheloud, N. Mihaylov, Michel Schinz, Erik Stenman ja Matthias Zenger. 2006. "An Overview of the Scala Programming Language Second Edition".

Odersky, Martin, ja Tiark Rompf. 2014. “Unifying Functional and Object-Oriented Programming with Scala”. *Commun. ACM* 57 (4). ISSN: 0001-0782. doi:10.1145/2591013. https://doi.org/10.1145/2591013.

“Perl website”. 2021. Viitattu 16. heinäkuuta. https://www.perl.org/.

“PHP website”. 2021. Viitattu 16. heinäkuuta. https://www.php.net/.

Putranto, Bambang Purnomosidi Dwi, Robertus Saptoto, Ovandry Chandra Jakaria ja Widyastuti Andriyani. 2020. “A Comparative Study of Java and Kotlin for Android Mobile Application Development”. Teoksessa *2020 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI),* 383–388. doi:10.1109/ISRITI51436.2020.9315483.

“PYPL PopularitY of Programming Language”. 2021. Viitattu 13. heinäkuuta. https://pypl.github.io/PYPL.html.

Python 3.9.6 documentation. 2021. Viitattu 12. heinäkuuta. https://docs.python.org/3/.

“Python integration wiki”. 2021. Viitattu 27. heinäkuuta. https://wiki.python.org/moin/IntegratingPythonWithOtherLanguages.

“Python Success Stories”. 2021. Viitattu 12. heinäkuuta. https://www.python.org/about/success/.

“Quotes about Python”. 2021. Viitattu 12. heinäkuuta. https://www.python.org/about/quotes/.

Rationale for International Standard Programming Languages C. 2003. Tekninen raportti. http://www.open-std.org/JTC1/SC22/WG14/www/docs/C99RationaleV5.10.pdf.

Rauschmayer, Axel. 2014. *Speaking JavaScript: An In-Depth Guide for Programmers.* O’Reilly Media, Inc. ISBN: 9781449365035.

Rebouças, Marcel, Gustavo Pinto, Felipe Ebert, Weslley Torres, Alexander Serebrenik ja Fernando Castor. 2016. “An Empirical Study on the Usage of the Swift Programming Language”. Teoksessa *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER),* 1:634–638. doi:10.1109/SANER.2016.66.

Ritchie, Dennis M. 1993. “The Development of the C Language”. *SIGPLAN Not.* 28 (3): 201–208. ISSN: 0362-1340. doi:10.1145/155360.155580.

Rossum, Guido van. 1998. “Glue It All Together With Python”. Viitattu 12. heinäkuuta 2021. https://www.python.org/doc/essays/omg-darpa-mcc-position/.

Rossum, Guido Van, ja Fred L. Drake. 2002. *Python Reference Manual Release 2.2.1.*

Salminen, A. 2011. “Mikä kirjallisuuskatsaus? : johdatus kirjallisuuskatsauksen tyyppeihin ja hallintotieteellisiin sovelluksiin”.

“ScienceDirect”. 2021. Viitattu 13. heinäkuuta. https://www.sciencedirect.com/.

Shaikhha, Amir, Yannis Klonatos ja Christoph Koch. 2018. “Building Efficient Query Engines in a High-Level Language”. *ACM Trans. Database Syst.* 43 (1). ISSN: 0362-5915. doi:10.1145/3183653. https://doi.org/10.1145/3183653.

Smaragdakis, Yannis. 2019. “Next-Paradigm Programming Languages: What Will They Look like and What Changes Will They Bring?” Teoksessa *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software,* 187–197. Onward! 2019. Association for Computing Machinery. ISBN: 9781450369954. doi:10.1145/3359591.3359739. https://doi.org/10.1145/3359591.3359739.

“State of Octoverse 2018”. 2021. Viitattu 13. heinäkuuta. https://octoverse.github.com/2018/.

“State of Octoverse 2019”. 2021. Viitattu 13. heinäkuuta. https://octoverse.github.com/2019/.

Stroustrup, Bjarne. 1998. “An Overview of the C++ Programming Language”. Teoksessa *The Handbook of Object Technology,* toimittanut Saba Zamir. CRC Press LLC. http://www.research.att.com/~bs/crc.pdf.

———. 2020. “Thriving in a Crowded and Changing World: C++ 20062020”. 4. doi:10.1145/3386320.

“The Computer Language Benchmarks Game”. 2021. Viitattu 12. lokakuuta. https://benchmarksgame-team.pages.debian.net/benchmarksgame/index.html.

The Rust Core Team. 2015. “Announcing Rust 1.0”. Viitattu 12. heinäkuuta 2021. https://blog.rust-lang.org/2015/05/15/Rust-1.0.html.

“TIOBE Index”. 2021. Viitattu 13. heinäkuuta. https://www.tiobe.com/tiobe-index/.

Torgersen, Mads. 2017. “The .NET Language Strategy”. Viitattu 16. heinäkuuta 2021. https://devblogs.microsoft.com/dotnet/the-net-language-strategy/.

Trivedi, Pratul, Pranav Kajgaonkar, Abhishek Kulkarni, Nikhil Kolte ja Bhavana Kanawade. 2019. “System Model for Syntax Free Coding”. Teoksessa *2019 Global Conference for Advancement in Technology (GCAT),* 1–5. doi:10.1109/GCAT47503.2019.8978461.

Wirfs-Brock, Allen, ja Brendan Eich. 2020. “JavaScript: The First 20 Years”. 4. doi:10.1145/3386327. https://doi.org/10.1145/3386327.

“Visual Basic documentation”. 2021. Viitattu 16. heinäkuuta. https://docs.microsoft.com/en-us/dotnet/visual-basic/.

Zhang, Jie, Feng Li, Dan Hao, Meng Wang, Hao Tang, Lu Zhang ja Mark Harman. 2019. “A Study of Bug Resolution Characteristics in Popular Programming Languages”. *IEEE Transactions on Software Engineering:* 1–1. doi:10.1109/TSE.2019.2961897.

Liitteet

Liite 1: Eri tahojen listaukset suosituimmista ohjelmointikielistä

“PYPL PopularitY of Programming Language” (2021) Popularity of Programming Language (Worldwide, Apr 2020 compared to a year ago):

1. Python 30.61 % (+3.9 %)
2. Java 18.45 % (-1.9 %)
3. JavaScript 7.91 % (-0.4 %)
4. C# 7.27 % (-0.0 %)
5. PHP 6.07 % (-1.1 %)
6. C/C++ 5.76 % (-0.2 %)
7. R 3.8 % (-0.2 %)
8. Objective-C 2.4 % (-0.4 %)
9. Swift 2.23 % (-0.2 %)
10. TypeScript 1.85 % (+0.2 %)
11. Matlab 1.77 % (-0.2 %)
12. Kotlin 1.63 % (+0.4 %)
13. VBA 1.33 % (+0.0 %)
14. Go 1.26 % (+0.2 %)
15. Ruby 1.23 % (-0.1 %)
16. Scala 0.99 % (-0.1 %)
17. Visual Basic 0.92 % (-0.2 %)
18. Rust 0.67 % (+0.2 %)
19. Abap 0.51 % (-0.1 %)
20. Perl 0.5 % (-0.1 %)

“TIOBE Index” (2021) for April 2020:

1. Java 16.73% (+1.69%)
2. C 16.72% (+2.64%)
3. Python 9.31% (+1.15%)
4. C++ 6.78% (-2.06%)
5. C# 4.74% (+1.23%)
6. Visual Basic 4.72% (-1.07%)
7. JavaScript 2.38% (-0.12%)
8. PHP 2.37% (+0.13%)
9. SQL 2.17% (-0.10%)
10. R 1.54% (+0.35%)
11. Swift 1.52% (+0.54%)
12. Go 1.36% (+0.35%)
13. Ruby 1.25% (-0.02%)
14. Assembly language 1.16% (-0.55%)
15. PL/SQL 1.05% (+0.26%)
16. Perl 0.97% (-0.30%)
17. Objective-C 0.94% (-0.57%)
18. MATLAB 0.93% (-0.36%)
19. Classic Visual Basic 0.83% (-0.23%)
20. Scratch 0.77% (+0.28%)

"GitHut 2.0" (2021) 2020 first quarter pushes:

1. JavaScript 21.0% (-5.9%)
2. Python 14.7% (+0.4%)
3. Java 11.7% (+0.6%)
4. C++ 8.0% (+0.6%)
5. PHP 5.9% (+0.2%)
6. Go 5.0% (+0.8%)
7. C# 4.8% (+0.7%)
8. TypeScript 4.8% (+1.1%)
9. Ruby 4.7% (+0.1%)

10. Shell 4.6% (+0.3%)
11. C 4.0% (+0.4%)
12. Scala 1.3% (+0.3%)
13. Rust 0.9% (+0.3%)
14. Swift 0.8% (-0.02%)
15. Kotlin 0.7% (+0.1%)
16. Perl 0.6% (+0.04%)
17. Objective-C 0.5% (+0.02%)
18. Groovy 0.4% (-0.006%)
19. Vim script 0.4% (+0.006%)
20. Lua 0.3% (-0.1%)

“Interactive: The Top Programming Languages 2018” (2021) 2019 ranking:

1. Python 100.0
2. Java 96.3
3. C 94.4
4. C++ 87.5
5. R 81.5
6. JavaScript 79.4
7. C# 74.5
8. Matlab 70.6
9. Swift 69.1
10. Go 68.0
11. Arduino 67.2
12. HTML,CSS 66.8
13. PHP 65.1
14. Assembly 63.7
15. SQL 63.4
16. Dart 57.4
17. Rust 55.5
18. Scala 55.3

19. Ruby 55.1
20. Visual Basic 55.1

Liite 2: Lyhyissä Pythonia koskevissa kommenteissa käytetyt termit

"Quotes about Python" (2021):

- fast enough
- allows us to produce maintainable features in record times, with a minimum of developers
- project size
- from crowd rendering to batch processing to compositing, Python binds all things together
- extend the capabilities of applications
- providing the glue between applications
- speed of development
- easy to read and use
- high level of a language as you can have without running into functionality problems
- 10 times more productive than Java programmers, and 100 times more than C programmers
- critical ingredient in this high performance system
- PSF is an invaluable resource
- create in record time
- over 50,000 simultaneous players in a shared space simulation
- the flexibilities
- short time-to-market philosophy
- concise, clear syntax
- powerful standard library
- development proceeds rapidly
- maintenance of existing code is straightforward and fast
- extremely productive
- makes maintaining a large and rapidly evolving codebase relatively simple

- flexibility
- many web libraries
- access, build and test in a matter of days rather then the months
- measurable productivity gain
- develop fast and proficient applications
- operate with minimal resources
- need of a scripting language
- powerful, documented, cross-platform standard
- well-supported scripting language
- writing code in a language with garbage collection simply goes faster than writing code in C++

Liite 3: Pidemmissä Python kertomuksissa käytetyt kuvaukset

"Python Success Stories" (2021):

Wingide:

- at least as good an end product
- cross-platform technologies
- speed of the resulting application
- scalability
- rock-solid stability
- strong support for mixed-language development
- a much more productive way to work (simple syntax, dynamic high-level data typing, powerful, easy-to-use data structures, extensive standard library, introspection, faster development and deeper prototyping)

Projectpipe:

- proof-of-concept experiments fast and painless to implement
- development velocity and runtime performance

Devil:

- libraries like PyQT
- reliable and portable system in a relative small time lapse
- simple syntax and clear coding style
- rapid development cycle
- extensive and complete standard library
- great external modules
- easy to build wrappers around a third-party library
- easy and fast code refactoring
- porting application to the different supported platforms is simple
- fast enough
- great exception handling mechanism

Frequentis:

- ability for Python to be embedded into other code
- documentation is excellent and the examples are easy to follow
- straightforward syntax
- extensive standard libraries
- easy to maintain
- runtime error handling

Gusto:

- higher productivity per man hour than Java
- capacity for automating this kind of testing
- OS Independence
- database tools are top notch
- almost everything is included in the Python distribution
- Python has managed to avoid the bloat seen in many other languages
- amount of online technical information available for Python is vast
- Jython for Java
- support for new developers and interns

Gravityzoo:

- developers generally finish projects much quicker than in C, C++, C# or Java
- easy to learn
- highly object oriented architecture
- easily interface to most other languages
- stability of Python
- performance of Python has proven to be much better than expected

St-andrews:

- dynamic nature, support for high-level data structures, and easy object-orientation all lower the barrier to writing well-structured reusable code in less time
- clear and simple syntax
- easy to learn
- Python's dynamic nature and flexibility makes it easier to write generic interfaces

Journyx:

- "1 line of Python is 10 lines of Java or 100 lines of C"
- speed with which features can be written and deployed
- write-once-run-anywhere cross-platform capabilities
- simplicity and clarity of Python
- object-oriented properties
- powerful and productive language
- rich standard library supports programmers in meeting aggressive development schedules
- responsive Python development community
- high degree of backwards compatibility and stability
- cross-platform standard library and platform-independent byte code file format allow the deployment of Python modules to any platform

Forecastwatch:

- many standard libraries useful in collecting, parsing, and storing data from the web
- more lines of code would have been needed working in Java or PHP and integration capabilities of those languages are not as strong, and their threading support is harder to use
- refactoring the Python code is easier than in other languages
- code ideas can easily be tested in the Python interactive shell for shorter edit/test cycle

Liite 4: Systemaattisen kirjallisuuskatsauksen tulokset listana

Tutkielmaa varten taulukkomuotoinen lomake on muotoiltu listaksi järkevämmän tilankäytön vuoksi. Lisäksi listaan on lisätty jokaiselle artikkelille tutkielmassa myöhemmin määritellyt kategoriat. Listan formaatti on seuraava:

Artikkelin nimi

- Artikkelin tallennuspäivämäärä
- Artikkelin kirjoittajat
- Artikkelin julkaisualusta
- Artikkelille itse määritelty kategoria
- Artikkelista havainnoista tehdyt tiiviit lainaukset

Tarkemmat tiedot jokaisesta artikkelista löytyvät lähdeluettelosta.

An Analysis on Python Programming Language Demand and Its Recent Trend in Bangladesh

- 26.03.2021
- Aaquib Javed, Monika Zaman, M. Monir Uddin, Tasnova Nusrat
- ICCPR ’19
- Implikaatio mahdollisesta kielen käytön määrän kehityssuunnasta
- According to our survey, in Bangladesh, students learn as their first programming language is 80.06% C/C++ where 48.06% are using C and 38% uses C++, 5.6% in Java, 8.2% is Python and 5.6% is others. Where the case in Python, 55.6% people hear about Python by other means 28.8% are by seniors and 16.6% are wanted to know

about it. There are 8.6% of professional Python user in Bangladesh and the rest 91.4% are either learning or using other programming languages. Using Python as a main or secondary then 17.6% users are using Python as their main language 70.6% uses it as their second language and the rest of the 11.8% answered that they will use it when they will feel comfortable.

An Overview of the Julia Programming Language

- 26.03.2021
- Tyler A Cabutto, Sean P Heeney, Shaun V Ault, Guifen Mao, Jin Wang
- ICCBD '18
- Kielen vaatimukset menestykseen
- Determining factors for this language to stay predominant and climb up the ladder of popularity will be maintaining a loyal fanbase. This can be done by continuing to have fluent documentation on how to use the language and documentation showing the user how to reach the performance needs as promised. Another factor would be continuing to hold the promise of having greater performance speeds in comparison to C and Python by using them together to break the boundaries each programming language holds.

Anti-Patterns for Multi-language Systems

- 26.03.2021
- Mouna Abidi, Foutse Khomh, Yann-Gaël Guéhéneuc
- EuroPLoP '19
- Tulevaisuuden mahdollinen kehityssuunta sovelluskehityksessä
- Most of the existing systems are multi-language systems and consist of components written in several, different programming languages. Multi-language systems provide many benefits because developers can reuse existing code and take advantage of existing libraries, even if written in different programming languages. Multilanguage systems also raised with the need to include and accommodate legacy code. However, multi-language systems also present challenges to developers: they are difficult to develop, maintain, and evolve because they are more complex than mono-language

systems.

Building Efficient Query Engines in a High-Level Language

- 26.03.2021
- AMIR SHAIKHHA, YANNIS KLONATOS, CHRISTOPH KOCH
- ACM Transactions on Database Systems, Vol. 43, No. 1, Article 4
- Tulevaisuuden mahdollinen kehityssuunta sovelluskehityksessä
- Our approach suggests using high-level programming languages for DBMS development without having to pay the associated abstraction penalty. The key technique to admit this productivity/efficiency combination is to apply generative programming and source-to-source compile the high-level Scala code to efficient low-level C code.

Characterizing the Transition to Kotlin of Android Apps: A Study on F-Droid, Play Store, and GitHub

- 26.03.2021
- Riccardo Coppola, Luca Ardito, Marco Torchiano
- WAMA '19
- Implikaatio mahdollisesta kielen käytön määrän kehityssuunnasta
- On our final set of 1232 applications, we found that 19% of projects featured Kotlin code. Among those projects the transition from Java to Kotlin was most of the times fast and unidirectional: the ratio of Kotlin over total code on those projects was, with few exception, always increasing during their evolution. Projects with Kotlin exhibited, on average, higher values for the popularity metrics.

Do Java Programmers Write Better Python? Studying Off-Language Code Quality on GitHub

- 26.03.2021
- Siegfried Horschig, Toni Mattis, Robert Hirschfeld
- <Programming'18> Companion
- Tulevaisuuden mahdollinen kehityssuunta sovelluskehityksessä
- Comparing the code quality of Python projects edited by Java or C++ programmers to

those edited by Python programmers has a visible effect regarding code quality. Our data supports the assumption that being knowledgeable in Java or C++ can actually make someone a better Python programmer regarding commonly accepted and object-oriented best practices, but not necessarily with respect to Python-specific conventions.

Next-Paradigm Programming Languages: What Will They Look Like and What Changes Will They Bring?

- 26.03.2021
- Yannis Smaragdakis
- Onward! '19
- Tulevaisuuden mahdollinen kehityssuunta sovelluskehityksessä
- Next-paradigm programming languages will need a revolutionary change in level of abstraction, if they are to ever realize large productivity improvements. Such a change will necessarily have many repercussions on the role of the compiler, on the programmer's mental model, and on development patterns.

Some Were Meant for C

- 26.03.2021
- Stephen Kell
- Onward!'17
- Kielen vaatimukset menestykseen
- C's enduring popularity is wrongly ascribed to performance concerns; in reality one large component of it owes to decades-old gaps in migration and integration support among proposed alternatives; another large component of it owes to a fundamental and distinctive property of the language which I have called its communicativity.

A comparative study of Java and Kotlin for Android mobile application development

- 31.03.2021
- Bambang Purnomosidi Dwi Putranto, Robertus Saptoto, Ovandry Chandra Jakaria, Widyastuti Andriyani
- ISRITI '20

- Kielien vertailu
- Testing on an application project showed that Kotlin is superior in terms of lines of code and the number of classes while Java is superior in terms of compilation time and APK size. For multiplatform development, Kotlin is better in framework support as KMM is freely available and fully supported. From a programming language constructs point of view, Kotlin has more advantages and less disadvantages compared with Java. Also, source code for Java is far more verbose than Kotlin which can lead to more bugs and / or code smell. Based on the overall measurement results, it can be concluded that Java should be used if mobile application priority is the APK size and compilation / build time while Kotlin should be used if mobile application development priority is lesser bugs, concise code, and faster development time.

A Study of Bug Resolution Characteristics in Popular Programming Languages

- 31.03.2021
- Jie M. Zhang, Feng Li, Dan Hao, Meng Wang, Hao Tang, Lu Zhang, Mark Harman
- IEEE Transactions on Software Engineering
- Kielien vertailu
- We found evidence that for projects written in Java, bug resolution consumes less time than other languages, while for those written in Ruby, bug resolution consumes more time; we also found that statically typed projects have fixes that occupy more lines and touch more files than dynamically typed ones.

An Empirical Study on the Usage of the Swift Programming Language

- 31.03.2021
- Marcel Reboucÿas, Gustavo Pinto, Felipe Ebert, Weslley Torres, Alexander Serebrenik, Fernando Castor
- 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering
- Kielen vaatimukset menestykseen
- Although experienced developers seem to find the language easy to understand and adopt, a significant proportion of questions on StackOverflow are about arrays and da-

ta types. Even though optional types are pervasive, most of the problems report trivial unwrapping errors. There are also many questions about bugs in the toolset (compiler, Xcode, libraries) and also about error messages that are either hard-to-understand or unhelpful. Interviewees were unanimous suggesting that the Swift compiler needs urgent improvement.

Concurrency Analysis of Go and Java

- 31.03.2021
- Abhinav P Y, Avakash Bhat, Christina Terese Joseph, K Chandrasekaran
- 2020 IEEE
- Kielien vertailu
- Concurrent applications with a large number of computations with fewer repeated thread creation it is advisable to use Java as the primary language. Go should be the preferred language for applications that require continuous production of threads with any amount of computation. Also if memory space is a constraint, it is better to choose Java since the binary sizes of Java are much lower than that of Go. In general, Go has much more built-in concurrency features in the language but Java can also obtain many of the advantages that Go has in that respect by importing thread libraries.

Empirical Analysis of the Growth and Challenges of New Programming Languages

- 31.03.2021
- Partha Chakraborty, Rifat Shahriyar, Anindya Iqbal
- 2019 IEEE 43rd COMPSAC
- Implikaatio mahdollisesta kielen käytön määrän kehityssuunnasta
- In Stack Overflow, we can expect adequate resources of Swift after two years of release while this period is three years for Go. In Stack Overflow, it takes significantly higher time to get the first answer in the evolving state than the matured state of a new language. In Stack Overflow, we found evidence that Go has comparatively less active community support and Rust has a small number of expert developers. New languages are benefited from the community base of the predecessor language.

Evaluating the Inherent Sensitivity of Programming Languages to Soft Errors

- 31.03.2021
- Frederico Cerveira, Alcides Fonseca, Raul Barbosa, Henrique Madeira
- 2018 14th European Dependable Computing Conference
- Kielien vertailu
- With the predicted increase of the soft error rate in coming years due to technological improvements and the popularization of energysaving techniques, the responsibility of tolerating transient hardware faults is no longer solely on the hardware-side, but is becoming more and more a software problem. Current results indicate that compiled languages (C, C++, Rust, Haskell) have lower sensitivity and vulnerability (i.e., soft errors in these languages were less likely to cause a failure) than their counterparts. However, noncompiled languages, despite being more likely to see failures, are significantly less likely to produce silent data corruption.

Impact of Using a Static-type System in Computer Programming

- 01.04.2021
- Ismail Rizky Harlin, Hironori Washizaki, Yoshiaki Fukazawa
- 2017 IEEE 18th International Symposium on High Assurance Systems Engineering
- Kielien vertailu
- When subjects coded from scratch, there is not a significant difference in terms of the number of successfully achieved requirement points between static- and dynamic-type solutions. This applies to both programs. In errors-fixing tasks, a static-type system may be beneficial. Subjects who used a static-type system tended to fix more errors. Additionally, this benefit is more pronounced in encryption programs, which contain more data types.

System Model for Syntax Free Coding

- 01.04.2021
- Pratul Trivedi, Pranav Kajgaonkar, Abhishek Kulkarni, Nikhil Kolte, Bhavana Kanawade

- 2019 GCAT
- Tulevaisuuden mahdollinen kehityssuunta sovelluskehityksessä
- We show that the system works efficiently with more than 80% accuracy. With each iteration, the dataset gets trained and updated, further increasing the precision and recall of the system. We also convey that, with the help of this system the syntax dependency can be eliminated, thereby increasing the user's efficiency.

Effectiveness of Kotlin vs. Java in android app development tasks

- 03.04.2021
- Luca Ardito, Riccardo Coppola, Giovanni Malnati, Marco Torchiano
- Information and Software Technology Volume 127 2020
- Kielien vertailu
- We did not observe any significant difference in terms of maintainability between the two languages.We found a significant difference regarding the amount of code written, which constitutes evidence of better conciseness of Kotlin. Concerning ease of development, the frequency of NullPointerExceptions reported by the subjects was significantly lower when developing in Kotlin. Finally, the IDE support was deemed better for Java than Kotlin