Ossi Johannes Jyrkinpoika Rantala

Automaattisen tekstinkorjauksen hyötyjä lukihäiriön näkökulmasta

Tietotekniikan kandidaatintutkielma

2. kesäkuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Ossi Johannes Jyrkinpoika Rantala

Yhteystiedot: ossi.j.j.rantala@jyu.fi

Ohjaaja: Tytti Saksa

Työn nimi: Automaattisen tekstinkorjauksen hyötyjä lukihäiriön näkökulmasta

Title in English: Automatic coding tools as help for those with dyslexia

Työ: Kandidaatintutkielma

Opintosuunta: Tietotekniikka / Mathematical Information Technology

Sivumäärä: 19+0

Tiivistelmä: Työssä käydään läpi minkälaisia apuja on tarjolla koodaamiseen niille henkilöille, joilla on lukihäiriö. Tutkimuksessa huomataan, että ei ole vain yhtä apuohjelmistoja, jota voisi käyttää aina. Toisin sanoen niin sanottua yleistä ohjelmaa ole olemasssa. Lukihäiriötä on tutkittu enimmäkseen ala- ja yläaste ikäisillä nuorilla, joille hyödyt tulevat oppimisen kannalta enemmän esiin. Näitä apukeinoja voidaan myös yleistää aikuisille, työssäkäyville henkilöille, jotka omaavat lukihäiriön.

Avainsanat: lukihäiriö, lukihäiriöapuri, automaattinen tekstinkorjaus

Abstract: The work explores the available aids for coding for individuals with dyslexia. The study recognizes that relying solely on one assistive software is not always feasible, and there is no one-size-fits-all solution. Dyslexia has mainly been researched among young people as in primary and secondary school, where the benefits are more evident in terms of learning. However, these assistive measures can also be extended to working adults with dyslexia.

Keywords: dyslexia, dyslexia helper, automatic textediting

Termiluettelo

ASAT	Automaattiset staattiset analyysityökalu
Bugi	Virhe ohjelmiston ajossa, joka ei tuota oikeaa tulosta
IDE	Integroitu Kehitysympäristö
linter	Automaattiset staattiset analyysityökalun yksi muoto

Esipuhe

Haluan antaa kiitosta oikolukemisesta ja kommenteista tätä kanditaatin tutkielmaa tehdessä ja neuvoista, kuinka kirjoittaa perempaa ja ymmärrettävämpää tekstiä. Kiitokset kuuluvat Triinu Grünille ja opiskelukavereille.

Kuviot

Sisällys

1 Johdanto

Tämän tutkimuksen tarkoituksena on selvittää lukihäiriöisten näkökulmasta niitä tekijöitä, jotka helpottavat koodaamista erityisesti niiden henkilöiden osalta, jotka eivät kykene tunnistamaan omia kirjoitus- tai huolimattomuusvirheitään koodaamisen aikana. Tutkimuksen avulla pyritään saamaan selville, millaisia apuvälineitä on olemassa ja miten niitä voidaan hyödyntää oppimisessa, työelämässä sekä muissa lukemisen ja kirjoittamisen tarpeissa. Vaikka tutkimuksen pääpainona on lukihäiriöstä kärsivät yksilöt, tuloksia voidaan jonkin verran yleistää myös muihin ihmisiin, joille tietyt konkreettiset apuvälineet eivät välttämättä ole yhtä selkeitä tai hyödyllisiä. Tutkimuskysymyksenä on selvittää lukihäiriöisen näkökulmasta, mitkä koodiapurit ja menetelmät auttavat ohjelmoinnissa.

Koodiapureita löytyy monenlaisia, jotka hyödyttävät toistettavuuden ja kirjotusvirheiden minimoinnissa (Hill ja Rideout 2004). Ueda, Ishio ja Matsumoto (2021) toisaalta tekivät tutkimusta siitä miten voidaan hyödyntää niin sanottua virheiden korjaamista koodin kirjoittamisessa käyttäen apuna analyyttisia työkaluja.

Lukihäiriö on määritetty ICD.Codes -verkkosivustolla, jossa lukihäiriöllä tarkoitetaan pysyvää oppimisvaikeutta, joka vaikuttaa lukemiseen, kirjoittamiseen ja luetunymmärtämiseen. Lisäksi joillain se vaikuttaa myös puheeseen. Tälle ei ole parannusta, koska se on fyysinen häiriö aivoissa, joka ei vaikuta henkilön älykkyyteen (*ICD-10-CM Code F81.0 - Specific reading disorder,*).

2 Minkälaisia apureita on olemassa

Maailmassa on monenlaisia koodiapureita, jotka helpottavat niitä, joilla on jonkinlainen lukivaikeus. Lisäksi paljon tekstiä tuottaessa, ja sen ymmärtämisessä, voi lukihäiriöisen tuottaman tekstin ymmärtäminen tuottaa ongelmia muille henkilöille.

2.1 Ennakointia ja syötettä tulkitseva

Apureita ovat esimerkiksi niin sanotut haisteluohjelmistot, jotka osaavat kertoa, että jokin asia on mahdollisesti kirjoitettu väärin, tai siinä on väärä sisältö koodissa. Apurit voivat auttaa myös kertomaan, mitä tekstissä haetaan. Automaattisessa ehdotuksessa ja täytössä, esimerkiksi osassa integroiduissa ohjelmistoympäristöissä, on myös joitain tietokantoja ja rakenteita, jotka käyttäen kokonaisia polkuja voivat luoda automaattista täyttöä. Tämän lisäksi myös Microsoft Wordissa oleva oikolukuominaisuus voidaan laskea tähän, koska se auttaa oikeinkirjoituksessa ja lauserakenteissa.

2.2 Tekoälypohjaiset

Näiden niin sanotusti kaikille saatavilla olevista valmiiksi asennetuissa työkaluista tulee lisäksi uusia versiota koko ajan, esimerkkinä voidaan käyttää GitHubin kehittämää Copilot-ohjelmistoa, joka on lisäosa Microsoftin Visual studio Codeen. Tämä ohjelmisto hyödyntää tekoälyteknologiaa, jolle voidaan syöttää tietty lyhyt kommentti siitä, mitä haluttaan sen tuottavan, ja se tekee sen automaattisesti. Lisäksi esimerkiksi ChatGPT 4 on hyvä apu näiden ongelmien ratkaisemiseksi kysymyksien avulla. Nämäkin tietysti vaativat sen, että osataan kysyä ja kirjoittaa, mitä halutaan tuottaa.

3 Koodiapureiden hyötyjä

Jokaisessa modernissa graafisessa koodiohjelmassa on sisäänrakennettu, nykyään osittain automaattinen, virheenkorjausominaisuus, joka tunnistaa kirjoitusvirheet. Osassa on myös olemassa automaattinen ehdotus siihen mitä tulisi kirjoittaa ensimmäisten parin kirjaimen avulla. Tämä on jo yleisesti käytössä kaikissa hakukoneissa. Lisäksi Tómasdóttir, Aniche ja Van Deursen (2020) kertovat tutkimuksessaan, että useimmat JavaScript kehittäjät käyttävät jo jonkin näköistä lintteriä pitääkseen oman koodinsa siistinä. Lintterin käyttöön löytyy enemmän syitä juuri koodin yhtäläisyyden näkökulmasta enemmin, kuin virheen korjauksesta. Monet ohjelmoijista haluavat käyttää valmiita pohjia omissa linttereissään ja toimia niiden mukaan. Yksilöintiä kumminkin tapahtuu jonkin verran riippuen minkälainen projekti on kyseessä (Tómasdóttir, Aniche ja Van Deursen 2020). Tähän vaikuttaa myös, tehdäänkö projektia millaisella henkilömäärällä. Kavaler ym. (2019) lisäävät tähän aiheeseen, että oikean työkalun valitseminen projektiin on tärkeää ja tehostaa adaptaatiota niiden käyttöön. Oikean työkalun valitseminen myös helpottaa työskentelyä. Intuitiivisuus, sekä työkalun ominaisuudet työkalun käytössä ovat tärkeä menetelmä sen valitsemiseen.

Kavaler ym. (2019) mukaan ohjelmoijat haluavat käyttää yhtä työkalua ja toimia sen mukaan, koska työkalun vaihtaminen on turhan työlästä, jonka lisäksi se tuottaa liikaa vaikeuksia hallita kaikkia mahdollisia muutoksia. Monet työntekijät valitsevat yhden työkalun projektiinsa, ja pysyttäytyvät siinä sen koko elinkaaren. Jos projektiin adoptoituu jonkin toinen työkalu, muuttavat he myös alkuperäisen työkalun niin sanotusti parempaan vaihtoehtoon, esimerkiksi JSHint vaihdetaan ESLint, joka toimii paremmin, tai on tehokkaampi. Ongelmista huolimatta, työkalut kuten standardJS, coveralls ja david esiintyvät eniten tilanteissa, joissa ne voivat vaikuttavat negatiivisesti niiden jälkitoimintaisten oppimiskäyrien takia. Osa työkaluista ei välttämättä ole alusta asti käyttäjäystävällisiä, tai toimivia ratkaisuja, vaan voivat vaikuttaa negatiivisesti projektiin. Ei ole väliä, missä järjestyksessä valitaan mikäkin työkalu vaan sen toimivuuden ja toiminnon piirteet ovat tärkeä osa sen valintaa. Kesken projektin siis voidaan vaihtaa käytössä olevia aputyökaluja, mutta vain niihin, jotka tarjoavat paremmat ominaisuudet (Kavaler ym. 2019).

Rafnsson ym. (2020) mukaan isot yritykset käyttävät koodiapureita välttääkseen ohjelmoin-

tivirheitä koodissa, sillä ohjelmointivirheet tuottavat enemmän rahallista tappiota (Rafnsson ym. 2020). Koodiapureiden avulla ohjelmista saadaan 'bugivapaita', eli niistä ei tule mitään negatiivista merkintää automaattityökaluihin. Valtaosa muista koodareista taas eivät halua käyttää kunnolla koodiapureita, sillä ne tuottavat äärimmäisen paljon virheilmoituksia. Tämä tuo lisähyötyä tietoturvaan, koska tiedostoissa ei ole mitään ylimääräistä, niin sanotusti hajalla olevaa koodia. Esimerkkinä näistä työkaluista voidaan nimetä Lint, FindBugs ja ESLint. Tärkeä osa linttereiden toimintaan on, että ne analysoivat kolmannen osapuolen kirjastoja. Tämä johtuu siitä että 93% verkkosovelluksista käyttävät avoimen lähdekoodin kirjaston sisältöä, joista 70,5% sisältää näiden kirjastojen haavoittuvuuksia, jotka tunnetaan tai on jo korjattu. Haavoittuvuuksien virheenkorjaus on vaikeaa, koska se vaatii asianmukaista tuntemusta ja syvällistä tietoa koodikannasta. Lintterin voivat ehkäistä tätä ja ovat siksi erittäin iso apu valtaosalle ohjelmoijista. ESLint omaa automaattikorjaustoiminnon ja toteuttaa sen asetettujen sääntöjen mukaan(Rafnsson ym. 2020).

3.1 Virheen korjaus

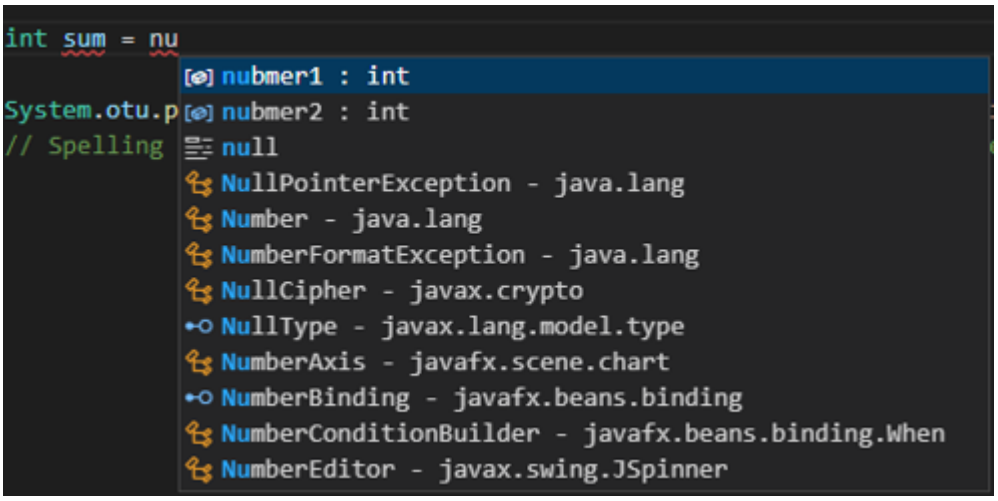
IDE:ssä, eli integroidussa kehitysympäristössä, on sisäänrakennettuja ominaisuuksia, jotka osaavat korjata yleisimpiä kirjoitusvirheitä ja muuttaa ne ymmärrettäväksi tekstiksi. Toimintaperiaate on samanlainen, kuten esimerkiksi Microsoft Wordissa, josta esimerkkinä on kuvio 1. Vassallo ym. 2020 artikkelin mukaan pitkät virheenkorjausistunnot saattavat ilmoittaa virheistä, koska käytetyn kirjaston versio on vaihtunut. CD-Linter pystyy, versiosta riippuen, ilmoittamaan virheet 73% - 100% tarkkuudella, 73% tarkkuudella manuaalisessa toiminnassa. Virheenilmoitustarkkuus on 100% toisintovirheessä, sekä virheellisillä onnistumisilla ja 81% Fuzzy versiolla, jolla tarkoitetaan likimääräistä merkkijonohakualgorimia. Yleensä lintterit kärsivät suurissa määrin virheilmoituksista, jopa 90% ajasta. Virheilmoituksien määrä saatiin laskettua 87%. Suurissa konfigurointitiedostoissa huomattiin, että 31% kehitysputkista reagoivat virheidenetsintään (Vassallo ym. 2020).

```
int nubmer1 = 10; // Spelling error: "nubmer" should be "number"
int nubmer2 = 20; // Spelling error: "nubmer" should be "number"
inr sum = nubmer1 + nubmer2; // Typo: "inr" should be "int"
```

Kuvio 1: Automaattinen kirjoituksen tarkastin, lukee syötettä ja ehdottaa, miten voitisiin kirjoittaa selkeämmin ja ilmoittaa virheistä.
Kuvankaappaus otettu Microsoftin Visual Studio Codesta versio 1.77.0.

3.2 Ennakointi

Ennakoivat syötöt antavat ehdotuksia muutaman kirjaimen perusteella. IDE osaa lukea syötettä ja katsoa, mitä on kirjoitettu jo valmiiksi, ja ehdottaa sen mukaan olemassa olevaa tekstiä. Tämä nopeuttaa kirjoittamista ja vähentää kirjoitusvirheitä. Ennakoinnista ja syötteen täytöstä kuvio 2. Ennakoivaa tekstinsyöttöä käytetäänkin nykyään lähes kaikkialla, kuten nettisivuilla hakuominaisuuksissa, sähköpostissa, sekä esimerkiksi Microsoft Visual Studio Code, Eclipse ja Atom IDE:t sisältää jonkin tavan korjata virheitä.



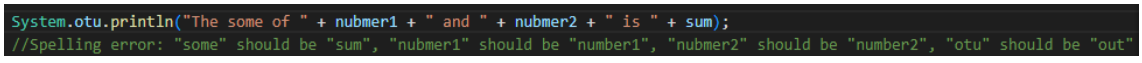
Kuvio 2: Ennakoiva ehdotukset, antaa ehdotuksia siitä mitä voitaisiin haluta ja ottaa huomioon myös aikaisemman kirjoitetun tekstin.
Kuvankaappaus otettu Microsoftin Visual Studio Codesta versio 1.77.0.

3.3 Muita oleellisia työkaluja

Lintterit, kuten ESLint, osaavat katsoa koodia ja ehdottaa toimintoja, jotka korjaavat kirjoitettua tekstiä, sekä ehkäisevät ohjelmointivirheitä ja tietoturvallisuusriskejä. Kuvio 3 kertoo, miten se toimii.

Hill ja Rideout (2004) esittelevät kolmeen kategoriaan jaettavissa olevat tunnistimet: tekstipohjainen, tietorakennepohjainen ja mittauspohjainen. Nämä kolme erilaista tunnistustapaa käyvät läpi koodia. Läpikäynnin aikana tunnistin tunnistaa yhtäläisyydet eri koodien välillä, ja siistii tätä yhdistämällä lähdekoodin ja poistamalla ylimääräisen kopion toisesta lähteestä. Toisaalta selvisi, että tunnistimien jaottelu kolmeen kategoriaan ei ole täysin tarkka, vaan siitä löytyy puutteita. Suurin osa koodista vaati joitain pienempiä koodin osia, jotta saatiin kaikki halutut toiminnat ohjelmistoon. Kopioita etsivä ohjelmistot kuten CPD ja Smian ovat vapaasti käytettävissä omilla lisensseillään (Hill ja Rideout 2004).

Automaattinen Statiikan Analysointityökalu (ASAT) tarkistaa lähdekoodia koodaussääntöjen kunkin ohjelmointikielen mukaan. Säännöissä on määritelty hyvin tunnetut ja usein esiintyvät mahdolliset virheet, sekä huonot käytänteet. Uudemmat työkalut käyttävät jotain modernimpaa teknologiaa, jonka lisäksi vanhentunutta tietoa olisi hyvä poistaa ohjelman mukaan (Ueda, Ishio ja Matsumoto 2021). ASAT automaattinen apuri JavaScript-kieleen auttaa tuottamaan parempaa koodia, jonka lisäksi se on tarkempi merkkaamaan virheet, jotka ovat aikaisemmin merkitty väärin. Tämä puolestaan vähentää turhien virheiden määrää. Rafnsson ym. (2020) kertovat tutkimuksessaan, että näillä työkaluilla voidaan ehkäistä haavoittuvuuksien luomista virheellisten kirjoitusten ja ohjelmistovirheiden avulla. Tutkimuksen mukaan JavaScript on tällä hetkellä suosituin ohjelmointikieli, mikä puolestaan tuottaa haavoittuvuuksia, sillä tekstiä on niin paljon (Rafnsson ym. 2020).



Kuvio 3: Tekoäly pohjaisen tai linterin tekemät ehdotukset kirjoitusvirheiden korjaukseen. kuvankaappaus otettu Microsoftin Visual Studio Codesta versio 1.77.0.

4 Hyötyjä lukihäiriön näkökulmasta

Tässä kappaleessa käsittelen koodiapureiden hyötyjä lukihäiriöisille. Wessel, Kennecke ja Heine (2021) kertovat tutkimuksessaan (Wessel, Kennecke ja Heine 2021), miten on otettu huomioon eri keinoja parantaa verkkosivujen toimivuutta niille henkilöille, joilla on lukihäiriö ja heille joilla sitä ei ole, jotta saadaan molemmille ryhmille toimivampi kokoisuus. Hiscoxin, Leonaviciuten ja Humbyn mukaan olemassa olevalla tekstillä niille ihmisille, joilla on lukihäiriö, sen vaikutukset voidaan huomata työmuistissa. Koodiapureiden ja automaattisen korjauksen on huomattu helpottavan tätä vaikeutta, ja sitä kautta helpottavan tekstin tuottamista ja sen ymmärtämistä. Lukihäiriön omaavat henkilöt eivät voi tunnistaa tekemiään virheitä helposti, ja jos tekstin määrä on suuri tai sen kirjoittamiseen on käytetty ei-erottavia tekijöitä, on ne vaikea erottaa toisistaan (Hiscox, Leonavičiūtė ja Humby 2014).

Saidah Sarpudin ja Zambri (2014) kertovat, että on oleellista, miten saadaan tehtyä verkkosivut käyttäjäystävällisemmäksi heille, joilla on lukihäiriö (Saidah Sarpudin ja Zambri 2014). Näitä samoja periaatteita voimmekin käyttää myös IDEssä, jolloin saadaan paljon helpommin ehkäistyä lukihäiriön omaavien henkilöiden yleisempiä virheitä. Esimerkiksi voidaan tehdä valmis teema lukihäiriöstä kärsiville, jossa olisi joitain pieniä muutoksia yksilöllisiin tarpeisiin. Kirjoitustyylin vaikutukset, kuten isompi fontti, tai tekstin tyylitys, eli millä tavalla teksti on kirjoitettu, ovat oleellisia vaikutteita, sillä osa on helpompilukuista kuin toiset. Kontrastin ero, tekstin väri ja taustan väri ovat myös vaikuttavia tekijöitä. Mitä isompi ero teksteissä on, auttaa lukemaan selkeämmin. Lisäksi tekstin välilyöntien koko, sekä rivivälit helpottavat lukemista, jotta sanojen hahmotus riveillä ei sekoittuisi keskenään. Hyvät värikoodaukset esimerkiksi tekstieditoriin, auttaa lukemaan tekstiä. Näistä hyvä esimerkki on Saidah Sarpudin ja Zambrin tutkimuksessa oleva Table 3 (Saidah Sarpudin ja Zambri (2014), Table 3). Tässä mallissa esitellään esimerkkejä mitä on käytetty tekstin ominaisuuksina. Tutkimuksessa esitellyt väriyhdistelmät voitaisiinkin siis ottaa käyttöön eri tekstieditorien ominaisuuksissa tai koko tietokoneessa. The HOPE on älylaiteohjelmisto, joka auttavat parantamaan kirjottamista, lukemista ja puhumista (Thelijjagoda ym. 2019). Visualinen työkalu erinäköisillä kuvioilla auttaa antamaan taas käyttäjälle paremman kuvan käyttäjän omasta tekstintuoton kehityksestä.

Bergetin ja Sandnesin mukaan tiedonhaussa henkilöt, joilla on lukihäiriö, hakee tietoa melkein yhtä nopeasti ja samalla tavalla kuin muut, mutta heille tulee enemmän kirjoitusvirheitä hakukriteereihin. Tähän auttaa automaattinen tekstinsyöttö. Eroa käyttötavoissa ei ollut automaattisessa tekstinsyötössä eri testihenkilöillä, joilla on lukihäiriö verrattuna testihenkilöihin, jotka eivät kärsi lukihäiriöstä (Berget ja Sandnes 2016).

Hiscoxin, Leonaviciuten ja Humbyn mukaan vaikutus työmuistiin kirjoittaessa on huomattava, ja automaattinen tekstinoikaisu ja tarkistaminen vähensi tätä taakkaa, jonka ansiosta henkilöt pystyivät kirjoittamaan tarkempaa tekstiä (Hiscox, Leonavičiūtė ja Humby 2014). Tästä päätellen se myös auttaa kirjoittamaan, ja keskittymään paremmin koodin tuottamiseen pidemmällä aikavälillä, koska ylimääräistä energiaa ei mene siihen, että pitää koko ajan seurata omaa kirjoittamista. Lukeš (2015) kertoo, että on olemassa monia erilaisia lukuohjelmistoja, mutta yksikään näistä ei tarjoa käyttäjäystävällistä versiota niille, jotka kärsivät jostain tekstintuoton vaikeudesta, esimerkiksi lukihäiriöstä (Lukeš 2015). Näitä lukuohjelmistoja voitaisiin yleistää myös apuvälineinä koodia lukiessa, jotta ei tarvitsisi käyttää ainoastaan manuaalista lukemista selvittämään mitä on koodattu. Hyödyllistä tästä olisi kehittää jotain jo olemassa olevaan IDE järjestelmään, ja personalisoida ne, jotta yksilöt erottavat helpommin tietyt asiat toisistaan. Apuna voitaisiin käyttää Tzouveli ym. (2008) tutkimuksessa ollutta AGED-DYSL-sovellusta, joka on kehitetty tukemaan lapsia, jotka kärsivät lukihäiriöstä. AGED-DYSL yhdistää muun muassa puheen- ja kuvantunnistusta, ja tätä kautta luo automaattisesti profiilin yksilön tarpeiden mukaan, jotta se helpottaisi lukemista (Tzouveli ym. 2008). Tätä voitaisiin hyödyntää myös muussa lukemisessa. Lukihäiriöstä ei voi parantua, joten AGED-DYSL voisi toimia esimerkiksi koodiapureissa, jossa omaversio helpottaisi yksilöä.

Wesselin, Kennecken ja Heinen mukaan monet verkkosivut eivät ole luettavissa yksilöllisesti niille, joilla on jokin ongelma lukemisessa. Apua käyttökokemuksen parantamiseksi voitaisiin lisätä esimerkiksi käyttämällä automaattista tekstinkorjausta, kirjoitusvirheen kompensaatiota haussa, antamalla positiivista palauttetta haun onnistuessa, termistön esittämisellä, tai välttämällä sitä, että kaikki on kirjoitettu isoilla kirjaimilla (Wessel, Kennecke ja Heine 2021).

5 Koodiapureiden tuottamat ongelmat

Tässä kappaleessa tarkastelen mahdollisia ongelmia, mitä koodiapureiden mukana voi tulla, kun niitä pyritään hyödyntämään lukihäiriöstä kärsivien henkilöiden tukena. Näistä kertoo hyvin Berget ja Sandnes (2016), sekä Hiscox, Leonavičiūtė ja Humby (2014). Tässä automaattinen syöttö ja täyttö, sekä virheenkorjaus ja tunnistamisohjelmat voivat taistella keskenään. Koodiapurit eivät osaa kertoa kokonaisuudessaan tilannetta, mikä on esimerkiksi tuotetun ohjelman tai ohjelmiston tarkoitus. Näitä polkuja on vaikea ennustaa tai luoda tyhjästä.

Koodiapurit itsessään eivät tuota ongelmia, mutta joissain niistä tulee huomattava määrä virhekoodeja. Jotkin näistä on harmittomia, mutta niiden määrä olisi hyvä minimoida. Puhdas koodi ei ole aina niin puhdasta, tai se ei ole niin nopeaa, kuin se voisi olla. Rafnsson ym. (2020) kertovat, että isoista yrityksistä ja heidän koodinsa puhtaudesta, niistä ei löydy virheitä. Tämä itsessään voi tarkoittaa, että ne eivät välttämättä ole optimoitu parhaalla mahdollisella tavalla (Rafnsson ym. 2020). Lukihäiriön näkökulmasta tämäkin on osittain hankalaa huomata, joten tekstit yleensä koitetaan saada mahdollisimman puhtaiksi koneen osalta.

Saidah Sarpudin ja Zambri (2014) osoittivat osaltaan väriyhdistelmien merkityksestä verkkosivuilla. Niiden toimivuutta ei ole tutkittu tarpeeksi integroiduissa ympäristöissä. Osa väriyhdistelmistä voi vaikuttaa siltä, että, virheentunnistus toimii, mutta käytännössä virhe voi jäädä huomioimatta (Saidah Sarpudin ja Zambri 2014). Tämä vaikeuttaa suoraan esimerkiksi avun pyytämiseen, sillä nämä ovat yksilöiviä tekijöitä.

6 Yhteenveto

Tässä tutkimuksessa keskityttiin esittelemään erilaisia koodiapureita ja niiden hyötyjä erityisesti lukihäiriöstä kärsivien yksilöiden näkökulmasta. Tutkielmassa koodiapurit jaettiin kolmeen eri luokkaan: ennakoiviin, virheenkorjaus- ja tekoälypohjaisiin koodiapureihin. Tämä jaottelu oli kuitenkin karkea, ja jaottelua voidaan vielä tarkentaa.

Tutkimuksessa tarkasteltiin yleisiä hyödyllisiä tekijöitä koodiapureissa, kuten automaattisen korjauksen vaikutusta kirjoittamisen nopeuteen ja kirjoitusvihjeiden vähentämiseen (Berget ja Sandnes 2016). Samankaltaisia hyötyjä havaittiin myös ennakoivasta syötöstä, joka tarjoaa ehdotuksia seuraavasta kirjoitettavasta sanasta tai lauseesta jo muutamien kirjainten perusteella. Ennakoivan syötön ja virheenkorjauksen havaittiin vapauttavan työmuistia, mikä auttoi tuottamaan laadukkaampaa tekstiä (Hiscox, Leonavičiūtė ja Humby 2014).

Linttereiden toiminnan osalta huomattiin, että ne voivat olla epätäydellisiä pelkästään vanhentuneen koodikannan takia (Ueda, Ishio ja Matsumoto 2021). Lisäksi niillä voi olla herkkiä virheenkertoimia, jotka voivat olla haitallisia optimoinnin näkökulmasta (Vassallo ym. 2020).

Tekoälypohjaisista koodiapureista ei ole vielä esiintynyt tutkimuksia tarpeeksi, jotta niiden hyödystä voitaisiin muodostaa vielä kunnollisia johtopäätöksiä. Tulevaisuudessa voitaisiin siis tutkia, mitä hyötyjä ja haittoja on olemassa näistä tekoälyä käyttävistä ohjelmointiapureista.

Hyödyt voivat olla apuna kaikille riippumatta siitä, kärsiikö jostain oppimisvaikeudesta, kuten lukihäiriöstä. Hyödyt voivat olla muille saavutettavissa, mutta näiden hyötyjen erittely ei ole tämän tutkielman tavoite.

Lähteet

Berget, Gerd, ja Frode Eika Sandnes. 2016. “Do autocomplete functions reduce the impact of dyslexia on information-searching behavior? The case of Google” [kielellä en]. _Eprint: https://asistdl.onlinelibrary.wiley.com/doi/pdf/10.1002/asi.23572, *Journal of the Association for Information Science and Technology* 67 (10): 2320–2328. ISSN: 2330-1643, viitattu 17. tammikuuta 2023. https://doi.org/10.1002/asi.23572. https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.23572.

Hill, R., ja J. Rideout. 2004. “Automatic method completion”. Teoksessa *Proceedings. 19th International Conference on Automated Software Engineering, 2004.* 228–235. ISSN: 1938-4300. Syyskuu. Viitattu 17. tammikuuta 2023. https://doi.org/10.1109/ASE.2004.1342740.

Hiscox, Lucy, Erika Leonavičiūtė ja Trevor Humby. 2014. “The Effects of Automatic Spelling Correction Software on Understanding and Comprehension in Compensated Dyslexia: Improved Recall Following Dictation” [kielellä en]. _Eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1 *Dyslexia* 20 (3): 208–224. ISSN: 1099-0909, viitattu 17. tammikuuta 2023. https://doi.org/10.1002/dys.1480. https://onlinelibrary.wiley.com/doi/abs/10.1002/dys.1480.

ICD-10-CM Code F81.0 - Specific reading disorder. Viitattu 12. maaliskuuta 2023. https://icd.codes/icd10cm/F810.

Kavaler, David, Asher Trockman, Bogdan Vasilescu ja Vladimir Filkov. 2019. “Tool Choice Matters: JavaScript Quality Assurance Tools and Usage Outcomes in GitHub Projects”. Teoksessa *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE),* 476–487. ISSN: 1558-1225. Toukokuu. Viitattu 17. tammikuuta 2023. https://doi.org/10.1109/ICSE.2019.00060.

Lukeš, Dominik. 2015. “Dyslexia friendly reader: Prototype, designs, and exploratory study”. Teoksessa *2015 6th International Conference on Information, Intelligence, Systems and Applications (IISA),* 1–6. Heinäkuu. Viitattu 17. tammikuuta 2023. https://doi.org/10.1109/IISA.2015.7388008.

Rafnsson, Willard, Rosario Giustolisi, Mark Kragerup ja Mathias Høyrup. 2020. “Fixing Vulnerabilities Automatically with Linters” [kielellä en]. Teoksessa *Network and System Security,* toimittanut Mirosław Kutyłowski, Jun Zhang ja Chao Chen, 224–244. Lecture Notes in Computer Science. Cham: Springer International Publishing. ISBN: 978-3-030-65745-1, viitattu 17. tammikuuta 2023. https://doi.org/10.1007/978-3-030-65745-1_13.

Saidah Sarpudin, Siti Nur, ja Suzana Zambri. 2014. “Web readability for students with Dyslexia: Malaysian case study”. Teoksessa *2014 3rd International Conference on User Science and Engineering (i-USEr),* 192–197. Syyskuu. Viitattu 17. tammikuuta 2023. https://doi.org/10.1109/IUSER.2014.7002701.

Thelijjagoda, Samantha, Mithila Chandrasiri, Dilan Hewathudalla, Pasindu Ranasinghe ja Isuru Wickramanayake. 2019. “The Hope: An Interactive Mobile Solution to Overcome the Writing, Reading and Speaking Weaknesses of Dyslexia”. Teoksessa *2019 14th International Conference on Computer Science & Education (ICCSE),* 808–813. ISSN: 2473-9464. Elokuu. Viitattu 17. tammikuuta 2023. https://doi.org/10.1109/ICCSE.2019.8845396.

Tómasdóttir, Kristín Fjóla, Maurício Aniche ja Arie Van Deursen. 2020. “The Adoption of JavaScript Linters in Practice: A Case Study on ESLint”. Conference Name: IEEE Transactions on Software Engineering, *IEEE Transactions on Software Engineering* 46, numero 8 (elokuu): 863–891. ISSN: 1939-3520, viitattu 17. tammikuuta 2023. https://doi.org/10.1109/TSE.2018.2871058.

Tzouveli, Paraskevi, Andreas Schmidt, Michael Schneider, Antonis Symvonis ja Stefanos Kollias. 2008. “Adaptive Reading Assistance for the Inclusion of Students with Dyslexia: The AGENT-DYSL Approach”. Teoksessa *2008 Eighth IEEE International Conference on Advanced Learning Technologies,* 167–171. ISSN: 2161-377X. Heinäkuu. Viitattu 17. tammikuuta 2023. https://doi.org/10.1109/ICALT.2008.236.

Ueda, Yuki, Takashi Ishio ja Kenichi Matsumoto. 2021. “Automatically Customizing Static Analysis Tools to Coding Rules Really Followed by Developers”. Teoksessa *2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER),* 541–545. ISSN: 1534-5351. Maaliskuu. Viitattu 17. tammikuuta 2023. https://doi.org/10.1109/SANER50967.2021.00062.

Vassallo, Carmine, Sebastian Proksch, Anna Jancso, Harald C. Gall ja Massimiliano Di Penta. 2020. “Configuration Smells in Continuous Delivery Pipelines: A Linter and a Six-Month Study on GitLab”. Teoksessa *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering,* 327–337. ESEC/FSE 2020. Virtual Event, USA: Association for Computing Machinery. ISBN: 9781450370431. https://doi.org/10.1145/3368089.3409709. https://doi.org/10.1145/3368089.3409709.

Wessel, D., A.-K. Kennecke ja M. Heine. 2021. “WCAG and Dyslexia Improving the Search Function of Websites for Users with Dyslexia (Without Making ItWorse for Everyone Else)” [kielellä English], 168–179. ISBN: 978-1-4503-8645-6, viitattu 17. tammikuuta 2023. https://doi.org/10.1145/3473856.3473867.