Valtteri Järvinen

HEIKKONÄKÖISTEN OHJELMOIJIEN KOHTAAMAT ONGELMAT OHJELMOINNISSA JA OHJELMAKOODIN ESITTÄMINEN

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2023

TIIVISTELMÄ

Järvinen, Valtteri
Heikkonäköisten ohjelmoijien kohtaamat ongelmat ohjelmoinnissa ja ohjelmakoodin esittäminen
Jyväskylä: Jyväskylän yliopisto, 2023, 103 s.
Kognitiotiede, pro gradu -tutkielma
Ohjaaja(t): Kujala, Tuomo; Häkkinen, Markku

Heikkonäköisyys on kasvava ongelma digitalisoituvassa maailmassa. Erityisesti heikkonäköisyys aiheuttaa ongelmia digitaalisen lukemisen parissa, joka johtaa heikentyneeseen kykyyn tehdä työtä ja joillain aloilla voi jopa estää työn teon tai tiettyihin työtehtäviin osallistumisen. Heikkonäköisyyden ja muiden näön ongelmien odotetaan vain kasvavan tulevaisuudessa, jonka vuoksi myös monet valtiot ja EU ovat säätäneet lakeja digitaalisten palvelujen saavutettavuudesta.

Heikkonäköiset ohjelmoijat ovat vielä lähes tutkimaton kohde. Digitaalinen lukeminen on selvästi ongelma heikkonäköisille ja nykyaikainen ohjelmakoodi lähentelee jo tietyllä tapaa luonnollista kieltä. Heikkonäköisten rajoittunut näkökyky voi heikentää ohjelmakoodin lukemista etenkin tilanteissa, joissa muut ohjelman ymmärtämisen strategiat eivät ole vaihtoehto ja aiheuttaa näin turhaa kognitiivista kuormitusta, joka taas voi johtaa virheisiin tai työtehtävien unohtamiseen. Ohjelmointiin liittyy ohjelmakoodin lukemisen lisäksi monia muita kognitiivisesti kuormittavia tehtäviä, jotka saattavat vaikeutua entisestään kognitiivisen kuormituksen vaikutuksesta.

Tutkimukseen osallistui heikkonäköisiä ohjelmoijia, jotka tekivät ohjelmakoodin lukemistehtäviä verkkotyökalussa. Ohjelmakoodia esitettiin normaalisti, katkaistuna tietyn merkkimäärän kohdalta sekä tutkimukseen luodulla automaattista tekstin rivittämistä mukailevalla ohjelmakoodin ruutuun sovittamisen menetelmällä. Ohjelmakoodin lukemisen tehtävien testaamisen tarkoituksena oli löytää voisiko ohjelmakoodin esittäminen vaikuttaa heikkonäköisen ohjelmoijan kognitiiviseen kuormitukseen. Tämän lisäksi tutkimuksessa kartoitettiin kyselyllä heikkonäköisten ohjelmoijien kohtaamia ongelmia ohjelmointiin liittyen.

Tutkimuksessa todettiin, että tutkimukseen luotu esitystapa ei poikennut tavallisesti esitetystä tai merkkimäärän kohdalta katkaistusta ohjelmakoodin esitystavasta. Tulokseen voi vaikuttaa esitystavan uutuus. Tutkimuksessa luotiin viisi kategoriaa, joissa heikkonäköiset ohjelmoijat kohtasivat ongelmia: haetun tiedon lukeminen, ohjelmakoodin lukeminen, ohjelmointityökalujen käyttäminen, ohjelmakoodin tulosten tulkinta ja sosiaalinen vaikutus. Tutkimus luo pohjaa heikkonäköisyyden ja ohjelmointityön tutkimukselle.

Asiasanat: Heikkonäköisyys, saavutettavuus, ohjelmointi, kognitiivinen kuormitus, digitaalinen lukeminen

ABSTRACT

Järvinen, Valtteri
Problems faced in programming work and presenting programming code for low vision programmers
Jyväskylä: University of Jyväskylä, 2022, 103 pp.
Cognitive Science, Master's Thesis
Supervisor(s): Kujala, Tuomo; Häkkinen, Markku

Digitalization of the world and the increasing amount of people with low vision has started to cause problems in the working environment. Low vision causes problems in reading digital text, which is a core task of many modern jobs, and can cause an inclusion problem in many modern lines of business. The amount of people with some form of low vision is only expected to grow in the future.

Programming is a line of work, which relies heavily on one's cognitive skills. Low vision is a known problem in digital reading, and thus can be a problem in reading programming code, which has started to close the gap between what is a written algorithm and programming code written close to natural language. Problems in digital reading cause cognitive load, which might lead to mistakes or even forgetting certain tasks in an already cognitively loading task such as programming.

Programmers with low vision and at least a beginner level skill in programming were tasked with programming code reading tasks. They were presented with a random set of normally structured code, code that was wrapped to the screen space by a certain number of characters and a style, that was made for this study, which aimed to wrap the code more intelligently to the space available by using the characteristics of programming code and responsive text style typically used on web pages. The aim of the test was to see whether a certain style of presenting programming code would allow for the participants to experience lower cognitive load. Another aim of this study was to begin categorizing problems that low vision programmers experience during programming work.

No difference was observed between normal, wrapped, or responsive style of presentation is SUS (System Usability Score), reading time or error rate. However, the effect of the style of presenting programming code that was made for this study might be weakened by the novelty of the style of presenting. Five categories of problems were developed based on the answers of the pariticipants: reading searched information, reading programming code, usability and accessibility of programming tools, interpreting the results of the program, and social effects. This study forms a base for future research combining programming code reading and low vision.

Keywords: Low vision, accessibility, programming, cognitive load, digital reading

KUVIOT

KUVIO 1 Atkinson-Shiffrin (1986) muistimalli. 14
KUVIO 2 Baddeleyn ja Hitchin (1973) muistin malli. 15
KUVIO 3 Cowanin (1988) työmuistin malli 16
KUVIO 4 Normaalisti rivitetty teksti, suurennettu teksti ilman rivitystä ja suurennettu teksti, joka on rivitetty saatavilla olevaan tilaan. 38
KUVIO 5 Ohjelmakoodi normaalisti esitettynä 40 merkin tilassa ja sovitettuna katkaistulla menetelmällä 40 merkin tilaan. 51
KUVIO 6 Esitystavat esiteltynä verkkosivun näkymässä 59
KUVIO 7 Tehtävien suoritusajat suoritusjärjestyksessä. 63
KUVIO 8 Tehtävien suoritusajat järjestettynä esitystavan ja altistumiskerran mukaan. 63
KUVIO 9 Esitystapojen SUS-pisteiden keskiarvo 64
KUVIO 10 Laatikkokaavio suoritusajoista 65
Kuvio 11 Rivitetyn esitystyylin suoritusajat altistumisjärjestyksessä 67
KUVIO 12 Tutkimustyökalun ensimmäinen sivu, jossa tutkittava perehdytetään tutkimukseen. 93
KUVIO 13 Tutkimustyökalun toinen sivu. Sivulla selvitetään koehenkilön olennaisia tietoja, sekä pyydetään kuvailemaan heikkonäköisyyttä ja ongelmia liittyen ohjelmointiin. 94
KUVIO 14 Tutkimustyökalun kolmas sivu, jossa tutkittavalle kerrotaan tutkimuksen tehtäväosiosta. 94
KUVIO 15 Tutkimustyökalun tehtävä sivun ensimmäinen näkymä, jossa tutkittava valmistautuu aloittamaan tehtävän. Sivu toistuu joka tehtävän aluksi. 95
KUVIO 16 Tutkimustyökalun tehtäväsivun toinen näkymä, jossa tutkittavalle esitetään ohjelmakoodi ja esitystapa satunnaisessa järjestyksessä. Näkymä toistuu jokaisen tehtävän aloittamisen jälkeen satunnaisella ohjelmakoodilla ja esitystavalla. Ohjelmakoodit eivät toistu ja tutkimuksen kolme eri esitystapaa esitetään jokainen kaksi kertaa tehtävien aikana. 95
KUVIO 17 Tutkimustyökalun tehtäväosion jälkeen täytettävä SUS-kysely. Sivu toistuu jokaisen vastauksen jälkeen. 96
KUVIO 18 Tutkimustyökalun viimeinen sivu, jossa koehenkilöt pystvät jättämään palautetta. 97

TAULUKOT

TAULUKKO 1 Esitystapojen ja ohjelmakoodien esiintyvyys suhteessa toisiinsa. 64
TAULUKKO 2 Koehenkilöiden ohjelmointikokemus, heikkonäköisyyden esiintymä ja ohjelmoinnissa kohdatut ongelmat 68

SISÄLLYS

TIIVISTELMÄ
ABSTRACT
KUVIOT JA TAULUKOT

1 JOHDANTO 7

2 KOGNITIIVINEN KUORMITUS 10
2.1 Swellerin kognitiivisen kuormituksen teoria 10
2.1.1 Lyhyt- ja pitkäkestoinen muisti 13
2.2 Kognitiivisen kuormituksen vaikutus 16

3 HEIKKONÄKÖISYYS JA SAAVUTETTAVUUS 20
3.1 Heikkonäköisyys 20
3.2 Avustavat teknologiat ja teknologian käyttö heikkonäköisenä 22
3.3 Saavutettavuus 24
3.3.1 Verkkosisällön saavutettavuusohje (WCAG) 2.1 26

4 KOGNITIIVINEN KUORMITUS JA DIGITAALISEN TEKSTIN LUKEMINEN 29
4.1 Kognitiivinen kuormitus teknologian käyttämisessä 30
4.2 Digitaalisen tekstin lukeminen 34

5 OHJELMOINTIKIELET JA HEIKKONÄKÖISYYS 40
5.1 Ohjelmointikielien lukeminen ja ymmärtäminen 41
5.2 Kognitiivinen kuormitus ohjelmoinnissa 45
5.3 Ohjelmointikielten lukemisen helpottaminen 49
5.3.1 Digitaalisen tekstin sovittaminen ruutuun 50
5.3.2 Värit 51
5.3.3 Kokemus 52
5.3.4 Ohjelmakoodin laatu 52
5.3.5 Tekstin korostaminen 52
5.3.6 Tekstin osien piilottaminen 53

6 TUTKIMUSMENETELMÄ 54
6.1 Hypoteesit ja tutkimuskysymykset 54
6.2 Tutkimusmenetelmät 55
6.3 Proseduuri 56
6.4 Esitystavat 58
6.5 Koehenkilöt 59
6.6 Analysointimenetelmät 60

7 TULOKSET 62
7.1 Tekstin automaattinen rivittäminen ohjelmakoodille 62

7.2 Heikkonäköisten ongelmat ohjelmoinnissa 67

8 POHDINTA 72
8.1 Menetelmän pohdinta 75
8.2 Tutkimuksen merkitys 77

9 YHTEENVETO 79

LÄHTEET 82

LIITE 1 TUTKIMUSTYÖKALUN RAKENNE 93

LIITE 2 OHJELMAKOODIT 98

LIITE 3 SUS-LOMAKKEEN KYSYMYKSET 101

1 JOHDANTO

Teknologian kehitys muuttaa maailmaa jatkuvasti teknologiapohjaisemmaksi ja siten maailma tarvitsee lisää teknologiaa kehittäviä ihmisiä. Maailman teknologisoituminen tarkoittaa kuitenkin sitä, että osa ihmisryhmistä voi jäädä syrjään erilaisten teknologian hyödyntämistä estävien rajoitteiden vuoksi (esim. Kirkpatrick ym., 2018). Suomessa on säädetty Laki digitaalisten palvelujen tarjoamisesta (306/2019), jonka tarkoituksena on tuoda julkishallinnon, sekä muiden yhteiskunnallisesti merkittävien toimijoiden palvelut saavutettavaksi kaikenlaisille käyttäjille. Saavutettavuudella tarkoitetaankin usein sitä, miten digitaalinen sisältö voidaan toteuttaa niin, että eri tavoilla vammaisten on niitä mahdollisimman helppo käyttää (Kirkpatrick ym., 2018). Saavutettavuuden puute voi siten jopa estää digitaalisen sisällön ymmärtämisen ja jopa työn teon ja aiheuttaa näin myös valtavan inkluusio-ongelman.

Heikkonäköisyys on kasvava ja erityisesti digitaaliseen työhön vaikuttava yhteiskunnallinen ongelma. Heikkonäköisyydellä tarkoitetaan näkökykyä, jota ei voida korjata tyypillisin menetelmin eikä salli normaalia elämistä, mutta ei kuitenkaan sokeutta (Maailman terveysjärjestö, 2019; Näkövammaisten liitto, n.d.a). Vaikka heikkonäköisyys on tyypillisesti ikääntyvien ihmisten ongelma (Maailman terveysjärjestö, 2019), niin se ei kuitenkaan tarkoita, etteikö ongelma koskisi myös lapsia, nuoria ja työikäisiä. Arvioiden mukaan maailmassa on jopa lähes 300 miljoonaa heikkonäköistä (Bourne, 2020; Pascolini & Mariotti, 2012), joista jopa 30 % voi olla nuoria tai työikäisiä (Flaxman ym., 2021; Ojamo & Tolkkinen, 2021). Heikkonäköisyyden on todettu vaikuttavan yhteiskunnallisella ja yksilöllisellä tasolla negatiivisesti moneen asiaan, kuten työllistymiseen, sosiaaliseen toimintakykyyn ja mielenterveyteen (Bassey & Ellison, 2020; Nyman, 2010; Welp ym., 2016). Heikkonäköisten määrän odotetaan myös kasvavan tulevaisuudessa (Maailman terveysjärjestö, 2019).

Digitaalisen sisällön saavuttamattomuus voi heikkonäköisellä vaikeuttaa työntekoa, sosiaalista elämistä tai harrastamista. WebAIMin (2018, 2021a, 2021b, 2022) tekemien tutkimusten perusteella suuri osa käyttäjistä ei koe saavutettavuuden paranevan ja osa kokee jopa sen huononevan, mutta myöskään saavu-

tettavuuden parissa työskentelevät eivät koe tilannetta hyväksi. Tilanteen helpottamiseksi muun muassa Euroopan Unioni on asettanut direktiivin 2016/2102 julkisen sektorin elinten verkkosivustojen ja mobiilisovellusten saavutettavuudesta (Direktiivi 2016/2102), jolla turvataan ainakin yhteiskunnallisesti merkittävien tahojen palveluiden saavutettavuus, jotta perusturva maailman digityalisoituessa säilyy myös rajoitteellisilla henkilöillä.

Maailman digitalisoituessa se tarvitsee myös erilaisia tekijöitä erilaisten digitaalisten työkalujen tuottamiseen ja tarpeiden täyttämiseen. Heikkonäköiset ohjelmoijat ja heikkonäköisyyden vaikutus ohjelmakoodin lukemiseen on todella heikosti tutkittu aihe tieteessä. Tekstipohjaisen ohjelmakielen ja visuaalisten ongelmien yhdistäminen voi aiheuttaa monenlaisia haasteita kuten digitaalisen tekstin lukemisessakin. Erityisesti heikkonäköisellä on vaikeuksia sivun ja hypertekstin navigoinnissa, tekstin lukemisessa, merkkien tunnistamisessa ja erityisesti tekstin ymmärryksessä, joka voi johtua esimerkiksi raskaasta kognitiivisesta kuormituksesta, joka taas johtuu toissijaisten toimintojen, kuten hypertekstin navigoinnin, vaikutuksesta (esim. Arditi & Cho, 2007; Chan, 2017; Dick, 2018; Hallett, Arnsdorff ym., 2015; Hallett, Roberts ym., 2015; Harrison, 2004; Legge, 2016; Sauvan ym., 2020). Ohjelmointikielten lukemista ei ole tutkittu riittävästi, mutta koska varsinkin korkean tason ohjelmointikielet vastaavat jo lähes luonnollisella kirjoitettuja algoritmeja, niin luonnollinen lähtökohta sen tutkimukselle on soveltaa digitaalisen tekstin lukemisen tutkimusta. Luonnollisen ja ohjelmointikielen erottaa kuitenkin välimerkkien ja tyhjän tilan käyttö (Oman & Cook, 1990), mutta toisaalta ohjelmakoodissa normaalista sanastosta ja kielestä poikkeaminen kasvattaa kognitiivista kuormitusta (esim. Fakhoury ym., 2018; Lawrie ym., 2006). Tämän tutkimuksen yhtenä tavoitteena onkin selvittää, että miten ohjelmakoodin lukemista voidaan helpottaa ja voidaanko digitaalisen tekstin lukemisen helpottamisen menetelmiä soveltaa ohjelmointikielille.

Tutkimus pohjautuu oletukseen, että heikkonäköisyys nostaa kognitiivista kuormitusta ohjelmoinnissa. Kognitiivisella kuormituksella tarkoitetaan tilannetta, jossa työmuisti ylikuormittuu liiallisen tiedon vaikutuksesta (Sweller, 1998). Swellerin (1998) kognitiivisen kuormituksen teoriassa kognitiivista kuormitusta on kolmenlaista: sisäistä, olennaista ja ulkoista. Sisäinen kuorma syntyy siitä, kuinka vaikea tehtävä tai materiaali on ymmärtää ja olennainen taas liittyy oppimiseen liittyviin muistitoimintoihin. Ulkoinen kuorma syntyy siitä, miten informaatio on esitetty. Etenkin heikkonäköisten tapauksessa ohjelmointiin liittyvä lukeminen ja muut digitaaliseen lukemiseen liittyvät oheistoiminnot saattavat aiheuttaa ulkoista kuormaa ja näin vaikeuttaa jo lähtökohtaisesti kognitiivisesti raskasta ohjelmointia entisestään. Tämä voi johtua siitä, että ohjelmakoodin lukeminen vaatii eri tiedon yhdistelyä ja riippuvuuksien havaitsemista (esim. Brooks, 1983; Kelly & Buckley, 2009; Weiser, 1981), mutta heikkonäköisyydestä johtuva suuremman tietomäärän muistissa pitäminen ja yhdistäminen voi häiritä tätä prosessia. Ohjelmoijat välttelevätkin usein ohjelmakoodin lukemista ja suosivat muita strategioita ymmärtää ohjelman toiminta

(Lee ym., 2016; Roehm ym., 2012). Ohjelmoinnin kognitiivinen kuormitus voi johtua erityisesti siihen liittyvistä alatehtävistä, jotka vaativat erimuotoisen tiedon ymmärtämistä ja yhdistelyä sekä eri taitoja kuten ohjelmointi, debuggaaminen ja ohjelman suunnittelu ja abstraktion ymmärtäminen, sekä niiden jatkuvasta vaihtelusta ohjelmaa luodessa tai ongelmaa ratkaistessa (Helgesson ym., 2019; Kelly & Buckley, 2009; Pennington & Grabowski, 1990; Roehm ym., 2012; Sillito ym., 2006). Kognitiivinen kuormitus ohjelmoinnissa voi johtaa jopa tiettyjen osatehtävien unohtamiseen (Huang, 2016).

Tutkimuksessa tavoitteena on tarkastella voidaanko digitaalisen tekstin lukemisen helpottamisen menetelmiä soveltaa ohjelmakoodille siten, että heikkonäköinen ohjelmoija kokisi pienempää kognitiivista kuormaa. Tässä tutkimuksessa erityisesti tarkastelun kohteena on tekstin automaattinen rivittäminen, joka tunnetaan myös responsiivisuutena, ja on todettu yhdeksi tehokkaimmista keinoista helpottaa heikkonäköisen henkilön digitaalisen tekstin lukemista erityisesti kognitiivisen kuormituksen osalta (esim. Chan, 2017; Dick, 2018; Hallett, Arnsdorff ym., 2015). Tutkimukseen luotua responsiivista ohjelmakoodin rivittämistä verrataan normaalisti kirjoitettuun ja rivitettyyn ohjelmakoodiin, sekä lähes kaikista moderneista ohjelmointiympäristöistä löytyvään tietyn merkkimäärän kohdalta katkaistuun ohjelmakoodin ruutuun sovittamismenetelmään. Vertailu tehdään virheiden ja suoritusaikojen, sekä esitystavan käyttökokemuksen perusteella ja näin voidaan todeta, että onko eri metodien välillä jotain eroa.

Tutkimukseen osallistui heikkonäköisiä henkilöitä, joilla on vähintään aloittelijan tasoinen kokemus ohjelmoinnista. Heikkonäköisyyksien esiintymien kirjo tutkimuksessa oli suuri. Tutkittavat suorittivat ohjelmoinnin lukemiseen liittyviä tehtäviä tutkimukseen luodussa verkkotyökalussa, joissa eri ohjelmakoodeja esitettiin joko normaalilla, merkkimäärän kohdalta katkaistulla tai tutkimukseen luodulla responsiivisella esitystavalla.

Toisena tavoitteena tutkimuksessa on kartuttaa heikkonäköisten ohjelmoijien kohtaamia ongelmia ohjelmointiin liittyen, jotta ongelma aluetta voidaan ymmärtää paremmin ja näin tukea tulevaisuuden tutkimusta. Tutkimuksen teoriaosiossa pyrittiin myös löytämään myös puutteita aihealueen tutkimuksessa. Tutkimusalueen uutuuden vuoksi on tärkeää ymmärtää aluetta monelta kannalta, jotta tutkimusta voidaan suunnata oikeaan suuntaan. Samalla tämä tutkimus luo pohjaa tulevalle tutkimukselle kategorioimalla erilaisia heikkonäköisten ohjelmoijien kohtaamia ongelmia, joka voi toimia tukena tai lähtökohtana tulevalle tutkimukselle.

Luvussa 2 esitetään tutkimuksen teoreettinen lähtökohta, eli kognitiivinen kuormitus. Luvussa 3 määritellään heikkonäköisyys ja saavutettavuus. Luvussa 4 selvitetään kognitiivisen kuormituksen, teknologian käyttämisen ja siten myös digitaalisen tekstin lukemisen yhteyttä yhdistämällä tutkimusta monelta alalta. Luvussa 5 pohditaan miten ohjelmointikielten lukeminen, kognitiivinen kuormitus ja heikkonäköisyys näyttäytyvät tieteellisessä tekstissä. Luvussa 6 esitellään, miten tutkimus on suoritettu. Luvussa 7 esitellään tutkimuksen tulokset ja Luvussa 8 pohditaan tutkimuksen tuloksia sekä itse tutkimuksen merkitystä ja toteutusta. Tutkimuksen yhteenveto tehdään Luvussa 9.

2 KOGNITIIVINEN KUORMITUS

Tämän tutkimuksen teoreettisena pohjana toimii kognitiivisen kuormituksen teoria. Kognitiivisen kuormituksen (eng. *cognitive load theory*), selittää sitä, miten oppimista voidaan edistää vähentämällä oppijan kognitiivista kuormaa (Sweller, 1988). Kognitiivinen kuorma (eng. *cognitive load*) rasittaa työmuistia ja vähentää työmuistissa käytössä olevien resurssien määrää. Teorian yksi merkittävimmistä ja tälle tutkimukselle olennaisimmista löydöistä on se, että kognitiivinen kuormitus voi vaikuttaa negatiivisesti tehtävästä suoriutumiseen. Negatiivinen suoriutuminen voi johtua joko informaation liiallisesta määrästä tai sen vaikeasta opittavuudesta.

2.1 Swellerin kognitiivisen kuormituksen teoria

Sweller (1988) tutki miten oppiminen ja ongelmanratkaisu kytkeytyvät toisiinsa. Swellerin (1988) argumentti kokeneiden ja aloittelevien ongelmanratkaisijoiden erolle on se, että kokeneilla on kehittyneemmät mielensisäiset rakenteet (skeema, eng. *schema*) erilaisista ongelmanratkaisutilanteista ja nämä skeemat toimivat pääasiallisena tekijänä ongelmanratkaisussa. Aloittelevilla ratkaisijoilla mielensisäiset rakenteet eivät ole vielä kehittyneet, joten ongelman ratkaisuun valittu strategia ei ole optimaalinen (Sweller, 1988).

Tutkimuksessaan Sweller (1988) tutki käyttäjän kognitiivisen kuorman kehittymistä ongelmanratkaisussa, kun tutkittava suorittaa kaksi peräkkäistä tehtävää. Jos toisen tehtävän suorittaminen ensimmäisen jälkeen hidastuu merkittävästi, niin sitä voidaan pitää merkkinä kognitiivisesta kuormittumisesta ensimmäisessä tehtävässä. Ensisijaisena tehtävänä tutkittavat ratkaisivat trigonomiatehtäviä, joissa annetun informaation määrä vaihteli. Toissijaisena tehtävänä tutkittavien tuli täydellisesti toisintaa nykyistä edeltävässä tehtävässä esitetty kuva. Tutkittavat jaettiin kahteen ryhmää, joista toista ohjeistettiin laskemaan niin monta kolmion sivun pituutta kuin mahdollista kun taas toista ryhmää pyydettiin laskemaan tietty sivu. Tuloksena löydettiin, että vaikka ensi-

sijaisesta tehtävästä suoriutumisessa tutkittavien ajat eivät juuri eronneet, niin ryhmä, jolle oli annettu tarkempi ohjeistus teki enemmän virheitä aiemman tehtävän kuvan jäljentämisessä. Ryhmällä, jolle ei ollut annettu tarkempaa ohjeistusta oli siis vähemmän kognitiivista kuormitusta.

Teoria rakentuu aiempien ihmisen muistin rajallisuutta käsittelevien tutkimuksen pohjalle. Swellerin (1988) teoria pohjautuu aiempien muistin rajallisuutta käsittelevien tutkimusten päälle. Teoriassa oletetaan työmuistin olevan rajallinen (esim. Miller, 1956) sekä, että oppiminen tapahtuu siirtämällä tietoa työmuistista pitkäkestoiseen muistiin (esim. Baddeley & Hitch, 1974). Kognitiivisella kuormituksella tarkoitetaan työmuistissa olevien resurssien määrää ja teorian yksi tärkeimmistä ajatuksista on se, että kun työmuistia kuormitetaan liiallisella informaatiolla niin informaation tallentamisen ja käsittelyyn liittyvät toiminnot häiriintyvät ja oppimista ei voi tapahtua (Sweller, 1988).

Tutkimuksellaan Sweller (1988) osoitti, että ongelmanratkaisu ja oppiminen voivat olla toisiaan häiritseviä muistitoimintoja. Swellerin (1988) mukaan skeemojen muodostaminen on tärkeimpiä ongelmanratkaisuun liittyviä tekijöitä. Skeema (eng. s*chema*) on mielensisäinen rakenne, jonka varaan voidaan rakentaa tietoa. Skeemat ovat rakenteita, jotka auttavat yhdistämään eri tiedonjyviä uusiksi muistirakenteiksi ja jotka sisältävät opittua tietoa (Sweller, 1988). Skeemojen avulla ihminen voi parantaa havaintoinformaation ymmärtämistä ja nopeuttaa informaation prosessointiin tarvittavia muistitoimintoja. Ihminen tallentaa samankaltaista tietoa skeemoihin, jolloin asiaan liittyvän tiedon oppiminen ja muistiin palauttaminen on tehokkaampaa. Swellerin (1988) mukaan perinteinen ongelmanratkaisu johtaa vain ratkaisun löytämiseen eikä skeemojen muodostumiseen, koska se johtaa isompaan kognitiiviseen kuormaan. Tällöin siis ongelmanratkaisua suorittamalla ei välttämättä päädytä parempiin ongelmanratkaisutaitoihin. Vähentämällä tehtävän luomaa kognitiivista kuormaa voitaisiin helpottaa skeemojen muodostamista, jolloin ongelmanratkaisukyky kasvaa (Sweller, 1988). Oppimateriaalin kuuluisi olla tehty niin, että se mahdollistaa skeemojen muodostumisen (Sweller, 1988).

Kognitiivinen kuorma voidaan jakaa kolmeen tyyppiin: sisäinen (eng. *intrinsic*), ulkoinen (eng. *extrinsic*) ja olennainen (eng. *germane*) kognitiivinen kuorma. Sweller, Marrienboer, Jeroen ja Paas (1998) esittelivät nämä kolme tyyppiä tutkimuksessaan, jossa he laajensivat kognitiivisen kuormituksen teoriaa. Sisäinen kuorma liittyy informaation monimutkaisuuteen. Esimerkiksi yhteen- ja vähennyslaskujen opettelu on helpompaa kuin integraalien ja derivaattojen opettelu, koska derivoimisen opettelu vaatii pohjalleen yhteen- ja vähennyslaskujen lisäksi muita taitoja, kuten kertolaskun hallitsemista. Swellerin ym. (1998) mukaan opittavaan asiaan liittyy jonkinasteinen vaikeustaso, joka määrittää kuinka paljon kyseinen materiaali aiheuttaa kognitiivista kuormaa. Vaikeustaso voi vaihdella oppijan mukaan, koska skeemoihin voidaan tallentaa tietoa, joka auttaa materiaalin oppimisessa ja tulkitsemisessa vähentäen kognitiivista kuormaa, mutta itse opittavan asian vaikeutta ei voida suoraan muuttaa (Sweller ym, 1998). Ulkoinen kuorma liittyy materiaalin esitystapaan ja on kontrolloitavissa toisin kuin sisäinen kuorma. Swellerin ym. (1998) mukaan ul-

koinen kuorma on eritoten tärkeä, koska kontrolloimalla sen aiheuttamaa kognitiivista kuormaa voidaan vähentää työmuistin kuormitusta, jolloin muun tyyppiselle kognitiiviselle kuormalle jää tilaa. Jos taas ulkoisen tyypin aiheuttama kuorma on suuri (esimerkiksi kun ohjeet ovat epäselvät), niin ulkoinen tyyppi vie kaiken työmuistin kapasiteetin ja suoriutuminen ja oppiminen voivat kärsiä. Esimerkki ulkoisen kuorman vähentämisestä on esimerkiksi tekstin tärkeimpien asioiden lihavointi, jolloin tekstin pääasiat on helppo poimia ja tekstiä kerratessa palauttaa mieleen. Olennainen tyyppi liittyy skeemojen muodostumiseen, prosessointiin ja käyttöön, eli muistitoimintoihin, jotka edesauttavat oppimista. Esimerkiksi reseptissä ruskean kastikkeen tekeminen saattaa olla yksi osa ohjetta ja sisältää useamman vaiheen. Kokemuksen karttuessa ohje muuttuu yksittäisistä ruskean kastikkeen teon vaiheista vain yhdeksi tehtäväksi "tee ruskea kastike". Sweller ym. (1998) ehdottavatkin, että ohjeistuksen suunnittelussa on tärkeää tukea olennaista tyyppiä, jotta skeemojen muodostamista ja prosessointia helpotetaan. Ohjeistuksella kognitiivisten toimintojen tukeminen vähentää ulkoisen tyypin aiheuttamaa kognitiivista kuormaa, jotta muille tyypeille jää enemmän kapasiteettia.

Kognitiiviseen kuormitukseen liittyy Swellerin ym. (2011) mukaan myös erilaisia vaikutuksia. Swellerin ym. (2011) listaamat yhdeksän vaikutusta kuvaavat erilaisia kognitiivisen kuormituksen esiintymiä, sekä niiden vaikutusta suoriutumiseen. Tavoitteettomuusefekti (eng. *goal-free effect*) kuvastaa tilannetta, jossa ohjeistuksen tavoitetilaa ei määritellä tarkasti, vaan ohjeistuksella sallitaan laajempi ratkaisujen joukko, jossa oppija voi ratkaista joukkoa pitkälle kuin osaa. Sweller ym. (2011) käyttävät esimerkkinä kolmion kulmien laskemista: sen sijaan, että pyydetään ratkaisemaan yksi tietty kulma, niin pyydetään ratkaisemaan mahdollisimman monta kulmaa. Käytännön esimerkkien efekti (eng. *worked example effect*) ja ongelman loppuun saattamisen efekti (eng. *problem-completion effect*) ovat läheisiä efektejä. Käytännön esimerkeissä opiskelijalle tarjotaan valmis esimerkki, kun taas ongelman loppuun viemisessä oppijalle tarjotaan osittain tai lähes valmiiksi ratkaistu tehtävä, joka hänen kuuluu tehdä loppuun. Kummassakin efektissä oppija ikään kuin lainaa kokeneelta tekijältä valmiita skeemoja ratkaisun opiskelun tueksi. Jaetun huomion efekti (eng. *split-attention effect*) havaitaan silloin, kun oppijalle tarjotaan informaatiota kahdesta lähteestä, jotka ovat joko ajallisesti tai paikallisesti erillään, mutta joista kumpikin on olennaista suuremman kontekstin ymmärtämiselle. Tässä efektissä havaitaan vahvaa ulkoista kuormitusta, koska järkevämpi vaihtoehto olisi tarjota tieto yhtenäisenä kokonaisuutena ilman, että oppijan tarvitsisi yhdistellä tietoa, jota on mahdollisesti jopa enemmän kuin kognitiivista kapasiteettia tarjolla. Efektin huomaamiseksi tiedon täytyy olla sellaista, että sitä ei voi yksinään tulkita tai yhdistää. Modaliteettiefekti (eng. *modality effect*) on hyvin lähellä jaetun huomion efektiä, koska modaliteettiefektissä tarjotaan tietoa kahdessa eri muodossa: visuaalinen ja audio. Samoin kuin jaetun huomion efektissä, niin tämän efektin havaitsemiseksi tiedon täytyy yhdessä muodostaa järkevä kokonaisuus. Tarpeettomuusefekti (eng. *redundancy effect*) kuvaa sitä, että sama informaatio esitetään kahteen kertaan eri muodossa, jolloin tehottomampi esitysmuoto on

oppimisen kannalta tarpeeton. Asiantuntemuksen käänteinen vaikutusefekti (eng. *expertise-reversal effect*) selittää, että mitä enemmän asiantuntemusta tai kokemusta aihealueesta on, niin sitä käytännöllisempää tarjotun tiedon tulee olla. Asiantuntijan kognitiivinen kuormitus kasvaa aloittelijoille tarkoitetusta materiaalista ja toisaalta aloittelija ei hyödy liian vaativista tehtävistä. Vastapaino asiantuntemuksen käänteiselle vaikutusefektille on ohjauksen häviämisen efekti (eng. *guidance fading effect*), jonka mukaan mitä kokeneempi oppija on, sitä vähemmän ohjausta ja ohjeistusta hän tarvitsee oppimiseen. Mielikuvitusefekti (eng. *imagination effect*) kuvastaa sitä, että asiantuntija voi siirtää tiedon pitkäkestoiseen muistiin parhaiten kuvittelemalla ohjeistuksen sisältämän tiedon.

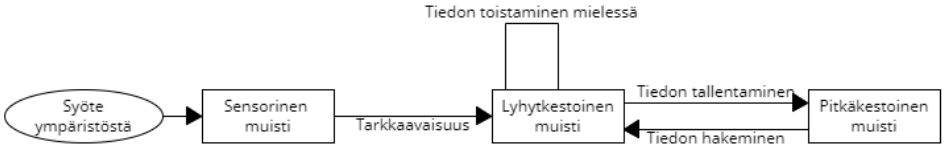
2.1.1 Lyhyt- ja pitkäkestoinen muisti

Pitkäkestoisen muistin rajattomuus ja pitkäkestoisuus ja lyhytkestoisen muistin (työmuisti) rajallisuus ja lyhytkestoisuus ovat ominaisia monille muistin teorioille, joista kognitiivisen kuormituksen teoria on syntynyt. Tämän tutkimuksen kannalta on tärkeää ymmärtää miten asiat säilyvät muistissa ja miten paljon muistia voidaan hyödyntää tehtävää suorittaessa. Swellerin (1988) teorian mukaan ihmisen työmuistin kuormittaminen vaikeuttaa tehtävistä suoriutumisesta. Työmuistin tarkoituksena on käsitellä tietoa, jota aistit vastaanottavat ja jotka ovat senhetkiselle tehtävälle olennaisia. Pitkäkestoisessa muistissa säilytetään opittua tietoa ja tieto voidaan säilyttää siellä hetkestä ikuisuuteen. Tieto käsitellään työmuistissa, jonka jälkeen se joko tallennetaan pitkäkestoiseen muistiin skeemoina tai unohdetaan (Sweller, 1988). Pitkäkestoisessa muistissa säilytetään skeemoja, joiden varaan voidaan rakentaa uutta tietoa. Muistin kapasiteetti vaikuttaa siis suoraan kognitiiviseen kuormitukseen: kun muisti täyttyy suuremmasta määrästä tietoa kuin mitä keretään prosessoida, niin kognitio kuormittuu. Skeemojen avulla tietoa voidaan kategorisoida, jolloin työmuistin tarve vähenee ja näiden skeemojen olemassaolo on olennainen ero kokeneen ja aloittelijan välillä (Sweller, 1988).

Swellerin (1988) tutkimuksen pohjalla on käsitys siitä, että työmuisti on rajallinen ja yksi teorian syntyyn vaikuttaneista teorioista oli Millerin (1956) teoria työmuistin kapasiteetista. Millerin (1956) teorian mukaan pitkäkestoisen muistin ja työmuistin erona on niiden koko ja kesto. Pitkäkestoisella muistilla on teoreettisesti loputon kapasiteetti ja kesto, kun taas työmuistissa voidaan käsitellä vain vähän tietoa kerrallaan ja tieto pysyy siellä maksimissaan sekunteja (Miller, 1956). Millerin (1956) teorian mukaan ihminen pystyy pitämään 7 ± 2 asiaa mielessään yhdellä kertaa. Vaikka monet myöhemmät tutkimukset pitävät työmuistin muistin rajallisuutta ja pitkäkestoisen muistin äärettömyyttä oikeana, niin kuitenkin työmuistin kapasiteetistä on myöhemmin kiistelty (esim. Cowan, 1988).

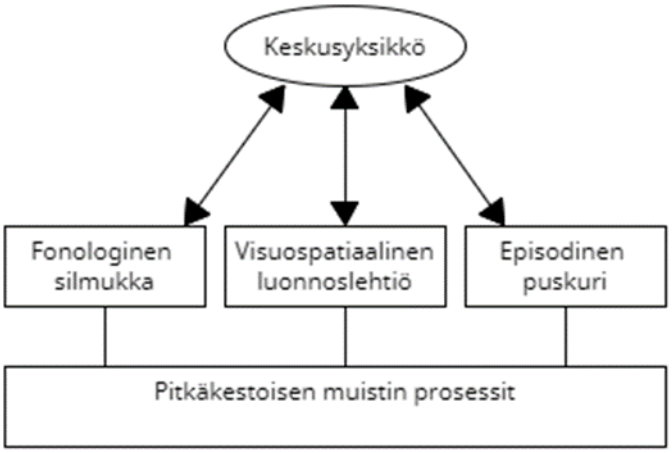
Muisti ei kuitenkaan ole niin yksinkertainen, että siellä vain prosessoidaan tietoa rajallisen kapasiteetin rajoittamana. Atkinsonin ja Shiffrin (1968) (Atkinson-Shiffrin muistimalli) mallissa muisti on monitasoinen ja koostuu kolmesta tasosta: aistimuisti, lyhytkestoinen muisti ja pitkäkestoinen muisti (kts. Kuvio 1).

Mallissa pohjana on sensorinen muisti (aistimuisti), jossa aistien ympäristöstä vastaanottama tieto käsitellään ja jossa tiedosta valikoidaan vain olennainen tieto käsiteltäväksi lyhytkestoiseen muistiin (“tarkkaavaisuus”). Mallin mukaan tietoa voidaan muokata kolmeen muotoon muistiin tallentamista varten: visuaalinen (kuvat), akustinen (ääni) ja semanttinen (merkitys). Sensorisessa muistissa jokaisella aistilla on oma muoto ja tieto pysyy muistissa alle sekunnin. Lyhytkestoisessa muistissa tieto prosessoidaan ja toistamalla tietoa se voidaan pitää mielessä huomattavasti pidempään, jopa 30 sekunnin ajan. Jos tietoa ei toisteta, niin se korvautuu uudella tiedolla tai unohdetaan. Toisin sanoen tietoa täytyy toistaa jatkuvasti mielessä tai se unohtuu, koska muistin kapasiteetti täyttyy muusta informaatiosta. Lyhytkestoisessa muistissa tieto on akustista. Tieto siirtyy lyhytkestoisesta muistista automaattisesti pitkäkestoiseen muistiin sen perusteella, miten pitkään tietoa on ylläpidetty (toistettu) työmuistissa. Pitkäkestoinen muisti on kapasiteetiltaan ääretön ja tietoa voidaan tallentaa äärettömän pitkän ajan. Pitkäkestoisesta muistista voidaan hakea tietoa lyhytkestoiseen muistiin prosessoitavaksi, jolloin uutta tietoa voidaan luoda linkittämällä se jo olemassa olevaan tietoon.



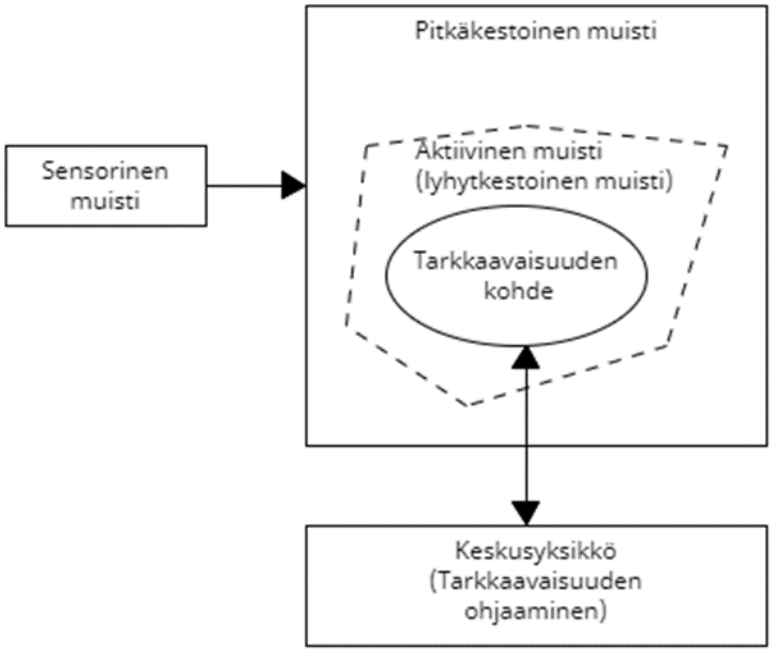
KUVIO 1 Atkinson-Shiffrin (1986) muistimalli.

Toisena tärkeänä innoittajana kognitiivisen kuormituksen teoriassa toimi Baddeleyn ja Hitchin (1974) teoria, joka osoitti, että muisti ei kuitenkaan ole aivan niin yksiselitteinen kuin Atkinson-Shiffrin (1968) mallissa. Baddeleyn ja Hitchin (1974) mallissa lyhytkestoisesta muistista puhutaan työmuistina ja se koostuu useista osista (Kuvio 2). Keskusyksikkö (eng. *central executive*) suorittaa tiedon käsittelyn ja apuyksiköt (eng. *slave system*) suorittavat tiedon ylläpitämisen. Keskusyksikkö pitää huolta siitä, että työmuisti ja pitkäkestoinen muisti toimivat yhdessä ja että tietoa päivitetään ja korvataan tarvittaessa. Keskusyksikön ja apuyksiköiden kapasiteettien oletetaan olevan erilliset. Apuyksiköt ovat fonologinen silmukka ja visuospatiaalinen luonnoslehtiö. Fonologinen silmukka (eng. *phonological loop*) koostuu fonologista tietoa säilyttävästä varastosta ja toistojärjestelmästä, jolla tietoa ylläpidetään muistissa toistamalla sitä mielessä äänettömästi (kuten Atkinson-Shiffrin mallissa toistaminen). Visuospatiaalinen lehtiö (eng. *visuo-spatial working memory*) ylläpitää kuva- ja paikkatietoa. Myöhemmin kritiikin perusteella malliin on lisätty episodinen puskuri (eng. *episodic buffer*), jonka tehtävänä on yhdistää pitkäkestoisen muistin tietorakenteet työmuistin ylläpitämään tietoon, jolloin ne voidaan tallettaa muistiin kokonaisuuksina tai tapahtumina (Baddeley, 2000).



KUVIO 2 Baddeleyn ja Hitchin (1973) muistin malli.

Cowanin (1988) työmuistin teoria esittää, että pitkäkestoista muistia ja työmuistia ei voida erottaa toisistaan, vaan työmuisti on pitkäkestoisen muistin aktiivinen osa (Kuvio 3). Cowanin (1988) työmuistin teoria ei ole vaikuttanut Swellerin (1988) alkuperäiseen työhön, mutta on yksi tunnetuimmista muistiin liittyvistä teorioista. Mallissa työmuisti on kokoelma kognitiivisia prosesseja, joiden tarkoitus on ylläpitää mentaalisia representaatioita helposti mieleen palautettavassa tilassa. Toisin kuin Baddeleyn ja Hitchin (1974) mallissa, jossa muistia ylläpitävät yksiköt kommunikoivat toistensa kanssa tiedon tallentamiseksi ja ylläpitämiseksi, niin Cowanin (1988) mallissa tieto voidaan tuoda työmuistin käsiteltäväksi aktivoimalla sen mentaalinen malli pitkäkestoisessa muistissa, eikä erityyppiselle tiedolle ole erillisiä käsitteleviä yksiköitä. Cowanin (1988) mallissa informaatio voi olla joko tarkkaavaisuuden kohteena, aktiivisessa muistissa valmiina käsittelyä varten tai pitkäkestoisessa muistissa tallennettuna. Cowanin (1988) mukaan tarkkaavaisuuden kohteena olevan tiedon määrää rajoittaa muistin kapasiteetti, kun taas aktiivisessa tilassa olevan muistin määrää rajoittaa aika. Toisin kuin Millerin (1956) teoriassa, jossa työmuistiin mahtuu 5–9 asiaa kerrallaan, niin Cowanin (1988) teoriassa työmuistin koko on vain 3–4 yksikköä kerrallaan. Teorian mukaan yksikköjä voidaan kasata mieltämisyksiköiksi (eng. *chunk*), eli mentaalisiksi representaatioiksi, niputtamalla tietoa esimerkiksi kategorioimalla. Tällöin esimerkiksi kissa, koira, marsu ja kani voidaan pakata yhdeksi mentaaliseksi representaatioksi “lemmikit”. Palauttamalla mentaalinen representaatio “lemmikit” voidaan palauttaa mieleen sen sisältämä tieto lemmikeistä, sen sijaan että tieto jokaisesta lemmikistä pidettäisiin erikseen muistissa. Tällä tavoin työmuistia voidaan ikään kuin laajentaa niputtamalla tietoa yhteen sen sijaan, että pidettäisiin yksittäisiä tietoyksiköitä mielessä.



KUVIO 3 Cowanin (1988) työmuistin malli

Swellerin (1988) kognitiivisen kuormituksen teoria rakentuu erityisesti Baddeleyn ja Hitchin (1974) muistin teorian päälle. Tästä huolimatta Swellerin (1988) ajatus skeemoista, eli muistirakenteista, joiden varaan voidaan rakentaa tietoa muistuttaa enemmän myöhemmin Cowanin (1988) ehdottamia kokonaisuuksia, joilla tietoa voidaan ikään kuin kasata erilaisiksi mieltämisyksiköiksi.

2.2 Kognitiivisen kuormituksen vaikutus

Kognitiivinen kuormitus tarkoittaa sitä, että työmuistille on tarjolla enemmän tietoa kuin mitä se pystyy käsittelemään. Syynä tähän on se, että muistin kapasiteetti ei riitä käsittelemään kaikkea informaatiota, jolloin informaatiota joko pudotetaan mielestä tai sitä ei valita ollenkaan käsiteltäväksi. Swellerin ym. (2011) mukaan suurin osa kognitiivisesta kuormituksesta syntyy ulkoisen kognitiivisen kuormituksen vaikutuksesta. Informaation esitystavalla voikin olla suuri vaikutus siihen, miten paljon työmuistin resursseja tarvitaan asian käsittelyyn (Sweller ym., 2011). Swellerin (1988) alkuperäisen tutkimuksen yksi tärkeimmistä huomioista oli se, että kognitiivinen kuormitus voi vaikuttaa negatiivisesti suoriutumiseen tehtävissä. Kuormitus voi vaikuttaa tehtävistä suoriutumiseen esimerkiksi vaikuttamalla keskittymiseen ja virheisiin tehtävän tekemisen aikana, tehtävien unohteluun tai keskeytymiseen, turhautumiseen, tai informaation vastaanottoon.

Raskas kognitiivinen kuormitus johtaa tehtävän suorittamisen aikana virheeseen tai häiriintymiseen. Paasin (1992) tutkimuksessa kognitiivinen kuorma yhdistettiin huomattavasti suurempaan määrään virheitä tehtävässä, vaikka

tehtävien suoritusajoissa ei ollut merkittävää eroa. Oppimistehtävän aiheuttama kognitiivinen kuorma vaikutti tehtävistä suoriutumiseen ja oppimisen laatuun (Paas, 1992). Käytännön esimerkin kautta ongelmanratkaisun oppiminen ja pidempi aika oppimiselle vähensi virheitä tehtävissä riippumatta oppijan taitotasosta (Cooper & Sweller, 1987). Käytännön esimerkkien kautta oppiminen aiheutti perinteistä ongelmanratkaisua vähemmän kognitiivista kuormaa ja edisti oppimisen nopeutta ja laatua (Cooper & Sweller, 1987). Sweller ym. (2003) taas löysivät, että aloittelijoille suunniteltu materiaali saattaa aiheuttaa enemmän kognitiivista kuormaa kokeneilla ongelmanratkaisijoilla ja vaikuttaa negatiivisesti tehtävästä suoriutumiseen, kun taas annetun ohjeistuksen vähentäminen johti parempaan suoriutumiseen. Kokeneiden ja aloittelijoiden strategiat eroavat toisistaan ja siten ylimääräinen materiaali voi aiheuttaa kokeneelle jopa enemmän kognitiivista kuormaa (Kalyga ym., 2003). Engström ym. (2017) osoittivat, että hyvin harjoitellut tehtävät aiheuttivat vähän kognitiivista kuormaa, kun taas kognitiivinen kuorma heikensi suoriutumista tehtävissä, joita ei ollut harjoiteltu ennakkoon. Tutkimusta kognitiivisen kuorman aiheuttamasta virheiden lisääntymisestä ja tehtävässä suoriutumisesta on paljon (esim. Chandler & Sweller, 1992; Lewis & Linder, 1997; Merriënboer ym., 2002). Kognitiivisen kuormituksen alaisena myös ihmisen kyky arvioida informaation laatua voi heikentyä (esim. Thüring ym., 1995), joka voi puolestaan vaikuttaa tehtävästä suoriutumiseen, jos valittu informaation laatu on heikkoa tai informaatiota ei täysin ymmärretä.

Kognitiivisen kuormituksen vaikutus ajamiseen on yksi tutkituimmista alueista kognitiivisen kuormituksen tutkimuksessa. Ajaminen on jatkuvaa kognitiivista kontrollia vaativa tehtävä, jossa kognitio voi helposti ylikuormittua ja virheillä voi olla merkittäviä vaikutuksia. Lee ym. (2007) tutkivat kognitiivisen kuormituksen vaikutusta ajamiseen ja löysivät, että kognitiivinen kuormitus oli merkittävä tekijä kuljettajien ajovarmuudessa. Myöhemmin Lee ym. (2009) totesivat myös, että kognitiivinen kuormitus vaikeutti ajajien kykyä havaita jalankulkijat. Engström ym. (2005) osoittivat, että kognitiivinen kuormitus ajaessa saattaa vaikuttaa negatiivisesti kuljettajan havainnointiin. Kognitiivisen kuormituksen seurauksena etenkin kuljettaja keskittyi liikenteen havainnoinnin sijaan enemmän suoraan eteen päin katsomiseen (Engström ym., 2005). Engström ym. (2017) osoittivat, että kognitiivinen kuormitus vaikutti ajamiseen, mutta useimmiten silloin kun häiriötehtävät eivät olleet autonomisia, eli joko tuttuja tai erikseen harjoiteltuja. Kognitiivinen kuormitus saattoi myös parantaa suoriutumista tietyistä tehtävistä, mutta toisaalta tämä voi joissain tapauksissa tarkoittaa esimerkiksi huomion kiinnittämistä vain tiettyyn asiaan kokonaisvaltaisen havainnoinnin sijasta (Engstrom ym., 2017).

Kognitiivinen kuormitus voi vaikuttaa myös havainnoinnin ja keskittymisen tasolla. Esimerkiksi korkea kognitiivinen kuorma näyttäisi vaikuttavan ihmisen puheentunnistamiseen havainnoinnin tasolla (Mattys, 2013; Hunter &

Pisoni, 2018). Kognitiivinen kuorma voi vaikuttaa myös näkökykyyn esimerkiksi heikentämällä näkökyvyn vastinetta, jolloin kognitiivisen kuorman aiheuttamalle prosessoinnille jää enemmän resursseja (Buetti & Leib, 2018). Williams (1982) tutki kognitiivisen kuormituksen vaikutusta ihmisen näkökenttään. Tutkimuksessa Williams (1982) löysi, että kognitiivinen kuormitus vaikuttaa ihmisen funktionaaliseen näkökenttään siten, että etenkin korkean kognitiivisen kuormituksen alla näkökenttä pienenee ja näkökentän reunalla olevat visuaaliset ärsykkeet jäivät huomaamatta. Myöhemmin Williams (1988) vahvisti efektin lisäämällä myös huomion vaikutuksen yhtälöön. Vredeveldt ym. (2011) löysivät, että silmien sulkeminen helpottaa visualisointia ja vähentää kognitiivista kuormitusta, joka voi johtua muun muassa siitä, että silmien sulkeminen poistaa turhan visuaalisen informaation. Head ja Helton (2014) argumentoivat, että tarkkaavuuden ylläpitäminen hankaloituu kognitiivisen kuorman kasvaessa. Tarkkaavaisuus ajaessa heikentyy kognitiivisen kuorman takia enemmän, kuin visuaalisen kuormituksen vaikutuksesta (Young ym., 2013). Sörqvist ym. (2016) löysivät tutkimuksessaan, että kognitiivisen kuormituksen vaikutus keskittymiseen saattaa olla sekä positiivinen, että negatiivinen. Kognitiivinen kuormitus saattaa suojata häiriötekijöiltä vaimentamalla irrelevantteja havainnointitoimintoja erityisesti silloin kun häiriötekijä ja keskittymistä vaativa asia ovat eri modaliteettejä, mutta modaliteettien ollessa samankaltaiset häiriötekijöiden vaikutus voi olla suurempi kognitiivisen kuormituksen alaisena (Sörqvist ym., 2016). Näyttäisi kuitenkin siltä, että kognitiivinen kuormitus suojaa häiriötekijöiltä silloin, kun yhteen asiaan voidaan keskittyä täysin, mutta voi korostaa häiriötekijöitä silloin kun tarkkaavaisuus ei ole hyvä tai kun tarkkaavaisuus jakautuu useamman asian kesken (Sörqvist ym., 2016).

Kognitiivisen kuormituksen mittaaminen ja sen aiheuttaman efektin luominen tutkimuskäyttöön ei ole aina helppoa. Swellerin ym. (2011) mukaan kognitiiviselle kuormitukselle on epäsuoria mittareita (kuten tietokonemallinnus), subjektiivisia mittareita (kuten koettu rasitus tai vaikeustaso) ja tehokkuuden mittaaminen (kahden vastaavan metodin vertailu). Kognitiivista kuormitusta voidaan mitata myös käyttämällä apuna toista kognitiivista kykyä rasittavaa tehtävää sekä mittaamalla kognitiivisen kuormituksen aiheuttamia fysiologisia vaikutuksia (esim. sykkeen nousu) (Sweller ym., 2011). Mittaaminen voi riippua myös siitä, minkä tyyppistä (sisäinen, ulkoinen, ominainen) kognitiivista kuormitusta mitataan (Sweller ym., 2011).

Kognitiivisen kuormitus tarkoitus siis auttaa sillä hetkellä tärkeimmässä tehtävässä suoriutumisesta. Ylimääräiset aktiviteetit, kuten toinen tehtävä, toimivat vain lähinnä häiriötekijöinä ja lisäävät virheen tai häiriön mahdollisuutta. Kuormitus näyttäisi kasvavan siinä suhteessa miten samanlaisia päätehtävä ja häiritsevä tehtävä ovat. Toisaalta kognitiivisen kuormituksen alaisena kognitio saattaa myös pyrkiä suojaamaan ylimääräisiltä häiriötekijöiltä muun muassa

heikentämällä tehtävälle epäolennaisia havaintotoimintoja ja siten ohjaamalla keskittymistä.

Seuraavassa kappaleessa tutustutaan heikkonäköisyyteen, sen vakavuuteen ja vaikutuksiin. Lisäksi tutustutaan saavutettavuuteen ja saavutettavuuteen liittyviä lakeja ohjaavaan ohjeistukseen.

3 HEIKKONÄKÖISYYS JA SAAVUTETTAVUUS

Heikkonäköiset ovat monessa kontekstissa sokeitakin vähemmän tutkittu ryhmä ihmisiä. Tähän voi olla yksi syy se, että heikkonäköisyys on laaja termi ja pitää sisällään monia eriasteisesti päivittäistä toimintaa heikentäviä näkökyvyn tiloja tai toisaalta se, että monet tilastot eivät sisällä lapsia, nuoria tai työikäisiä. Monet heikkonäköiset käyttävät päivittäiseen toimimiseen apunaan avustavia teknologioita. Heikkonäköiset ovat ryhmä ihmisiä, joilla on omat tarpeensa ja tarvitsevat erilaisia suunnittelu- ja toteutusratkaisuja elämiseen (Szpiro ym., 2016). On kuitenkin selvää, että avustavien teknologioiden ja muiden heikkonäköisille luotujen ratkaisujen taso ei ole vielä aivan toivotulla tasolla etenkin, kun heikkonäköisyyden ennustetaan kasvavan tulevaisuudessa. Saavutettavuus onkin noussut lainsäädännössä esille ja uusia ohjeistuksia saavutettavuudesta päivitetään jatkuvasti.

3.1 Heikkonäköisyys

Näkövammaisten liitto (n.d.a) määrittää heikkonäköisyyden sellaiseksi näkökyvyksi, joka heikentää jokapäiväistä toimimista ja jota ei voida korjata silmälaseilla, lääkkeillä tai leikkauksella. Maailman terveysjärjestö (2019) (eng. *World Health Organization, WHO*) luokittelee heikkonäköisyyden (eng. *low vision*) näkövammaksi (eng. v*isual impairment*). Sokeus ja heikkonäköisyys muodostavat näkövammojen kirjon (Maailman terveysjärjestö, 2019, s. 10). Heikkonäköisyys on siis näkökyky, jota ei voida korjata (tyypillisillä menetelmillä), mutta jota ei luokitella sokeudeksi.

Heikkonäköisyys heikentää päivittäistä toimimista, koska siihen liittyy näön sameutta, näkökentän rajoittumista tai estymistä, sekä muita muutoksia näössä ja näkökentässä. Suomessa heikkonäköiseksi määritellään henkilö, jonka toiminnallinen näkökyky on merkittävästi heikentynyt tai jonka paremman silmän korjattu näkökyky on alle 0.3 (Näkövammaisten liitto, n.d.a). Maailman terveysjärjestö (2019, s. 11) määrittää näkövammaisuudelle eri tasoja: parem-

man silmän näöntarkkuus alle 0.3 tarkoittaa kohtalaista näkövammaa ja alle 0.1 tarkoittaa merkittävää näkövammaa. Erilaiset tekijät kuten ikä, sairaudet ja vammat voivat aiheuttaa näkövammoja tai edistää erilaisten näkövammoja aiheuttavien tilojen syntymistä. Esimerkkejä sairauksista ja vammoista ovat silmänpohjan rappeuma, harmaakaihi ja glaukooma (Maailman terveysjärjestö, 2019, s. 6—7). Heikkonäköisyys voi johtua myös synnynnäisistä vioista.

Heikkonäköisyys on tyypillisesti ikääntyvien henkilöiden ongelma ja suurin osa luvuista perustuu arvioihin, joten vaikutuksen alaisten henkilöiden määrää on vaikea arvioida tarkkaan. Maailman terveysjärjestö (2019) antaa useita syitä sille, miksi näkövammojen määrän arviointi on vaikeaa: Joissain tutkimuksissa ei oteta huomioon tietyn tyyppisiä heikkonäköisyyksiä ja näkövammoja saatetaan mitata ja määritellä eri tavoilla. Arviot keskittyvät pääasiassa yli 40- tai 50-vuotiaisiin. Maailman terveysjärjestö (2019) arvioi, että maailmassa on noin 2.2 miljardia henkilöä, joilla on vähintään lievä näkövamma ja vähintään 1 miljardi näistä olisi estettävissä tai hoidettavissa. Vuonna 2012 Pascolini ja Mariotti (2012) arvioivat, että maailmassa on lähes 285 miljoonaa näkövammaista, joista yli 85 % on heikkonäköisiä. Myöhemmin Bourne ym. (2020) arvioivat, että maailmassa noin 43 miljoonaa ihmistä on sokeita, 295 miljoonaa kärsii vakavasta näkövammasta ja 258 miljoonaa kärsii lievästä näkövammasta. Näiden lisäksi 510 miljoonaa kärsii korjaamattoman ikänäön aiheuttamasta näkövammasta. Suomessa on karkean arvion mukaan noin 55 000 näkövammaista ja heistä jopa 30 % on työikäisiä, nuoria tai lapsia (Ojamo & Tolkkinen, 2021). Suomessa näkövammaisista noin 74 % on heikkonäköisiä (Ojamo & Tolkkinen, 2021). Rekisterissä olevista vähintään 32 % on vammautunut jo syntyessään, nuorena tai työikäisenä ja lähes kolmasosa ei ole ilmoittanut vammautumisen ajankohtaa (Ojamo & Tolkkinen, 2021). Flaxman ym. (2021) arvioivat, että Amerikassa on yli 1,6 miljoonaa alle 40-vuotiasta näkövammaista (22 % kaikista näkövammaisista), joista yli 85 % on heikkonäköisiä. Luvut on saatu pääasiassa maista, joissa terveydenhuolto on kehittyneempää ja joissa on olemassa edes jonkinlainen rekisteri näkövammaisille. Maailman terveysjärjestön (2019) mukaan näön ongelmat, jotka voitaisiin korjata terveydenhuollon piirissä aikaisin ovat tyypillisempiä maissa, joissa terveydenhuolto ei ole niin kehittynyttä.

Luvuista voidaan päätellä, että merkittävä osa heikkonäköisistä on lapsia, nuoria tai työikäisiä, jolloin heikkonäköisyydellä voi olla merkittävä vaikutus suoriutumiseen elämässä, opiskelussa tai työssä (esim. Maailman terveysjärjestö, 2019). Maailman terveysjärjestön (2019) mukaan näkövammaisten työllisyys on yleisesti heikompaa kuin näkövammattomien työllisyys samoilla aloilla. Kirjassaan Welp ym. (2016) listaavat monia tutkimuksia näkövammaisuuden negatiivisista vaikutuksista Amerikassa. Welp ym. (2016) mukaan näkövammaisuus voi vaikuttaa muun muassa elämänlaatuun, liikkuvuuteen ja itsenäisyyteen, sekä erityisesti vaikuttaa mielenterveyteen, kognitioon, sosiaaliseen toimimiseen, sekä opiskeluissa ja töissä suoriutumiseen. Myös Nyman ym. (2010) sekä Bassey ja Ellison (2020) totesivat, että työikäisillä näkövammaisuus vaikuttaa

mielenterveyteen, sosiaalisuuteen ja elämänlaatuun sekä vähentää osallistumista työelämään, yhteisöön ja opiskeluun. Heikkonäköisyys vaikuttaa myös ekonomisesti. Heikkonäköisten pienempi toimeentulo, työllisyys ja koulutustaso, sekä heikkonäköisyydestä johtuvat terveydenhuollon kustannukset ja pienempi yhteiskunnallinen tuottavuus ovat yhteiskunnan kannalta ajateltuna selviä ongelmia (Welp ym., 2016; Maailman terveysjärjestö, 2019).

Maailman terveysjärjestön (2019) arvion mukaan lievä heikkonäköisyys sekä näkövammat tulevat kasvamaan tulevaisuudessa. Pääsyinä tähän ovat väestön vanheneminen ja elämäntyylin muutokset (vähenevä ulkoileminen, kaupungistuminen ja työn digitalisoituminen ja niiden sekä muiden syiden vaikutus diabeteksen yleistymiseen) (Maailman terveysjärjestö, 2019). Bourne ym. (2020) arvioivat, että vuoteen 2050 mennessä vakavien näkövammojen määrä kasvaa yli 60 % ja lievien näkövammojen määrä kasvaa lähes 40 %. Vuonna 2014 Witterborn ja Rein (2014) ennustivat, että Amerikassa näkövammojen määrä tulee yli kaksinkertaistumaan vuoteen 2050 mennessä. Holden ym. (2016) arvioivat, että vakava likinäköisyys yli viisinkertaistuu vuodesta 2000 vuoteen 2050 mennessä, kun taas lievä likinäköisyys jopa kolminkertaistuu samassa ajassa. Holdenin ym. (2016) arvion mukaan siis yli puolet maailmasta kärsisi likinäköisyydestä tai vakavasta likinäköisyydestä vuonna 2050.

3.2 Avustavat teknologiat ja teknologian käyttö heikkonäköisenä

Rajoitteelliset henkilöt, kuten heikkonäköiset, tarvitsevat usein elämisen avuksi avustavia teknologioita. Avustava teknologia (eng. a*ssistive technology*) tarkoittaa teknologista ratkaisua, joka helpottaa rajoitteellista henkilöä ymmärtämään tietoa tai suoriutumaan paremmin eri tehtävistä kuten arjen askareet tai liikkuminen. Näkövammaisten liitto (n.d.) kertoo, että näkövammaiset käyttävät teknologiaa usein avustavien teknologioiden, kuten ruudunlukuohjelmien avulla, muokkaamalla näytöllä näkyviä elementtejä tai käyttäen muita avustavia menetelmiä. W3C (2017) (World Wide Web Consortium) jakaa nämä teknologiat avustaviksi teknologioiksi, jossa teknologia on suoraan luotu avustamaan tietokoneen käyttöä, ja kutsuu muita menetelmiä adaptiivisiksi strategioiksi, joissa käytetään hyödyksi eri ohjelmien tai laitteiden ominaisuuksia.

Ruudunlukuohjelma tai tekstiä puheeksi muuttava menetelmä auttaa näkövammaista muuttamalla näytöllä olevaa tekstiä puheeksi puhesynteesin avulla tai auttamalla navigoimaan sovelluksia ja verkkosivuja (Näkövammaisten liitto, n.d.; W3C, 2017). Tekstiä puheeksi muuttavat menetelmät ovat hyvin tyypillisiä tapoja näkövammaisille käyttää tietokonetta. Ruudunlukijasta voi olla heikkonäköiselle hyötyä, jos käyttäjän silmät väsyvät tai näkökyky heikkenee pitkän lukemisen seurauksena. Ominaisuus löytyy nykyään jo monista selaimista, sovelluksista ja käyttöjärjestelmistä esiasennettuna tai vakio-

ominaisuutena. Usein heikkonäköiset käyttävät ruudunlukuohjelmaa, joka toimii useimmissa ohjelmissa ja käyttöjärjestelmissä, jolloin sitä ei tarvitse erikseen kytkeä eri ohjelmissa päälle tai konfiguroida eri sovelluksessa uudelleen (W3C, 2017). Szpiro ym. (2016) totesivat, että heikkonäköiset suosivat kuitenkin visuaalista informaatiota ja saattoivat käyttää jopa useampia avustavia teknologioita tämän saavuttamiseksi.

Laitteen tai näkymän mukauttaminen tarkoittaa sitä, että laitteella näytettäviä elementtejä, kuten tekstiä, fonttia, osoitinta tai näkymän värejä muutetaan tai piilotetaan (Näkövammaisten liitto, n.d.; W3C, 2017). Laitteen tai näytön ominaisuuksia, kuten kokoa, kirkkautta, kallistuskulmaa tai etäisyyttä muuttamalla voidaan myös helpottaa informaation hahmottamista (Näkövammaisten liitto, n.d.; W3C, 2017). Heikkonäköiselle hyvin tyypillistä on käyttää erilaisia näkymää tai ruutua suurentavia menetelmiä tai olennaista informaatiota korostavia menetelmiä, esimerkiksi fonttikoon suurentaminen lukiessa. Moreno ym. (2021) totesivat, että heikkonäköiset eivät usein käytä vain yhtä menetelmää ja useilla on erityyppiset tarpeet ja strategiat sisällön saavuttamiseen. Tutkimuksessa todettiin myös, että kun sivut ovat lähtökohtaisesti saavutettavat, niin käyttäjät suosivat ruudun suurentamiseen tarkoitettujen ohjelmien sijaan fonttikoon suurentamista. Ruudun suurentamismenetelmiä käyttävät hyötyivät muista näkymää mukauttavista menetelmistä, koska suurennetun alueen jatkuva siirtäminen saattoi tehdä verkkosivun käytön ongelmalliseksi (Moreno ym., 2021). Hallett ym. (2017) huomasivat, että ruudun suurentaminen voi aiheuttaa pahoinvointia tai epämukavuutta lukiessa, kun taas ruudun automaattinen rivittäminen koettiin käytettävämmäksi.

Avustavien teknologioiden käytössä ja saannissa on ongelmia. W3C (2017) kertoo, että vaikka ominaisuudet ja teknologiat ovat yleistymässä, niin niiden käyttö tai olemassaolo ei ole ilmiselvää ja joissain tapauksissa sovellukset tai laitteisto voivat olla epäsopivia eri menetelmien kanssa. Szpiro ym. (2016) tutkivat sitä millaisia ongelmia heikkonäköiset kohtaavat verkon käytössä ja löysivät, että erilaiset avustavat teknologiat eivät tukeneet teknologian käyttöä riittävästi. Rodrigues ym. (2022) löysivät tutkimuksessaan, että ongelmia voi olla sovellusten kehittäjien ymmärryksessä käyttäjien tarpeista ja myös eri työkalujen ja menetelmien käytettävyyden ja tehokkuuden arvioinnissa. Tutkimuksessa nostetaan esille, että yleisesti käytetyt kvantitatiiviset mitat kuten nopeus tai tarkkuus eivät itsessään ole hyviä mittareita avustavien teknologioiden käytössä, koska ne eivät ota huomioon kaikkia käytettävyyden aspekteja ja niihin keskittyminen saattaa vähentää esimerkiksi avustavia teknologioita tarvitsevien ihmisten osallistamista niiden kehityksessä. Lorenzini ja Wittich (2020) tutkivat avustavien teknologioiden käyttöä ja löysivät, että suuri määrä niistä, joilla on avustavia teknologioita eivät välttämättä käytä niitä. Vastaavasti Rodrigues ym. (2022) löysivät tutkimuksessaan, että laitteiden käyttöön liittyy myös emotionaalisia puolia, kuten esimerkiksi että avustavia teknologioita ei välttämättä

haluta käyttää, jos ne ovat rumia tai vaikuttavat negatiivisesti ulkonäköön. Bouck (2016) tutki sitä, miten vammaiset opiskelijat ovat saaneet avustavia teknologioita avukseen. Kaikkien vammaisten kesken vain 7 % oli saanut avustavia teknologioita avukseen edellisen vuoden aikana. Tutkimuksessa parhaiten avustavia teknologioita edellisen vuoden aikana avukseen saaneet olivat näkövammaiset (62,3 %) ja kuurosokeat (73,2 %). Syynä vähäiseen avustavien teknologioiden käyttöön opetuksessa voi olla tutkimuksen ja ymmärryksen puute (esim. Thomas ym., 2015). Bouck (2016) mainitsee myös, että opiskelijoilta sekä opettajilta voi puuttua ymmärrys siitä, mitä avustavat teknologiat ovat ja miten niitä voidaan hyödyntää. Wittich ym. (2015) tutkivat avustavien teknologioiden käyttöä heikkonäköisillä yli 60-vuotiailla ja totesivat, että laitteesta, tehtävän vaikeudesta ja ohjeistuksesta riippuen laitteen käyttö onnistui jossain tapauksissa vain 20 prosentilla vanhuksista. Pienelläkin ohjeistuksella oli merkittävä vaikutus laitteen käytössä onnistumiseen (Wittich ym., 2015).

Tilannetta voitaisiin helpottaa tuottamalla ratkaisuja, jotka eivät ole riippuvaisia avustavan teknologian käytöstä ja mukautuvat riittävän hyvin eri käyttäjien tarpeisiin, jolloin niiden käyttö ei olisi riippuvainen tiedon tai ymmärryksen puutteesta. Tällaiset ratkaisut voisivat esimerkiksi ottaa huomioon käyttäjien erilaiset asetukset, jolloin laite tarvitsisi konfiguroida vain kerran. Tyypillinen esimerkki tästä ovat responsiiviset verkkosivut, jotka rivittävät ja siirtävät elementtejä automaattisesti esimerkiksi laitteen koon tai fonttikoon perusteella, jolloin sivuston käyttö ei ole riippuvainen käyttäjän taidosta tai laitteesta, vaan sivusto mukautuu käyttäjän tarpeisiin automaattisesti.

3.3 Saavutettavuus

Saavutettavuus on käsitteenä jo melko hyvin ymmärretty ja tunnettu, mutta käsitteestä käytetään silti eroavia määritelmiä. Esimerkiksi jotkin tutkimukset ja tahot määrittävät saavutettavuuden kohderyhmänä olevan vammaiset, kun taas toisaalla se kattaa laajemmin kaikki ihmiset, joilla voi olla erilaisia ongelmia digitaalisten palvelujen käytössä. Ero ei kuitenkaan yleisesti aiheuta ongelmaa, koska saavutettavuuden koetaan olevan hyödyksi kaikille kohderyhmästä riippumatta (Kirkpatrick ym., 2018). Aluehallintovirasto (n.d.) määrittää, saavutettavuuden tarkoittavan sitä, että mahdollisimman moni ihminen pystyy käyttämään verkkosivuja ja mobiilisovelluksia. EU:n asettamassa saavutettavuusdirektiivissä (Direktiivi 2016/2102) ja Suomen digipalvelulaissa (Laki digitaalisten palvelujen tarjoamisesta (306/2019)) saavutettavuus kattaa verkkosivustot ja mobiilisovellukset suunnittelusta ylläpitoon ja kohderyhmänä ovat kaikki käyttäjät, mutta erityisesti vammaiset. W3C (n.d.), joka toimii internetin standardointijärjestönä, määrittelee saavutettavuuden tarkoitukseksi tuottaa digitaaliset palvelut ja verkkosisältö siten, että niitä on mahdollisimman helppo käyttää mahdollisimman monella eri laitteella mahdollisimman monessa eri

tilanteessa käyttäjän digitaalisista ja kognitiivisista taidoista tai rajoitteista riippumatta. Saavutettavuudesta on siis hyötyä rajoitteisten ihmisten lisäksi muillekin käyttäjille, kuten väliaikaisesti kätensä loukanneelle, muistisairaalle vanhukselle tai teknisesti ja kielellisesti taitamattomalle käyttäjälle kehittyvässä maassa. Tämän lisäksi termistö eroaa englanniksi ja suomeksi jonkin verran. Suomessa esteettömyys tarkoittaa sitä, että fyysisessä maailmassa, kuten rakennuksissa ja tai kaupungissa, on helppo kulkea rajoitteista huolimatta, kun taas saavutettavuudella tarkoitetaan digitaalisten palveluiden helppoa käytettävyyttä (Aluehallintovirasto, n.d.). Englanniksi esteettömyys ja saavutettavuus ovat usein sama termi *accessibility* ja termiin lisätään joskus etuliite riippuen käytöstä (esim. w*eb/physical accessibility*). Tässä tutkimuksessa saavutettavuudella tarkoitetaan sitä, että kaikki erityyppinen media, tieto ja digitaaliset palvelut ovat kaikkien saatavilla.

Kansainvälisesti saavutettavuudella on merkittävä vaikutus. Maailman terveysjärjestön (2011) arvion mukaan maailman väkiluvusta jopa 15 % (yli miljardi ihmistä) elää jonkinasteisen vammaisuuden kanssa. Aluehallintoviraston (n.d.) mukaan Suomessa elää jopa yli miljoona ihmistä, jotka tarvitsevat saavutettavia palveluita. WebAIM on voittoa tavoittelematon organisaatio, joka muun muassa tuottaa resursseja verkon saavutettavuuteen liittyen. WebAIM toteuttaa vuosittain testin, jossa testataan saavutettavuutta testaamalla miljoona verkkosivua, joilla on paljon liikennettä. Vuonna 2022 näistä sivuista 96,8 % ei saavuttanut eri standardeissa asetettua minimitasoa ja keskimäärin sivuilla oli yli 50 saavutettavuusvirhettä (WebAIM, 2022). Vuonna 2018 WebAIM toteutti kyselyn näkövammaisille ja vuonna 2021 kyselyn ruudunlukulaitteiden käyttäjille. Ruudunlukulaitteita käyttävistä 18.5 % oli sitä mieltä, että saavutettavuus on heikentynyt vuoden aikana ja 42,3 % oli sitä mieltä, että sen taso ei ole muuttunut (WebAIM, 2021a). Näkövammaisista 19,8 % koki saavutettavuuden heikentyneen ja 45.4 % ei koe, että se olisi kehittynyt suuntaan tai toiseen (WebAIM, 2018). Myös saavutettavuuden parissa toimiville henkilöille toteutetussa kyselyssä vastaajat olivat pessimistisiä saavutettavuuden kehityksen suhteen (WebAIM, 2021b). Saavutettavuus on hyödyksi myös väliaikaisesti rajoitteellisille. Esimerkiksi Aluehallintovirasto (n.d.) listaa huonon kielitaidon, heikot digitaaliset taidot ja tilapäiset haasteet, kuten väliaikainen vamma tai meluisa ympäristö, ongelmatilanteiksi, joita saavutettavuudella voidaan helpottaa. Maailman terveysjärjestön (2011) mukaan toimintaympäristöstä olevat rajoitteet voivat vaikuttaa vammaisuuteen, kuten esimerkiksi rajoitteet ja esteet, jotka estävät ihmistä toimimista itsenäisesti työssä tai kotona.

Saavutettavuuden merkitys kansainvälisesti on noussut esille lähivuosina. Suurimpia syitä saavutettavuuden kasvuun on lainsäädäntö ja tietotaidon kasvaminen. Saavutettavuusdirektiivi, Euroopan parlamentin ja neuvoston asettama direktiivi 2016/2102 julkisen sektorin elinten verkkosivustojen ja mobiilisovellusten saavutettavuudesta (Direktiivi 2016/2102) (usein myös “Saavutettavuusdirektiivi”) on yksi maailman merkittävimmistä saavutettavuuteen liittyvistä ohjeistuksista. Saavutettavuusdirektiivissä säädetään, että valtioiden

julkishallinnon verkossa ja mobiililaitteilla toimivien palveluiden tulee olla saavutettavia etenkin vammaisille. Suomessa direktiivistä on johdettu Laki digitaalisten palvelujen tarjoamisesta (306/2019) (usein myös "digipalvelulaki"), joka velvoittaa julkishallintoa, pankkeja ja vakuutusyhtiöitä ja muita yhteiskunnallisesti merkittäviä toimijoita tuottamaan digitaaliset palvelunsa saavutettavaksi. Digipalvelulaki velvoittaa toimijat arvioimaan palveluidensa saavutettavuutta ja esittelemään sen tila ja puutteet erillisessä saavutettavuusselosteessa. Tämän lisäksi palveluiden tulee tarjota käyttäjille mahdollisuus antaa palautetta saavutettavuuteen liittyen. Amerikassa on rehabilitaatiolaki, jossa on jo vuonna 1973 säädetty vammaisille kansalaisoikeudet ja oikeudet tasavertaisesta kohtelusta koulutuksen ja työllistämisen piirissä (Amerikan rehabilitaatiolaki, 1973). Lakiin on vuonna 1998 lisätty sektio 508, jossa määrätään, että julkisten palveluiden käyttämien ja tarjoamien kommunikaatio- ja informaatioteknologisten laitteiden ja palveluiden tulee olla saavutettavia (Amerikan rehabilitaatiolaki, 1973). Digipalvelulaki, saavutettavuusdirektiivi sekä sektio 508 määrittävät, että saavutettavan palvelun minimitaso on Verkkosisällön saavutettavuusohjeessa (WCAG) 2.1 määritellyt tasot A ja AA, joista kerrotaan tarkemmin myöhemmin.

Monet valtiot ja kansainväliset ja julkiset tahot ylläpitävät, tarjoavat ja tuottavat nykyään omia palveluita. Esimerkiksi Suomessa Kela työllistää yli 800 informaatioteknologia-alan osaajaa ja tarjoaa töitä niin palvelukehityksessä, -tuotannossa, IT-arkkitehtuurissa ja tietoturva-alalla ja onkin yksi Suomen suurimmista IT-alan työllistäjistä (KELA, 2022). Saavutettavuusdirektiivin tukena toimii harmonisoitu standardi (EN 301 549 V3.2.1), joka perustuu Verkkosisällön saavutettavuusohjeen versioon 2.1 ja sen tarkoituksena on yhtenäistää direktiivin vaatimukset, vaatimuksien testaaminen ja arviointi sekä vaatimuksien kohteet. EN 301 549 V3.2.1 määrittelee myös vaatimuksia, jotka eivät ole osa Verkkosisällön saavutettavuusohjetta tai eivät ole suoraan relevantteja Saavutettavuusdirektiivin kannalta. Dokumentissa määritellään saavutettavuuden kohteeksi palveluiden, verkkosivujen ja laitteiden lisäksi esimerkiksi ohjelmistot, dokumentit, laitteisto ja mediatyökalut. EN-standardit eivät ole kuitenkaan velvoittavia, mutta vahvoja suosituksia ja niihin viitataan useissa EU:n asettamissa direktiiveissä. Amerikan rehabilitaatiolaki määrittää, että vammaisilla tulee olla samat oikeudet työssä kuin rajoitteettomalla (Amerikan rehabilitaatiolaki, 1973). On siis vain ajankysymys, kun eri valtioiden julkishallinnon ja muiden merkittävien tahojen tulee tarjota saavutettavia työkaluja työntekijöilleen.

3.3.1 Verkkosisällön saavutettavuusohje (WCAG) 2.1

Verkkosisällön saavutettavuusohje (eng. *Web Content Accessibility Guidelines, WCAG*) versio 2.1 on World Wide Web Consortiumin (lyhennetään W3C) kehittämä ja ylläpitämä ohjeistus siitä, miten verkkosisältö ja sovellukset voidaan toteuttaa, kehittää ja suunnitella saavutettaviksi. Verkkosisällön saavutettavuusohje versio 2.1 (tai edellinen versio 2.0) toimii pohjana hyvin monelle saavutettavuutta edistävälle laille tai ohjeistukselle (esimerkiksi saavutettavuusdirektiivi, rehabilitaatiolain sektio 508 ja digipalvelulaki) ja sitä ylläpitää Web Ac-

cessibility Initiative (lyhennetään WAI), joka on W3C:n aloite, jonka tarkoituksena on luoda materiaalia, kuten ohjeistuksia ja ohjeita, verkon käyttäjille, kehittäjille, suunnittelijoille, päättäjille ja sisällöntuottajille. W3C on eri yritysten, yliopistojen ja organisaatioiden muodostama kansainvälinen ryhmä, joka kehittää verkon standardeja ja ohjeistuksia vapaaseen käyttöön.

Verkkosisällön saavutettavuusohjeista on aiemmin julkaistu kaksi versiota: vuonna 1999 versio 1.0 ja vuonna 2008 versio 2.0. Versio 1.0 (Chisholm ym. 1999) sisälsi 14 ohjetta, jotka sisältävät eritasoisia onnistumiskriteereitä. Ohjeet on jaettu kolmeen tasoon: A (matalin taso), AA, AAA (korkein taso). Saavuttaakseen tietyn tason verkkosivun tuli täyttää haluamansa tason vaatimukset, sekä mahdollisten alempien tasojen vaatimukset. Versiota 1.0 on kritisoitu muun muassa siitä, että ohjeistukset eivät sopineet verkkosovelluksille (Jani & Schrepp, 2005), sekä siitä että ne olivat liian keskittyneet verkkosivun tekniseen toteutukseen (Petrie & Kheir, 2007).

Verkkosisällön saavutettavuusohjeiden versio 2.0 (Caldwell ym., 2008) toteutettiin ensimmäisen version kritiikin pohjalta helpommin testattaviksi, soveltumaan useampiin teknologioihin ja soveltumaan paremmin tulevaisuuden teknologian kehitystä ja kehittymistä varten. Versio 2.0 hyväksyttiin standardiksi ISO/IEC 405000 vuonna 2012 (W3C, 2012). Uudessa versiossa ohjeita vähennettiin 12:een, jotka on jaettu neljään eri saavutettavuuden periaatteeseen. Periaatteet ovat havaittavuus (käyttäjän täytyy pystyä havaitsemaan sivulla esitetty informaatio sekä käyttöliittymän elementit), hallittavuus (navigoinnin ja käyttöliittymän elementtien tulee olla toimivia), ymmärrettävyys (käyttäjän tulee pystyä ymmärtämään sivulla esitetty informaatio, sekä ymmärtämään käyttöliittymän elementtien tarkoitus) ja lujatekoisuus (verkkosivun sisältö pitää pystyä tulkitsemaan luotettavasti eri laitteilla ja teknologioilla, joilla sen sisältöä voidaan esittää) (Caldwell ym., 2008). Ohjeet ja niiden sisältämät kriteerit määriteltiin uudelleen, jotta ne olisivat helpommin testattavissa ja ymmärrettävissä. Versiossa 2.0 ohjeistuksen noudattamisen tasot ovat samat kuin versiossa 1.0 (tasot A, AA ja AAA). Samoin kun versiossa 1.0, saavuttaakseen tietyn tason verkkosivun tulee täyttää kaikki sen tason ja alemman tason vaatimukset. Ohjeistuksessa kehotetaan tyydyttämään taso AA, koska osa verkkosisällöstä ei pysty saavuttamaan tasoa AAA. Tästä syystä myös ohjeistuksen pohjalta kehitetyt tai ohjeistukseen viittaavat lait ja ohjeistukset vaativat pääasiasiassa tyydyttämään tasot A ja AA.

Verkkosisällön saavutettavuusohje versio 2.1 (Kirkpatrick ym., 2018) julkaistiin vuonna 2018. Version 2.1 tarkoituksena on tehdä vaihtoehtoinen tapa täyttää version 2.0 vaatimukset sekä vastata aikaisemman version jälkeisiin teknologian muutoksiin ja varautua tulevaisuuden muutoksiin. Versio 2.1 laajentaa ohjeistusta sisältämään mobiililaitteiden saavutettavuuden ja ottamaan paremmin huomioon henkilöt, joilla on näkökyvyn, kielellisiä tai kognitiivisia rajoitteita tai oppimisvaikeuksia. Versiossa 2.1 tehdään vain lisäyksiä versioon 2.0 eikä alkuperäisiä version 2.0 ohjeita ole muutettu. Ohjeeseen on lisätty yksi uusi ohje, syötetavat, sekä 17 uutta kriteeriä.

Verkkosisällön saavutettavuusohjeita työstävällä työryhmällä on alla uusi versio 2.2 (Adams ym. 2022), joka täydentää version 2.1 ohjeistusta samalla tavalla kuin 2.1 täydensi versiota 2.0. Versiossa 2.2 ohjeistukseen lisätään 9 uutta kriteeriä, joiden on tarkoitus ottaa paremmin huomioon kognitiiviset rajoitteet ja oppimisvaikeudet, heikkonäköiset ja rajoitteet mobiilikäytössä. Verkkosivut, jotka täyttävät version 2.2 kriteerit täyttävät myös version 2.1 kriteerit ja siten myös version 2.0 kriteerit.

Seuraavassa kappaleessa esitellään kognitiivisen kuormituksen vaikutusta teknologian käyttöön ja digitaalisen tekstin lukemiseen.

4 KOGNITIIVINEN KUORMITUS JA DIGITAALISEN TEKSTIN LUKEMINEN

Heikkonäköisten kognitiivista kuormitusta eri tilanteissa on tutkittu melko vähän ja onkin selvää, että harvoista kognitiivista kuormitusta ja heikkonäköisiä yhdistävästä tutkimusalasta löytyy edes minkäänlaista tutkimusta. Iso osa kognitiivisen kuormituksen ja heikkonäköisyyden yhdistävä tutkimus keskittyy spatiaaliseen hahmottamiseen ja tilassa navigoimiseen (esim. Creem-Regehr ym., 2021; Giudice ym., 2008; Zhao ym., 2020), mutta tutkimusta löytyy myös esimerkiksi lukemiseen ja digitaaliseen lukemiseen liittyen (esim. Arditi & Cho, 2007; Harrison, 2004; Sass ym., 2006).

Maailman teknologisoituminen merkitsee sitä, että meillä on uusia tapoja saavuttaa tietoa ja oppia asioita. Olisikin tärkeää löytää ratkaisuja heikkonäköisille, jotka helpottavat eri asioiden ymmärtämistä ja vähentävät kognitiivista ja visuaalista kuormaa. Saavutettavuus on ensiaskel, mutta kuten kappaleessa 3.3 todettiin, niin saavutettavuus ei ole vielä riittävällä tasolla. Avustavat teknologiat ovat myös apuna, mutta kuten kappaleessa 3.2 todettiin, niin niidenkin käyttöön liittyy ongelmia. Teknologian kehittyminen tuo myös paljon uusia mahdollisuuksia, mutta esimerkiksi Buchner ym., (2022) ovat todenneet, että esimerkiksi uudet lisätyn todellisuuden ratkaisut saattavat auttamisen sijaan vain lisätä kognitiivista kuormitusta. Corn ym. (2001) huomasivat, että nuorilla oppijoilla kielen kehittyminen saattoi jäädä muita ikäisiä heikommalle tasolle muun muassa visuaalisen kuormituksen vuoksi. Optiset avustavat teknologiat paransivat lukemisnopeutta etenkin vakavasti heikkonäköisillä, kun taas lievästi heikkonäköisten lukemisnopeus kärsi (Corn ym., 2001). Visuaalinen kuormitus on yksi suurimmista tietokoneeseen liittyvistä haitoista (esim. Sheppard & Wolffson, 2018).

Heikkonäköisten tutkimuksessa pitää ottaa huomioon, että heikkonäköisyys ilmenee hyvin eri muodoissa ja on useissa tutkimuksissa osallistujilla voi olla hyvin erilaisia heikkonäköisyyden ilmentymiä tai tutkijat saattavat rajata tietynasteiset tai tietynlaiset heikkonäköisyyden ilmentymät tutkimuksesta. Olisi tärkeää erottaa, miten eri heikkonäköisyyden muodot eroavat toisistaan ja miten ne voivat vaikuttaa eri aktiviteetteihin. Jotkin tutkimukset (esim. Creem-

Regehr ym. 2021) käyttävät tutkimiseen simuloitua heikkonäköisyyttä, jossa normaalisti näkevän henkilön näkökenttää heikennetään simuloimalla eri heikkonäköisyyden muotoja. Vaikka simuloitu heikkonäköisyys heikentää näkevän ihmisen kykyä suoriutua näkökykyä vaativista tehtävistä (Boumenir ym., 2014), lukemisessa (Christen & Abegg, 2017) ja heikentää etäisyyden arviointia (Rand ym., 2019), niin simuloinnista saatu tieto ei välttämättä ole täysin luotettavaa tai aidosti heikkonäköisiä vastaavaa, koska näkevät eivät lyhyellä harjoittelulla pysty simuloimaan heikkonäköisten strategioita selviytyä heikkonäköisyyden kanssa (esim. Sass ym., 2006). Simuloidulla näkökyvyllä saatu informaatio voi kuitenkin olla arvokasta esimerkiksi sellaisissa tilanteissa, jossa heikkonäköisen silmät saattaisivat kokea liikaa rasitusta ja sillä voidaan mahdollisesti tehdä parempaa tutkimusta esimerkiksi äkillisesti vammautuneiden kyvyistä ja mukautumisesta uuteen näkökykyyn.

4.1 Kognitiivinen kuormitus teknologian käyttämisessä

Heikkonäköisillä kognitiivinen kuormitus johtaa haastavampiin ongelmiin kuin normaalinäköisen kanssa. Swellerin (1988) kognitiivisen kuormituksen teorian merkittävä löytö on se, että kognitiivinen kuormitus voi vaikuttaa negatiivisesti tehtävästä suoriutumiseen. Heikkonäköisillä heikentynyt näkökyky rajoittaa informaation saantia ja siten muun muassa hidastaa tai jopa estää arkielämässä tai työssä toimimisen (Maailman terveysjärjestö, 2019). Tämän lisäksi tavalliset kognition kuormittumisesta johtuvat virheiden lisääntyminen ja tehtävästä suoriutuminen voivat lisääntyä merkittävästi, jos useita tehtäviä tehdessä jossakin tehtävässä tarvitaan avuksi heikentynyttä näkökykyä (Zhao ym., 2020). Useissa ratkaisuissa käytetään avuksi ääntä: esim. tilassa liikkuminen äänimerkkien avulla (esim. Giudice ym., 2008) tai tietokoneen tai mobiililaitteen käyttäminen ruudunlukijan avulla, jolloin eri modaliteettien informaatio voi olla kognitiiviselle kuormitukselle helpompaa (kuten todettu kappaleessa 2.2) tai voi sopia erityyppisille henkilöille, koska kaikki eivät koe kognitiivista kuormitusta samalla tavalla. Visuaalisen ja auditorisen informaation yhdistäminen voi olla myös tehokasta oppimisen kannalta (Mousavi ym., 1995). Heikkonäköiset pitävät kuitenkin monia ratkaisuja normaalia näkökykyä heikompina ja pääasiassa suosivat visuaalista informaatiota avustavien teknologioiden tai adaptiivisten strategioiden sijaan (esim. Szpiro ym., 2016; Zhao ym., 2018).

Kuten kappaleessa 2.2 todettiin, niin kognitiivinen kuormitus johtaa muun muassa virheisiin tehtävässä ja havainnoinnin heikentymiseen, mutta myös keskittymiskyvyn paranemiseen tai heikkenemiseen tehtävästä ja tilanteesta riippuen. Heikentynyt näkökyky lisää virheen mahdollisuutta entisestään, joten kognitiivisen kuormituksen vähentäminen monessa tilanteessa on tärkeää heikkonäköiselle. Ei ole olemassa kuitenkaan tutkimusta, joka tutkisi miten kognitiivinen kuormitus vaikuttaa heikkonäköisellä havainnointikykyyn, joten ei ole varmaa tietoa siitä miten kognitiivinen kuormitus vaikuttaa havainnointiin, kun tutkittava on oppinut käyttämään erilaisia ja adaptiivisia strategioita

esimerkiksi teknologian kanssa vuorovaikuttamiseen. Vaikka on selvää, että heikkonäköisyys vaikuttaa moniin visuaalisiin toimintoihin, kuten matkan arviointiin (Rand ym., 2019) tai navigointiin (Creem-Regehr ym., 2021; Giudice ym., 2008; Zhao ym., 2020), niin on kuitenkin mahdollista, että tämä johtuu ennemmin siitä, että visuaalista informaatiota ei ole tarjolla riittävästi esimerkiksi matkan arvioimiseen (Rand ym., 2019) kuin siitä, että heikkonäköisten havainnointikyky heikentyisi kognitiivisen kuormituksen alaisena. Esimerkiksi Zhao ym. (2020) löysivät, että näkökyvyn käyttäminen navigointiin aiheutti vähemmän kognitiivista kuormaa kuin äänen avulla navigoiminen, vaikka tutkittavat suosivat ääntä visuaalisen informaation sijaan. Vredeveldt ym. (2011) tutkimuksessa silmien sulkeminen poisti turhaa visuaalista informaatiota näkökentästä ja siten paransi visualisointia ja vähensi kognitiivista kuormitusta. Voisi siis olla mahdollista, että joissain tapauksissa heikkonäköisyys voi jopa vähentää kognitiivista kuormitusta, koska heikentynyt näkökenttä voi suodattaa tehtävälle turhaa informaatiota, kuten esimerkiksi häiritsevät mainokset verkkosivulla.

Kognitiivisen kuormituksen teorian perusteella suoraan oppimiseen liittymättömät lisäaktiviteetit toimivat vain työmuistia ylikuormittavina tekijöinä. Yksi suurimmista kognitiivisen kuormituksen aiheuttajista teknologian kanssa on informaation esitystapa ja käyttöliittymän käyttö. Kognitiivisen kuormituksen teoriaan (Sweller, 1988) viitaten tässä tapahtuu kaksi toisiaan häiritsevää tapahtumaa, jolloin esimerkiksi käyttäjä joutuu ylläpitämään ja yhdistämään tietoa samalla, kun hän joutuu käyttämään käyttöliittymää saadakseen lisää tietoa. Tämän lisäksi käyttäjä voi joutua tekemään useampia kognitiota rasittavia toimia, kuten päättämään mitä linkkejä klikkaa, muistamaan eri elementtien relatiiviset suhteet ja koordinaatit, selvittämään millaiset elementit ovat klikattavia tai mikä navigointielementti vie käyttäjän parhaiten tavoitettaan kohti. DeStefano & LeFevre (2007) totesivat, että hypertekstin lukeminen vaatii kognitiivisesti enemmän resursseja kuin tavallinen lukeminen, koska materiaalin selaaminen vaatii päätöksentekoa ja visuaalista prosessointia. Dick (2018) väittääkin rullaamisen olevan vain pakollinen väline digitaalisen materiaalin katselulle. Tutkimuksessaan Hallett, Arnsdorff ym. (2015) totesivat, että normaalinäköiset käyttäjät kokivat edestakaisen vaakasuoraisen ruudun vierittämisen haittaavan informaation ymmärtämistä, joka voi johtua siitä, että käyttäjät joutuivat turvautumaan kovemmin työmuistin käyttöön, kun ruudulla pystyttiin kerrallaan näyttämään vähemmän informaatiota. Chen ja Lin (2014) taas löysivät, että kontrollin vieminen käyttäjältä lukiessa voi johtaa kognitiiviseen kuormitukseen, koska ilman kontrollia käyttäjä ei voi sovittaa tekstiä lukutyyliinsä ja -nopeuteensa. Chenin ja Linin (2014) tutkimuksessa tutkittavat korostivat erityisesti häiriöttömän ja jatkuvan lukemisen merkitystä. Youngin (2014) tutkimuksessa käyttäjät pitivät muun muassa verkossa lukemista ahdistavana, koska toisin kuin sanomalehteä lukiessa, niin verkossa lukiessa ei pysty niin helposti näkemään kokonaiskuvaa. Tutkittavat pitivät myös digitaalista lukemista vaikeampana, muun muassa sen takia, että verkossa on paljon muita häiriöitä (Young, 2014). Hallettin, Arnsdorffin ym. (2015) tutkimuksessa uskotaan, että

ruudun suurentaminen on sisällön automaattista sovittamista huonompi, koska käyttäjä ei voi ennustaa tulevaa materiaalia, eikä voi helposti tarkastella jo ohitettua materiaalia esimerkiksi täyttäessään verkkolomaketta. Wangin ym. (2012) tutkimuksessa huomattiin, että verkkosivun monimutkaisuus vaikuttaa kognitiiviseen kuormitukseen, etenkin jos verkkosivulla tehtävä on lisäksi monimutkainen. Mielenkiintoista on kuitenkin, että vaikka käyttäjät näyttäisivät häiriintyvän monimutkaisilla sivuilla enemmän, niin se ei vaikuta merkitsevästi yksinkertaisen tehtävän tekemiseen, joka voi johtua esimerkiksi siitä, että yksinkertaisessa tehtävässä tehtävän tavoitetila on hyvin selkeä ja siten havainnoitujen häiriöiden vaikutus ei ole liian suuri syrjäyttämään tavoitetilaa työmuistista (Wang ym., 2012). Chandler ja Sweller (1996) totesivatkin, että kun tietokoneohjelman käyttö vaatii paljon vuorovaikutusta (näppäimistön käyttö ja näytöltä ohjeistuksen lukeminen, jotka kuvastavat tehtävän sisäistä kuormaa), niin ohjeistuksen ymmärtämisen ulkoinen kognitiivinen kuorma kasvaa suureksi ja virheiden määrä kasvaa, kun taas vähäinen vuorovaikutus (vähäinen sisäinen kuorma) ei näyttäisi vaikuttavan merkitsevästi tehtävistä suoriutumiseen. Szpiron ym. (2016) tutkimuksessa mainitaan, että heikkonäköisyys lisää tarvetta erilaisille avustaville teknologioille, jotta sisältöä voidaan edes selata tai ymmärtää. Brachten ym., (2020) totesivat, että erilaisten virtuaalisten avustajien avulla, jotka vastaavat käyttäjän teksti- tai äänikomentoihin, voidaan laskea käyttäjän kognitiivista kuormaa.

Heikkonäköisyys voi johtaa jo lähtökohtaisesti kognitiivista kuormitusta lisäävään tilanteeseen teknologiaa käyttäessä. Graafinen käyttöliittymä on yleisin tapa käyttää tietokonetta, mobiililaitteita ja erilaisia sovelluksia ja ohjelmistoja. Graafisen käyttöliittymän eri elementtien tunnistaminen, havainnoiminen ja erottaminen toisistaan voi olla jopa mahdotonta tai hankaloitua heikentyneen näkökyvyn vuoksi. Siitä huolimatta heikkonäköiset suosivat visuaalista informaatiota (Szpiro ym., 2016). Boumernirin ym. (2014) mukaan simuloimalla heikkonäköisyyttä monet tutkittavat kokivat muotojen, esineiden ja etäisyyksien tunnistamisen vaikeaksi. Etenkin sumeaa näköä ja putkinäköä vastaavat tilat johtivat siihen, että tutkittavat joutuivat tulemaan lähemmäksi ruutua ja suoriutuivat näkökykyä vaativista tehtävistä hitaammin (Boumenir ym., 2014). Arditin ja Chon (2007) mukaan jo fonttikoko ja aakkoslaji voi vaikuttaa siihen, että heikkonäköisen on vaikeampi lukea kuin normaalinäköisen. Kuten kappaleessa 3.2 todettiin, niin heikkonäköiset käyttävät avustavia teknologioita teknologian käyttämiseen, mutta nekään eivät ole ongelmattomia. Esimerkiksi Szpiron ym. (2016) mukaan heikkonäköiset tarvitsevat jo pelkästään sisällön selaamiseen ja ymmärtämiseen avuksi erilaisia avustavia teknologioita. Avustavat teknologiat voivat kyllä helpottaa lukemista, mutta voivat kuitenkin asettaa erilaista kognitiivista kuormitusta (Szpiro ym., 2016). Heikentynyt näkökyky voi siis lisätä visuaalisen prosessoinnin ja tiedon yhdistelyn tarvetta normaalinäköiseen verrattuna ja siten teknologian käyttäminen voi aiheuttaa suuremman kognitiivisen kuorman.

Avustavat teknologiat ovat ratkaisuja muun muassa kognitiivisen kuorman vähentämiseen, mutta kaikki ratkaisut eivät helpota kaikissa tapauksissa. Esimerkiksi Langin ja Machullan (2021) tutkimuksessa lisätyn todellisuuden ratkaisujen hyödyt olivat hyvin erilaiset eri lailla heikkonäköisille. Buchner ym. (2022) taas toivat esille tutkimuksessaan, että etenkin lisätyn todellisuuden ratkaisut voivat olla kognitiivisesti vähemmän kuormittavia kuin muut vastaavat ratkaisut, mutta kehnoilla ratkaisuilla kognitiivinen kuorma voi lisääntyä merkittävästi. Buchner ym. (2022) huomauttavat kuitenkin, että ratkaisuja verrataan usein perinteiseen mediaan, joka ei välttämättä kerro koko totuutta. Lang ja Machulla (2021) tutkivat lisätyn todellisuuden hyötyjä heikkonäköiselle. Tutkimuksessa todettiin, että lisätyn todellisuuden avulla heikkonäköiset pystyivät käyttämään laitteita, joita he eivät pystyneet käyttämään ilman lisättyä todellisuutta, jonka lisäksi lisätty todellisuus koettiin hyödylliseksi kosketusnäyttöjen käytössä. Etenkin tuntemattomissa elementtien asetelmassa kognitiivinen kuorma väheni (Lang & Machulla, 2021). Hallett, Roberts ym. (2015) tutkimuksessa todettiin, että sisältö sääntelee sitä mitä avustavia teknologioita heikkonäköinen tarvitsee ja joissain tapauksessa sisältö voi olla täysin saavuttamatonta joillain avustavilla teknologioilla.

Erityisesti erilaiset visuaaliset representaatiot näyttäisivät parantavan muistia ja materiaalin tulkitsemista. Amadieu ym. (2010) löysivät tutkimuksessaan, että kun tutkittavan huomio kiinnitettiin animaatiolle tärkeisiin elementteihin, niin tutkittavien kognitiivinen kuorma oli vähäisempi kuin sellaisen ryhmän, jolle animaatio näytettiin sellaisenaan. Lisäksi huomattiin, että animaation eri elementtien suhteiden ymmärtäminen parani vain huomiota kiinnittävän animaation katsojilla, jonka lisäksi se näytti parantavan mentaalisen mallin luomista (Amadieu ym., 2010). Vastaavasti Yung ja Paas (2015) löysivät, että aritmeettisten operaatioiden oppimiseen visuaalinen representaatio vähensi kognitiivista kuormaa ja paransi materiaalin oppimista kuin ilman visuaalista representaatiota. Visuaalinen representaatio voi mahdollisesti esittää numeroita ja niiden suhteita paremmin ymmärrettävässä muodossa, jolloin oppilaat voivat helpommin poimia tehtävälle tärkeän informaation (Yung & Paas, 2015). Visuaalisella informaatiolla voitaisiin helpottaa materiaalin ymmärtämistä heikkonäköisen tapauksessa. Ei ole kuitenkaan luotettavaa määrää tutkimusta, jossa otettaisiin heikkonäköisyys riittävän hyvin huomioon

Heikkonäköisyyttä ja teknologian käyttöä yhdistävää tutkimusta on vielä hälyttävän pieni määrä. On todennäköistä, että kognitiivinen kuorma esiintyy heikkonäköisillä ja normaalinäköisillä samankaltaisena ja samoista ilmiöistä, mutta sen merkitys ja vaikutus korostuu heikkonäköisillä pienentyneen tai estyneen näkökentän vuoksi. Erityisesti teknologian käytössä suositut graafiset käyttöliittymät voivat aiheuttaa suurempaa kognitiivista kuormitusta heikkonäköiselle, koska erilaisten häiriötekijöiden vaikutus voi olla suurempi tai koska navigoimiseen tai tiedon ymmärtämiseen tarvittava informaatio on sovitettu sivustolle huonosti. Esimerkiksi Chenin ja Linin (2014) tutkimus osoittaa, että käyttäjien pitää säilyttää jatkuva kontrolli järjestelmästä kognitiivisen

kuormituksen minimoimiseksi. Kontrollin menettäminen voi olla heikkonäköiselle tavanomaisempaa, koska informaatiota navigoimiseen ei ole näkökentässä tarpeeksi (esim. Hallett, Arnsdorff ym., 2015). Tiedon tulee myös olla selvästi ja järkevästi esitetty, jotta käyttäjät voivat palata aiempaan tietoon ilman, että sen joutuu uudelleen etsimään, jolloin riski kognitiiviselle kuormalle kasvaa huomattavasti (Hallet, Arnsdorff ym., 2015).

4.2 Digitaalisen tekstin lukeminen

Digitaalisen tekstin lukeminen on nykyään jopa yleisempää kuin tulostetun tekstin lukeminen. Monissa tutkimuksissa on verrattu digitaalisen ja tulostetun tekstin lukemisen eroja muun muassa lukemisen nopeuden ja ymmärtämisen tason (Alisaari ym., 2018; Kazazoğlu, 2020), sekä esimerkiksi silmille aiheutuvan rasituksen kautta (Kang ym., 2009). Esimerkiksi Chan (2017) ehdottaa, että visuaalisen kuormituksen vähentäminen voisi auttaa käyttäjiä ja etenkin heikkonäköisiä lukemaan pidempään. Digitaaliseen lukemiseen liittyy useita ongelmia, kuten kognitiivinen kuormitus, joka voi johtua kappaleessa 4.1 mainituista tekijöistä, informaation saatavuuteen liittyvät ongelmat kuten käytettävyys sekä visuaalinen kuormitus sekä silmien väsyminen. Toisaalta digitaalinen formaatti mahdollistaa sisällön muokkaamisen, joka voi toimia myös erityisen hyvänä mahdollistajana esimerkiksi heikkonäköiselle lukijalle (esim. Legge, 2016). Esimerkiksi e-kirja-alustojen muokattavuus voisi jopa parantaa lukukokemusta ja -nopeutta (esim. Tanner, 2014). Myös esimerkiksi erilaiset muokatut lukukokemukset ja yhdistetyt luku- ja kuuntelukokemukset voivat olla digitaalisen lukemisen etuna (esim. Larson, 2015).

Perinteinen tapa tutkia digitaalisen tekstin lukemista on verrata digitaalisen ja painetun tekstin lukemisen eroja. Kuten kappaleessa 4.1 todettiin, niin teknologian käyttöön liittyy suuri kognitiivinen kuormitus, joka johtuu muun muassa verkkosivuilla olevista häiriötekijöistä (kuten mainoksista) ja teknologian kanssa vuorovaikutuksessa olemisesta, kuten käyttöliittymän käytöstä. Yksi suurimmista eroista painetun ja digitaalisen välillä onkin digitaalisen tekstin navigoimiseen tarvittavat työkalut, kuten sivun rullaaminen vaaka- ja pystysuuntaisesti. Cho ym. (2015) löysivätkin, että lukemiseen liittymättömät ylimääräiset aktiviteetit, jotka aiheuttavat kognitiivista kuormitusta vaikuttivat negatiivisesti lukemiseen.

Hillesund (2010) tutki kokeneiden lukijoiden tapoja lukea eri medioita. Tutkimuksessaan hän löysi kaksi merkittävää haastetta digitaaliselle lukemiselle: sellaisen tilan luominen, jossa tekstiin voidaan uppoutua (lukeminen ilman katkoksia) ja jossa tekstiä voidaan lukea reflektiivisesti (kuten opiskelu). Tutkimuksessa päädyttiin siihen, että kokeneet lukijat käyttävät verkkoa hyvin epäjohdonmukaisesti, eli lukevat pintapuolisesti, selailevat ja hyppivät sivulla otsikosta toiseen, kun taas novellin lukeminen paperilta on jatkuvaa. Oppimistarkoituksessa lukeminen vastasi taas enemmän paperilta lukemista: se on jollain tasolla jopa sekavaa eikä artikkeleita tai tieteellisiä papereita lueta millään taval-

la johdonmukaisesti kuten esimerkiksi kirjaa. Erona digitaalisen ja painetun tiedon välillä oli se, että tietoa voi olla helpompi yhdistellä paperilta, esimerkiksi kun paperit levitetään pöydälle, jolloin voi olla helpompi hahmottaa tietynlainen kokonaiskuva tiedosta.

Pardeden (2019) koosti tutkimustietoa digitaalisen ja painetun tekstin ymmärtämisen välillä liittyen englannin oppimiseen toisena kielenä. Pardedenin (2019) mukaan ei voida vielä sanoa selvästi, onko digitaalinen vai painettu teksti tehokkaampaa oppimisen kannalta, mutta uudemmat tutkimukset näyttäisivät saavan kirjavia tuloksia, kun taas ennen vuotta 2010 tutkimustulokset kaatuvat pääasiassa painetun tekstin puoleen. On huomattava, että tämä voi etenkin johtua siitä, että digitaalinen teksti on yleistynyt rajusti vuoden 2010 jälkeen muun muassa mobiililaitteiden yleistyttyä ja oppijat ovat olleet jo taitavia digitaalisten laitteiden käyttäjiä lapsena. Pardeden (2019) erityisesti huomauttaa, että vaikka lukemisen tehokkuuden välillä ei olisikaan suuria eroja, niin digitaalisen ja painetun tekstin lukemisessa on selviä eroja, koska digitaalinen teksti sisältää muita suoraan lukemiseen liittymättömiä, mutta lukemisen kannalta olennaisia komponentteja (kuten hyperlinkkejä). Digitaalisen ja painetun tekstin lukemisen strategioissa ei ole kuitenkaan merkittävästi eroja, mutta eroja syntyy muun muassa digitaalisen tekstin multimodaalisuuden ja tarvittavien navigointimenetelmien vuoksi (Pardeden, 2019). Alisaari ym. (2018) totesivat myös, että ylipäätään parempi lukutaito ja ymmärrys omasta lukutaidosta vaikutti positiivisesti lukemiseen sekä painetun, että digitaalisen tekstin lukemisessa. Digitaalinen ei kuitenkaan korvaa täysin painettua, koska kumpaakin mediaa suosivat olivat merkitsevästi parempia lukijoita, kuin pelkkää digitaalista suosivat (Alisaari ym., 2018).

Digitaalisen lukeminen voi siis olla hyvin tehokas väline oppimisen kannalta, eikä välttämättä tarkoita huonompaa suoriutumista kuin painettua tekstiä lukiessa. Kuitenkin heikkonäköisillä lukeminen ruudulta voi olla hyvin erilaista kuin normaalinäköisillä. Etenkin mahdollisten avustavien teknologioiden avulla lukeminen voi jopa aiheuttaa suurempaa kognitiivista kuormaa, kuin normaali lukeminen, vaikkakin ne voivat olla heikkonäköiselle ainut keino saavuttaa tietoa esimerkiksi verkossa. Esimerkiksi Cornin ym. (2002) tutkimuksessa heikkonäköisten lasten lukemistaito kehittyi nopeasti, kun heille annettiin laitteita, jotka paransivat lukemista. Corn ym. (2002) pitivät mahdollisena, että optisilla laitteilla ei ollut merkitystä varsinaisen lukemisen mekanismeihin, vaan siihen että lapset pystyivät ylipäätään lukemaan koska avustavat teknologiat helpottivat tekstin näkyvyyttä.

Digitaalinen teksti saattaa myös aiheuttaa enemmän silmien rasitusta kuin painettu teksti. Esimerkiksi Jeongin (2012) tutkimuksessa normaalinäköiset käyttäjät lukivat elektronista ja painettua tekstiä ja kokivat merkittävästi suurempaa silmien väsymistä elektronisen tekstin lukemisen jälkeen. Varsinkin heikkonäköisyyteen liittyy usein myös muita tiloja, jotka saattavat esimerkiksi

pitkäaikaisen lukemisen seurauksena heikentää näkökykyä entisestään. Monet näkökyvyn ongelmat aiheuttavat kipua silmissä tai silmien väsymistä (Maailman terveysjärjestö, 2019) ja aiheuttavat siten mahdollisesti tilan, jossa näkökyky voi olla väsymisen vuoksi tavallistakin heikompi. Sheppard ja Wolffsohn (2018) toteavat, että digitaalilaitteiden käyttämisestä johtuva silmien rasitus voi haitata jopa yli 50 %:a tietokoneen käyttäjistä. Tietokoneen käytöstä johtuva rasitus johtaa silmien kuivamiseen ja ongelmiin näön kanssa, kuten vaikeuksiin kohdistaa näköä ja taittovirheistä johtuvaan näön epätarkkuuteen. Nämä voivat erityisesti vaikuttaa heikkonäköiseen käyttäjään, jonka näkökyky on jo valmiiksi heikko. Tutkimusta heikkonäköisten visuaalisesta rasituksesta digitaalisessa lukemisessa on kuitenkin melko vähän.

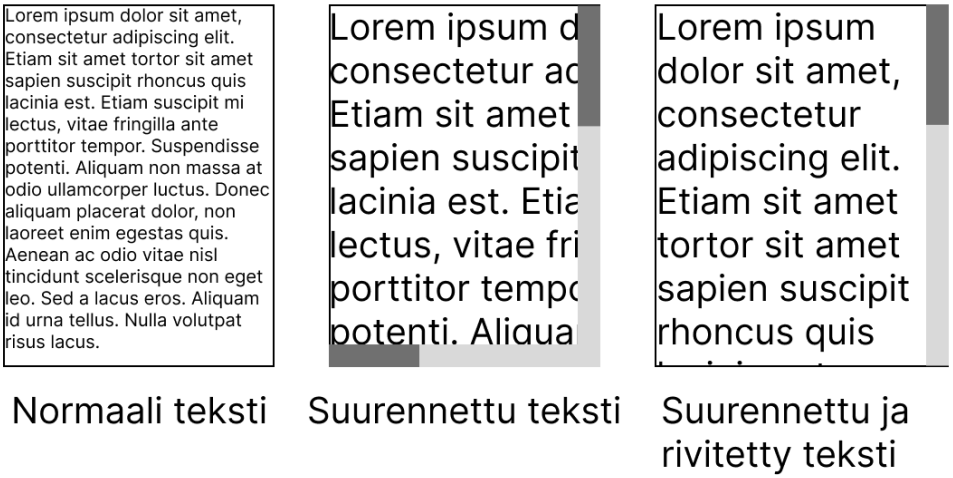
Heikkonäköiset lukevat yleisesti ottaen hitaammin, mutta yhtä tarkasti, kuin normaalinäköiset. Ero voi johtua siitä, että heikkonäköiset saattavat turvautua tekstiä lukiessa kontekstitietoon, kuten sanojen väleihin, rivien väleihin, välimerkkeihin tai muuhun tekstiä erottavaan tai selventävään informaatioon useammin kuin normaalinäköiset. Ongelmana heikkonäköisen lukemisessa on visuaalisen informaation rajallisuus. Esimerkiksi vaikeasti heikkonäköinen saattaa käyttää lukemiseen isoa suurennusta, jolloin heikkonäköinen saattaa nähdä vain parista muutamaan merkkiä kerrallaan, jolloin kokonainen sana ei välttämättä edes mahdu näkökenttään (Harrison, 2004). Sanojen pituudella ei kuitenkaan välttämättä ole suurta merkitystä lukemisen sujuvuudessa heikkonäköisillä (Sauvan ym., 2020). Joillakin heikkonäköisillä esimerkiksi pälvisokeus (näkökentässä on sokeita tai sumeita kohtia) saattaa estää samankaltaisten sanojen erottamisen toisistaan, koska eri merkkien yksityiskohdat saattavat jäädä huomaamatta (Sauvan ym., 2020). Heikkonäköiset kuitenkin pystyvät lukemaan paremmin silloin, kun välimerkit ja välilyönnit ovat tarpeeksi selvästi näkyvillä (Harrison, 2004). Sass ym. (2006) tutkivat kuinka heikkonäköisten ja normaalinäköisten lukeminen eroa. Tutkimuksessa tekstissä verrattiin lukemisnopeutta, kun tutkittavat lukivat normaalia tekstiä ja tekstiä, josta oli poistettu välilyönnit ja osassa teksteistä sanajärjestys tai kirjainjärjestys oli sekoitettu. Heikkonäköiset lukivat sekoitettuja sanoja lähes yhtä hyvin kuin normaalinäköiset, joka voisi viitata siihen, että heikkonäköisillä on enemmän kokemusta lukemisessa vähäisen kontekstuaalisen informaation avulla, mutta lukeminen vaikeutui selvästi, kun erotinmerkit oli poistettu (Sass ym., 2006). Sauvan ym. (2020) saivat mielenkiintoiseksi tulokseksi sen, että mitä yleisemmin sana esiintyy kielessä, niin sitä nopeammin heikkonäköiset pystyvät sen lukemaan. Tulos eroaa normaalinäköisten tuloksesta merkittävästi, joka voi tarkoittaa sitä, että heikkonäköiset käyttävät lukiessa apunaan sanaston tuntemustaan (Sauvan ym., 2020). Atilgan ym. (2020) taas löysivät, että heikkonäköiset pystyvät saavuttamaan siedettävän lukunopeuden jopa silloin, kun ruututilaan mahtuu vain kahdeksan merkkiä kerrallaan, kun taas vastaava luku normaalinäköiselle on 13. Heikkonäköiset saattavat siis käyttää lukemiseen hyvinkin erilaisia strategioita

kuin normaalinäköiset ja saattavat eritoten turvautua työmuistiinsa lukiessaan ja yhdistäessään tietoa. Heikkonäköiset tarvitsevat lukemisen tueksi selviä erotinmerkkejä (kuten välit, pilkut ja pisteet) sanojen ja lauseiden välillä, koska heikkonäköiset eivät pysty heikentyneen näkökyvyn seurauksena käyttämään apunaan visuaalista kontekstia, kuten esimerkiksi tietoa, jonka normaalinäköinen poimii ääreisnäön avulla.

Digitaalisen tekstin ja sisällön etuna on se, että sitä voidaan muokata esimerkiksi heikkonäköisen tarpeisiin. Legge (2016) kokosi tietoa siitä, miten digitalisoituminen vaikuttaa heikkonäköisten lukemiseen ja lukemistottumuksiin. Leggen (2016) mukaan digitaaliset laitteet mahdollistavat monia etuja heikkonäköisille, esimerkiksi mobiililaitteen etäisyyden ja katselukulman voi säätää tarpeisiin nähden ja parhaimmillaan mobiililaitetta voi jopa käyttää apuna lukemiseen. Tärkeimpinä heikkonäköisen digitaaliseen lukemiseen vaikuttavina tekijöinä pidetään fontin ja ruudun kokoa (sekä mahdollisuus ruudun tai fontin suurentamiseen), kontrastia ja sen muuttamista, ruudun kirkkautta, rivien ja sanojen välitystä ja jollain tasolla myös itse fonttia, jolla voi olla merkitystä jossain ääritilanteissa (Legge, 2016). Hallett, Roberts ym. (2015) tutkimuksessa todettiin, että etenkin sisällön esitystapa vaikuttaa siihen mitä avustavia teknologioita sisällön saavuttamiseksi tarvitaan ja vaikuttaako sisältö työskentelyyn esimerkiksi aiheuttamalla pään tai silmien särkemistä. Digitaalisen tekstin käyttökokemusta muokkaamalla voidaan lukemisen helpottamisen lisäksi päästä eroon esimerkiksi silmien liiallisesta rasituksesta, jolloin voidaan lukea pidempään ja lukeminen on miellyttävämpää.

Heikkonäköisille yleisimmät tavat helpottaa verkossa lukemista ja navigointia on suurentamalla ruudulla näkyviä elementtejä. Ruudun suurentamisessa (eng. *screen magnification*) näytön osa suurennetaan ikään kuin suurennuslasilla lukien, kun taas tekstin automaattinen rivittäminen (eng. *text reflow*) auttaa lukemista rivittämällä tekstin automaattisesti ruudulla tarjolla olevaan tilaan, jolloin esimerkiksi fonttikokoa voidaan suurentaa ilman, että teksti häviää ruututilan ulkopuolelle (Kuvio 4). Verkkosivujen yhteydessä elementtien automaattisesta muovautumisesta ruututilaan puhutaan responsiivisena suunnitteluna (eng. *responsive web design*). Hallett, Arnsdorff ym. (2015) tutkivat sitä, miten ruudun suurentaminen ja tekstin automaattinen rivittäminen vaikuttavat verkon käyttöön. Tutkimuksessa tekstin automaattinen rivittäminen laski suoritusaikoja ja nosti lukemisen tarkkuutta merkitsevästi normaalinäköisillä tutkittavilla, joten informaation esitystavalla on digitaalisessa lukemisessa merkitystä. Etenkin pidempiä tekstejä lukiessa tutkittavat valittivat silmien särystä ja uupumuksesta (Hallett, Arnsdorff ym., 2015). Myöhemmin Hallett ym. (2017) taas löysivät tutkimuksessaan, että tekstin ymmärtämisen taso oli ruudun suurentamismenetelmällä ja tekstin automaattisella rivittämisellä sama. Toisaalta tutkimuksessa käyttäjät kokivat ruudun suurentamisen aiheuttavan pahoinvointia ja kokivat sen käyttämisen uuvuttavammaksi. Myös Chan (2017) tutki sitä, miten verkossa tekstin esittäminen vaikuttaa heikkonäköisten käyttäjien visuaaliseen kuormitukseen ja löysi, että erityisesti ruudun suurentaminen aiheutti vi-

suaalista kuormitusta ja vaikutti merkittävästi lukemiseen kuluvaan aikaan, kun taas tekstin automaattinen rivittäminen ei aiheuttanut merkittävää kuormitusta. Dick (2018) totesi tekstin automaattisen rivittämisen olevan parempi, jopa hyvin suurilla fonttiko'oilla. Voidaankin todeta, että erityisesti tekstin automaattinen rivittäminen näyttäisi olevan ruudun osien suurentamista parempi vaihtoehto etenkin lukemisen kannalta.



KUVIO 4 Normaalisti rivitetty teksti, suurennettu teksti ilman rivitystä ja suurennettu teksti, joka on rivitetty saatavilla olevaan tilaan.

Arditin ja Chon (2007) tutkimuksessa todettiin, että isot kirjaimet helpottavat lukemista. Etenkin mielenkiintoinen tulos tutkimuksessa on se, että isot kirjaimet ovat luettavampia jopa puolet pienemmällä fonttikoolla, mutta etu alkaa hävitä isompia fonttikokoja lähestyessä (Arditi & Cho, 2007). Käyttämällä isoja kirjaimia voitaisiin siis ehkä mahduttaa enemmän merkkejä heikkonäköisen lukijan näkökenttään, jolloin lukeminen olisi helpompaa. Kuten aiemmassa kappaleessa todettiin, niin erityisesti vaakasuuntainen vierittäminen aiheuttaa kognitiivista kuormitusta. Voisi siis olla mahdollista, että yhdistämällä isot kirjaimet ja tekstin automaattinen rivittäminen voitaisiin helpottaa lukemista ja mahduttaa enemmän merkkejä pieneenkin ruututilaan.

Digitaalista tekstiä luetaan siis kuten painettua tekstiäkin, mutta erona on digitaalisen tekstin lukemiseen liittyvät oheistoiminnat, kuten navigointi ja tekstiin liittyvät hyperlinkit. Tekstin ymmärtämisen taso näyttäisi riippuvan lukijan omasta lukutaidosta, sekä kokemuksesta eri tekstimedioiden välillä. Lukutaito näyttäisi olevan myös merkittävä tekijä normaalinäköisten ja heikkonäköisten digitaalisessa lukemisessa. Suurimmaksi ongelmaksi digitaalisessa lukemisessa heikkonäköiselle muodostuu tekstin saavutettavuus, eli se, että tekstiä on jopa mahdotonta lukea esimerkiksi, koska eri sanojen erottaminen tai rajoitettu merkkien määrä ruudulla aiheuttaa liikaa kuormitusta. Ensisijaisesti heikkonäköistä näyttäisivät tukevan tekstin selkeys (suurikokoinen ja selkeä fontti, selkeät välit ja erotinmerkit) ja muokattavuus, sekä tekstin ja sen sisältämän tiedon jäsentäminen helposti ymmärrettävään muotoon.

Seuraavassa kappaleessa yhdistetään ymmärrystä digitaalisesta lukemisesta ja sen ja ohjelmakoodin lukemisen yhteyksistä ja eroista. Lisäksi kappaleessa kerrotaan ohjelmoinnin kognitiivisesta kuormituksesta ja sen vaikutuksista ohjelmakoodin lukemiseen.

5 OHJELMOINTIKIELET JA HEIKKONÄKÖISYYS

Ohjelmointikielet ovat formaaleja kieliä, eli kieliä, jotka ihminen on rakentanut tiettyä tarkoitusta varten. Useat ohjelmointikielet ovat tekstimuotoisia ja sisältävät usein paljon välimerkkejä ja numeroita. Yleisesti ohjelmakoodi koostuu erilaisista tekstin erotteluun tarkoitetuista visuaalisista elementeistä, kuten väleistä, sisennyksistä, muista rakenteellisuuksista (esim. listat) sekä välimerkeistä. Ohjelmointikielet eroavat toisistaan useilla tavoilla: jotkut ovat lähempänä konekieltä, eli muistuttavat hyvin pitkälti sitä, miten tietokone ymmärtäisi ohjelmointikieltä luettavan, kun taas ylemmän tason ohjelmointikielet on suunniteltu helppolukuisiksi, jolloin joitakin helppoja ohjelmia on mahdollista ymmärtää jopa ilman ohjelmointitaitoa.

Ohjelmointi aiheuttaa paljon kognitiivista kuormitusta. Ohjelmoidessa ohjelmoijan täytyy ymmärtää logiikkaa, muistaa ohjelmointikielen syntaksia, muodostaa ja ymmärtää algoritmeja, muistaa eri funktioille ja muuttujille annettuja nimityksiä, hakea vanhaa tietoa muistista ja yhdistää sitä uuteen tietoon, joka saattaa olla verkosta haettua eikä ei ole varmuutta, että informaatio on helppo hakea ja ymmärtää (esim. Brooks, 1983). Kuten kappaleessa 4.1 on todettu, niin teknologia voi aiheuttaa tarpeetonta kognitiivista kuormitusta etenkin lukemisessa. Lukemiseen verrattuna vaakasuuntaisen vierittämisen lisäksi ohjelmoinnissa tapahtuu paljon tiedon ymmärtämistä ja abstraktien rakenteiden muodostamista työmuistissa, johon digitaaliseen lukemiseen liittyvät ylimääräiset oheistoiminnot voivat vaikuttaa negatiivisesti. Sauvan ym. (2020) totesivat heikkonäköisten käyttävän sanastoa lukemisen apuna, koska etenkin kun näkökenttä on pieni, niin lukija joutuu yhdistelemään sanan ja lauseen eri osia toisiinsa. Ohjelmointikielten lukemisessa voitaisiin todennäköisesti havaita sama efekti, etenkin kun ohjelmointikielten lukeminen vaatii hyvin paljon tiedonpalasten yhdistämistä abstrakteiksi kokonaisuuksiksi ja ohjelmointikielten syntaksi on hyvin selkeästi määriteltyä.

Näkövamma saattaa vaikeuttaa ohjelmointikielten rakenteiden ymmärtämistä ja voi siten vaikuttaa kirjoitetun ohjelman ymmärtämiseen. Aiheesta ei ole kuitenkaan selvää tutkimusta, joten tässä tutkimuksena pidetään hyvänä lähtökohtana tutkia aihetta tekstin muokkaamiseen tarkoitettujen menetelmien poh-

jalta. Etenkin ohjelmointikielten ymmärtäminen saattaa poiketa suuresti tekstin ymmärtämisestä, joka voi johtua muun muassa siitä, että ohjelmointikielten ymmärtäminen vaatii abstraktien rakenteiden ymmärtämistä ja yhdistelemistä. On kuitenkin selvää, että kuten kappaleessa 5.2 todettiin, niin digitaalinen lukeminen on heikkonäköisillä jonkin verran erilaista verrattuna normaaliin näkökykyyn ja saattaa perustua nimenomaan erilaisiin lukemisen strategioihin, jotka taas muovautuvat tekstin ominaisuuksien, kuten välimerkkien ja välien pohjalta. Ohjelmointikielet perustuvat myös hyvin paljon samanlaisiin ominaisuuksiin kuten tavallinen teksti (esimerkiksi tekstin erotteluun tarkoitetut menetelmät, kuten rakenteellisuudet, välit ja välimerkit), joten tavallisen tekstin lukemiseen vaikuttavilla tekijöillä voi olla vaikutusta myös ohjelmointikielten lukemiseen heikkonäköisillä.

Ohjelmointikielten syntaksi saattaa kuitenkin aiheuttaa sekavuutta lukiessa. Sauvan ym. (2020) totesivat, että heikkonäköisyys voi vaikutta varsinkin samankaltaisten merkkien tunnistamiseen. Tämä voisi olla mahdollinen ongelma, kun otetaan huomioon ohjelmoinnissa yleisesti käytetyt, lähes samankaltaiset erottimet, kuten erilaiset sulkumerkit ja pystypalkit. Toisaalta Harrisonin (2004) mukaa erotinmerkit ovat olennaisia ja helpottavat heikkonäköisen lukemista.

Yksi parhaiksi todetuista digitaalisen lukemisen helpottamisen mekanismeista on tekstin responsiivisuus, joka helpottaa etenkin heikkonäköisen lukijan lukemistyötä (esim. Chan, 2017; Dick, 2018; Hallett, Arnsdorff ym., 2015). Tekstimuotoinen ohjelmointikieli voi myös hyötyä muista digitaalisen tekstin lukemisen helpottamisen menetelmistä. Ohjelmointikielten lukemisen ja ymmärtämisen mekanismeja ei kuitenkaan vielä ymmärretä riittävästi.

5.1 Ohjelmointikielien lukeminen ja ymmärtäminen

Ohjelmakoodin lukeminen eroaa luonnollisen kielen lukemisesta monella tavalla, mutta niissä on myös paljon yhtäläisyyttä. Suuri ja helposti huomattava ero on ohjelmakoodin lukemiseen liittyvä suurempi kognitiivinen kuormitus. Ohjelmakoodia ymmärtääkseen ohjelmoijan on yhdisteltävä ja ymmärrettävä suuri määrä tietoa ohjelmointikielen syntaksista ja tietokoneen toiminnasta lähtien. Ohjelmakoodin ymmärtämisellä tässä tutkimuksessa tarkoitetaan sitä, miten ohjelmoija pystyy erottamaan erilaiset koodin osat toisistaan, yhdistämään niiden merkityksiä ja koostamaan niistä mentaalisen mallin koko ohjelmasta ja sen toiminnasta. Lukemisella tarkoitetaan tekstimuotoisen ohjelmakoodin havainnointia näkökyvyn avulla. Kappale keskittyy vertaamaan luonnollista kieltä ja ohjelmointikieltä.

Ohjelmakoodi eroaa tekstistä merkittävästi siinä, että teksti täytyy muodostaa erittäin tarkasti tai ohjelmaa ei voida suorittaa. Kielessä saman asian voi usein ilmaista eri tavoilla, mutta ohjelmoinnissa päädytään harvoin tuottamaan

merkittävästi erilaisia ratkaisuja yhteen ongelmaan. Toinen ero tekstiin on se, että ohjelmakoodissa käytetään usein paljon välimerkkejä, kuten sulkeita, erottamaan koodin eri osia ja koodi formaalisena kielenä muistuttaa muodoltaan enemmän esimerkiksi listaa ohjeista, kuin luonnollisella kielellä kirjoitettua tekstiä. Algoritmi kirjoitettuna selkokielellä voi olla helppo lukea, mutta luettuna voi kuulostaa hieman tönköltä, koska algoritmin vaiheet on usein lueteltu ohjeen muodossa. Vaikka välimerkkejä käytetään tekstissäkin, niin niitä käytetään ohjelmakoodissa huomattavasti useammin.

Ohjelmointikielet esitetään usein tiettyjen tekstiä helpottavien rakenteiden avulla. Ohjelmoinnissa ohjelman kirjoitusasua kutsutaan ohjelman muotoiluksi (eng. *formatting*). Ohjelman kirjoitusasu perustuu usein sisennyksiin, välimerkkeihin ja tyhjän tilan käyttöön (Oman & Cook, 1990). Ohjelmointikielten kirjoittamiseen liittyvät myös isojen ja pienten kirjainten käyttö, konventiot nimeämisen suhteen, välimerkkien ja tyhjän tilan tietynlainen rivitys tai tyylittely, elementtien tyylittely samanarvoisuuden esittämiseksi, sekä kommenttien käyttäminen ohjelman ymmärtämisen helpottamiseksi (esim. Oman & Cook, 1990). Ohjelmakoodin oikea muotoilu helpottaa ohjelmakoodin lukemista ja siten myös ymmärtämistä (Roehm ym., 2012).

Useimmissa ohjelmointikielissä rakenne ja kirjoitusasu ovat pohjimmillaan merkittäviä vain luettavuuden kannalta. Esimerkiksi Java-ohjelmointikielellä kirjoitetun ohjelman voi kirjoittaa yhdelle riville, mutta sen ymmärtäminen ja luettavuus kärsivät huomattavasti. Joissain ohjelmointikielissä, kuten Python-ohjelmointikielessä, rakenne on osa kieltä, jolloin ilman oikean rakenteen hallitsemista ohjelma ei käänny oikein. Tällöin rakenteen oikeellisuus on osa kielen ymmärtämistä. Tämän lisäksi ohjelmointikielissä käytetään erilaisia välimerkkejä ja matemaattisia merkintöjä erottamaan ja toteuttamaan koodin eri osia. Esimerkiksi sulkumerkeillä osoitetaan usein, että niiden väliin jäävä osio merkitsee sulkeiden yhteyteen määritellylle funktiolle (tietynlainen joukko sääntöjä/komentoja) annettuja parametreja (alkuarvot, joiden perusteella funktio saa tuloksen), kun taas kaarisulkeiden väliin jäävä osa pitää sisällään jonkin merkitsevän osion, kuten esimerkiksi funktion sisällön**.** Matemaattiset operaatiot, kuten yhteen- ja vähennyslasku ovat myös hyvin yleisiä ohjelmoinnissa.

Ohjelmakoodin ja luonnollisen kielen välillä on myös merkittäviä yhtäläisyyksiä. Varsinkin korkeamman tason ohjelmointikielet perustuvat pitkälti luonnolliseen kieleen ja samankaltaiseen merkistöön (kuten kirjaimet), kuin luonnollinen kieli. Funktiot, muuttujat ja muut ohjelmakoodin osat nimetään usein niiden merkityksen mukaan luonnollisella kielellä. Esimerkiksi jos jotain arvoa kasvatetaan ja se ei saa kasvaa tiettyä arvoa ylemmäksi, niin usein kasvatettavaa arvoa verrataan muuttujaan, joka on nimetty suomeksi *raja* tai englanniksi *limit* tai jos funktion tarkoituksena on laskea kaksi arvoa yhteen, niin sen nimi on suomeksi *summa* tai englanniksi *sum*. Lawrien ym. (2006) tutkimuksessa todettiinkin, että etenkin kokonaisilla sanoilla nimetyt muuttujat ja funktiot ovat parempia ohjelmakoodin ymmärtämisen kannalta, kuin yksittäiset kirjaimet. Selkeissä tapauksissa lyhenteet ovat myös yhtä ymmärrettäviä kuin koko-

naiset sanat (Lawrie ym., 2006). Fakhoury ym. (2018) taas löysivät, että mitä enemmän lähdekoodissa poiketaan normaalista sanastosta, sitä enemmän se aiheuttaa kognitiivista kuormitusta ohjelmakoodia lukiessa.

Ohjelmakoodin lukeminen ja ymmärtäminen on selvästi haastavampaa kuin luonnollisen kielen lukeminen, koska ohjelmakoodin lukeminen ja ymmärtäminen vaatii paljon kognitiivista kapasiteettia. Esimerkiksi Roehm ym. (2012) totesivat tutkimuksessaan, että kokeneet ohjelmoijatkin välttelevät ohjelmakoodin ymmärtämistä lukemalla ohjelmakoodia ja turvautuvat mieluummin selitykseen ohjelman toiminnasta. Woodfieldin ym. (1981) tutkimuksessa todettiinkin, että ohjelmakoodin kommentointi vaikutti ymmärtämiseen positiivisesti. Ohjelmakoodin ylläpito ei kuitenkaan aina johda siihen, että kommentteja ylläpidettäisiin samalla tavalla (Wen ym., 2019). Ohjelmakoodia joudutaan siis usein lukemaan kommenteista huolimatta, jolloin ohjelmakoodin muotoilun laatu on merkittävä tekijä etenkin heikkonäköiselle henkilölle, jonka lukemiskyky on luonnostaan rajoittunutta.

Kognitiivinen vaativuus voi johtua esimerkiksi siitä, että ohjelmakoodin lukeminen ei ole vain lukemista, vaan ohjelmoinnin työskentely vaatii myös muita toimia, kuten ongelmanratkaisua ja debuggausta, eli ongelmakohtien löytämistä (esim. Kelly & Buckley, 2009). Ohjelmakoodin lukeminen ja ymmärtäminen koettiinkin Roehmin ym. (2012) tutkimuksessa toissijaiseksi tehtäväksi ohjelman ylläpitotehtävissä. Näyttäisi myös siltä, että mitä staattisempi kieli on, niin sitä helpompi sitä on lukea ja ymmärtää (esim. Ray ym., 2014). Staattinen tarkoittaa tässä tapauksessa sitä, että ohjelmakoodin kääntäjä vaatii tarkempaa kuvailua ja tarkkuutta ohjelman kirjoittamisessa, kuin dynaaminen kieli, jolla kirjoitettu ohjelma saattaa lähteä käyntiin, vaikka ohjelmassa olisi virheitä (mutta ohjelma ei toimi virheen takia oikein). Kansankielisemmin sanoen voisi sanoa, että mitä tarkemmin ohjelmakoodi joudutaan kuvailemaan tietokoneen ymmärrettäväksi, niin sitä ymmärrettävämpää se on myös ohjelmoijalle. Tarkempi kuvailu voi johtaa siihen, että erilaisten riippuvuuksien löytäminen on helpompaa.

Ohjelmakoodin ymmärtämistä on tutkittu, mutta sitä ei vielä ymmärretä täysin. Yleisesti ohjelman ymmärtämisen on perusteltu riippuvan tiedonhankinnasta ja kognitiivisista prosesseista (esim. Brooks, 1983; Guéhéneuc, 2009; Von Mayrhauser & Vans, 1995). Näissä malleissa ohjelmoija luo ohjelmasta mentaalisen mallin erilaisten muistitoimintojen avulla. Myös Bloomin taksonomiaa on käytetty ohjelmien ymmärtämisen tasojen tunnistamiseksi (esim. Kelly & Buckley, 2009; Xu ym., 2005). Kuitenkaan ei ole vielä täysin yhtenäistä käsitystä siitä millaiset prosessit avustavat ohjelmoijaa ymmärtämään ohjelmakoodia paremmin.

Tutkimusten perusteella näyttäisi siltä, että ohjelmakoodin ymmärtäminen eroaa luonnollisen kielen ymmärtämisestä (Binkley ym., 2013, Ivanovna ym. 2020). Ohjelmakoodin ymmärtämisen on perinteisesti ajateltu liittyvän joko luonnollisen kielen ymmärtämiseen tai matemaattisiin taitoihin kuten logiikkaan (Fedorenko ym., 2019). Ivanovna ym. (2020) tutkivat sitä, miten ohjelma-

koodin lukemiseen liittyvät tehtävät vaikuttavat neurologisella tasolla ja tutkimus on ensimmäisiä aihealueen tutkimuksia. Tutkimuksessa huomattiin aivojen aktivoituvan myös muilla alueilla, kuin pelkästään logiikkaan liittyvillä alueilla, mutta aivot eivät aktivoituneet kaikissa tapauksissa luonnollisen kielen alueilla. Ei ole kuitenkaan selvää, miksi joissain tapauksissa kielellinen alue aktivoitui ja toisissa ei, jonka lisäksi tutkimuksessa on monia rajoitteita kuten esimerkiksi tutkimuksen melko kokematon tutkimusjoukko (Ivanovna ym., 2020). Tämän lisäksi Peitekin ym. (2021) tutkimuksessa todettiin satunnaista aktivoitumista aivoalueella, joka vastaa sanojen yhdistämisestä lauseeksi. Leen ym. (2016) tutkimuksessa havaittiin suuria eroja aivoalueiden aktivoitumisessa kokemattomien ja kokeneiden ohjelmoijien välillä. Kokeneempien todettiin olevan parempia lukujen koodaamisessa, lyhytkestoisen muistin käytössä ja jälkeen päin ohjelman funktioiden mieleen palauttamisessa (Lee ym., 2016). Lisäksi Pratin ym. (2020) tutkimuksessa todettiin, että kielellisellä taidolla on merkitystä ohjelmoinnin oppimisessa päättelykyvyn ja lyhytkestoisen muistin kapasiteetin lisäksi. Mielenkiintoisesti Pratin ym. (2020) tutkimuksessa todettiin, että laskutaito oli lähes merkityksetön, joka voi kertoa siitä miten lähellä luonnollista kieltä ohjelmakoodi nykyään on. Ivanovna ym. (2020) muistuttavat myös, että koodin ymmärtäminen on välttämätön taito, mutta ei välttämättä ole päätaito monissa ohjelmointiin liittyvissä aktiviteeteissa kuten ohjelmakoodin tuottamisessa tai debuggauksessa. Tutkimustiedon perusteella ei siis voida luotettavasti sanoa paljonko luonnollisen kielen taito vaikuttaa ohjelmoinnissa.

Tämän tutkimuksen kannalta ei ole välttämätöntä ymmärtää ohjelmointikielten ominaisuuksia syvällisesti, koska tässä tutkimuksessa keskitytään etenkin ohjelmointikielten tekstimuotoiseen esitystapaan. On kuitenkin mainitsemisen arvoista, että ohjelmointikielten ominaisuuksien, kuten rakenteen, ja ohjelmointikielten lukemisen ja ymmärtämisen mekanismien ymmärtämisellä voi olla suuri merkitys sille, miten ohjelmointikieliä voitaisiin tulevaisuudessa luoda helpommin luettavaksi ja ymmärrettäväksi. Myös graafiset tai visuaaliset ohjelmointikielet, kuten Scratch, voivat olla hyödyllisiä abstraktien rakenteiden ymmärtämisessä ja ovatkin suosittuja ohjelmointikieliä lapsien opetuksessa. Chen ym. (2019) totesivatkin, että ennen nuoruusikää graafisen ohjelmointikielen opettelu vaikutti positiivisemmin oppilaiden arvosanoihin, kuin tekstipohjaisen ohjelmointikielen opettelu.

Graafisia ohjelmointikieliä on yritetty kehittää jo vuosia, mutta eivät ole laajassa käytössä. Voi olla mahdollista, että etenkin monimutkaiset rakenteet voi olla helpompi esittää tai käsittää tekstimuotoisena (esim. Boshernitsan & Dowes, 2004). Toisaalta ongelmana voi olla se, että graafisten ohjelmointikielien käyttäminen vaatii jonkinlaista ajattelutavan muutosta. Graafisella tai visuaalisella ohjelmoinnilla toteutetaankin usein ulkoasu (kuten verkkosivun tai sovelluksen elementtien asettelu) ja toiminta toteutetaan ohjelmointikielen avulla.

Vaihtoehtoisten ohjelmointikielien, kuten graafisen ohjelmoinnin tai erilaisten taktiilisten ohjelmointitapojen tutkimus on kuitenkin vielä alkuvaiheessa.

Ohjelmakoodin ymmärtäminen, kuten moni muukin kognitiivisten prosessien tutkimus, on vielä melko tutkimatonta aluetta. Emme esimerkiksi tiedä varmaksi voiko luonnollisen kielen lukemisen helpottamiseen tarkoitettuja tekniikoita käyttää tehokkaasti formaaleille kielille. Luonnollisen kielen ja ohjelmointikielien merkistö on monissa tapauksissa sama ja kielitaito näyttäisi vaikuttavan ohjelmointitaitoon, jonka lisäksi monet ohjelmointikielet käyttävät luonnollista kieltä helpottamaan ohjelmakoodin lukemista. Luonnollinen lähtökohta tekstimuotoisen ohjelmakoodin lukemisen tutkimiselle on siis verrata voiko luonnollisen kielen tekstin lukemisen helpottamiseen tarkoitettuja menetelmiä soveltaa ohjelmakoodin lukemiseen.

5.2 Kognitiivinen kuormitus ohjelmoinnissa

Ohjelmointi on kognitiivisesti raskas toiminto, koska ohjelmoijan täytyy luoda ja pitää mielessä abstrakteja, tietokoneen tai ohjelman toimintaa kuvaavia malleja, tuottaa ohjelmakoodia, ymmärtää toisten tekemää ohjelmakoodia, etsiä ja yhdistää tietoa, oppia uusia käsitteitä ja asioita, tehdä jatkuvaa ongelmanratkaisua ja löytää virheitä ohjelmakoodista. Klemola ja Rilling (2002) esittävät, että ohjelmointiin liittyvät kognitiiviset toiminnot ovat tunnistaminen, etsiminen, muistaminen, oppiminen ja luominen. Tässä tutkimuksessa keskitytään ohjelmakoodin lukemiseen ja siihen, miten heikkonäköisyyden aiheuttama rajoitus vaikuttaa kognitiiviseen kuormitukseen, mutta on tärkeä ymmärtää, että ohjelmakoodin lukemiseen liittyy muitakin kognitiivisia rasitteita pelkän lukemisen ja ymmärtämisen lisäksi.

Ohjelmakoodin lukeminen ja ymmärtäminen on tietynlainen oppimisprosessi, koska jokainen uusi ohjelma vaatii uusien yhteyksien luomista ja ohjelman ja ohjelmakoodin ymmärtämistä ja oppimista. Ohjelmoinnin oppimista pidetäänkin yhtenä vaikeimmista aiheista sen kognitiivisen vaatimuksen takia ja siksi kognitiivisen kuormituksen teoria onkin usein käytetty teoria etenkin ohjelmoinnin opetuksen tutkimuksissa (Berssanette & de Fransisco, 2021). Kognitiivisen kuormituksen teoria esittää, että oppiminen häiriintyy, kun työmuistille on tarjolla enemmän informaatiota kuin mitä se pystyy käsittelemään (Sweller, 1988). Vaikka kognitiivisen kuormituksen vaikutus ohjelmoinnin opiskeluun tunnetaan, niin ohjelmointityön aiheuttamaa kognitiivista kuormaa ei kuitenkaan ole tutkittu vielä riittävän tarkasti (esim. Gonçales ym., 2019). Tämä voi johtua siitä, että ohjelmointityössä on niin paljon eri tekijöitä, jotka vaikuttavat kognitiiviseen kuormitukseen, joten varsinaisen kuormituksen vaikutuksen mittaaminen voi olla haastavaa (esim. Ivanovna ym., 2020).

Heikkonäköisten ohjelmoijien kokema kognitiivinen kuormitus on vielä hyvin tuntematon alue. On todennäköistä, että heikkonäköisten ohjelmoijien

kognitiiviset kuormitustekijät ovat vastaavia kuin normaalinäköisillä ohjelmoijilla, mutta niiden vaikutus on suurempi, koska heikkonäköinen joutuu käyttämään enemmän muistikapasiteettia heikentyneen näkökyvyn vuoksi. Toisaalta heikkonäköisten ohjelmoijien strategioita lukea ja ymmärtää ohjelmakoodia ei tunneta, joten vaikutukset ja syyt voivat olla myös poiketa normaalinäköisen kognitiivisista kuormitustekijöistä.

Sweller ym. (1998) esittelivät kognitiivisen kuorman kolme tyyppiä: sisäinen, ulkoinen ja olennainen, jotka on paremmin esitelty Kappaleessa 2.1. Sisäinen kuorma tarkoittaa informaation monimutkaisuudesta aiheutuvaa kuormaa eli toisin sanoen sitä, miten vaikea tehtävä tai materiaali on ymmärtää. Ulkoinen kuorma taas liittyy siihen, miten informaatio on esitetty. Olennainen taas kattaa oppimista auttavat muistitoiminnot. Ulkoisen kuorman voidaan katsoa olevan niin sanotusti "jarru" oppimiselle, joka voi estää tai vaikeuttaa oppimista. Tehtävän sisäistä kuormaa ei juuri voida muuttaa ja olennainen kuorma on välttämätöntä oppimisen kannalta, joten kuormitusta voidaan merkitsevästi muuttaa vain ulkoisen kuorman tasolla.

Ohjelmoinnissa sisäinen ja olennainen kuorma, esimerkiksi tietylle ratkaisulle, voivat olla muuttuvia. Esimerkiksi monet ohjelmoinnissa käytettävät kirjastot tai viitekehykset saattavat helpottaa ohjelmointia muuttamalla monimutkaisia toimintoja yhdeksi funktioksi, mutta samalla näiden funktioiden käyttäminen vaatii enemmän osaamista, koska ymmärtääkseen funktion toiminnan ohjelmoijan tulee ymmärtää myös sen muodostavien osien merkityksen. Tämä voi hankaloittaa etenkin kokemattoman ohjelmoijan työtä, koska uusi ohjelmoija ei ole välttämättä tuttu funktion toteuttaman konseptin kanssa. Funktion osien piilottaminen aiheuttaa myös toisenlaista kuormaa. Konseptin osaaminen ja ymmärtäminen on yksi kuromaa aiheuttava tekijä, mutta toisenlainen kuorma on esimerkiksi se, mitä ohjelmoijan täytyy tehdä muuttaakseen ohjelman toimintaa. Erilaisten toimintojen kokoaminen funktioiksi voi siis tietyllä tapaa vähentää kuormaa esimerkiksi vähentämällä ohjelman kirjoittamiseen tarvittavien rivien määrää (vähemmän mahdollisuuksia virheille ja osaava ohjelmoija pystyy ymmärtämään ohjelman toiminnan nopeammin), mutta myös kasvattaa sitä esimerkiksi piilottamalla funktion toiminnan (kokemattoman ohjelmoijan täytyy ymmärtää koko funktion konsepti ymmärtääkseen ohjelman toiminnan tai ohjelmoija joutuu käyttämään enemmän resursseja funktion toiminnan muuttamiseen).

Ulkoinen kuorma ohjelmoinnissa voi aiheutua monesta tekijästä. Yksi vaikuttavista tekijöistä on abstraktio, kuten yllä mainittu tietyn konseptin kokoaminen yhden funktion alle. Abstraktiosta on myös paljon hyötyä, kuten se että voimme kirjoittaa ohjelmakoodia lähes luonnollisen kielen tasolla konekielen sijaan ja voimme tehdä ainakin joissain tapauksissa paljon helpommin luettavaa ohjelmakoodia, mutta sen haittapuolena voi olla se, että olennaiset toiminnot kuten ohjelman ymmärtäminen ja muuttaminen voivat muuttua vaikeammaksi kuin ilman abstraktiota.

Ohjelmoinnin kognitiivinen kuormitus syntyy monien eri tehtävien yhdistelystä, tiedon hankkimisesta ja yhdistämisestä, sekä abstraktien rakenteiden

ymmärtämisestä. Pennington ja Grabowski (1990) jakavat ohjelmoinnin alatehtävät neljään osaan: ongelman ymmärtäminen, suunnittelu, koodaaminen ja ylläpito. Koodaamisen eri alatehtävissä tarvitaan eri aihealueen taitoja ja ymmärrystä, kuten esimerkiksi suunnitteluvaiheessa algoritmien tuntemusta ja ohjelmointivaiheessa ohjelmoinnin konventioiden ja syntaksin tuntemusta (Pennington & Grabowski, 1990). Kellyn ja Buckleyn (2009) tutkimuksen mukaan ohjelmoijat toimivat Bloomin taksonomian viidellä tasolla: tietää, ymmärtää, soveltaa, analysoi ja arvioi. Roehm ym. (2012) taas totesivat tutkimuksessaan ohjelmakoodin ymmärtämisen olevan toissijainen tehtävä ohjelmoinnissa. Sen sijaan ohjelmoijat käyttävät erilaisia kontekstisidonnaisia strategioita ohjelman toiminnan ymmärtämiseen (Roehm ym., 2012). Sillito ym. (2006) luettelevat tutkimuksensa pohjalta 44 kysymystä, joita sovelluskehittäjät joutuvat jatkuvasti kysymään yrittäessään ymmärtää ohjelman toimintaa. Weiser (1981) taas väitti, että ohjelmakoodin ymmärtäminen on erityisesti erilaisten riippuvuuksien ymmärtämistä ja löytämistä.

Ohjelmoijan kognitiivinen kuorma vähenee kokemuksen myötä. Klemola ja Rilling (2002) esittävät, että varsinkin pidemmällä aikavälillä ohjelman ymmärtämistä helpottavat tekijät ovat sellaisia, jotka sallivat ohjelmoijan oppia ohjelman toiminnan sopivissa palasissa. Roehm ym. (2012) löysivät, että kokeneetkin ohjelmoijat välttävät itse ohjelmakoodin lukemista ja osaavat turvautua toisenlaisiin strategioihin ohjelman ymmärtämiseksi. Peitekin ym. (2021) tutkimuksessa ohjelmakoodin määrän huomattiin vaikuttavan ohjelman ymmärtämiseen ja sanaston koko taas aiheutti rasitusta työmuistille. Ohjelmoijat käyttävät kuitenkin eri keinoja ohjelmakoodin lukemisen välttämiseksi (Peitek ym., 2021) ja ohjelmoijan sanasto kehittyy kokemuksen myötä, joten kognitiivisen kuormituksen pitäisi myös laskea kokemuksen myötä. Esimerkiksi Woodfieldin ym. (1981) tutkimuksessa hyvä kommentointi helpotti ohjelman ymmärtämistä. Roehm ym. (2012) totesivat myös, että standardit ja kokemus ovat helpottavia tekijöitä ohjelman ymmärtämisessä.

Kognitiivinen kuormitus aiheutuu monesti häiriöistä. Kuten aiemmin todettu kappaleessa 4.1 ja 4.2, niin tietokoneen käytössä kognitiivista kuormaa aiheutuu muun muassa toissijaisista toiminnoista, kuten vaakasuuntaisesta rullaamisesta tai ruudulla olevista muista häiriöistä. Tämän lisäksi ohjelmoinnin aikana tapahtuva tehtävien vaihtaminen aiheuttaa häiriöitä, kun ohjelmoija joutuu vaihtamaan eri taitojen ja tiedon välillä (vrt. Pennington ja Grabowski, 1990). Raskas kognitiivinen kuormitus voi myös edistää virheiden tapahtumista, joten kognitiivinen kuormitus voi vaikuttaa myös virheisiin ohjelmoidessa. Esimerkiksi Huangin (2016) mukaan on myös tyypillistä, että ohjelmistokehittäjät saattavat usein unohtaa tärkeitä alatehtäviä, kuten ohjelman viimeistelemiseen vaadittavan toiminnon luomisen, joka voi johtua muun muassa kognitiivisen kuormituksen vaikutuksesta.

Abad ym. (2018) tutkivat häiriötekijöiden vaikutusta sovelluskehityksessä. Tutkimuksessa todettiin, että häiriön tyyppi, vuorokaudenaika ja konteksti ovat parempia mittareita häiriön haittaavuudelle kuin tehtävän kiireellisyys, vaadit-

tu taso tai tehtävän etenemistaso. Häiriöiden vaikutus todettiin merkittäväksi erityisesti ohjelmakoodin testaamiseen ja tuottamiseen liittyvissä työtehtävissä (Abad ym., 2018). Ohjelmoinnissa onkin yleistä tehdä pieniä virheitä tai kohdata esteitä, jolloin ohjelmoijan täytyy vaihtaa tehtävä ohjelman kirjoittamisesta tiedonhakemiseksi esimerkiksi lukemalla dokumentaatiota, ongelman korjaamiseen tehtyä ohjetta tai yrittämällä ymmärtää ohjelman tuottamaa virhettä esimerkiksi debuggerin avulla. Dillon ja Thompson (2016) tutkivat miten erilaiset ohjelmistokehityksessä tarvittavat työkalut vaikuttavat työn tehokkuuteen ja totesivat, että työkalujen heikko käytettävyys voi heikentää ohjelmistokehitystä. Sedanon ym. (2017) mukaan turha kognitiivinen kuormitus on yksi yhdeksästä ohjelmistokehityksen hukkatekijöistä, eli turhaan resursseja kuluttavista asioista.

Helgesson ym. (2019) tekivät kirjallisuuskatsauksen sovelluskehityksen kognitiivisista kuormitustekijöistä ja tutkivat myös sovelluskehittäjien kuormitustekijöitä haastattelututkimuksella. Tutkimuksen pohjalta tunnistettiin kolme päätekijää sovelluskehittäjien kognitiiviselle kuormitukselle: työkalut, informaatio ja työprosessi. Työkaluihin ja informaatioon liittyvät ongelmat koettiin suuremmiksi kuin työprosessiin liittyvät. Työkaluihin liittyvä kognitiivinen kuormitus johtui sisäisestä ongelmasta (esim. työkalujen huono mukautuvuus ja toimintojen päällekkäisyys sekä huono toteutus ja toimintojen puute), viivästymisistä (pienet, keskittymistä häiritsevät tauot esimerkiksi, kun ohjelma lataa, sekä pidemmät tauot jotka estävät työskentelyn täysin), sekä vuorovaikutuksen ongelmista (työkalun epäintuitiivisuus ja epäjohdonmukaisuus sekä työkalujen ominaisuuksien puute tai niiden huonon implementaatio). Informaatiosta johtuva kognitiivinen kuormitus voidaan jakaa kahteen pääkohtaan: Informaation yhtenäisyys (tietojen puute tai luotettavuus) ja informaation järjestely (tietojen tallennus- ja hakupaikka ei ole tiedossa, tiedosta ei voi saada yleiskatsausta tai tiedon rakenne on ongelmallinen). Työprosessiin liittyvät ongelmat koettiin pääasiassa tuen puutteena, joka johti turhaan resurssien kuluttamiseen.

Heikkonäköisyys voi vaikuttaa ohjelmoinnin kognitiivisen kuormituksen määrään. Kuten tässä kappaleessa on jo todettu, niin ohjelmoinnin kognitiivinen kuormitus on monien osien summa. Heikkonäköisellä heikentynyt tai rajallinen näkökenttä voi aiheuttaa lisäongelmia esimerkiksi ohjelman kokonaiskuvan hahmottamisen tai ohjelmakoodin lukemisen kanssa. Esimerkiksi Peitek ym. (2021) totesivat tutkimuksessaan ohjelmakoodin määrän vaikuttavan siihen, miten helposti ohjelmakoodi on ymmärrettävissä. Tällöin esimerkiksi heikentynyt näkökenttä voi aiheuttaa lisää kuormitusta, kun tietoa joudutaan pitämään työmuistissa enemmän kuin tavallisella näkökyvyllä.

Heikkonäköisyys voi vaikuttaa myös virheiden määrään ja siihen miten paljon virheet vaikuttavat ohjelmointiin. Kappaleessa 4.2 todettiin, että heikkonäköisten lukeminen voi olla hankalampaa kuin lukeminen normaalilla näkökyvyllä, koska esimerkiksi sanojen tai kirjaimien erottaminen voi joissain ta-

pauksissa olla hankalaa. Tällöin virheiden huomaaminen kirjoittaessa tai niiden löytäminen ohjelmakoodia lukemalla voi hankaloitua merkittävästi.

5.3 Ohjelmointikielten lukemisen helpottaminen

Ohjelmoijat lukevat ohjelmakoodia päivittäin joskus jopa enemmän kuin kirjoittavat. Aiemmin tässä tutkimuksessa on todettu, että digitaalinen lukeminen aiheuttaa kognitiivista kuormaa sekä, että ohjelmoinnissa kognitiivinen kuormitus on suuri jo työn vaativuuden vuoksi. Roehm ym. (2012) ja Peitek ym. (2021) toteavat, että ohjelmakoodin lukeminen on raskasta, jonka vuoksi ohjelmoijat yrittävät ensin muita strategioita ohjelman ymmärtämiseksi lukemisen sijaan sen kognitiivisen rasituksen vuoksi. Ohjelmakoodin lukemista ei kuitenkaan voi välttää kaikissa tapauksissa ja se voidaan joutua tekemään muualla kuin ohjelmointiin liittyvässä ympäristössä, kuten esimerkiksi verkkosivulla tai mobiililaitteella ja joskus ohjelmakoodia voidaan joutua lukemaan esimerkiksi paperilta tai tekstitiedostosta, jota ei ole muotoiltu ohjelmointikielen konventioiden mukaan. Ohjelmakoodin lukemista helpottamalla voitaisiin mahdollisesti vähentää sen aiheuttamaa kognitiivista kuormaa ja auttaa ohjelmoijia työskentelemään tehokkaammin. Erityisesti heikkonäköiselle ohjelmoijalle lukeminen on heikentyneen näkökyvyn vuoksi vaikeampaa, joten erilaiset helpottamisen menetelmät voivat olla kriittisiä, jotta heikkonäköinen pystyy edes lukemaan ohjelmakoodia.

Aiemmin Kappaleessa 5.1 todettiin, että ohjelmakoodi ja luonnollisella kielellä kirjoitettu teksti perustuvat hyvin paljon samankaltaisiin rakenteisiin, sanastoon ja merkistöön. Varsinkin korkeamman tason ohjelmointikielet myös lähestyvät jo luonnollisen kielen kirjoittamista. Lisäksi ohjelmointikielet asetellaan usein helpommin havaittaviin ja ymmärrettäviin osioihin eri merkkien ja rakenteiden avulla. Tyhjän tilan käyttö, rivitys, ja muut rakenteet auttavat ohjelmoijaa lukemaan ja ymmärtämään ohjelman toimintaa paremmin (Lawrie ym., 2006; Oman & Cook, 1990).

Tässä tutkimuksessa esitetyn tiedon pohjalta ei voida sanoa varmaksi miten paljon tavallisen tekstin ja ohjelmakoodin lukeminen eroavat toisistaan. Lähtökohdaksi otetaan oletus, että tekstin lukemisen helpottamisen menetelmät toimivat myös ohjelmakoodin lukemisen helpottamiseen. Ohjelmakoodi on tekstipohjaista, se perustuu samoihin rakenteisiin ja merkistöön kuin tavallinen teksti ja osaa tekstin lukemisen helpottamiseen käytettyjä menetelmiä käytetään ohjelmakoodin muokkaukseen tarkoitetuissa työkaluissa kuten Visual Studio Code, Sublime Text ja Atom.

Myers ym. (2016) ehdottavat, niin uusien ohjelmointia helpottavien menetelmien pitäisi aloittaa siitä, mitä ohjelmoijat tekevät jo entisestään. Heikkonäköiset ohjelmoijat joutuvat muovaamaan lukukokemustaan (esimerkiksi ruudun tai tekstin suurentamisen menetelmät) monilla eri tavoilla helpottaakseen lukemista. Erityisesti tekstin automaattinen rivittäminen on heikkonäköi-

sille olennaista tekstin lukemisessa (Chan, 2017; Hallett, Arnsdorff ym., 2015), mutta monissa ohjelmointityökaluissa ei ole tällaista ominaisuutta tai ominaisuus ei välttämättä tue lukemista täysin, koska ne eivät välttämättä ota huomioon sitä, että ohjelmakoodin rivityksen muuttaminen voi tehdä ohjelmakoodista vaikeammin luettavaa.

5.3.1 Digitaalisen tekstin sovittaminen ruutuun

Digitaalisen tekstin etuna on se, että sitä voidaan muokata eri tarpeisiin ja tarkoituksiin. Muokattavuus onkin suuri etu heikkonäköiselle lukijalle, koska erilaiset heikkonäköisyyden muodot vaikuttavat lukemiseen eri tavoin ja käyttäjillä voi olla eri mieltymyksiä monien asioiden suhteen. Muokattavuus sallii heikkonäköisen esimerkiksi lisätä tekstin kokoa, muuttaa sen väriä tai lisätä tai vähentää tekstin tai kirjaimien välissä olevaa tilaa.

Tekstin eri ominaisuudet vaikuttavat lukemiseen eri tavalla. Heikkonäköiset tarvitsevat tietoa siitä, milloin edellinen sana loppuu ja seuraava sana alkaa (Sass ym., 2006) ja ylipäätään hyötyvät siitä, että sanojen ja kappaleiden alut ja loput ovat selkeät (Harrison, 2004). Heikkonäköiset hyötyvät isoista kirjaimista, mutta myös isosta fontista lukiessaan tekstiä, jossa on pieniä sekä isoja kirjaimia (Arditi & Cho, 2007). Tekstin kokoa voidaan muokata jopa hyvin suureksi ilman, että luettavuus kärsii (Dick, 2018). Näkökenttään pitäisi siis saada mahdollisimman paljon merkkejä siten, että niiden suhde toisiinsa on edelleen selkeä.

Legge (2016) ehdottaa, että digitaalisen lukemisen helpottamiseksi tekstin on oltava muokattavissa heikkonäköisen tarpeisiin. Tekstin muokkausmenetelmiä on kuitenkin paljon. Leggen (2016) mukaan heikkonäköisille tärkeitä tekstin ja laitteen muokattavia ominaisuuksia ovat fontin ja ruudun koko, korkea kontrasti tekstin ja taustan välillä, kirkas näyttö ja kontrastin kääntäminen, rivivälit ja sanavälit, sekä vähäisesti myös fontti itsessään. Tekstin automaattinen rivittäminen on heikkonäköisille tehokas menetelmä mahduttaa tekstiä ruutuun ilman, että luettavuus kärsii (Chan, 2017; Hallett, Arnsdorff ym., 2015) ja hyvin toteutettuna se sopii erinomaisesti myös suuria fonttikokoja käyttäville lukijoille.

Tekstin rivityksen muokkaaminen on saatavilla yleisimpiin ohjelmoinnissa käytettäviin ohjelmakoodin muokkaustyökaluihin. Sen on todettu olevan tehokas visuaalisen ja kognitiivisen kuorman vähentämisessä (esim. Hallett ym, 2017). Kuitenkaan nykyisissä editoreissa käytettävä rivitys ei ota kontekstia hyvin huomioon, koska usein rivitys katkaisee ohjelmakoodin tietyn merkkimäärän kohdalta ilman, että se ottaa huomioon ohjelmakoodin luettavuutta. Esimerkiksi Kuvassa 5 40:n merkin kohdalta katkaistu ohjelmakoodi katkeaa jokaisen rivin kohdalla eri pisteestä, joka vaikeuttaa ohjelmakoodin tulkitsemista. Esimerkissä etenkin ohjelmakoodin eri osiot toistavat samaa kaavaa, jonka huomaa helposti normaalisti rivitettyä ohjelmakoodia lukiessa, mutta toistuvuutta ei ole aivan yhtä helppo huomata merkkimäärän kohdalta katkaistusta ohjelmakoodista. Ohjelmakoodin järkevämpi rivittäminen voisi helpottaa oh-

jelmakoodille olennaisien osien mahduttamista rajalliselle ruututilalle, kun kyseessä on heikkonäköinen lukija.

KUVIO 5 Ohjelmakoodi normaalisti esitettynä 40 merkin tilassa ja sovitettuna katkaistulla menetelmällä 40 merkin tilaan.

5.3.2 Värit

Värien käyttö ei ole välttämättä paras lähtökohta ratkaisun kehittämiselle. Ratkaisun tulisi olla riippumaton väreistä, jotta se olisi mahdollisimman yleiskäyttöinen. Monesti näkövammaisilla henkilöillä on heikentyneen näön tarkkuuden tai esteellisyyden lisäksi myös jonkinasteinen värisokeus (Kirkpatrick, 2018). Tämän vuoksi esimerkiksi värisokeat, eri kontrasteja käyttävät ja mustavalkoista väriskaalaa käyttävät eivät välttämättä pysty tulkitsemaan eri värien merkityksiä (Kirkpatrick, 2018). Leggen (2016) mukaan heikkonäköiset tarvitsevat lukemisen tueksi muun muassa kontrastien mukauttamista. Tämän vuoksi on yleistä ja suositeltavaa käyttää värin lisäksi muitakin visuaalisia korostuskeinoja ja ikoneita digitaalisessa sisällössä. Ohjelmointikielissä rakenne ja erilaiset välimerkit auttavat tulkitsemaan ohjelmakoodin hierarkiaa ja erottamaan eri osia, joten värit toimivat tehosteena tulkitsemiselle. Tämän lisäksi esimerkiksi aloittelevat ohjelmoijat selvisivät ohjelmakoodin lukemisesta yhtä hyvin, kun ohjelmakoodin osat olivat korostettu väreillä ja silloin kun ohjelmakoodi esitettiin mustavalkoisena (Hannebauer ym., 2018).

5.3.3 Kokemus

Kokemuksella on myös selvä vaikutus ohjelmakoodin lukemiseen ja ymmärtämiseen. On selvää, että kokemuksen myötä tietotaito kasvaa ja näin ollen ohjelmakoodin ymmärtäminen on myös helpompaa. Kokemuksen vaikutuksesta ohjelmakoodin ymmärtämiseen on tehty paljon tutkimusta. Yleisesti on huomattu, että kokeneemmat ohjelmoijat pystyvät käyttämään parempia strategioita ongelmien ratkaisuun, joka voidaan nähdä esimerkiksi suurempana kognition aktivoitumisena (Lee ym., 2016), kun taas aloittelevat ohjelmoijat saattavat käyttää samankaltaisien strategioiden alkuasteita, mutta eivät pysty käyttämään strategioita oikein tai yhtä tehokkaasti (Murphy ym., 2008; Weiser & Shertz, 1983; Wiedenbeck, Fix & Scholtz, 1993). Roehmin ym. (2012) tutkimuksessa kokemuksen määrä vaikutti ohjelmakoodin ymmärtämiseen.

Ohjelmointikokemuksen lisäksi myös tottumuksella on vaikutus. Heikkonäköisetkin ohjelmoijat tottuvat tiettyyn ohjelmointikielten kirjoitus- ja esitystapaan, joten on todennäköistä, että tottumus tietynlaiseen ohjelmakoodin lukemiseen voi vaikuttaa ohjelmakoodin lukemiseen ja ymmärtämiseen.

5.3.4 Ohjelmakoodin laatu

Tärkeä huomioida, että ohjelmakoodin laatu vaikuttaa hyvin paljon ohjelmointikielen ymmärtämiseen ja lukemiseen. Hyvin kommentoitua ohjelmakoodia ei välttämättä tarvitse edes lukea ymmärtääkseen ohjelman toiminnan. Toisaalta hyvin rakennettua ja nimettyä, mutta kommentoimatonta ohjelmakoodia pystyy seuraamaan myös helposti. On kuitenkin huomioitava, että maailmassa on paljon ohjelmakoodia, jota ei ole kommentoitu tai tehty järkevästi. On myös mahdollista, että vanhassa ohjelmakoodissa on paljon asioita, joita uudemmat ohjelmoijat eivät ole joutuneet kohtaamaan, jolloin ohjelmakoodin lukemisen tärkeys korostuu. Tässä tutkimuksessa ratkaisua pyritään löytämään nimenomaan siihen tilanteeseen, kun ohjelmakoodin tarkempi tarkastelu on tarpeen sen toiminnan selvittämiseksi.

5.3.5 Tekstin korostaminen

Tekstin korostamiseen käytetään monia tekniikoita, kuten kursivointi, lihavointi, sekä tekstin tai taustan värit ja kontrasti ja fontin koko. Leggen (2016) mukaan erityisesti taustan ja tekstin kontrasti on tärkeää heikkonäköiselle lukemisessa. Kuten jo Kappaleessa 5.3.2 todettiin, niin värit eivät sovellu ainakaan kaikille heikkonäköisille, joten kontrastien luominen tässä tapauksessa pitää ajatella vain valkoisen ja mustan värin suhteena toisiinsa. Kontrasti on tässä tapauksessa jo olemassa oleva ratkaisu (vaalea ohjelmakoodi mustalla taustalla tai musta ohjelmateksti valkealla taustalla). Esimerkiksi ohjelmarivien hierarkian korostaminen ei kuitenkaan ole hyvä ratkaisu, koska

kontrastien suhde toisiinsa ei välttämättä ole selkeä, jos kontrasti eri tasoisten rivien välillä on liian pieni tai korostavia tasoja ei voida tehdä tarpeeksi, jos ero eri rivien välillä on liian suuri. Tämän lisäksi useimmissa ohjelmakoodin ja tekstin muokkaamiseen tarkoitetuissa työkaluissa esimerkiksi valittu tai valitut rivit osoitetaan jo kontrastin avulla.

Kursivointi ja lihavointi ovat tyypillisiä tekniikkoja korostaa tärkeitä osia tekstistä. Nämä ovat kuitenkin erittäin rajallinen keino korostaa ohjelmakoodin eri osia. Leggen (2016) tutkimuksessa esitellään eri fontteja ja niiden näkyvyyttä erilaisilla näöntarkkuuksilla. Leggen (2016) mukaan näyttäisi siltä, että fontit, joissa on selkeät linjat ja riittävästi tilaa kirjaimen muodostavien linjojen ympärillä ja välissä. Lihavointi ja kursivointi muuttavat fontin ominaisuuksia, kuten linjojen paksuutta, joka saattaa vääristää helposti luettavan fontin vaikeasti luettavaksi.

5.3.6 Tekstin osien piilottaminen

Digitaalisen tekstin lukemista helpotetaan usein piilottamalla lisätietoa esimerkiksi erilaisten klikkaamalla aukeavien elementtien alle. Tämä on tyypillisesti helppo keino jaotella teksti helposti luettaviin ja ymmärrettäviin osuuksiin tai piilottaa pääasialle epäolennaista tietoa. Ohjelmakoodin ymmärtäminen on kuitenkin vaikeaa ja joskus pienimmätkin yksityiskohdat saattavat olla olennaisia ohjelman toiminnan ymmärtämiseksi. Ei ole kuitenkaan olemassa tutkimusta, jossa olisi tutkittu miten tarkasti ohjelmakoodia pitää esittää tai mitkä osat ovat olennaisia ohjelmien ymmärtämiseksi. Tämä voi johtua siitä, että erilaiset ohjelmat voidaan abstrahoida eri tasolle ja eri tavoilla esimerkiksi ohjelman monimutkaisuuden tai tärkeiden yksityiskohtien vuoksi, joten tietynlaisen säännöstön luominen tähän tarkoitukseen on vaikeaa.

Seuraavassa kappaleessa esitellään tutkimuksen hypoteesit ja tutkimuskysymykset sekä tutkimuksen menetelmä ja analysointimenetelmät.

6 TUTKIMUSMENETELMÄ

Tässä osiossa esitellään tutkimuksen menetelmä, sekä olennaisia asioita tutkimuksesta. Tutkimuksella on kaksi tavoitetta: kokeilla tekstin automaattisen rivityksen menetelmää ohjelmakoodille heikkonäköisen ohjelmoijan lukemisen helpottamiseksi, sekä tehdä katsaus ongelmiin, joita heikkonäköiset ohjelmoijat kohtaavat ohjelmointiin liittyen. Tutkimuksessa on empiirinen osio, jossa testataan automaattista tekstin rivittämistä, tavallisesti esitettyä ohjelmakoodia, sekä merkkimäärän kohdalta katkaistua esitystapaa. Empiirisen osion pohjana toimii käsitys siitä, että heikkonäköisyys johtaa kognitiiviseen kuormitukseen digitaalista tekstiä lukiessa (esim. Hallett, Arnsdorff ym., 2015) ja siten voi johtaa kognitiivisesti raskaan ohjelmoinnin aikana tarpeettoman suureen kognitiiviseen kuormitukseen (esim. Helgesson ym., 2019). Ohjelmoinnin kognitiivisen kuormituksen tekijöiden kirjo on kuitenkin kattava ja etenkin heikkonäköisyyttä ja ohjelmointia yhdistävää tutkimus on lähes olematonta. Tutkimuksessa kartoitetaan ja kategorioidaan ongelmia, joita heikkonäköiset kohtaavat ohjelmointiin liittyen, jotta tulevaisuuden tutkimukselle löytyy lähtökohta.

6.1 Hypoteesit ja tutkimuskysymykset

Tällä hetkellä ohjelmointikielten esitystapa perustuu konventioihin, jotka ovat pääasiassa tehty helpottamaan normaalisti näkevän henkilön työskentelyä. Tutkimuksen lähtökohtana on kartoittaa sitä, voiko heikkonäköisten ohjelmoijien ohjelmakoodin lukemista helpottaa erilaisin tekstin lukemisen helpottamiseen tarkoitetuin menetelmin. Lisäksi vielä lähes tuntemattomasta aihealueesta halutaan kartoittaa lisää tietoa. Tämän perusteella tutkimuskysymyksiksi ovat valikoituneet:

A. Voidaanko ohjelmakoodia esittää heikkonäköiselle henkilölle siten, että ohjelmakoodin lukeminen ja ymmärtäminen on helpompaa?
B. Mikä on tehokas tapa esittää ohjelmakoodia heikkonäköiselle henkilölle siten, että ohjelmakoodin lukeminen ja ymmärtäminen on helpompaa?
    i. Ovatko nykyiset ohjelmakoodin kirjoittamiseen ja lukemiseen tarkoitettujen sovellusten heikkonäköisille tarkoitetut avustavat teknologiat tehokkaita?
    ii. Onko tekstin uudelleenrivitys toimiva metodi ohjelmakoodin ymmärrettävyyden parantamiseen, kun lukija on heikkonäköinen?

iii. Ovatko muut yleiset tekstin luettavuuden helpottamiseen käytetyt tekniikat tehokkaita ohjelmakoodin ymmärrettävyyden parantamiseen, kun lukija on heikkonäköinen?

Vastaamalla näihin tutkimuskysymyksiin voidaan saada kattava kuva siitä, miten ohjelmakoodin lukemista voitaisiin helpottaa, kun lukija on heikkonäköinen. Kaikkiin tutkimuksen kysymyksiin ei voida vastata empiirisen tiedon perusteella eikä aihealueesta ole vielä tarpeeksi kattavaa tutkimusta, jotta kaikkiin kysymyksiin voitaisiin vastata tutkimustiedon perusteella. Tutkimuksen rajauksen vuoksi suuri osa tekstin lukemisen helpottamisen menetelmistä jää vielä tutkimatta. Empiirisen tiedon perusteella voidaan kuitenkin todeta, että onko kognitiivinen kuormitus pienempää, kun ohjelmakoodille käytetään tekstin rivittämisen menetelmiä. Tutkimuksen hypoteesi on siis:

> H1: Ohjelmakoodin lukemisessa normaalisti rivitetyllä tai merkkimäärän kohdalta rivitetyllä esitystavalla esiintyy enemmän virheitä, suuremmat suoritusajat ja pienempi koettu kuorma (SUS-pisteet), kuin automaattisen tekstin rivittämisen menetelmällä esitetyllä ohjelmakoodilla.

6.2 Tutkimusmenetelmät

Tässä tutkimuksessa selvitettiin voiko tekstin uudelleen rivittäminen helpottaa heikkonäköisen ohjelmoijan kognitiivista kuormaa ohjelmakoodin lukemisessa. Heikkonäköiset ohjelmoijat kokevat suurempaa kognitiivista kuormaa lukiessaan ohjelmakoodia muun muassa vaakasuuntaisen vierittämisen vuoksi (esim. Hallett, Arnsdorff ym., 2015) ja toisaalta tekstin automaattinen rivittämisen on todettu olevan erittäin tehokas keino (Chan, 2017; Hallett, Arnsdorff ym., 2015) jopa suurilla fontti ko'oilla (Dick, 2018), joten tekstin automaattisella rivittämisellä voisi olla positiivinen vaikutus siihen, miten heikkonäköinen lukee ja ymmärtää ohjelmakoodia. Aiheen uutuuden vuoksi empiirisen osion lisäksi toteutettiin laadullinen osio, jonka avulla toivottiin löytyvän lisää tietoa aihealueesta ja kysymyksiä jatkotutkimusta varten. Tutkimukseen osallistuminen toteutettiin tutkimusta varten toteutetussa työkalussa, jossa tutkittavat vastasivat ensin laadulliseen osioon, jossa kerättiin tietoa tutkittavasta sekä kysyttiin avoimia kysymyksiä. Laadullisen osion jälkeen tutkittavat suorittivat ohjelmoinnin lukemiseen liittyviä tehtäviä.

Tekstin uudelleen rivittämisen on luvussa 4.2 todettu olevan erittäin tehokas lukemisen helpottamisen menetelmä heikkonäköisille ja sen vaikutusta on kohtalaisen helppo tutkia empiirisesti: lukemiseen kuluva aika ja virheiden määrä. Empiirinen osio toteutettiin tutkimusta varten tehdyn työkalun avulla, jossa tutkittavat suorittivat ohjelmakoodin lukemiseen liittyviä tehtäviä. Tämän lisäksi apuna käytettiin SUS-mittaria (Brooken, 1996), jolla voidaan arvioida järjestelmän (tässä tapauksessa ohjelmakoodin esitystavan) aiheuttamaa kuor-

maa ja käytettävyyttä. Tutkittavat täyttivät SUS-kyselyn jokaisen ohjelmakoodin lukemisen tehtävän jälkeen. Luvussa 4.1 esitellyt kognitiivisen kuormituksen vaikutukset voidaan kaikki mitata empiirisin menetelmin, jolloin empiirinen menetelmä sopii hyvin tähän tutkimukseen. Useissa luvussa 4.2 esitellyissä tutkimuksissa mitataan joko tehtävään kulunutta aikaa, virheiden määrää tai molempia.

Tutkimuksessa koeasetelma oli within-subjects. Osallistujat vastasivat samoihin kysymyksiin ja tehtäviin, mutta ohjelmakoodin esitystavat ja esitysjärjestys oli satunnaistettu. Erityyppisten heikkonäköisyyksien ja kokemuksien vuoksi suoritusaikoja ei voinut suoraan verrata tutkittavien välillä. Tehtävissä tutkittavat lukivat ohjelmakoodin ja vastasivat sen perusteella millaisen tulosteen kyseinen ohjelma antaa suorituksen päättyessä. Tehtäviin luodut ohjelmakoodit olivat aloittelijalle sopivia ja vaativat, että koko ohjelma luetaan läpi, jotta oikea syöte voidaan antaa. Toteutettu verkkotyökalu mittasi vastaamiseen käytetyn ajan sekä tallensi tutkittavan antaman vastauksen. Tehtävän tekemisen jälkeen tutkittavalle esitettiin SUS-lomake, jossa tutkittavaa pyydettiin arvioimaan tehtävässä käytetyn ohjelmakoodin esitystapaa. Tutkimuksen empiirinen tutkimustieto analysoitiin käyttäen SPSS-ohjelmaa.

Tutkimuksen laadullisen osion pohjalta voitiin kartuttaa tietoa heikkonäköisten ohjelmoijien kokemista ongelmista ohjelmointiin liittyen. Kyselyn tarkoituksena on nimenomaan luoda kokonaiskuvaa tilanteesta, josta ei tiedetä vielä paljoa. Tutkimuksen laadullinen osio toteutettiin kyselylomakkeen avulla, joka täytettiin ohjelmointitehtävien tekemisen yhteydessä.

Tutkimuksen aikana tutkittavien ruutua nauhoitettiin Zoom-videopalvelun välityksellä. Ennen tutkimuksen aloittamista tutkittavia vaadittiin olemaan häiriöttömässä tilassa, jotta tehtävien tekeminen ei häiriytyisi ja näin vaikuttaisi suoritusaikoihin. Nauhoituksen tarkoitus oli varmentaa, että tutkittavat eivät häiriintyneet tehtävien tekemisen aikana eikä sen aikana tallennettu tutkittavan kasvoja, nimeä tai ääntä.

6.3 Proseduuri

Tutkimus toteutettiin kyselytutkimuksena, johon liittyi kyselyn lisäksi tehtävien suorittamista. Ennen kyselytutkimuksen avaamista tutkimusta varten rakennettiin tutkimustyökalu (Liite 1). Tutkimuksen perusjoukon ja metodin vuoksi oli vaikea löytää työkalua, joka täyttäisi saavutettavuuden vaatimuksen sekä tutkimuksen tehtäväosion vaatimukset. Työkalu rakennettiin yksinkertaiseksi ja siten, että se taipuisi mahdollisimman hyvin heikkonäköisen tutkittavan omiin asetuksiin ja eri tapoihin käyttää sovelluksia. Työkalua testattiin useamman henkilön toimesta ja kaikki todennäköiset saavutettavuusongelmat kitkettiin työkalusta pois sekä palautteen, että automaattisen saavutettavuutta testaavan ohjelmiston avulla. Samat henkilöt testasivat myös tehtävien vaativuutta ja so-

veltuvuutta, sekä varmistivat, että tehtävissä ei ole virheitä. Testauksen perusteella todettiin, että kuusi tehtävää (kaksi tehtävää esitystapaa kohden) on sopiva, jotta tehtävien kuormitus ei kasva liian suureksi.

Koehenkilöitä hankittiin sähköpostilistojen ja muiden soveltuvien kanavien kautta. Sähköpostilistoille ja erilaisille kanaville lähetettiin kutsukirje tutkimukseen ja linkki ajanvaraukseen. Suppean perusjoukon vuoksi tutkittavia jouduttiin lähestymään myös organisaatioiden kautta ja jopa henkilökohtaisella viestillä. Tutkittavien tuli varata aika tutkimukseen, koska tutkittavia vaadittiin jakamaan näyttöä tehtäväosion aikana. Kutsukirjeessä ilmaistiin vaatimukset tutkittaville, sekä yleisesti tietosuoja-asioista. Tutkittaville lähetettiin linkki tutkimustyökaluun vasta varatun ajan alussa, kun yhteys tutkittavaan oli luotu Zoom-videopalvelun kautta. Video- tai puheyhteys ei ollut tutkimukseen vaatimus, mutta tutkittavan ruutua piti pystyä nauhoittamaan tehtäväosion aikana. Tutkittaville kerrottiin, että ruutua ei nauhoiteta ennen kuin tehtäväosio alkaa ja nauhoitus loppuu tehtäväosion lopuksi. Tutkittavia muistutettiin nauhoituksen alkamisesta ja loppumisesta myös suullisesti ennen nauhoituksen aloittamista ja lopettamista.

Ennen kyselyn aloittamista tutkittavat siirtyivät linkin kautta tutkimusta varten rakennettuun verkkotyökaluun, jossa kysely ja tehtävät toteutettiin. Kyselylomakkeen ensimmäisellä sivulla tutkittavia muistutettiin kyselyn tarkoituksesta, ohjeistettiin kyselyn etenemisestä, informoitiin datan tallentamisesta ja käsittelystä, sekä pyydettiin varmistamaan, että tutkittava ymmärtää tutkimukseen liittyvät ehdot. Tutkittaville annettiin myös mahdollisuus kysyä huolenaiheista tai tutkimukseen liittyvistä asioista ennen tutkimuksen aloittamista.

Ensimmäisessä osiossa tutkittavilta kysyttiin perustietoja, kuten ikä, ohjelmointikokemus ja tietoa käytettävästä laitteistosta, jonka lisäksi työkalu tallensi olennaista laitteistotietoa, kuten ruudun koon ja selaimessa käytetyn suurennuksen tason. Tämän lisäksi laadullista tutkimusta varten kysyttiin vapaavalintaista tietoa näkövammasta ja ohjelmointiin liittyvistä ongelmista. Kyselylomakkeessa tutkittavia muistutettiin, että näkövammaisuuteen liittyvät kysymykset ovat valinnaisia, mutta niihin vastaaminen antaa arvokasta tietoa.

Toinen osio koostuu tehtäväosiosta. Tehtäväosion alussa tutkittavia ohjeistettiin siitä, mitä tehtävissä tehdään sekä siitä, että heidän tulisi keskittyä tehtävän aikana. Tutkittavia muistutettiin myös siitä, että heidän tulisi olla häiriöttömässä paikassa ja tarvittaessa he voivat vielä etsiä häiriöttömän paikan. Tässä vaiheessa tutkittavia muistutettiin, että tallennus aloitetaan, kun he ovat valmiit jatkamaan.

Tehtävissä tutkittavien tuli lukea esitetty ohjelmakoodi ja vastata lukemansa perusteella mitä ohjelma tulostaa, jos se ajettaisiin. Ohjelmakoodien tutkittavat saivat täysin saman ohjeistuksen, mutta ohjelmakoodin esitystapaa ja ohjelmakoodien esitysjärjestystä vaihdettiin satunnaisesti. Tutkittaville esitettiin siis satunnaisia ohjelmakoodien ja esitystapojen yhdistelmiä. Tehtäviä oli kuusi ja esitystapoja kolme, joten jokainen tutkittava kohtasi jokaisen esitystavan kaksi kertaa. Ohjelmakoodien esitysruutu oli rajattu 40:een merkkiin, jotta ohjelmakoodi saatiin pidettyä yksinkertaisena ja silti rivitysefekti saatiin aikaiseksi.

Jokaisen tehtävän jälkeen tutkittavalle esitettiin SUS-lomake, joka liittyi juuri esitettyyn ohjelmakoodin esitystapaan. Lomakkeessa muistutettiin, että arviointi tulisi kohdentua ohjelmakoodin esitystapaan eikä tehtävään itseensä. Lomake koostui SUS-menetelmään kuuluvista kymmenestä Likert-asteikollisesta kysymyksestä, joista järjestelmä laski automaattisesti pisteet valmiiksi. Tutkimustyökalu tallensi käyttäjien vastauksen tehtävään, tehtävän suoritusajan ja SUS-lomakkeen vastaukset.

Tehtävät alkoivat näkymällä, jossa muistutettiin mitä tutkittavan kuuluu tehdä tehtävässä. Näkymässä näytettiin ruutu, johon ohjelmakoodi tulee luettavaksi ja nappi "Start task", jota painamalla tehtävä ja ajan laskeminen aloitettiin. Napin painamisen jälkeen ruudulle ilmestyi ohjelmakoodi ja laatikko, johon vastaus voitiin kirjoittaa, sekä nappi "End task", jota painamalla tehtävä ja ajan laskeminen voitiin lopettaa, kun vastaus oli annettu. Tehtävää ei voinut lopettaa ilman, että vastaus oli annettu ja jos sitä yritettiin, niin tutkittavalle näytettiin selvä ilmoitus varoitusmerkillä sekä tekstillä vastauksen puuttumisesta. Tutkija oli myös valmistautunut antamaan saman ohjeistuksen suullisesti, mikäli tutkittavan oli heikkonäköisyyden vuoksi vaikea lukea tai havaita ilmoitusta.

Ohjelmakoodin lukemisen tehtävät oli suunnattu aloittelijatasoiseksi ja niiden luomisessa keskityttiin erityisesti siihen, että ohjelmakoodia joudutaan lukemaan (Liite 2). Ohjelmakoodi toteutettiin pseudokoodina, jotta se ei sulkisi tietyn kielien osaajia pois tutkimuksesta. Samankaltaisten ja tarpeeksi samanlaisten tehtävien luominen oli tärkeää, jotta tehtävien suoritusajat tai vaikeusaste ei vaikutu liikaa tehtävien erilaisuudesta. Apuna tehtävien luomiseen sovellettiin Luxton-Reilleyn ja Petersenin (2017) sekä Kaston ja Whalleyn (2013) ehdotuksia siitä, millaiset ohjelmakoodit sopivat aloitteleville ohjelmoijille. Ehdotuksien perusteella luotiin ohjelmakoodeja, jotka poikkeavat toisistaan merkkimäärien perusteella, mutta ovat syntaksiltaan yksinkertaisia ja joiden kompleksisuus on vastaava. Tehtävissä vaihdettiin yksinkertaisia operaatioita ja niiden järjestystä, jotta tehtävien samankaltaisuus ei vaikuttaisi suoriutumiseen. Tehtäviä testattiin useammalla henkilöllä, ilman eri esitystapojen vaikutusta, ja todettiin eri tehtävien lukemiseen kuluvan ajan olevan lähes samankaltainen.

Tehtäväosion jälkeen tutkittaville kerrottiin, että ruudun nauhoittaminen loppuu siihen ja tutkittava voi sulkea ruudun jakamisen. Tutkittaville esitettiin tekstilaatikko, johon he voivat vapaaehtoisesti jättää vapaata palautetta liittyen tutkimukseen. Kun palaute oli annettu, järjestelmä tallensi tiedot tietokantaan.

6.4 Esitystavat

Tutkimuksessa käytettiin ohjelmakoodille kolmea eri esitystapaa (Kuvio 6). Ensimmäinen esitystapa on ohjelmointikielille tyypillinen, normaali esitystapa,

jossa ohjelmakoodin yksi osio sijoitetaan yhdelle riville, jonka jälkeen rivinvaihdot ja sisennykset määrittävät seuraavan luettavan rivin. Toisena käytettiin merkkimäärän kohdalta katkaisua esitystapaa, joka on tyypillinen ominaisuus ja pikanäppäintoiminto monissa ohjelmointiympäristöissä. Merkkimäärän kohdalta katkaistussa esitystavassa määritellään merkkimäärä, jonka jälkeen tuleva teksti siirretään seuraavalle riville. Toimintoja on hieman "viisaampia" ja "tyhmempiä". Etenkin "tyhmä" toiminto katkaisee koko rivin vain suoraan merkkimäärän kohdalta välittämättä sen sisällöstä. Osa "viisaammista" toiminnoista saattaa katkaista rivin jonkin merkitsevän sisällön välimerkin kohdalta. Kolmanneksi esitystavaksi luotiin viisaampi, automaattista rivitystä imitoiva esitystapa. Esitystavassa ohjelmakoodi rivitetään merkitsevien osuuksien kohdalta ja merkitsevät osuudet yritetään mahduttaa samalle riville, jotta lukeminen olisi helpompaa.

KUVIO 6 Esitystavat esiteltynä verkkosivun näkymässä

6.5 Koehenkilöt

Tutkimukseen osallistui kahdeksan heikkonäköistä ohjelmoijaa. Kohderyhmän pienuuden vuoksi osallistujien tavoittaminen osoittautui erittäin haasteelliseksi. Osallistujilta vaadittiin jotain heikkonäköisyyden esiintymismuotoa, jonka vuoksi osallistuja joutuu käyttämään apunaan ruudun suurennusmenetelmää (200—400 % suurennus). Kaikilla osallistujilla oli vähintään aloittelijan taso ohjelmoinnissa. Tutkimus toteutettiin englannin kielellä, joten osallistujilta vaadit-

tiin myös englannin kielen taitoa. Osallistujista vain yksi puhui englantia äidinkielenään, vaikka kolme osallistujaa oli maista, joissa ainakin yksi virallinen kieli on englanti. Tutkimukseen osallistujista kaksi oli naisia ja kuusi miehiä. Tutkittavien ikä vaihteli välillä 20—52. Osallistujien ohjelmointikokemus vaihteli välillä 1—31 vuotta.

Koehenkilöiden heikkonäköisyyksien esiintymien kirjo oli kattava. Kysymyksen vapaaehtoisuudesta huolimatta jokainen tutkittava halusi kuvailla heikkonäköisyyden esiintymää. Heikkonäköisyyksiä esitellään myöhemmin Kappaleessa 7.2. Tutkittavista kaikki käyttivät ruudun suurennusta väliltä 200—400 %.

6.6 Analysointimenetelmät

Tutkimus on määrällisesti ja laadullisesti analysoitu. Määrällinen analyysi on toteutettu SPSS-ohjelmistolla, jossa eri esitystapojen suoritusaikoja, vastauksen oikeellisuutta ja SUS-pisteitä verrattiin toisiinsa. Tutkimuksen hypoteesin H1 hyväksymiseksi, pitäisi suoritusajoissa, vastauksen oikeellisuudessa ja SUS-pisteissä olla tilastollisesti merkitsevä ero. Laadullisessa analyysissa haluttiin kartoittaa heikkonäköisten ohjelmoijien kohtaamia ongelmia ohjelmointiin liittyen.

Tutkimuksessa asetelma oli within-subjects. Tutkimuksen pienen osallistujamäärän vuoksi (N = 8) lukuja vertailtiin Kruskal-Wallisin epäparametrisellä testillä sekä kategorisia muuttujia Fisherin tarkalla testillä. Epäparametrisen Kruskal-Wallisin testin valinta johtui pääasiassa otoskoon pienuudesta, mutta myös osa datasta ei ollut normaalisti jakautunut. Testillä voitiin verrata eri esitystavan vaikutusta suoritusaikoihin ja SUS-pisteisiin. Fisherin tarkka testi valikoitiin, koska tutkimuksessa kategoriset muuttujat olivat pieniä, jonka lisäksi useammassa taulukon kohdassa tulos oli alle 5, joka vaikuttaisi Khiin neliön -testiin. Fisherin tarkalla testillä verrattiin esitystavan vaikutusta vastauksen oikeellisuuteen.

Tutkimuksessa kuorman mittaamiseen käytettiin SUS-mittaria (eng. *system usability scale*). SUS-mittari on Brooken (1996) kehittämä mittari järjestelmän käytettävyydelle. SUS-mittarin etuna on se, että se on nopea toteuttaa eikä vaadi käyttäjältä liikaa kognitiivista kuormaa, joten sen sopivuus muuten kognitiivista kykyä vaativiin tehtäviin on hyvä (Brooke, 2013). SUS- mittarilla mitataan erityisesti käyttökokemusta, eikä käyttäjän mielipidettä järjestelmästä. SUS-mittarissa käyttäjältä kysytään kymmenen Likert-asteikollista kysymystä (asteikko 1—5, kts. Liite 3), joita painotetaan eri tavoin ja tulokseksi saadaan luku väliltä 0—100. SUS-mittarin eri osien tulokset eivät ole itsessään merkitsevät, joten niitä ei voida käyttää vain tietyn osan tutkimiseen. Bangor ym. (2009) ja Sauro (2011) ovat todistaneet, että SUS-mittaria voidaan käyttää monenlaisten järjestelmien testaamiseen. Tässä tutkimuksessa järjestelmällä tarkoitetaan oh-

jelmakoodin esitystapaa. Tässä tutkimuksessa SUS-tulosta arvioidaan Bangorin ym. (2008) kehittämän arviointiasteikon avulla. SUS-mittari soveltuu myös hyvin pienille otoksille (Tullis & Stetson, 2004).

Tutkimuksessa oli olennaista mitata myös eri ohjelmakoodien vaikutus suoritusaikoihin, vastauksen oikeellisuuteen ja SUS-pisteisiin. Testaamisella varmistettiin, että ohjelmakoodi ei ollut vaikuttavana tekijänä tuloksiin. Vastaavasti suoritusaikoja ja SUS-pisteitä verrattiin epäparametrisellä Kruskal-Wallisin testillä ja vastauksen oireellisuutta Fisherin tarkalla testillä.

Tutkimuksen palautteen perusteella huomattiin, että tutkittavien kokemukset eri esitystavoista eivät olleet odotetun laiset. Esitystapojen ensimmäistä ja toista esityskertaa verrattiin Wilcoxonin merkittyjen sijalukujen testillä, jotta voidaan todeta onko esitystavoissa eroa. Testillä oli tarkoitus kartoittaa, oliko esitystavan ensimmäisen ja toisen kohtaamisen välillä tarpeeksi suuri ero, että sillä voisi olla merkitystä jatkotutkimuksen kannalta.

Tutkimuksen laadullisessa osiossa haluttiin löytää heikkonäköisten ohjelmoijien ohjelmointiin liittyviä ongelmia. Todetut ongelmat kategorioitiin oman kokemukseni perusteella yläluokiksi, joiden alle on koottu samankaltaisia kohdattuja ongelmia. Ongelmia selvitettiin kysymyksellä “Kuvaile ongelmia, joita kohtaat ohjelmoidessasi heikkonäköisyytesi vuoksi”.

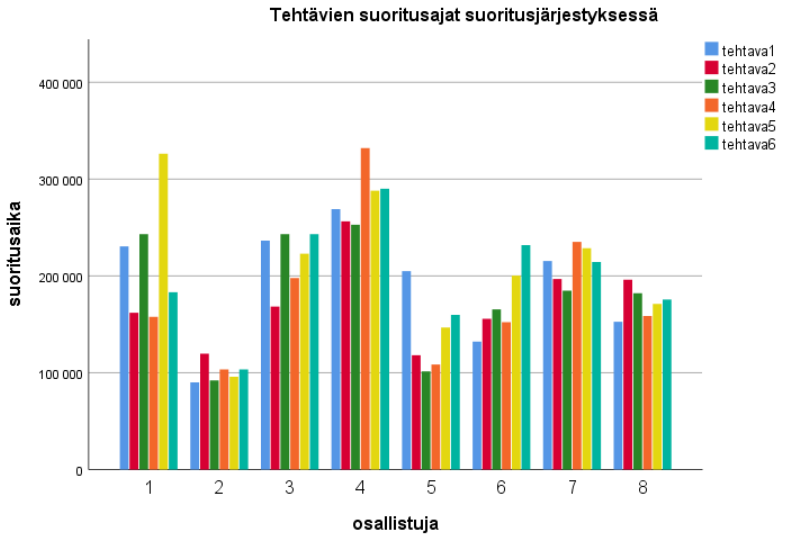
Kategoriointi oli tässä tutkimuksessa aineistolähtöistä, mutta apua kategoriointiin lainattiin teoriatiedosta. Kategorioinnissa huomioitiin samankaltaiset vastaukset, joista luotiin kokoavia yläkategorioita perustuen löyhästi Penningtonin ja Grabowskin (1990) kokoamiin ohjelmoinnin alatehtäviin. Kategoriointi aloitettiin kokoamalla aineistosta lauseita Penningtonin ja Grabowskin (1990) esittämiin alatehtäviin. Alatehtävien täyttyessä vastauksissa huomattiin selviä samankaltaisuuksia, joiden löytämistä helpotti myös selkeät ja yksiselitteiset vastaukset, joita koehenkilöt antoivat. Vastaukset jaettiin eri ohjelmoinnin osatehtäviin ja osalle vastauksista luotiin omat kategoriat, joita Penningtonin ja Grabowskin (1990) alatehtävissä ei ollut. Kategoriat jaoteltiin tämän jälkeen perustuen omaan kokemukseeni ohjelmoinnista. Kategorioiden perustana on ohjelmoinnin eri tehtävät, kuten tiedon hakeminen, ohjelmakoodin lukeminen ja ohjelmakoodin ajamisen jälkeen tapahtuva toimivuuden varmistaminen. Kategoriat olivat luonnollinen tapa kasata yhteen ohjelmoinnin osaan liittyvät ongelmat yhdeksi yläluokaksi. Osassa kategorioissa on päällekkäisyyttä, esimerkiksi ohjelmakoodia luetaan sekä verkossa, että ohjelmointityökaluissa ja ongelmissa on myös samankaltaisuutta, mutta tehtävät ja tarkoitukset ovat erilaiset, joten niiden jakaminen eri kategorioihin on järkevää.

7 TULOKSET

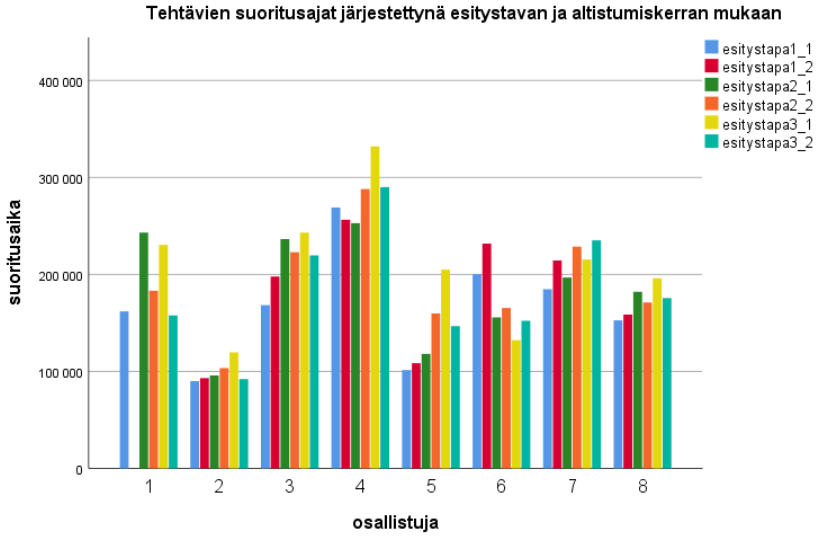
7.1 Tekstin automaattinen rivittäminen ohjelmakoodille

Tässä tutkimuksessa haluttiin tietää, voidaanko ohjelmakoodia esittää heikkonäköiselle henkilölle siten, että ohjelmakoodin lukeminen on helpompaa. Ohjelmakoodin lukemista testattiin ohjelmakoodin lukemista vaativilla tehtävillä, joita esitettiin kolmella eri esitystavalla: ohjelmakoodi muotoiltuna normaalisti (myöhemmin myös *normaali*), ohjelmakoodi rivitettynä tietyn merkkimäärän kohdalta (myöhemmin myös *katkaistu*) ja ohjelmakoodi rivitettynä ohjelmakoodin (myöhemmin myös *rivitetty*) ominaisuuksien perusteella. Ohjelmakoodin lukemisen tehtävistä tallennettiin tehtävän vastaus ja tehtävän suoritusaika. Lukemisen jälkeen tutkittavat täyttivät SUS-kyselyn, jonka perusteella toivottiin saatavan tietoa mahdollisesta esitystavan kuormituksesta. Vastauksen oikeellisuuden, tehtävän suoritusajan ja SUS-tuloksen perusteella voidaan todeta, että onko eri esitystapojen välillä eroa. Koehenkilön 1 yksi tulos on jätetty pois datasta, koska tehtävän tekemisen aikana koehenkilö häiriintyi ja koetulos ei vastannut todellista tilannetta (poistettu aika on kuitenkin selvyyden vuoksi esitetty Kuviossa 7).

Kuviossa 7 on esitelty osallistujien suoritusajat tehtävittäin. Kaaviosta voidaan huomata, että osallistujien suoritusajat ovat pääasiassa samankaltaiset eri tehtävien välillä ja näyttäisivät kasvavan tehtävien edetessä. Kuviossa 8 on esitelty tehtävien suoritusajat järjestyksessä tehtävästä yksi tehtävään kuusi. Kaavion perusteella on vaikea tulkita selvä vastauksia, mutta siitä voidaan huomata, että tavallinen esitystapa johti usein nopeampiin suoritusaikoihin kuin merkkimäärän kohdalta katkaistu ja rivitetty esitystapa.

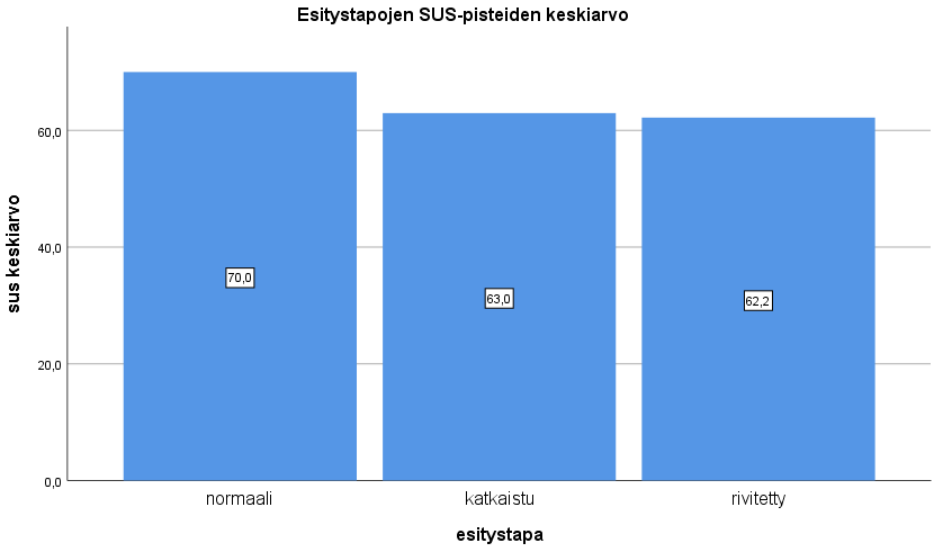


KUVIO 7 Tehtävien suoritusajat suoritusjärjestyksessä.



KUVIO 8 Tehtävien suoritusajat järjestettynä esitystavan ja altistumiskerran mukaan.

Eri esitystapojen aiheuttamaa kuormaa mitattiin käyttämällä apuna SUS-mittaria. Kuviossa 9 on esitelty eri esitystapojen saamat SUS-tulokset. Tavallinen esitystapa sai selvästi paremman tuloksen (keskiarvo 70) kuin kuin merkkimäärän kohdalta katkaistu (keskiarvo 63,0) ja rivitetty esitystapa (keskiarvo 62,2), jotka saivat lähes samanlaisen tuloksen. Bangorin ym. (2008) ehdottaman arviointiskaalan mukaan siis tavallisen esitystavan SUS-tulos on lähellä arvosanaa “hyvä” ja on niin sanotusti korkea hyväksyttävyydeltään, kun taas merkkimäärän kohdalta katkaistu ja rivitetty esitystapa sijoittuvat hyväksyttävyyden marginaalialueelle ja saavat arvosanan “ok”. Sauron (2011) mukaan SUS-tulosten keskiarvo sijoittuu kohdalle 68, joka tarkoittaa sitä, että normaali esitystapa on juuri paremmalla puoliskolla, kun taas katkaistu ja rivitetty esitystapa ovat huonommalla puoliskolla.



KUVIO 9 Esitystapojen SUS-pisteiden keskiarvo

Analyysin tekemiseksi datasta täytyy tietää ovatko esitystapojen ja tehtävien esitysjärjestykset olleet satunnaisia. Taulukossa 1 esitellään eri esitystapojen ja ohjelmakoodien esiintyvyydet suhteessa toisiinsa. Tietyt ohjelmakoodi ja esitystapa –parit esiintyvät selvästi useammin kuin toiset, mutta kokonaisuudessaan parit jakautuvat riittävän satunnaisesti.

TAULUKKO 1 Esitystapojen ja ohjelmakoodien esiintyvyys suhteessa toisiinsa.

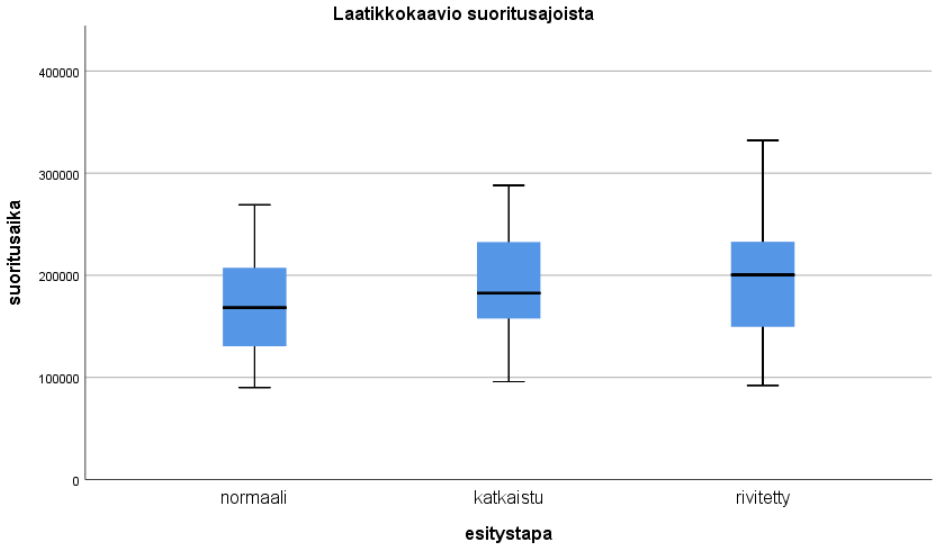
		esitystapa		
		1	2	3
ohjelmakoodi	1	5	1	2
	2	1	6	1
	3	5	1	2
	4	3	1	4
	5	0	3	5
	6	2	4	2

Eri esitystapojen eroja verrattiin vertaamalla esitystavan vaikutusta suoritusaikaan, vastauksen oikeellisuuteen sekä esitystavan saamaan SUS-tulokseen. Nämä olivat siten tutkimuksen riippuvat muuttujat. Suoritusaika on välimatka-asteikollinen, oikeellisuus on dikotominen ja SUS-tulos on järjestyslukuasteikollinen muuttuja. Tämän lisäksi koin tärkeäksi verrata suoritusaikaa, vastauksen oikeellisuutta ja SUS-tulosta esitettyyn ohjelmakoodiin, jotta voitiin todeta, että eri ohjelmakoodi ei ole vaikuttava tekijä tulosten välillä. Tutkimuksen riippumattomia muuttujia ovat esitystapa, sekä ohjelmakoodi.

Tähän tutkimukseen osallistuneiden henkilöiden määrä on melko pieni (N = 8), joka vaikuttaa analysointitapojen valintaan. Otoskoon ollessa pieni on perusteltua valita epäparametrinen testi, jolloin myöskään datan normaaliudella ei ole merkitystä testien kannalta. Otokset ovat toisistaan riippumattomat, kos-

ka suoritusaikoja ei verrata yhden henkilön välillä. Täten epäparametrisenä testinä suoritusajan ja SUS-tuloksen vertaamiseen käytetään Kruskall-Wallisin testiä. Kategoristen muuttujien (esitystapa ja vastauksen oikeellisuus) vertaamiseen voidaan käyttää Fisherin testiä. Fisherin testi soveltui tässä tutkimuksessa Khiin neliön testiä paremmin, koska taulukossa on arvoja, jotka ovat alle viisi.

Esitystavan vaikutusta suoritusaikaan testattiin Kruskal-Wallisin testillä. Esitystavalla ei huomattu olevan tilastollisesti merkitsevää vaikutusta suoritustaikaan (H (2) = 0,830, p = 0,660). Kuviossa 10 on esitelty eri esitystapojen laatikko-janakuviossa. Vaikka tehtävien suoritusajoissa voidaan nähdä jonkin verran vaihtelua, niin koehenkilöt suoriutuivat omista tehtävistään hyvin samankaltaisin ajoin. Tavallisen esitystavan mediaani (Mdn = 168392ms) on hieman alhaisempi kuin katkaistun (Mdn = 182618ms) ja rivitetyn (Mdn = 200463ms). Kuitenkin tehtävien suoritusajoissa on selvää päällekkäisyyttä. Esitystapa ei siis vaikuta suoritusaikaan.



KUVIO 10 Laatikkokaavio suoritusajoista

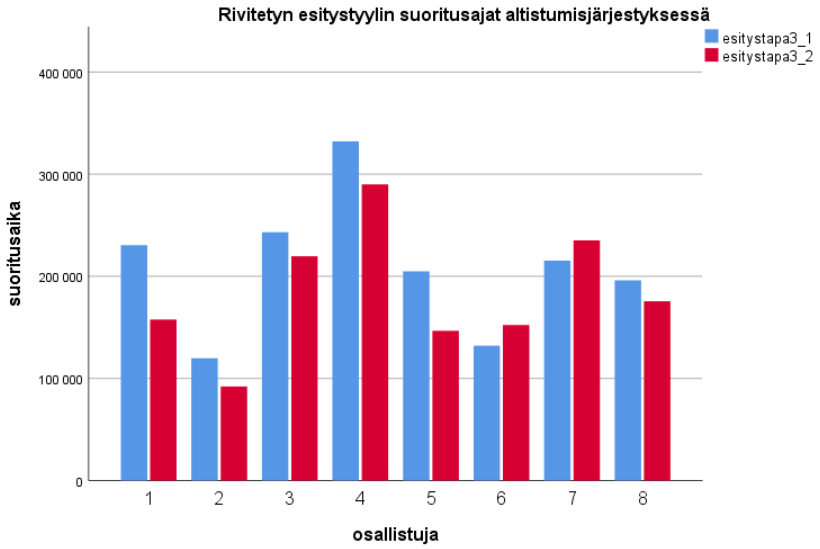
Esitystavan vaikutusta vastausten oikeellisuuteen verrattiin Fisherin tarkalla testillä. Esitystavalla ei huomattu olevan tilastollisesti merkitsevää yhteyttä tehtävien vastausten oikeellisuuteen (p = 0,345). Tulos ei ole yllättävä, sillä vain viisi (10 % vastauksista) oli väärin. Normaali esitystavalla vääriä vastauksia ei tullut, kun taas katkaistulla esitystavalla huomattiin kolme virhettä ja rivitetyllä esitystavalla kaksi virhettä. Testin pohjalta ei voida kuitenkaan sanoa johtuuko virheiden esiintyminen esitystavasta.

Esitystavan saamaa SUS-tulosta testattiin Kruskal-Wallisin testillä. Esitystapojen SUS-tulokset eivät eronneet tilastollisesti merkitsevästi toisistaan (H (2) = 3,619, p = 0,164). Aiemmin esitetyssä Kuviossa 9 on esitelty eri esitystapojen saamat SUS-tulokset. Normaalin esitystavan saama SUS-tulos on korkeampi (Mdn = 70,0) kuin katkaistun (Mdn = 66,25) ja rivitetyn (Mdn = 63,75) esitystavan. Normaalin esitystavan keskihajonta (SD = 9,06) on myös pienempi kuin

katkaistun (*SD* = 10,73) ja rivitetyn (*SD* = 12,44). Tämä voidaan huomata siinä, että esitystapojen saamissa tuloksissa on suurta eroa. Tavallisen esitystavan saamien tulosten ero on vain 30 (*Min* = 55, *Max* = *85*) kun taas katkaistun oli 40 (*Min* = 40, *Max* = *80*) ja rivitetyn 42,5 (*Min* = 37,5, *Max* = *80*). Testin perusteella ei voida kuitenkaan väittää normaalin esitystavan olevan selvästi parempi kuin katkaistu tai rivitetty esitystapa. Katkaistu ja rivitetty esitystapa näyttäisivät kuitenkin jakavan mielipiteitä enemmän kuin normaali esitystapa.

Ohjelmakoodin vaikutusta suoritusaikaan, vastausten oikeellisuuteen ja SUS-tulokseen, jotta tiedetään että ohjelmakoodi ei itsessään vaikuta suoritukseen. Ohjelmakoodin vaikutusta suoritusaikaan ja SUS-tulokseen verrattiin Kruskal-Wallisin testillä. Ohjelmakoodilla ei huomattu olevan tilastollisesti merkitsevää vaikutusta suoritustaikaan (H (5) = 0,396, p = 0,995). Ohjelmakoodien SUS-tulokset eivät eronneet tilastollisesti merkitsevästi toisistaan (H (5) = 3,053, p = 0,692). Ohjelmakoodin vaikutusta vastausten oikeellisuuteen verrattiin Fisherin tarkalla testillä. Ohjelmakoodilla ei huomattu olevan tilastollisesti merkitsevää vaikutusta tehtävien vastausten oikeellisuuteen (p = 0,453). Suoritusaika, vastausten oikeellisuus ja SUS-tulos eivät siis riipu esitetystä ohjelmakoodista, joten voidaan sanoa, että ohjelmakoodit olivat riittävän samankaltaisia.

Esitystavalla ja tehtävällä ei ole tilastollisesti merkitsevää vaikutusta suoritusaikaan, tehtävien oikeellisuuteen tai SUS-tulokseen. Hypoteesia H1 ei voida siis vahvistaa tulosten perusteella, joten nollahypoteesi jää voimaan. Tuloksia tulkitessa huomattiin kuitenkin mielenkiintoinen efekti. Kuten Kuviossa 11 voidaan huomata, niin monessa tapauksessa rivitetyn esitystavan ensimmäinen kohtaaminen vaikuttaisi negatiivisesti tehtävän suoritusaikaan eli ensimmäinen altistuminen näyttäisi vievän kauemmin aikaa kuin toinen. Kaikkien esitystapojen ensimmäisen ja toisen altistumisen suoritusaikoja verrattiin Wilcoxonin merkittyjen sijalukujen testillä. Tavallisen (Z = -1,69, p = 0,91) ja merkkimäärän kohdalta katkaistun (Z = -0,70, p = 0,484) esitystavan suoritusajoissa ei ollut eroa, kun verrataan ensimmäistä ja toista altistumista esitystavalle, kun taas rivitetyn tekstin suoritusaikojen välillä havaitaan tilastollisesti merkitsevä ero, Z = -2,10, p = 0,036 (p < 0,05). On siis mahdollista, että jos tutkittavat olisi altistettu uudelle esitystavalla useammin tai ennen tehtävien suorittamista, niin tuloksissa voitaisiin havaita erilainen efekti suoritusajoissa. Tätä tulkintaa vahvistaa myös se, että palautetta antaessa viisi koehenkilöä mainitsi, etteivät ole käyttäneet vastaavaa ratkaisua ja kokivat sen käyttämisen epämukavaksi koska heillä ei ollut kokemusta siitä. Samat viisi koehenkilöä mainitsivat, että saattaisivat hyötyä vastaavasta ratkaisusta ohjelmoidessaan. Suurin osa koehenkilöistä ei ollut myöskään käyttänyt katkaistua esitystyyliä kuin harvoissa tilanteissa.



Kuvio 11 Rivitetyn esitystyylin suoritusajat altistumisjärjestyksessä

7.2 Heikkonäköisten ongelmat ohjelmoinnissa

Tutkimuksessa haluttiin kartoittaa myös tietoa siitä millaisia ongelmia heikkonäköiset kohtaavat ohjelmoinnissa. Tutkimus toteutettiin englannin kielellä, joten kaikki vastaukset on suomennettu. Kaikki tutkimuksen osallistujat (N = 8) vastasivat avoimiin kysymyksiin "Kuvaile heikkonäköisyyttäsi" ja "Kuvaile ongelmia, joita kohtaat ohjelmoidessasi heikkonäköisyytesi vuoksi". Koehenkilöillä oli mahdollisuus jättää vastaamatta kumpaankin kysymykseen. Tässä osiossa esitellään koehenkilöiden kohtaamat ongelmat.

Taulukossa 2 on esitelty tiivistettynä koehenkilöiden ilmoittamat näön ongelmat ja ongelmat, joita koehenkilöt ovat kohdanneet ohjelmointiin liittyen. Kohdatut ongelmat on kategorioitu teemoiksi oman ohjelmointikokemukseni ja aihealueen ymmärrykseni perusteella ja niiden merkitys avataan myöhemmin. Taulukkoon on lisäksi ilmoitettu ohjelmointikokemus vuosina, koska koin sen olevan merkitsevä tekijä tiettyjen tulkintojen kanssa.

Koehenkilöillä 1 ja 5 toinen silmä oli sokea ja toinen silmä erittäin heikkonäköinen. Koehenkilöillä 2, 7 ja 8 osa näkökentästä on sumea ja lisäksi koehenkilöllä 2 on näkökentässä kohtia, jotka estävät näkemistä. Koehenkilöllä 4 on näkökentässä esteitä, jonka lisäksi kummankin silmän näöntarkkuus on huono. Koehenkilöllä 3 on putkinäkö, eli henkilö näkee vain näkökentän keskialueen ja reunat näyttäytyvät sumeina. Koehenkilöllä 6 kummankin silmän näöntarkkuus on huono. Koehenkilöt 5 ja 6 ilmoittivat, että heillä on myös värisokeus. Koehenkilö 5 on täysin puna-vihersokea, mutta koehenkilö 6 ei tarkentanut värisokeuden ilmentymää. Koehenkilöiden heikkonäköisyyksien ilmentymien kirjo on siis melko kattava, mutta pienen otannan vuoksi se myös vaikeuttaa tarkempien tulkintojen tekemistä.

Koehenkilöiden kokemusten kirjo on myös melko kattava. Puolet koehenkilöt jakautuvat erittäin kokeneisiin (50 %:lla koehenkilöistä on yli 16 vuotta kokemusta) ja loput koehenkilöistä ovat sekalaisesti aloittelijoista jo kokeneisiin (1—7 vuotta kokemusta).

TAULUKKO 2 Koehenkilöiden ohjelmointikokemus, heikkonäköisyyden esiintymä ja ohjelmoinnissa kohdatut ongelmat

Koehenkilö	**Ohjelmointi-kokemus (vuotta)**	**Heikkonäköisyys**	**Kohdatut ongelmat**
1	4	Toinen silmä sokea ja toisen silmän näöntarkkuus erittäin huono	Haetun tiedon lukeminen, ohjelmakoodin lukeminen, ohjelmakoodin tulosten tulkinta, ohjelmointityökalujen käyttäminen
2	28	Estynyt näkökenttä ja osa näkökentästä sumea	Haetun tiedon lukeminen, ohjelmointityökalujen käyttäminen
3	31	Putkimainen näkökenttä	Ohjelmakoodin lukeminen, tulosten tulkinta, ohjelmointityökalujen käyttäminen
4	20	Estynyt näkökenttä ja kummankin silmän näöntarkkuus huono	Haetun tiedon lukeminen, ohjelmakoodin lukeminen, ohjelmointityökalujen käyttäminen, ohjelmakoodin tulosten tulkinta, sosiaalinen vaikutus
5	16	Toinen silmä sokea ja toisen silmän näöntarkkuus erittäin huono, puna-vihersokeus	Ohjelmakoodin lukeminen, ohjelmakoodin tulosten tulkinta, ohjelmointityökalujen käyttäminen
6	7	Kummankin silmän näöntarkkuus erittäin huono, värisokeus	Haetun tiedon lukeminen, ohjelmakoodin lukeminen, ohjelmointityökalujen käyttäminen, sosiaalinen vaikutus
7	1	Osa näkökentästä sumea	Haetun tiedon lukeminen, ohjelmakoodin lukeminen, ohjelmakoodin tulosten tulkinta, ohjelmointityökalujen käyttäminen, sosiaalinen vaikutus
8	5	Osa näkökentästä sumea	Haetun tiedon lukeminen, ohjelmakoodin lukeminen, ohjelmointityökalujen käyttäminen

> #1 Dokumentaation lukeminen on minulle hankalaa, koska usein etenkin erilaiset ohjelmakoodiesimerkit eivät muutu tekstin asetuksien mukaan.
>
> #4 Verkkosivuilla koodin lukeminen aiheuttaa ongelmia ja välillä alkaa turhauttaa ja koodin kirjoittamisesta ei tule mitään.
>
> #8 - - usein dokumentaation ohjelmakoodi eri verkkosivuilla on vaikea lukea. Tämän huomaa usein lisäosien dokumentaatiossa. - - Monet muutkin [ohjelmointiin liittyvät] sivut ovat täysin saavuttamattomia.

Haetun tiedon lukeminen näyttäisi aiheuttavan heikkonäköisille ongelmia. Haetulla tiedolla tässä viitataan esimerkiksi hakutyökalua käyttämällä haettuun tietoon kuten ohjelmakielen tai työkalun dokumentaatioon sekä toisten luomiin ratkaisuihin. Viisi koehenkilöä ilmaisi, että erityisesti dokumentaation lukeminen on ongelma. Koehenkilöt pitivät erityisesti virallisia dokumentaatioita ja yleisimmin käytettyjä tietolähteitä saavutettavuudeltaan hyvänä. Usein esimerkiksi blogikirjoitukset, lisäosien dokumentaatiot ja forumeilla esitetyt ratkaisut aiheuttavat ongelmia lukemisen kanssa. Koehenkilöt erittelivät pääasialliseksi ongelmaksi erityisesti sen, että verkkosivun teksti on responsiivista ja muovautuu käyttäjän asetuksiin, mutta usein koodin sisältävä osio muovautuu vaikeaksi lukea tai ei muovaudu ollenkaan. Erityisesti nuoremmat ohjelmoijat mainitsivat myös siitä, että ohjelmakoodin lukeminen ohjelmointiympäristöjen ulkopuolella oli haastavaa. Saavutettavuuden puute eri lähteissä koettiin myös kuormittavaksi tekijäksi, jolloin sivulla esitetyn tiedon ymmärtäminen vaikeutuu, kun tietoa joudutaan yhdistelemään.

> #5 Näköni tekee hankalaksi lukea [ohjelmakoodia], koska käytän niin suurta zoomausta. Käytän usein apuna ruudunlukusovellusta, jos ohjelmakoodi on pitkä. Yleensä luen kommentteja tai käytän apuna ohjelman suunnitelmaa.
>
> #6 Ohjelmakoodin lukeminen on alusta asti ollut turhat hankalaa ja jotkin ratkaisut jopa vain hankaloittavat sitä. Joskus olen joutunut kopioimaan ohjelmakoodia Notepadiin, että pystyin edes lukemaan sitä. Yleensä ohjelman pystyy ymmärtämään paremmin kommenteista.
>
> #7 Hyvän ohjelmointiympäristön etsiminen oli alkuun hankalaa ja uskon, että en ole ainoa, jolla on ollut tuo ongelma.

Kaikki koehenkilöt kokivat ohjelmakoodin lukemisessa olevan ongelmia. Ohjelmakoodin lukemisella tarkoitetaan nimenomaan ohjelmakoodin lukemista ohjelmointiympäristöissä. Etenkin kokemattomammat ohjelmoijat kokivat ohjelmakoodin lukemisen olevan hankalaa. Kokeneemmat ohjelmoijat kokivat sopeutuneensa ohjelmakoodin lukemiseen kohtalaisen hyvin, mutta kokivat silti sen olevan joissain tilanteissa hankalaa. Erityisesti ohjelmointiympäristöjen taipumattomuus saavutettavuuteen esiintyi usein ongelmana ja esitellään myöhemmin. Saavutettavuuden puute tai mahdollisuudet muokata lukukokemusta koettiin vaikuttavan ohjelmakoodin lukemiseen. Yleisesti katsottuna ongelmat liittyvät enemmän siihen, että ohjelmakoodia ei voida muokata omiin tarpeisiin

nähden, kuin siihen että ohjelmakoodi itsessään olisi liian vaikeata lukea. Osa koehenkilöistä kertoi pärjäävänsä ohjelmakoodin kanssa kohtalaisen hyvin vaikka saavutettavuus olisikin huono, mutta toiset kokivat sen tekevän ohjelmakoodin lukemisen liian vaikeaksi, jolloin ohjelmakoodin lukeminen on liian raskasta. Erityisesti kokeneemmat ohjelmoijat korostivat myös sitä, että ohjelmakoodin lukemista voi välttää lukemalla kommentteja tai käyttämällä apuna ohjelmistoa kuvaavia kaavioita.

> #2 Ohjelmointiympäristöt ja muut ohjelmistot ovat yleensä aika huonosti saavutettavia. Kehitystä on tapahtunut paljon ja ainakin yleisimmät tuntuvat olevan jo lähes 70 % saavutettavia, mutta kehitystä voisi silti vielä tapahtua. Itse en ole kokenut, että mikään olisi täysin mahdoton käyttää, mutta usein työ häiriintyy ongelmien vuoksi.

> #3 Olen vuosien varrella joutunut käyttämään monenlaisia ohjelmointiympäristöjä. Osa (VS Code ja Visual Studio) ovat jo ainakin hyvin saavutettavia ja kehitystä on tapahtunut paljon. Olen joutunut joskus taistelemaan työnantajan kanssa, että saan vaihtaa [ohjelmointiympäristöä], koska se ei ollut saavutettava.

> #5 Pidän Visual Studiosta, koska se on suurimmaksi osaksi saavutettava ja sitä pystyy käyttämään ihan hyvin myös ruudunlukijan kanssa. Eri ohjelmointiympäristöt eivät aina ole saavutettavia. Myös monet ohjelmointia tukevat ohjelmat ovat saavutettavuudeltaan huonoja. Esimerkiksi Microsoftin Access on mahdoton käyttää.

Erilaisten ohjelmointiin liittyvien työkalujen ja ohjelmointiympäristöjen käyttäminen koettiin ongelmaksi. Useat vastaajat olivat sitä mieltä, että varsinkin yleisimmässä käytössä ohjelmointiympäristöt ovat hyvin saavutettavia. Etenkin kokeneemmat ohjelmoijat kokivat saavutettavuuden olevan hyvällä tasolla. Koehenkilöt eivät eritelleet kaikkia ohjelmointiympäristöissä ongelmia aiheuttavia tekijöitä. Ongelmiksi koettiin kuitenkin saavutettavuuden puute, joka häiritsi ohjelmointia tai pahimmillaan jopa teki työn mahdottomaksi. Erilaisten ohjelmointityökalujen aiheuttamat häiriötekijät vaikuttivat ohjelmointiin muun muassa keskeytyksien muodossa.

> #4 Joskus virheiden löytäminen on hankalaa ja riippuen vähän siitä mitä koodi tekee, niin joskus on vaikea tietää, että tekikö koodi sen mitä halusi. Onneksi suurimman osan koodista voi testata.

> #5 - - myös debuggaaminen ja ohjelmakoodin tarkastaminen on hankalaa. En pysty tekemään käyttöliittymäkoodia ainakaan loppuun asti, koska en välttämättä näe pienien muutoksien eroja.

Koehenkilöillä oli myös hankaluuksia ohjelmakoodin tulosten tulkinnassa. Tällä tarkoitetaan niitä tuloksia, jotka syntyvät, kun koodi ajetaan. Toisin sanoen ohjelmakoodi suoritetaan onnistuneesti tai sen suorittaminen keskeytyy virheen vuoksi. Koehenkilöt mainitsivat ongelmaksi ohjelmakoodissa syntyneen virheen havaitsemisen, tulkitsemisen sekä joissain tapauksissa myös onnistuneesti suoriutuneen ohjelmakoodin tuloksen havaitsemisen. Yleisin ongelma tuloksen havaitsemisessa on joko monimutkaisen tulosteen oikeaksi toteaminen

esimerkiksi ison datataulukon tapauksessa tai näytöllä tapahtuvien visuaalisten muutoksien toteaminen. Ratkaisuna osa koehenkilöistä ohjelmoi testejä, jotka tarkistavat ohjelmakoodin toimintaa. Testien tekeminen on hyvinkin tavallista ohjelmoinnissa, mutta tässä tapauksessa tulkitsin, että kyseessä on joskus nimenomaan ohjelmakoodia, joka ei välttämättä tarvitsisi testejä, ellei heikkonäköisyys estäisi tulosten tulkintaa.

> #4 Työtapani vaikeuttaa joskus työtä kollegoiden kanssa, koska minun tapani koodata eivät välttämättä aina ole samanlaisia kuin kollegoiden.
>
> #6 Joskus kohtaan ongelmia työkavereiden kanssa. Tuntuu välillä, että hitaampi työtapani haittaa kollegoita.
>
> #7 Muut opiskelijat ovat kehittyneet paljon nopeammin, koska näkökyky haittaa oppimistani.

Osa koehenkilöistä koki myös heikkonäköisyyden aiheuttavan tietynlaista sosiaalista vaikutusta ohjelmointiin liittyen. Etenkin kokemattomammat ohjelmoijat kokivat painetta siitä, että eivät ole yhtä tehokkaita ja taitavia ohjelmoijia kuin vertaisensa tavallisesti näkevät henkilöt. Kokeneempi koehenkilö taas koki painetta siitä, että kollegoiden kanssa yhteistyö ei ole aina sujuvaa.

8 POHDINTA

Tämän tutkimuksen yksi tarkoituksista oli tutkia, voidaanko heikkonäköisen ohjelmoijan kokemaa kognitiivista kuormaa laskea käyttämällä tekstin automaattisen rivittämisen menetelmää ohjelmointikielille. Tutkimuksessa verrattiin ohjelmakoodille tyypillistä tekstin rivitystä, merkkimäärän kohdalta katkaistua rivitystä ja automaattista rivitystä. Vertaamalla tehtävien suoritusaikoja ja vastausten oikeellisuutta, sekä tehtävän saamaa SUS-pistettä todettiin, että eri menetelmien välillä ei ole tilastollisesti merkitsevää eroa. Tutkimuksen hypoteesia H1 ei voida siis tulosten perusteella varmentaa, joten tutkimuksen nollahypoteesi jää voimaan. Tulos ei myöskään riippunut tutkimukseen luoduista ohjelmakoodeista. On kuitenkin huomattava, että tutkittavat eivät olleet kokeneita merkkimäärän kohdalta tai automaattisesti rivitetyn esitystavan kanssa, joten erojen puute on jopa yllättävä tulos. Useamman altistumisen tai ennen tehtäviä tehtävän altistumisen perusteella voitaisiin ehkä huomata erilainen efekti erityisesti, koska osa koehenkilöistä kertoi, että voisi hyötyä vastaavasta ratkaisusta ohjelmakoodin lukemisessa. Heikkonäköiset hyötyvätkin tavallisen tekstin lukemisessa tekstin automaattisesta rivittämisestä suuresti (esim. Chan, 2017; Hallett, Arnsdorff ym., 2015; Hallett ym, 2017), joten voi olla, että erilaiset rivitysmenetelmät tehostavat ohjelmakoodin lukemista heikkonäköisellä ja siksi tavallisen ja muokattujen esitystapojen välillä ei voida nähdä tilastollisesti merkitsevää eroa, vaikka koehenkilöt eivät olleet kokeneita vaihtoehtoisten ohjelmakoodin esitystapojen kanssa. Ohjelmointi on kognitiivisesti raskas tehtävä, johon liittyy paljon muitakin osa-alueita ja alatehtäviä kuin lukeminen (Helgesson ym., 2019; Kelly & Buckley, 2009; Pennington & Grabowski, 1990; Roehm ym., 2012; Sillito ym., 2006), joten toisena tulkintana voidaan pitää myös sitä, että tähän tutkimukseen valituilla esitystavoilla ei ole vahvaa merkitystä ohjelmakoodin lukemisen kannalta. Tämän tutkimuksen kahdella altistumisella ja koehenkilöiden hyvin vähäisellä esitystapojen kokemuksella, sekä määrällä, ei voida kuitenkaan sanoa onko kumpikaan tulkinta oikea tai onko efektille jokin muu selittävä tekijä.

Tutkimuksen toinen tarkoitus oli kartuttaa tietoa siitä, millaisia ongelmia heikkonäköiset kohtaavat ohjelmointiin liittyen. Koehenkilöt saivat vastata vapaaehtoiseen kysymykseen ongelmista, joita he kohtaavat ohjelmointiin liittyen. Vastausten perusteella luotiin viisi ongelmakategoriaa: haetun tiedon lukeminen, ohjelmakoodin lukeminen, ohjelmakoodin tulosten tulkinta, ohjelmointityökalujen käyttäminen ja sosiaalinen vaikutus. Kategorioista kerrottiin tarkemmin ja niiden vaikutusta arvioitiin koehenkilöiden kokemuksen suhteen. Kokonaisuudessaan kategoriat kertovat, että heikkonäköisyys aiheuttaa ongelmia laajasti ohjelmointiin liittyen sen eri vaiheissa. Pennington ja Grabowski (1990) esittivät ohjelmoinnin alatehtäviksi ongelman ymmärtämisen, suunnittelun, koodaamisen ja ylläpidon. Tässä tutkimuksessa havaitut ongelmat kattavat kaikki nämä alatehtävät. Ongelman ymmärtämiseen liittyvät ongelmat esimerkiksi ohjelmakoodin lukeminen ja tulosten tulkinta (esim. virheiden etsiminen) ja suunnitteluun taas haetun tiedon hakeminen. Koodaamiseen ja ylläpitämiseen taas liittyvät ohjelmakoodin lukemisen, haetun tiedon lukemisen sekä tulosten tulkinnan kategoriat. Helgessonin (2019) tutkimuksessa ohjelmoijien kognitiivisiksi kuormitustekijöiksi taas löydettiin informaation, työprosessit ja työkalut. Informaatioon liittyviä ongelmia olivat muun muassa informaation yhtenäisyys ja sen järjestely, johon liittyy tässä tutkimuksessa todettu haetun tiedon lukeminen. Työkaluihin liittyvät ongelmat vastaavat tässä tutkimuksessa löydettyjä ongelmia: työkalujen mukautuvuuden puute, ominaisuuksien huono toteutus tai puute, viivästymiset sekä vuorovaikutuksen ongelmat. Työprosessien ongelmat olivat taas pääasiassa tuen puutteen kokeminen. Tässä tutkimuksessa ei varsinaisesti löydetty vastaavaa ongelmaa, vaan vastaavan kaltaiset ongelmat liittyivät ryhmäpaineeseen ja kollegoiden sekä heikkonäköisen yhteisten prosessien epäsopivuuteen.

Haetun tiedon lukemisessa nähdään samoja ongelmia kuin aikaisemmissa tutkimuksissa on todettu. Swellerin (1988) kognitiivisen kuormituksen teoria määrittää, että tiedon oppiminen vaikeutuu, kun ihmisen kognitiota kuormitetaan liiallisella tiedolla. Tutkittavat kertoivat, että erityisesti verkosta haetun ohjeistuksen ja tiedon lukeminen on hankalaa, koska ohjelmakoodi ja teksti eivät muotoudu aina yhtäläisesti. Responsiivisuus on yksi digitaalisen tekstin tärkeimpiä avustavia menetelmiä heikkonäköisille (esim. Chan, 2017). Hallett, Arnsdorff ym. (2015) mukaan ihmisen kognitio voi rasittua digitaalista tekstiä lukiessa esimerkiksi vaakasuuntaisen vierityksen takia. Chenin ja Linin (2014) tutkimuksessa taas käyttäjät kokivat, että häiriötön ja jatkuva lukeminen oli olennaista digitaalisen tekstin lukemiseksi. Sivuilla olevien ohjelmakoodien mukautuvuus oli eritysesti mainittu puutteellisena. Leggen (2016) mukaan erityisesti heikkonäköisille sisällön mukautuvuus on saavutettavuuden kannalta olennaista. Yleinen saavutettavuus on siis edelleen ongelma erityisesti, jos verkkosivulla on useampaa erilaista sisältöä. Heikkonäköiset ohjelmoijat kärsivät tällä hetkellä sisällön mukautuvuuden puutteista heidän tarpeisiinsa.

Ohjelmakoodin lukeminen, jota myös tämän tutkimuksen empiirisessä osiossa pyrittiin testaamaan, oli selvä ongelmatekijä. Koehenkilöt mainitsivat muun muassa ohjelmakoodin saavutettavuuden ongelmaksi ja joutuivat käyttämään erityylisiä strategioita välttääkseen itse ohjelmakoodin lukemista. Erityisesti kaaviot ohjelmasta, sekä kommenttien tärkeys nousivat esille. Roehm ym. (2012) löysivät myös tutkimuksessaan, että ohjelmoijat käyttävät erityylisiä kontekstisidonnaisia strategioita ohjelmakoodin ymmärtämiseen. Woodfield ym. (1981) löysivät myös kommentoinnin auttavan ohjelmakoodin ymmärtämistä. Ohjelmakoodin mukautuvuuden ja sen oikeanlaisen esittämisen tutkimukselle on siis tarvetta myös jatkossa. Tämä on olennaista myös haetun tiedon lukemiseen liittyen, koska heikkonäköisille sopivimman esitystavan löytäminen voisi helpottaa myös verkossa erilaisten ohjelmakoodien lukemista.

Helgessonin ym. (2019) mukaan työkalujen mukautuvuuden puute ja ominaisuuksien puute tai ongelmat toteutuksessa olivat yksiä ohjelmoijien kognitiivisia kuormittajia. Myös Dillon ja Thompson (2016) ovat todenneet työkalujen huonon saavutettavuuden vaikeuttavan ohjelmoimistyötä. Tässä tutkimuksessa heikkonäköiset ohjelmoijat mainitsivat työkalujen suurimmaksi ongelmaksi etenkin ohjelmointia tukevien tuotteiden puutteet saavutettavuudessa. Etenkin ohjelmistoympäristöt koettiin riittävän saavutettaviksi. Heikkonäköiset ohjelmoijat mainitsivat myös, että jotkin työkalut ovat mahdottomia käyttää, jonka voi helposti nähdä olevan suuri ongelma, koska silloin työtä ei voida toteuttaa ollenkaan. Tässä tutkimuksissa koehenkilöt eivät eritelleet moniakaan ongelmia, joka voi johtua siitä, että yleisesti vastaukset olivat pääsääntöisesti kaksijakoisia: työkalut ovat joko riittävän saavutettavia tai mahdottomia käyttää. Ongelmien määrä tietyissä ohjelmissa voi olla niin suuri, että heikkonäköinen ohjelmoija ei voi edes käyttää työkalua. Heikkonäköisten ohjelmoijien kohtaamat ongelmat ohjelmointiin liittyvissä työkaluissa ovat selvä ongelmakohta, josta pitäisi tehdä enemmän tutkimusta.

Ohjelmakoodin tulosten tulkinnassa käyttäjät kokivat visuaalisten muutosten havaitsemisen, monimutkaisen tulosteen tulkitsemisen ja ongelmien löytäminen koodin seasta. Tulosten tulkitseminen ja ongelmien löytäminen koodin seasta liittyvät hyvin vahvasti yleisesti ohjelmakoodin lukemisen ongelmiin. Visuaalisten muutosten havaitseminen taas voi olla monimutkaisempi ongelma ja liittyy suoraan heikkonäköisyyteen ja sen eri ominaisuuksiin. Esimerkiksi kuten koehenkilötkin totesivat, niin vahva heikkonäköisyys voi johtaa siihen, että pienet muutokset ruudulla jäävät täysin havaitsematta, joka voi rajoittaa heikkonäköisen ohjelmoijan mahdollisuuksia toteuttaa erityylisiä ohjelmia.

Sosiaalinen vaikutus näkyi erityisesti heikkonäköisten ohjelmoijien heikentyneenä itsetuntona omasta osaamisestaan. Heikkonäköisyys on selvä vaikuttaja työssä ja opiskeluissa menestymisessä (esim. Maailman terveysjärjestö, 2019). Heikkonäköisyys voi vaikuttaa monella tapaa esimerkiksi mielenterveyteen, kognitioon, sosiaaliseen toimimiseen ja yhteiskunnalliseen suoriutumiseen

(esim. Bassey ja Ellison, 2020; Numan ym., 2010; Welp ym., 2016), joten saman vaikutuksen näkyminen tässäkin tutkimuksessa ei ole poikkeavaa. Heikkonäköisyyden yhteiskunnalliset vaikutukset ja yleiset vaikutukset esimerkiksi työssä suoriutumiseen ovat melko hyvin tutkittu, mutta heikkonäköisten ohjelmien kokemat ongelmat esimerkiksi työ- ja opiskeluympäristöissä ovat vielä kartoittamattomat. Tutkimusta pitäisi tehdä siitä, miten heikkonäköisyys vaikuttaa erityisesti ohjelmointityön sujumiseen ja miten heikkonäköisyys vaikuttaa ohjelmoijan itsetuntoon omista kyvyistä ja soveltuvuudestaan työ- tai opiskeluympäristöön.

8.1 Menetelmän pohdinta

Tämän tutkimuksen osalta menetelmää täytyy pohtia kahdelta kannalta. Tutkimuksessa toteutettiin sekä empiirinen, että laadullinen osio, joiden validiteetti ja reliabiliteetti ovat erilaiset. Empiirisen osion kannalta on tärkeä, että tutkimusmenetelmä ja sen analysoimiseen valitut mittarit on valittu oikein. Laadullisen osion kannalta on taas tärkeää, että tutkimuksen kysymykset on valittu oikein ja kyselyt on suunniteltu huolella tarkoituksellisesti.

Tutkimuksen empiiriseen osioon voidaan palautteen perusteella löytää selviä ongelmakohtia. Koehenkilöt mainitsivat palautteessa, että tutkimuksessa verrokkina käytetyt merkkimäärän kohdalta rajattu esitystapa sekä tutkimusta varten kehitetty automaattista rivitystä muistuttava tapa eivät olleet tuttuja. Koehenkilöt kokivat, että tuntemattomuus vaikutti suoritukseen. Tutkimuksen validiteetin kannalta olisi ollut hyvä altistaa koehenkilöt esitystavoille ennen tehtävien suorittamista, jolloin koehenkilöt olisivat saattaneet saada tasalaatuisempia tuloksia. Toisaalta tutkimuksessa huomattiin myös mielenkiintoinen tulos, jossa esitystavan alkuhämmennyksen jälkeen suoritusaika putosi keskimäärin nopeasti. Tämän lisäksi tutkimusta varten olisi kannattanut tehdä jonkinlainen esiselvitys ongelmista, jolloin tutkimus olisi voitu toteuttaa parempien esitietojen perusteella.

Koehenkilöt eivät kokeneet, että tutkimuksessa käytetyt ohjelmakoodit olisivat poikenneet liikaa toisistaan. Ohjelmakoodien luomiseen käytettiinkin huomattava aika ja apuna käytettiin jo Luxton-Reilleyn ja Petersenin (2017) ja Kaston ja Whalleyn (2013) tutkimaa tietoa siitä, mikä vaikuttaa ohjelmakoodin hankaluuteen ja aloittelijaystävällisyyteen. Ohjelmakoodien samankaltaisuutta testattiin useampaan otteeseen normaalisti näkevillä henkilöillä ja kaikki ohjelmakoodi esitettiin heille ohjelmakoodin normaalilla rivityksellä, jotta voitiin minimoida esitystavan ja näkökyvyn vaikutus.

Palautteen perusteella myös SUS-kysely tuotti tutkittaville pientä hämmennystä. Tutkittavat ymmärsivät, että SUS-kysely liittyi erityisesti esitystapaan, mutta ennen altistumista muille esitystavoille tutkittavat kokivat häm-

mennystä etenkin, jos ensimmäisenä esitetty esitystapa oli normaalisti rivitetty ohjelmakoodi. Tässäkin tapauksessa tutkittavien altistaminen eri esitystavoille olisi palvellut sitä, että tutkittavat ymmärtäisivät täysin mitä esitystavalla tarkoitetaan ja näin voitaisiin saada luotettavampia vastauksia.

Tutkimuksen laadullinen osio on tarkoituksellisesti suppea ja tutkimuksen tarkoitus olikin enemmän kartoittava kuin asiaa syvemmin selvittävä. Tutkimuksessa käytetyt suorat kysymykset saivat suoria vastauksia, joita oli helppo tulkita. Tutkimukseen toteutetut ongelmien kategoriat ovat toteutettu oman mieleni mukaan. Koin kuitenkin, että kokemukseni ohjelmoinnista ja ymmärrykseni aihealueesta on riittävä vastausten kategorioimiseen. Vastaukset olivat lisäksi myös hyvin kategoriamaisia, jolloin vastausten kategorioiminen ja kategorioiden luominen oli lähes luonnollista. Vastaavan kaltaisesta tutkimuksesta on löydetty myös samoja efektejä muissa ympäristöissä, jolloin on luonnollista, että niitä löytyy ohjelmointiin liittyen.

Toisaalta voidaan kyseenalaistaa, että onko suoran kysymyksen vastausten laatu tarpeeksi kattava, koska se perustuu koehenkilöiden omiin kokemuksiin ja voi jättää paljon tulkinnan varaan. Tällöin voi olla, että tutkittavat jättävät vastaukselle olennaista kertomatta, joka voi vaikuttaa vastausten todellisuuteen ja tulkittavuuteen, jonka lisäksi koehenkilö voi hyvinkin unohtaa monia ongelmakohtia, jotka olisivat kokonaisuuden kannalta mainitsemisen arvoisia. Koronatilanteen ja tutkimuksen perusjoukon suppeuden vuoksi koehenkilöiden tarkempi observointi oli kuitenkin mahdotonta. Vastauksista olisi saatu todennäköisesti laajempi kirjo observoimalla tutkittavia tai päiväkirjan pitämisellä kohdatuista ongelmista. Toisaalta vastauksien perusteella voitiin kategorisoida hyvinkin yleisen tiedon perusteella pääteltäviä ongelmia, joten vastaukset antavat ainakin suuntaviivat eri ongelmien kategorioimiselle.

Tutkimukseen saatiin mukaan näkökyvyiltään hyvin monimuotoinen joukko. Tyypillisesti monimuotoisuus on hyvä ja auttaa tutkimusten tulosten yleisteettävyyttä. Kuitenkin tässä tutkimuksessa pienen osallistujamäärän (N = 8) vuoksi se heikentää yleistettävyyttä. Pienellä osallistujamäärällä eri heikkonäköisyyden tietynlaisten esiintymien maksimimäärä oli kaksi, josta tulosta ei voida vielä yleistää suurempaan joukkoon. Toisaalta laadullisen tiedon kannalta eri esiintymien määrä on hyvä ja varsinkin tämän tutkimuksen tavoitteisiin, eli jatkokysymysten kartuttamiseen, erittäin soveltuva. Suuri määrä eri heikkonäköisyyden esiintymiä antaa parhaan mahdollisuuden saada tarpeeksi kattava kirjo ongelmista, joita heikkonäköiset kohtaavat. Jos heikkonäköisyyksien kirjo olisi suppea, niin laadullisen osion tulokset olisivat vain yhdistettävissä tiettyyn perusjoukkoon. Toisaalta suppeampi kirjo voisi vaikuttaa positiivisesti kokonaisuudessaan varsinkin empiirisen osion tulosten yleistettävyyteen.

Tutkimukseen osallistuvien ohjelmoijien ohjelmointikokemuksen määrä oli myös kirjava. Toisaalta tutkimukseen voidaan sanoa osallistuneen hyvin laaja joukko eri kokemustaustaisia heikkonäköisiä ohjelmoijia, mutta toisaalta puolet vastaajista oli hyvin kokeneita, kun taas toinen puoli jakautui kokenei-

den ja aloittelijoiden kesken. Tutkimuksen yleistettävyys kärsii vastaavasti tässäkin siitä, että tuloksia ei voida yleistää tiettyyn perusjoukkoon. Toisaalta hyötynä on taas se, että laadullisen osion vastauksia voitiin arvioida myös ohjelmointikokemuksen mukaan.

Tutkimuksen reliabiliteettiin ja validiteettiin vaikuttaa myös tutkimukseen itse toteutettu työkalu. Tutkimusta varten toteutettiin työkalu, jotta eri tutkimustyökalujen rajoitteet eivät vaikuttaisi heikkonäköisen tutkimiseen. Eri työkalut eivät tarjonneet mahdollisuutta esittää ohjelmakoodia samalla tavalla kuin tutkimukseen luodussa työkalussa, jonka lisäksi ohjelmakoodin sekä valmiin työkalun saavutettavuus olisi kärsinyt vaihtoehtoratkaisujen vuoksi. Valmiin ratkaisun käyttäminen olisi tarkoittanut sitä, että ohjelmakoodia ei olisi voitu esittää täysin halutuilla tyyleillä tai ohjelmakoodin olisi esitetty kuvina, jolloin ohjelmakoodi voisi olla saavuttamaton sen mukautuvuuden puutteen vuoksi. Tutkimukseen toteutettua työkalua voidaankin pitää tutkimuksessa vahvuutena, koska sen toteuttaminen mahdollisti tutkimuksen toteuttamisen ja lisäsi tutkimuksen tarkkuutta, koska se poisti saavutettavuusongelmia.

Työkalulla mitattujen aikojen validiteettia yritettiin parantaa ruudunjakamisen kautta. Ruudunjakamisella varmistettiin, että tehtävien suoritusajat ovat todelliset. Puheyhteyden välityksellä tutkittaville pystyttiin myös antamaan lisää ohjeistusta, joka osaltaan paransi tutkimuksen etenemistä. Tutkimus olisi erittäin mielekästä suorittaa laboratorioasetelmassa, jolloin heikkonäköisien koehenkilöiden strategioita lukea tekstiä voitaisiin tutkia paremmin. Koronatilanteen vuoksi tällainen ei ollut kuitenkaan tässä tutkimuksessa mahdollista.

8.2 Tutkimuksen merkitys

Tämä tutkimus toimii ponnahduslautana uudelle tutkimuksen suunnalle. Omaan tietooni ei ole vielä olemassa tutkimusta, joka olisi yhdistänyt heikkonäköisten ohjelmoinnin ja sen esitystavan parantamisen. Tämän lisäksi en tiedä tutkimuksesta, jossa olisi kartoitettu heikkonäköisten kohtaamia ongelmia ohjelmoinnissa.

Tutkimuksen empiirisen osion perusteella ei voida tehdä vielä tarkempia ehdotuksia. Tässä tutkimuksessa saadun datan perusteella näyttäisi siltä, että normaali, merkkimäärän kohdalta katkaistu ja automaattisesti rivitetty ohjelmakoodin esitystavat eivät eroa lukemisen suhteen merkitsevästi. Kuten tutkimuksessa huomattiin, niin erityisesti automaattisesti rivitetyn ohjelmakoodin suoritusajat erosivat merkitsevästi ensimmäisen ja toisen altistumisen välillä. Olisikin tärkeä tutkia, voisiko useampi altistumiskerta vaikuttaa tulokseen.

Tutkimuksen tuloksena luotiin myös kategorioita ongelmille, joita heikkonäköiset ohjelmoijat kohtaavat. Mielestäni tutkimuksen tärkein tulos ovat nimenomaan näiden kategorioiden tunnistaminen. Itsessään kategoriat antavat

jo ideoita jatkotutkimukselle ja ajatuksia avattiin jo aiemmin pohdintaosiossa. Kategoriat paljastavat, että heikkonäköiset kohtaavat ongelmia suoraan ohjelmointiin liittyen, mutta myös sen ulkopuolella ohjelmointiin liittyvässä elämässä, kuten työssä ja opiskelussa.

Jatkossa tutkimusta pitäisi tehdä nimenomaan siitä, millaisia todelliset ongelmat heikkonäköisten ohjelmoijien keskuudessa ovat, jotta ne voitaisiin ratkaista oikealla tavalla. Ylipäätään tutkimustietoa heikkonäköisistä ohjelmoijista on erittäin vähän, joten tutkimustiedon kartuttaminen olisi ylipäätään kriittistä koko aihealueen tutkimiselle. Tässäkin tutkimuksessa huomattiin, että vaikka tietoa yritettiin hakea ja yhdistää digitaalisen lukemisen ja kognitiivisen kuormituksen kautta, niin suuri osa tiedosta ei kuitenkaan ole tutkittu heikkonäköisillä.

9 YHTEENVETO

Tutkimuksessa tavoitteena oli selvittää, että voidaanko digitaalisen tekstin lukemisen helpottamisen menetelmiä soveltaa ohjelmointikielten esittämiseen heikkonäköisille ohjelmoijille. Tutkimuksen toisena tavoitteena oli kartoittaa millaisia ongelmia heikkonäköiset ohjelmoijat kohtaavat. Tutkimusalue on täysin uusi, joten tutkimuksessa jouduttiin yhdistämään tietoa eri alueilta, jotta voitiin luoda kattava kuva kognitiivisen kuormituksen vaikutuksesta ohjelmointiin ja sen lukemiseen, sekä ymmärtää miten heikkonäköisyys vaikuttaa teknologian käyttöön, digitaalisen tekstin lukemiseen ja ohjelmointiin. Lisäksi tutkimuksessa todettiin, että heikkonäköisyyteen liittyvä tutkimus on vielä melko kehnolla tasolla etenkin liittyen ohjelmointiin tai kognitiiviseen kuormitukseen.

Tutkimuksessa verrattiin ohjelmakoodin lukemista tavallisesti rivitetyllä ja muotoillulla esitystavalla, merkkimäärän kohdalta katkaistua ohjelmakoodin ruutuun sovittamisen menetelmällä, sekä tutkimusta varten luotiin kolmas tyyli, joka mukailee verkkosivuilla usein käytettyä automaattista tekstin rivittämistä ja pyrki rivittämään ohjelmakoodia luettavampaan muotoon. Tutkittavat suorittivat kuusi ohjelmakoodin lukemistehtävää, joissa ohjelmakoodi täytyi lukea kokonaisuudessaan ja lukemisen jälkeen vastata mitä ohjelmakoodi tulostaa. Jokaisen esitystavan jälkeen tutkittavat täyttivät SUS-lomakkeen esitystapaan liittyen. Esitystavat ja ohjelmakoodit oli satunnaistettu ja jokainen esitystyyli esitettiin kaksi kertaa. Eri esitystyylien välillä mitattiin ohjelmakoodin lukemiseen kulunut aika, vastauksen oikeellisuus ja käyttäjiltä kysyttiin subjektiivinen kokemus esitystyylistä. Tutkimuksessa ei havaittu eroa eri esitystyylien välillä, joka voi tarkoittaa sitä, että tässä tutkimuksessa esitetyillä esitystyyleillä on muita tekijöitä pienempi vaikutus ohjelmakoodin lukemiseen tai automaattinen rivitys voi olla helpompi lukea, mutta efektiä ei voida selvästi tulkita tämän tutkimuksen otannan ja altistumiskertojen perusteella. Datasta havaittiin merkitsevä ero ensimmäisen ja toisen altistumiskerran välillä tutkimukseen luodun automaattisen tekstin rivittämisen esitystavalla. Normaalissa esitystavassa ja merkkimäärän kohdalta katkaistussa tavassa vastaavaa eroa ei löydy. Ohjelmakoodi on rakenteellista formaalia kieltä ja kuten Luvussa 5.1 on esitelty, niin

ohjelmointi on kognitiivisesti raskas tehtävä, joten monet suoraan lukemiseen liittymättömätkin tekijät voivat vaikuttaa lukemisen kuormitukseen. Asiasta ei voi tehdä vielä tarkempia päätelmiä, vaan tarvittaisiin useampia altistumiskertoja joko valmistavasti ennen tutkimusta, jolloin altistumisen uutuus ei vaikuttaisi tuloksiin tai tutkimuksen aikana useampia tehtäviä samalla esitystyylillä, jolloin toistuvien altistumisten vaikutus voitaisiin nähdä selvemmin tuloksissa.

Tutkimuksen toisena tavoitteena oli kartoittaa ongelmia, joita heikkonäköiset kohtaavat ohjelmointiin liittyen. Tutkittavilta kysyttiin tietoa kohtaamistaan ongelmista, sekä tutkittavia pyydettiin kuvailemaan heikkonäköisyyttään. Tutkimuksen koehenkilöiden vastausten perusteella luotiin viisi ongelmakategoriaa: haetun tiedon lukeminen, ohjelmakoodin lukeminen, ohjelmointityökalujen käyttäminen, ohjelmakoodin tulosten tulkinta ja sosiaalinen vaikutus. Erityisesti mainittavaa kategorioista on se, että saavutettavuuden puute esiintyy useasti jokaisessa kategoriassa, jossa se on relevanttia. Ohjelmointiin liittyvien työkalujen, informaation ja verkkosivujen saavutettavuus koettiin pääasiassa huonoksi ja osassa tapauksista saavutettavuuden puute sulki heikkonäköisen henkilön täysin ulos tietyistä ohjelmoinnin osatehtävistä. Ongelmakategoriat kuvastavat hyvin sitä, että heikkonäköisyys vaikuttaa esimerkiksi ohjelmointityössä monella tasolla työn tekemisestä työyhteisössä toimimiseen. Tämä tutkimus on kuitenkin vain yksittäinen otos ongelmista, vaikkakin ongelmakategoriat pystyttiin liittämään jo olemassa olevaan tietoon. Ongelma-aluetta ei ymmärretä vielä täysin ja lisää tutkimusta tarvitaan ongelmien ja niiden syntyperien selvittämiseksi ja mahdollisten ratkaisujen luomiseksi. Tässä tutkimuksessa luotuja kategorioita voi käyttää lähtökohtana aihealueen tarkemmalle selvittämiselle ja paremman kokonaiskuvan muodostamiselle.

Tutkimuksen luotettavuuteen vaikuttavat monet seikat. Ensimmäisenä vaikuttavana tekijänä on pieni osallistujamäärä (N = 8), joka on tyypillistä heikkonäköisyyteen liittyville tutkimuksille. Oma kokemukseni erityisesti heikkonäköisten ihmisten tutkimisesta varsinkin uudella aihealueella on vähäinen. Kokemuksen puute voi vaikuttaa tutkimuksen toteutukseen ja etenkin ongelmien kategorioinnissa käytin apunani aihealueen ymmärrystäni ja kokemustani aihealueesta, joka voi vaikuttaa kategorioinnin laatuun. Luodut kategoriat ovat kuitenkin enemmän tutkittavien vastauksista esille nousevia ja ne pystytään liittämään olemassa olevaan tietoon. Tutkimuksen toteutuksessa työkalua, kysymyksiä ja metodia mietittiin tarkoin ja testattiin usein. Kuitenkin tutkimuksen palautteessa nousi ilmi, että tutkittavat eivät juuri käytä tutkimuksessa käytettyä merkkimäärän kohdalta rajattua esitystapaa ja tutkimukseen luodun esitystavan uutuus koettiin vaikuttavana tekijänä suoriutumiseen tehtävissä. Tarkempi esiselvitys, etenkin tutkimustiedon puutteen vuoksi, olisi voinut ohjata paremmin tutkimuksen toteutusta. Minulla ei ole myöskään kokemusta luettavuudeltaan toisiaan vastaavien ohjelmakoodien luomisesta, mutta ohjelmakoodien luomiseen käytettiin apuna tutkimustietoa ohjelmakoodien vaativuuteen vaikuttavista tekijöistä ja ohjelmakoodien lukemiseen kuluvaa aikaa testattiin useaan otteeseen normaalisti näkevien ohjelmoijien kesken. Lisäksi tutkimuksessa todettiin empiirisesti, että ohjelmakoodeilla ei ollut vaikutusta tuloksiin.

Tutkimuksen tulosten yleistettävyyteen vaikuttaa hyvin eriävä kokemustaso ohjelmoinnista, sekä heikkonäköisyyksien esiintymien laaja kirjo, mutta myös pieni otanta ja aihealueen kontekstisidonnaisuus.

Tämän tutkimuksen pohjalta voidaan todeta, että aihealueesta tarvitaan lisää tietoa. Heikkonäköisyyttä ja ohjelmointia yhdistävää tutkimusta on erittäin vähän ja aihealuetta koskevan tiedon määrän kasvattaminen olisikin erityisen tarpeellista. Yleisestikin heikkonäköisyyttä tutkivaa tietoa on verrattain vähän. Tutkimus on kuitenkin vaikeaa, koska usein heikkonäköisiä koskeviin tutkimuksiin saadaan heikosti osallistujia ja aihealueen tutkimusta varjostaa vielä ohjelmointiosaamisen tarpeellisuus. Ohjelmointi saattaa haastavuutensa takia sulkea pois potentiaalisia heikkonäköisiä osaajia, koska ohjelmoinnin kognitiiviset vaatimukset ovat korkeat ja heikkonäköisyys saattaa toimia niitä korottavana tekijänä ja vaikuttaa siten myös mahdollisten osallistujien määrään pidemmälläkin aikavälillä.

Tutkimuksen perusteella ei voida vielä todeta toimiiko tekstin automaattisen rivittämisen menetelmä ohjelmakoodin lukemiseen heikkonäköiselle ohjelmoijalle. Olisikin tärkeä tutkia mitkä tekijät erityisesti vaikuttavat ohjelmakoodin lukemiseen, koska ohjelmoinnin kognitiivinen vaativuus on monen tekijän summa. Vaikka heikkonäköisillä digitaalisen tekstin lukemiseen vaikuttavat tekijät liittyvät usein tekstin esittämiseen ja tekstin visuaalisiin ominaisuuksiin, niin se ei välttämättä tarkoita, että tekstipohjaisen ohjelmakoodin lukemiseen vaikuttavat samat tekijät. Jatkossa tutkimus voisi keskittyä selvittämään millä tavalla digitaalisen tekstin ja ohjelmakoodin lukeminen eroaa toisistaan, mitkä tekijät vaikuttavat ohjelmakoodin lukemiseen ja miten heikkonäköisyys vaikuttaa ohjelmakoodin lukemiseen. Tämän lisäksi olisi myös olennaista tutkia mitkä tekijät nykyisissä työkaluissa helpottavat ohjelmakoodin lukemista ja miten ne jo helpottavat heikkonäköisen lukemista tai miten niitä voitaisiin soveltaa tai parantaa heikkonäköisten lukemisen tueksi.

Tutkimukseen luotu kategoriointi voi toimia alustavana tekijänä jatkotutkimukselle. Pienen otannan ja kysymyksenasettelun vuoksi tutkittavien vastaukset voivat antaa suppean kuvan aihealueesta. Jatkossa tutkimus voisi selvittää laajemmin millaisia todellisia ongelmia heikkonäköiset ohjelmoijat kohtaavat ohjelmointiin liittyen, joka antaisi taas mahdollisuuden pohtia uusia tai soveltaa olemassa olevia ratkaisuja ongelmien helpottamiseen ja tutkimiseen.

LÄHTEET

Abad, Z. S. H., Karras, O., Schneider, K., Barker, K., & Bauer, M. (2018)Task interruption in software development projects: What makes some interruptions more disruptive than others?. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*, 122—132.

Adams, C., Campbell, A., Montgomery, R., Cooper, M., & Kirkpatrick, A. (2022). *Web Content Accessibility Guidelines (WCAG) 2.2*. Haettu 1.3.2022 sivulta https://www.w3.org/TR/WCAG22/.

Alisaari, J., Turunen, T., Kajamies, A., Korpela, M., & Hurme, T. R. (2018). Reading comprehension in digital and printed texts. *L1 Educational Studies in Language and Literature, 18*.

Aluehallintoviorasto (n.d.) *Yleistä saavutettavuudesta*. Haettu 1.3.2022 sivulta https://www.saavutettavuusvaatimukset.fi/yleista-saavutettavuudesta/.

Amadieu, F., Mariné, C., & Laimay, C. (2011). The attention-guiding effect and cognitive load in the comprehension of animations. *Computers in Human Behavior, 27*(1), 36—40.

Amerikan rehabilitaatiolaki (1973). *Vuoden 1973 Amerikan rehabilitaatiolaki*. https://www.eeoc.gov/statutes/rehabilitation-act-1973.

Arditi, A., & Cho, J. (2007). Letter case and text legibility in normal and low vision. *Vision research, 47*(19), 2499—2505.

Atilgan, N., Xiong, Y. Z., & Legge, G. E. (2020). Reconciling print-size and display-size constraints on reading. *Proceedings of the National Academy of Sciences, 117*(48), 30276—30284.

Atkinson, R. C., & Shiffrin, R. M. (1968). Chapter: Human memory: A proposed system and its control processes. In Spence, K. W., & Spence, J. T. *The psychology of learning and motivation 2*, 89-195. Academic Press.

Baddeley, A. D., & Hitch, G. (1974). Working memory. In *Psychology of learning and motivation 8*, 47—89. Academic press.

Bassey, E., & Ellison, C. (2022). Psychological changes among working-age adults with acquired vision impairment: The need for psychological intervention?. *British Journal of Visual Impairment, 40*(1), 61—74.

Berssanette, J. H., & de Francisco, A. C. (2021). Cognitive Load Theory in the Context of Teaching and Learning Computer Programming: A Systematic Literature Review. *IEEE Transactions on Education*.

Binkley, D., Davis, M., Lawrie, D., Maletic, J. I., Morrell, C., & Sharif, B. (2013). The impact of identifier style on effort and comprehension. *Empirical software engineering, 18*(2), 219—276.

Boshernitsan, M., & Downes, M. S. (2004). Visual programming languages: A survey. Computer Science Division, University of California.

Bouck, E. C. (2016). A national snapshot of assistive technology for students with disabilities. *Journal of Special Education Technology, 31*(1), 4—13.

Boumenir, Y., Kadri, A., Suire, N., Mury, C., & Klinger, E. (2014). Impact of simulated low vision on perception and action. *International Journal of Child Health and Human Development (IJCHD), 7*(4), 441—450.

Bourne, R., Steinmetz, J. D., Flaxman, S., Briant, P. S., Taylor, H. R., Resnikoff, S., ... & Tareque, M. I. (2021). Trends in prevalence of blindness and distance and near vision impairment over 30 years: an analysis for the Global Burden of Disease Study. *The Lancet global health, 9*(2), e130—e143.

Brachten, F., Brünker, F., Frick, N. R., Ross, B., & Stieglitz, S. (2020). On the ability of virtual agents to decrease cognitive load: an experimental study. *Information Systems and e-Business Management, 18*(2), 187—207.

Brooke, J. (1996). SUS-A quick and dirty usability scale. *Usability evaluation in industry, 189*(194), 4-7.

Brooke, J. (2013). SUS: a retrospective. *Journal of usability studies, 8*(2), 29-40.

Brooks, R. (1983). Towards a theory of the comprehension of computer programs. *International journal of man-machine studies, 18*(6), 543—554.

Buchner, J., Buntins, K., & Kerres, M. (2022). The impact of augmented reality on cognitive load and performance: A systematic review. *Journal of Computer Assisted Learning, 38*(1), 285—303.

Buetti, S., & Leib, S. (2018). Evaluation of visual sensitivity across the visual field under varying levels of cognitive load. *Journal of Vision, 18*(10), 479—479.

Caldwell, B., Cooper, M., Reid, L. G., & Vanderheiden, G. (2008). *Web Content Accessibility Guidelines (WCAG) 2.0*. Haettu 1.3.2022 sivulta https://www.w3.org/TR/WCAG20/.

Chan, M. L. (2017). *Web-Based Usability Evaluation of Text-Resizing Methods and Users' Visual Fatigue on Online Reading Tasks*. California State University.

Chandler, P., & Sweller, J. (1992). The split-attention effect as a factor in the design of instruction. *British Journal of Educational Psychology, 62*(2), 233—246.

Chandler, P., & Sweller, J. (1996). Cognitive load while learning to use a computer program. *Applied cognitive psychology, 10*(2), 151—170.

Chen, C., Haduong, P., Brennan, K., Sonnert, G., & Sadler, P. (2019). The effects of first programming language on college students' computing attitude and achievement: a comparison of graphical and textual languages. *Computer Science Education, 29*(1), 23—48.

Chen, C. M., & Lin, Y. J. (2016). Effects of different text display types on reading comprehension, sustained attention and cognitive load in mobile reading contexts. *Interactive Learning Environments, 24*(3), 553—571.

Chisholm, W., Vanderheiden, G., & Jacobs, I. (1999). Web Content Accessibility Guidelines 1.0. Haettu 1.3.2022 sivulta https://www.w3.org/TR/WAI-WEBCONTENT/.

Cho, K. W., Altarriba, J., & Popiel, M. (2015). Mental juggling: when does multitasking impair reading comprehension?. *The Journal of general psychology, 142*(2), 90—105.

Christen, M., & Abegg, M. (2017). The effect of magnification and contrast on reading performance in different types of simulated low vision. *Journal of Eye Movement Research, 10*(2).

Cooper, G., & Sweller, J. (1987). Effects of schema acquisition and rule automation on mathematical problem-solving transfer. *Journal of educational psychology, 79*(4), 347.

Corn, A. L., Wall, R. S., Jose, R. T., Bell, J. K., Wilcox, K., & Perez, A. (2002). An initial study of reading and comprehension rates for students who received optical devices. *Journal of Visual Impairment & Blindness, 96*(5), 322—334.

Cowan, N. (1988). Evolving conceptions of memory storage, selective attention, and their mutual constraints within the human information-processing system. *Psychological bulletin, 104*(2), 163.

Creem-Regehr, S. H., Barhorst-Cates, E. M., Tarampi, M. R., Rand, K. M., & Legge, G. E. (2021). How can basic research on spatial cognition enhance the visual accessibility of architecture for people with low vision?. *Cognitive Research: Principles and Implications, 6*(1), 1—18.

DeStefano, D., & LeFevre, J. A. (2007). Cognitive load in hypertext reading: A review. *Computers in human behavior, 23*(3), 1616—1641.

Dick, W. E. (2017). Usability of Enlargement Methods: How Enlargement Method Influences the Amount of Scrolling Actions Needed to Read Publications. In *International Conference on Applied Human Factors and Ergonomics*, 271—281. Springer.

Dillon, B., & Thompson, R. (2016, May). Software development and tool usability. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, 1—4. IEEE.

Direktiivi 2016/2102. *Julkisen sektorin elinten verkkosivustojen ja mobiilisovellusten saavutettavuudesta*. Euroopan parlamentti ja neuvosto. http://data.europa.eu/eli/dir/2016/2102/oj

EN 301 549 V3.2.1 (2021-03). https://www.etsi.org/deliver/etsi_en/301500_301599/301549/03.02.01_60/en_301549v030201p.pdf.

Engström, J., Johansson, E., & Östlund, J. (2005). Effects of visual and cognitive load in real and simulated motorway driving. *Transportation research part F: traffic psychology and behaviour, 8*(2), 97—120.

Engström, J., Markkula, G., Victor, T., & Merat, N. (2017). Effects of cognitive load on driving performance: The cognitive control hypothesis. *Human factors, 59*(5), 734—764.

Fakhoury, S., Ma, Y., Arnaoudova, V., & Adesope, O. (2018, May). The effect of poor source code lexicon and readability on developers' cognitive load. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, 286—28610. IEEE.

Fedorenko, E., Ivanova, A., Dhamala, R., & Bers, M. U. (2019). The language of programming: a cognitive perspective. *Trends in cognitive sciences, 23*(7), 525—528.

Flaxman, A. D., Wittenborn, J. S., Robalik, T., Gulia, R., Gerzoff, R. B., Lundeen, E. A., ... & Vision and Eye Health Surveillance System study group. (2021). Prevalence of visual acuity loss or blindness in the US: a Bayesian meta-analysis. *JAMA ophthalmology, 139*(7), 717—723.

Giudice, N. A., Marston, J. R., Klatzky, R. L., Loomis, J. M., & Golledge, R. G. (2008). Environmental learning without vision: Effects of cognitive load on interface design. In *9th international conference on low vision*.

Gonçales, L., Farias, K., da Silva, B., & Fessler, J. (2019, May). Measuring the cognitive load of software developers: A systematic mapping study. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, 42—52. IEEE.

Guéhéneuc, Y. G. (2009). A theory of program comprehension: Joining vision science and program comprehension. *International Journal of Software Science and Computational Intelligence (IJSSCI), 1*(2), 54—72.

Hallett, E. C., Arnsdorff, B., Sweet, J., Roberts, Z., Dick, W., Jewett, T., & Vu, K. P. L. (2015). The usability of magnification methods: A comparative study between screen magnifiers and responsive web design. In *International Conference on Human Interface and the Management of Information*, 181—189. Springer.

Hallett, E. C., Dick, W., Jewett, T., & Vu, K. P. L. (2017). How screen magnification with and without word-wrapping affects the user experience of adults with low vision. In *International Conference on Applied Human Factors and Ergonomics*, 665—674. Springer.

Hallett, E. C., Roberts, Z., Sweet, J., Chan, M. L., Sun, Y., Dick, W., ... & Vu, K. P. L. (2015). Computer accessibility: How individuals with low vision adjust

the presentation of electronic text for academic reading. *Procedia Manufacturing, 3*, 5206—5213.

Hannebauer, C., Hesenius, M., & Gruhn, V. (2018). Does syntax highlighting help programming novices?. *Empirical Software Engineering, 23*(5), 279—2828.

Harrison, C. M. (2004). Low-vision reading aids: reading as a pleasurable experience. *Personal and Ubiquitous Computing, 8*(3), 213-220.

Head, J., & Helton, W. S. (2014). Sustained attention failures are primarily due to sustained cognitive load not task monotony. *Acta psychologica, 153*, 87—94.

Helgesson, D., Engström, E., Runeson, P., & Bjarnason, E. (2019, May). Cognitive load drivers in large scale software development. In *2019 IEEE/ACM 12th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 91—94. IEEE.

Hillesund, T. (2010). *Digital reading spaces: How expert readers handle books, the Web and electronic paper*.

Holden, B. A., Fricke, T. R., Wilson, D. A., Jong, M., Naidoo, K. S., Sankaridurg, P., ... & Resnikoff, S. (2016). Global prevalence of myopia and high myopia and temporal trends from 2000 through 2050. *Ophthalmology, 123*(5), 1036—1042.

Huang, F. (2016, May). Post-completion error in software development. In *2016 IEEE/ACM Cooperative and Human Aspects of Software Engineering (CHASE)*, 108—113. IEEE.

Hunter, C. R., & Pisoni, D. B. (2018). Extrinsic cognitive load impairs spoken word recognition in high-and low-predictability sentences. *Ear and Hearing, 39*(2), 378.

Ivanova, A. A., Srikant, S., Sueoka, Y., Kean, H. H., Dhamala, R., O'reilly, U. M., ... & Fedorenko, E. (2020). *Comprehension of computer code relies primarily on domain-general executive brain regions*. Elife.

Jani, R., & Schrepp, M. (2005). Are the Web Accessibility Guidelines applicable and sufficient for Web Applications. *Assistive Technologies Research Series, 16*, 499—503.

Jeong, H. (2012). A comparison of the influence of electronic books and paper books on reading comprehension, eye fatigue, and perception. *The Electronic Library, 30*(3), 390—408.

Kang, Y. Y., Wang, M. J. J., & Lin, R. (2009). Usability evaluation of e-books. *Displays, 30*(2), 49—52.

Kasto, N., & Whalley, J. (2013). Measuring the difficulty of code comprehension tasks using software metrics. In *Proceedings of the Fifteenth Australasian Computing Education Conference, 136*, 59—65.

Kazazoğlu, S. (2020). Is printed-text the best choice? A mixed-method case study on reading comprehension. *Journal of Language and Linguistic Studies, 16*(1), 458—473.

KELA (2022). *IT-työssä Kelassa*. Haettu 1.3.2022 osoitteesta https://www.kela.fi/uramahdollisuuksia-it-tyo.

Kelly, T., & Buckley, J. (2009). Cognitive levels and Software Maintenance Sub-tasks. In *PPIG*, 9.

Kirkpatrick, A., O Connor, J., Campbell, A., & Cooper, M. (2018) *Web content accessibility Guidelines (WCAG) 2.1*. Haettu 1.3.2022 sivulta https://www.w3.org/TR/WCAG21/.

Klemola, T., & Rilling, J. (2002, August). Modeling comprehension processes in software development. *In Proceedings First IEEE International Conference on Cognitive Informatics,* 329—336. IEEE.

Lang, F., & Machulla, T. (2021). Pressing a Button You Cannot See: Evaluating Visual Designs to Assist Persons with Low Vision through Augmented Reality. In *Proceedings of the 27th ACM Symposium on Virtual Reality Software and Technology*, 1—10.

Laki digitaalisten palvelujen tarjoamisesta (306/2019). https://www.finlex.fi/fi/laki/alkup/2019/20190306.

Larson, L. C. (2015). E-books and audiobooks: Extending the digital reading experience. *The Reading Teacher, 69*(2), 169—177.

Lawrie, D., Morrell, C., Feild, H., & Binkley, D. (2006, June). What's in a name? a study of identifiers. *IEEE International Conference on Program Comprehension*, 3—12.

Lee, Y. C., Lee, J. D., & Ng Boyle, L. (2007). Visual attention in driving: The effects of cognitive load and visual disruption. *Human Factors, 49*(4), 721—733.

Lee, Y. C., Lee, J. D., & Ng Boyle, L. (2009). The interaction of cognitive load and attention-directing cues in driving. *Human factors, 51*(3), 271—280.

Legge, G. E. (2016). Reading digital with low vision. *Visible language, 50*(2), 102.

Lewis, B. P., & Linder, D. E. (1997). Thinking about choking? Attentional processes and paradoxical performance. *Personality and social psychology bulletin, 23*(9), 937—944.

Lorenzini, M. C., & Wittich, W. (2020). Factors related to the use of magnifying low vision aids: a scoping review. *Disability and Rehabilitation, 42*(24), 3525—3537.

Luxton-Reilly, A., & Petersen, A. (2017, January). The compound nature of novice programming assessments. In *Proceedings of the Nineteenth Australasian Computing Education Conference*, 26—35.

Maailman terveysjärjestö (2011). *World report on disability*. World Health Organization.

Maailman terveysjärjestö (2019). *World report on vision*. World Health Organization.

Mattys, S. L., Barden, K., & Samuel, A. G. (2014). Extrinsic cognitive load impairs low-level speech perception. *Psychonomic bulletin & review, 21*(3), 748—754.

Miller, G. A. (1956). The magical number seven, plus or minus two: Some limits on our capacity for processing information. *Psychological review, 63*(2), 81.

Moreno, L., Valencia, X., Pérez, J. E., & Arrue, M. (2021). An exploratory study of web adaptation techniques for people with low vision. *Universal access in the information society, 20*(2), 223—237.

Mousavi, S. Y., Low, R., & Sweller, J. (1995). Reducing cognitive load by mixing auditory and visual presentation modes. *Journal of educational psychology, 87*(2), 319.

Murphy, L., Lewandowski, G., McCauley, R., Simon, B., Thomas, L., & Zander, C. (2008). Debugging: the good, the bad, and the quirky--a qualitative analysis of novices' strategies. *ACM SIGCSE Bulletin, 40*(1), 163—167.

Myers, B. A., Ko, A. J., LaToza, T. D., & Yoon, Y. (2016). Programmers are users too: Human-centered methods for improving programming tools. *Computer, 49*(7), 44—52.

Nyman, S. R., Gosney, M. A., & Victor, C. R. (2010). Psychosocial impact of visual impairment in working-age adults. *British Journal of Ophthalmology, 94*(11), 1427—1431.

Näkövammaisten Liitto. (n.d.). *Näkövammaisuus*. Haettu 1.3.2022 verkkosivulta https://www.nkl.fi/fi/nakovammaisuus.

Näkövammaisten Liitto. (n.d.). *Miten nä-kö-vam-mai-nen käyttää tie-to-ko-net-ta tai mo-bii-li-lai-tet-ta?*. Haettu 1.3.2022 verkkosivulta https://www.nkl.fi/fi/miten-nakovammainen-kayttaa-tietokonetta-tai-mobiililaitetta.

Ojamo, M., & Tolkkinen, L. (2021). *Näkövammarekisterin vuosikirja 2020*. Jyväskylä.

Oman, P. W., & Cook, C. R. (1990). Typographic style is more than cosmetic. *Communications of the ACM, 33*(5), 506—520.

Paas, F. G. (1992). Training strategies for attaining transfer of problem-solving skill in statistics: a cognitive-load approach. *Journal of educational psychology, 84*(4), 429.

Pardede, P. (2019). Print vs Digital Reading Comprehension in EFL. *Journal of English Teaching, 5*(2), 77—90.

Pascolini, D., & Mariotti, S. P. (2012). Global estimates of visual impairment: 2010. *British Journal of Ophthalmology, 96*(5), 614—618.

Peitek, N., Apel, S., Parnin, C., Brechmann, A., & Siegmund, J. (2021, May). Program comprehension and code complexity metrics: An fmri study. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 524—536. IEEE.

Pennington, N., & Grabowski, B. (1990). The tasks of programming. In *Psychology of programming*, 45—62. Academic Press.

Petrie, H., & Kheir, O. (2007). The relationship between accessibility and usability of websites. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, 397—406.

Prat, C. S., Madhyastha, T. M., Mottarella, M. J., & Kuo, C. H. (2020). Relating natural language aptitude to individual differences in learning programming languages. *Scientific reports, 10*(1), 1—10.

Rand, K. M., Barhorst-Cates, E. M., Kiris, E., Thompson, W. B., & Creem-Regehr, S. H. (2019). Going the distance and beyond: simulated low vision increases perception of distance traveled during locomotion. *Psychological research, 83*(7), 1349—1362.

Ray, B., Posnett, D., Filkov, V., & Devanbu, P. (2014, November). A large scale study of programming languages and code quality in github. In *Proceedings of the 22nd ACM SIGSOFT international symposium on foundations of software engineering*, 155—165.

Rodrigues, A., Machado, M. B., Almeida, A. M. P., Abreu, J. F. D., & Tavares, T. A. (2022). Interaction Devices as Assistive Technology: Current Practices about Evaluation Methodologies. *International Journal of Human–Computer Interaction, 38*(3), 201—212.

Roehm, T., Tiarks, R., Koschke, R., & Maalej, W. (2012, June). How do professional developers comprehend software?. In *2012 34th International Conference on Software Engineering (ICSE)*, 255—265. IEEE.

Sass, S. M., Legge, G. E., & Lee, H. W. (2006). Low-vision reading speed: Influences of linguistic inference and aging. *Optometry and Vision Science, 83*(3), 166—177.

Sauro, J. (2011). *A practical guide to the system usability scale: Background, benchmarks & best practices*.

Sauvan, L., Stolowy, N., Aguilar, C., François, T., Gala, N., Matonti, F., ... & Calabrese, A. (2020, May). Text simplification to help individuals with low vision read more fluently. In *Proceedings of the 1st Workshop on Tools and Resources to Empower People with REAding DIfficulties (READI)*, 27—32.

Sheppard, A. L., & Wolffsohn, J. S. (2018). Digital eye strain: prevalence, measurement and amelioration. *BMJ open ophthalmology, 3*(1), e000146.

Sillito, J., Murphy, G. C., & De Volder, K. (2006). Questions programmers ask during software evolution tasks. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*, 23—34.

Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive science, 12*(2), 257—285.

Sweller, J., Ayres, P., & Kalyuga, S. (2011). *Cognitive load theory*. New York: Springer.

Sweller, J., Ayres, P. L., Kalyuga, S., & Chandler, P. (2003). *The expertise reversal effect*.

Szpiro, S. F. A., Hashash, S., Zhao, Y., & Azenkot, S. (2016). How people with low vision access computing devices: Understanding challenges and opportunities. In *Proceedings of the 18th International ACM SIGACCESS Conference on Computers and Accessibility*, 171—180.

Sörqvist, P., Dahlström, Ö., Karlsson, T., & Rönnberg, J. (2016). Concentration: The neural underpinnings of how cognitive load shields against distraction. *Frontiers in human neuroscience, 10*, 221.

Tanner, M. J. (2014). Digital vs. print: Reading comprehension and the future of the book. *School of Information Student Research Journal, 4*(2), 6.

Thomas, R., Barker, L., Rubin, G., & Dahlmann-Noor, A. (2015). Assistive technology for children and young people with low vision. *Cochrane database of systematic reviews*, (6).

Tullis, T. S., & Stetson, J. N. (2004). A comparison of questionnaires for assessing website usability. In *Usability Professionals Association (UPA) 2004 Conference*.

Thüring, M., Hannemann, J., & Haake, J. M. (1995). Hypermedia and cognition: Designing for comprehension. *Communications of the ACM, 38*(8), 57—66.

Van Merrienboer, J. J., Schuurman, J. G., De Croock, M. B. M., & Paas, F. G. W. C. (2002). Redirecting learners' attention during training: Effects on cognitive load, transfer test performance and training efficiency. *Learning and instruction, 12*(1), 11—37.

Von Mayrhauser, A., & Vans, A. M. (1995). Program comprehension during software maintenance and evolution. *Computer, 28*(8), 44—55.

Vredeveldt, A., Hitch, G. J., & Baddeley, A. D. (2011). Eyeclosure helps memory by reducing cognitive load and enhancing visualisation. *Memory & cognition, 39*(7), 1253—1263.

W3C (n.d.). *Accessibility*. Haettu 1.3. 2022 verkkosivulta https://www.w3.org/standards/webdesign/accessibility.

W3C (2012). *WCAG 2.0 IS NOW ALSO ISO/IEC 40500!*. Haettu 1.3.2022 verkkosivulta https://www.w3.org/blog/2012/10/wcag-20-is-now-also-isoiec-405/.

W3C (2017). *Tools and Techniques*. Haettu 1.3. 2022 verkkosivulta https://www.w3.org/WAI/people-use-web/tools-techniques/.

Wang, Q., Yang, S., Liu, M., Cao, Z., & Ma, Q. (2014). An eye-tracking study of website complexity from cognitive load perspective. *Decision support systems, 62*, 1—10.

WebAIM (2018). *Survey of Users with Low Vision #2 Results*. https://webaim.org/projects/lowvisionsurvey2/.

WebAIM (2021-a). *Screen Reader User Survey #9 Results*. https://webaim.org/projects/screenreadersurvey9/.

WebAIM (2021-b). *Survey of Web Accessibility Practitioners #3 Results*. https://webaim.org/projects/practitionersurvey3/.

WebAIM (2022). *The WebAIM Million*. https://webaim.org/projects/million/.

Weiser, M. (1984). Program slicing. *IEEE Transactions on software engineering*, (4), 352—357.

Weiser, M., & Shertz, J. (1983). Programming problem representation in novice and expert programmers. *International Journal of Man-Machine Studies, 19*(4), 391—398.

Welp, A., Woodbury, R. B., McCoy, M. A., & Teutsch, S. M. (2016). Making eye health a population health imperative: Vision for tomorrow.

Wen, F., Nagy, C., Bavota, G., & Lanza, M. (2019, May). A large-scale empirical study on code-comment inconsistencies. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)*, 53—64. IEEE.

Wiedenbeck, S., Fix, V., & Scholtz, J. (1993). Characteristics of the mental representations of novice and expert programmers: an empirical study. *International Journal of Man-Machine Studies, 39*(5), 793—812.

Williams, L. J. (1982). Cognitive load and the functional field of view. *Human Factors, 24*(6), 683—692.

Witterborn, J., & Rein, D. (2014). *The future of vision: Forecasting the prevalence and costs of vision problems*.

Wittich, W., Southall, K., & Johnson, A. (2016). Usability of assistive listening devices by older adults with low vision. *Disability and Rehabilitation: Assistive Technology, 11*(7), 564—571.

Woodfield, S. N., Dunsmore, H. E., and Shen, V. Y.(1981). The effect of modularization and comments on program comprehension. In *Proceedings of the 5th international conference on Software engineering (ICSE '81)*, 215—223. IEEE Press.

Young, J. (2014). A study of print and computer-based reading to measure and compare rates of comprehension and retention. *New Library World, 115*(7/8), 376—393.

Young, R. A., Hsieh, L., & Seaman, S. (2013). The tactile detection response task: preliminary validation for measuring the attentional effects of cognitive load. In *Driving Assesment Conference* 7(2013). University of Iowa.

Yung, H. I., & Paas, F. (2015). *Effects of computer-based visual representation on mathematics learning and cognitive load*.

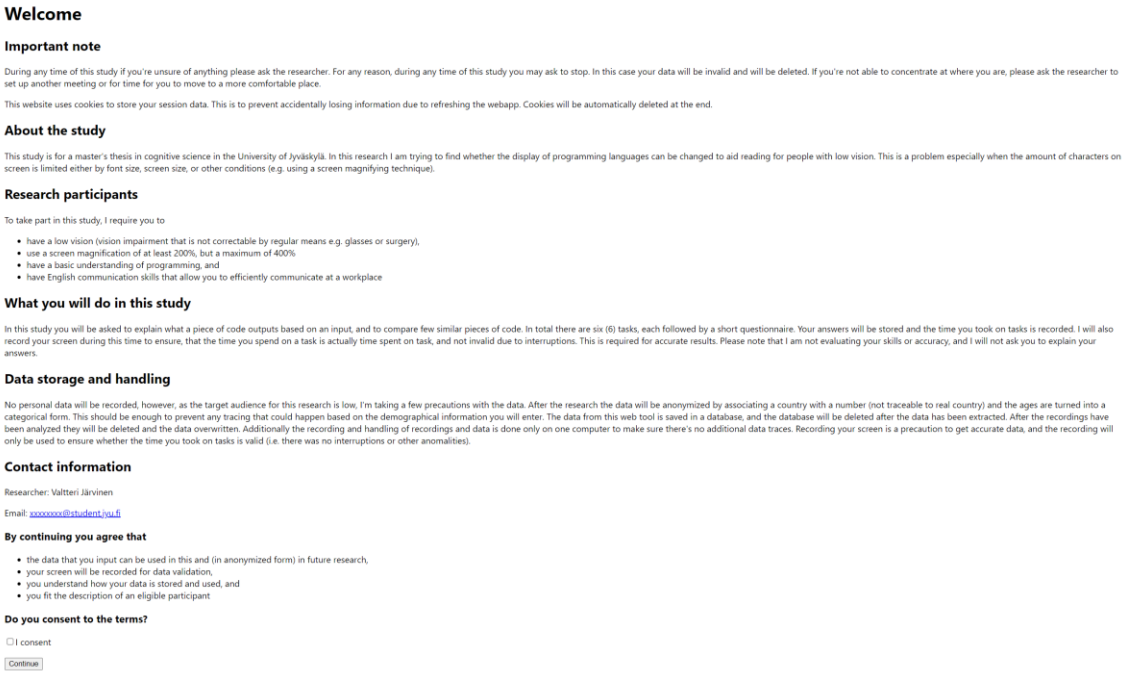
Zhao, Y., Kupferstein, E., Rojnirun, H., Findlater, L., & Azenkot, S. (2020). The effectiveness of visual and audio wayfinding guidance on smartglasses for people with low vision. In *Proceedings of the 2020 CHI conference on human factors in computing systems*, 1—14.

Zhao, Y., Kupferstein, E., Tal, D., & Azenkot, S. (2018). " It Looks Beautiful but Scary" How Low Vision People Navigate Stairs and Other Surface Level Changes. In *Proceedings of the 20th International ACM SIGACCESS Conference on Computers and Accessibility*, 307—320.

Xu, S., Rajlich, V., & Marcus, A. (2005). An empirical study of programmer learning during incremental software development. In *Fourth IEEE Conference on Cognitive Informatics, 2005 (ICCI 2005)*, 340—349. IEEE.

LIITE 1 TUTKIMUSTYÖKALUN RAKENNE

Seuraavissa kuvioissa esitetään tutkimustyökalun eri näkymät. Kuvioiden 15, 16 ja 17 näkymät näytetään peräkkäin jokaisen tehtävän kohdalla. Nämä näkymät esitetään peräkkäin kuusi kertaa. Viimeiseen tehtävään vastattuaan ja täytettyään SUS-kyselyn koehenkilölle näytetään työkalun viimeinen sivu.



Welcome

Important note

During any time of this study if you're unsure of anything please ask the researcher. For any reason, during any time of this study you may ask to stop. In this case your data will be invalid and will be deleted. If you're not able to concentrate at where you are, please ask the researcher to set up another meeting or for time for you to move to a more comfortable place.

This website uses cookies to store your session data. This is to prevent accidentally losing information due to refreshing the webapp. Cookies will be automatically deleted at the end.

About the study

This study is for a master's thesis in cognitive science in the University of Jyväskylä. In this research I am trying to find whether the display of programming languages can be changed to aid reading for people with low vision. This is a problem especially when the amount of characters on screen is limited either by font size, screen size, or other conditions (e.g. using a screen magnifying technique).

Research participants

To take part in this study, I require you to

- have a low vision (vision impairment that is not correctable by regular means e.g. glasses or surgery),
- use a screen magnification of at least 200%, but a maximum of 400%
- have a basic understanding of programming, and
- have English communication skills that allow you to efficiently communicate at a workplace

What you will do in this study

In this study you will be asked to explain what a piece of code outputs based on an input, and to compare few similar pieces of code. In total there are six (6) tasks, each followed by a short questionnaire. Your answers will be stored and the time you took on tasks is recorded. I will also record your screen during this time to ensure, that the time you spend on a task is actually time spent on task, and not invalid due to interruptions. This is required for accurate results. Please note that I am not evaluating your skills or accuracy, and I will not ask you to explain your answers.

Data storage and handling

No personal data will be recorded, however, as the target audience for this research is low, I'm taking a few precautions with the data. After the research the data will be anonymized by associating a country with a number (not traceable to real country) and the ages are turned into a categorical form. This should be enough to prevent any tracing that could happen based on the demographical information you will enter. The data from this web tool is saved in a database, and the database will be deleted after the data has been extracted. After the recordings have been analyzed they will be deleted and the data overwritten. Additionally the recording and handling of recordings and data is done only on one computer to make sure there's no additional data traces. Recording your screen is a precaution to get accurate data, and the recording will only be used to ensure whether the time you took on tasks is valid (i.e. there was no interruptions or other anomalities).

Contact information

Researcher: Valtteri Järvinen

Email: xxxxxxxx@student.jyu.fi

By continuing you agree that

- the data that you input can be used in this and (in anonymized form) in future research,
- your screen will be recorded for data validation,
- you understand how your data is stored and used, and
- you fit the description of an eligible participant

Do you consent to the terms?

☐ I consent

Continue

KUVIO 12 Tutkimustyökalun ensimmäinen sivu, jossa tutkittava perehdytetään tutkimukseen.

Background information

Please answer all questions.

You may opt out of some of the more personal questions, but any information you can give is valuable.

What is your gender?

- Male
- Female
- Other
- Prefer not to answer

What is your age?

Enter your age:

What is your country?

Type or select from the dropdown menu:

Is English your native language?

- Yes
- No

How much programming experience do you have? (full years, if less than one year enter 0)

Programming experience in years:

What is your navigation and input method?

- Mouse and Keyboard
- Mouse only (On-screen keyboard)
- Keyboard only
- Other

Please, shortly describe your disability (or check the box below if you prefer not to)

Describe your disability:

☐ I prefer not to describe my disability, but I verify that I have low vision and use some form of screen magnification to use the computer

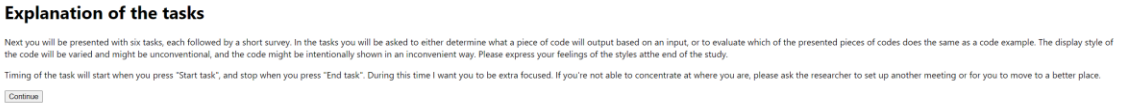
Please, shortly describe the usual problems you encounter related to programming and your vision (or check the box below if you believe there isn't anything worth mentioning)

Describe your usual problems here:

☐ I don't find anything worth mentioning

Continue

KUVIO 13 Tutkimustyökalun toinen sivu. Sivulla selvitetään koehenkilön olennaisia tietoja, sekä pyydetään kuvailemaan heikkonäköisyyttä ja ongelmia liittyen ohjelmointiin.



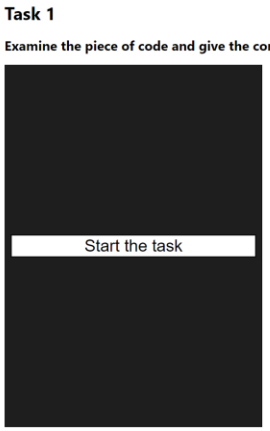
Explanation of the tasks

Next you will be presented with six tasks, each followed by a short survey. In the tasks you will be asked to either determine what a piece of code will output based on an input, or to evaluate which of the presented pieces of codes does the same as a code example. The display style of the code will be varied and might be unconventional, and the code might be intentionally shown in an inconvenient way. Please express your feelings of the styles atthe end of the study.

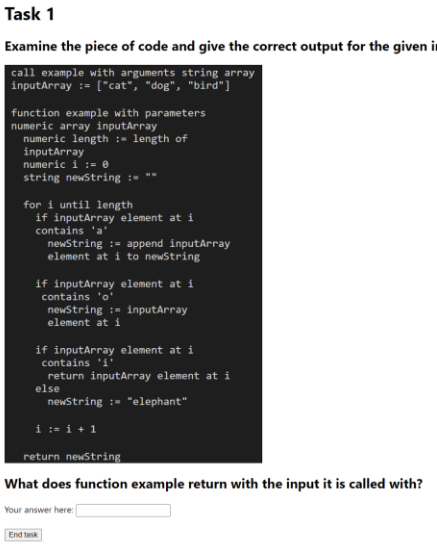
Timing of the task will start when you press "Start task", and stop when you press "End task". During this time I want you to be extra focused. If you're not able to concentrate at where you are, please ask the researcher to set up another meeting or for you to move to a better place.

Continue

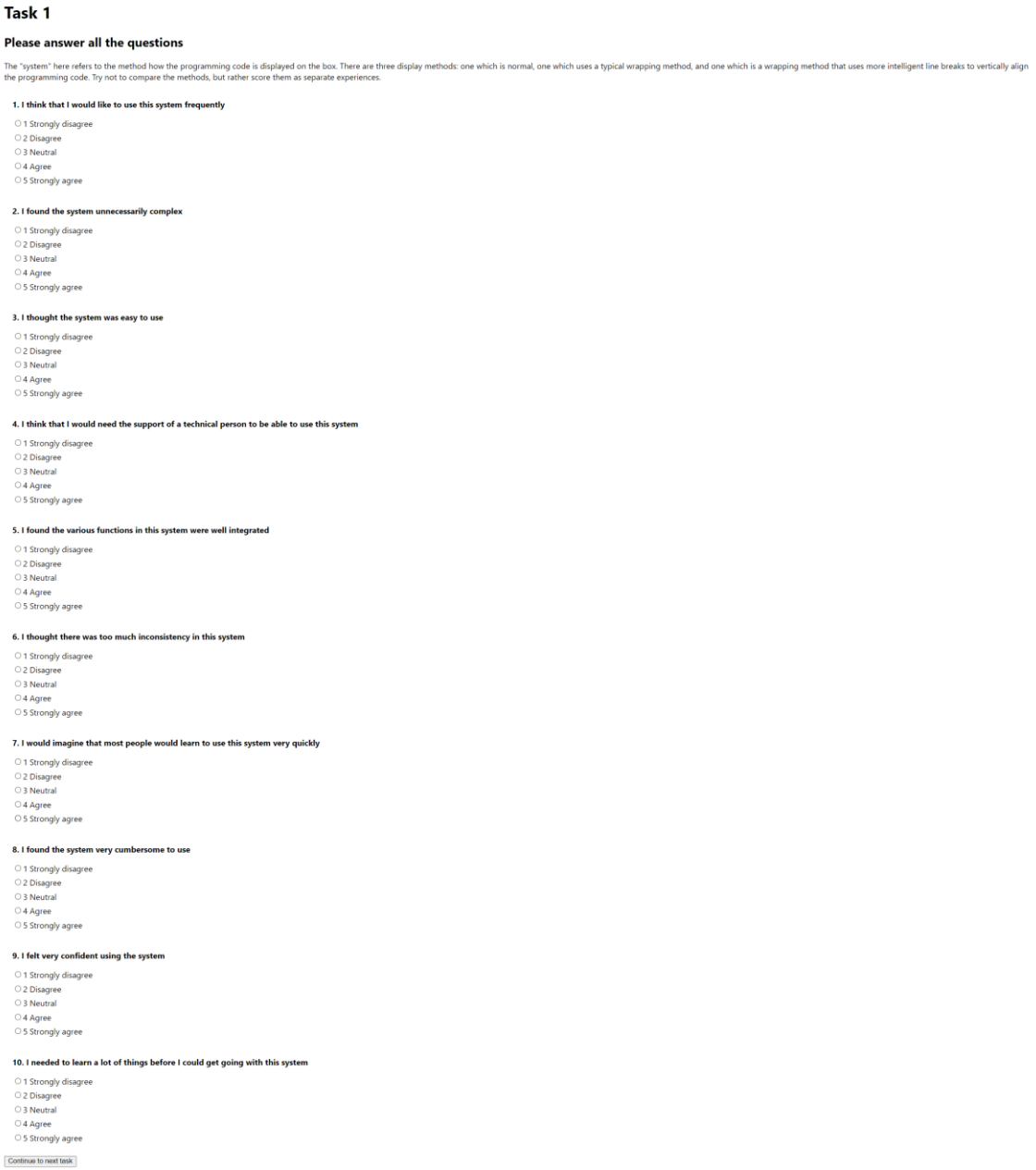
KUVIO 14 Tutkimustyökalun kolmas sivu, jossa tutkittavalle kerrotaan tutkimuksen tehtäväosiosta.



KUVIO 15 Tutkimustyökalun tehtävä sivun ensimmäinen näkymä, jossa tutkittava valmistautuu aloittamaan tehtävän. Sivu toistuu joka tehtävän aluksi.



KUVIO 16 Tutkimustyökalun tehtäväsivun toinen näkymä, jossa tutkittavalle esitetään ohjelmakoodi ja esitystapa satunnaisessa järjestyksessä. Näkymä toistuu jokaisen tehtävän aloittamisen jälkeen satunnaisella ohjelmakoodilla ja esitystavalla. Ohjelmakoodit eivät toistu ja tutkimuksen kolme eri esitystapaa esitetään jokainen kaksi kertaa tehtävien aikana.



Task 1

Please answer all the questions

The "system" here refers to the method how the programming code is displayed on the box. There are three display methods: one which is normal, one which uses a typical wrapping method, and one which is a wrapping method that uses more intelligent line breaks to vertically align the programming code. Try not to compare the methods, but rather score them as separate experiences.

1. I think that I would like to use this system frequently

- 1 Strongly disagree
- 2 Disagree
- 3 Neutral
- 4 Agree
- 5 Strongly agree

2. I found the system unnecessarily complex

- 1 Strongly disagree
- 2 Disagree
- 3 Neutral
- 4 Agree
- 5 Strongly agree

3. I thought the system was easy to use

- 1 Strongly disagree
- 2 Disagree
- 3 Neutral
- 4 Agree
- 5 Strongly agree

4. I think that I would need the support of a technical person to be able to use this system

- 1 Strongly disagree
- 2 Disagree
- 3 Neutral
- 4 Agree
- 5 Strongly agree

5. I found the various functions in this system were well integrated

- 1 Strongly disagree
- 2 Disagree
- 3 Neutral
- 4 Agree
- 5 Strongly agree

6. I thought there was too much inconsistency in this system

- 1 Strongly disagree
- 2 Disagree
- 3 Neutral
- 4 Agree
- 5 Strongly agree

7. I would imagine that most people would learn to use this system very quickly

- 1 Strongly disagree
- 2 Disagree
- 3 Neutral
- 4 Agree
- 5 Strongly agree

8. I found the system very cumbersome to use

- 1 Strongly disagree
- 2 Disagree
- 3 Neutral
- 4 Agree
- 5 Strongly agree

9. I felt very confident using the system

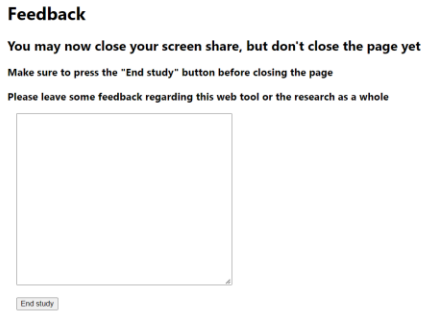
- 1 Strongly disagree
- 2 Disagree
- 3 Neutral
- 4 Agree
- 5 Strongly agree

10. I needed to learn a lot of things before I could get going with this system

- 1 Strongly disagree
- 2 Disagree
- 3 Neutral
- 4 Agree
- 5 Strongly agree

Continue to next task

KUVIO 17 Tutkimustyökalun tehtäväosion jälkeen täytettävä SUS-kysely. Sivu toistuu jokaisen vastauksen jälkeen.



KUVIO 18 Tutkimustyökalun viimeinen sivu, jossa koehenkilöt pystvät jättämään palautetta.

LIITE 2 OHJELMAKOODIT

KOODI 1

```
call example with numeric number := 2

function example with parameters numeric number
  numeric i := 1
  numeric x := 0

  while number >= i
    if x >= 0 and number > 0
      x := x + number * number
      number := number + 1

    while number > 0 and x < 10
      x := number * i + x * 2

    i := i + 2

  return x
```

KOODI 2

```
call example with numeric array inputArray := ["grape", "mango", "fig"]

function example with parameters numeric array inputArray
  string newString := ""
  numeric length := length of inputArray
  numeric i := 0

  while i < length
    if inputArray element at i contains 'a'
      newString := append newString to newString
    if inputArray element at i contains 'e'
      newString := append newString to inputArray element at i
    if inputArray element at i contains 'i'
      newString := inputArray element at i
    i := i + 1

  return newString
```

KOODI 3

```
call example with numeric array inputArray := [30, 40, 39]

function example with parameters numeric array inputArray
```

```
  numeric i := 0
  numeric length := length of newArray

  for i until length
    if newArray element at i % 10 = 0
      newArray element at i := newArray element at i * 2

    if newArray element at i % 3 = 0
      newArray element at i := newArray element at i - 4

    if newArray element at i % 15 = 0
      newArray element at i := 0

    i := i + 1

  return newArray
```

KOODI 4

```
call example with numeric limit := 2

function example with parameters numeric limit
  numeric i := 0
  numeric number := 2

  while i < limit
    while number > 0
      number := number - limit

    if number <= -1
      number := number - i

    while number <= 0
      number := number + 3

    i := i + 1

  return number
```

KOODI 5

```
call example with numeric array inputArray := [22, 10, 0]

function example with parameters numeric array inputArray
  numeric length := length of inputArray
  numeric i := 0
  numeric number := 0
```

```
 for i until length
  if number <= 0 and inputArray element at i < 10
   number := number + 10

  if number >= 0 and inputArray element at i > 20
   number := number - 5

  if number = 10
   return -1
  else
   number := 0

  i := i + 1

 return number
```

KOODI 6

```
call example with string array inputArray := ["cat", "dog", "bird"]

function example with parameters numeric array inputArray
 numeric length := length of inputArray
 numeric i := 0
 string newString := ""

 for i until length
  if inputArray element at i contains 'a'
   newString := append inputArray element at i to newString

  if inputArray element at i contains 'o'
   newString := inputArray element at i

  if inputArray element at i contains 'i'
   return inputArray element at i
  else
   newString := "elephant"

  i := i + 1

 return newString
```

LIITE 3 SUS-LOMAKKEEN KYSYMYKSET

1. I think that I would like to use this system frequently
2. I found the system unnecessarily complex
3. I thought the system was easy to use
4. I think that I would need the support of a technical person to be able to use this system
5. I found the various functions in this system were well integrated
6. I thought there was too much inconsistency in this system
7. I would imagine that most people would learn to use this system very quickly
8. I found the system very cumbersome to use
9. I felt very confident using the system
10. I needed to learn a lot of things before I could get going with this system