Tallinna Reaalkool

Pranglimisrakenduse loomine välearenduse põhimõtetel

Uurimistöö

Hermann Käbi

140.a reaal-programmeerimise õppesuund

Juhendaja: õp Hanna Britt Soots

Tallinn 2024

Sisukord

Sissejuhatus

Peastarvutamine on matemaatikatunnis õpetatavatest oskustest üks eluliselt kasulikem. See on oskus, mida vajab ja kasutab pea igaüks. Põhikooli matemaatika õppekavasse on peastarvutamine oskuse omandamine ka sisse kirjutatud. Paraku on peastarvutamise oskuse arendamine paberil problemaatiline nii õpilasele, kes saab tagasisidet liiga hilja, kui õpetajale, kelle jaoks oleks selliste testide tegemine ajaliselt koormav. Õpilastel, kes soovivad oma peastarvutusoskust hoida ja parandada, tuleb seega leida selleks mingi muu võimalus.

Üheks enimsoovitatud võimaluseks on pakutud pranglimist. Pranglimine on võisteldes aja peale peastarvutamine. Pranglimine on olnud populaarne koolides üle Eesti, seda eeskätt tänu Miksikese keskkonnale. Miksike ongi olnud pranglimise algatajaks ja eestvedajaks. Kahjuks sulges Miksike 2022. aastal oma veebiserverid, peale mida on pranglimise populaarsus nii tunnisiseselt kui -väliselt langenud. Alternatiiviks on laialt levinud Nutispordi keskkond, millel aga esineb Miksikesega võrreldes mitmeid puudusi. Eeltoodust tulenevalt tekkis õpetajatel soov luua Tallinna Reaalkoolile oma pranglimisrakendus, mis vastaks võimalikult täpselt nende ja õpilaste vajadustele.

Käesoleva uurimistöö eesmärgiks on selline veebirakendus luua ja arendusprotsessi käigus vastata uurimisküsimusele: kuidas kasutada välearenduse põhimõtteid reaalses elus pranglimisrakenduse arenduse näitel? Uurimistöö koosneb välearenduse tsükli kirjeldusest, pranglimisrakenduse arendamiseks kasutatud tehnoloogiate kirjeldusest, arendusprotsessi kirjeldusest ja õpilaste ning õpetajate antud tagasiside analüüsist. Peamiseks allikaks välearenduse kohta on James Shore'i ja Shane Warden'i teos „The Art of Agile Development“ („Välearenduse kunst“), samuti erinevad välearendust käsitlevad artiklid ja uurimistööd. Erinevate tehnoloogiate kirjeldamisel on aluseks võetud vastavad dokumentatsioonid.

Uurimistöö autor soovib tänada juhendaja Hanna Britt Sootsi, pranglimisrakenduse kaasarendajat Jarl Justus Hellatit, pranglimisrakenduse valmimisega aidanud õpetajaid Helli Juurma, Jaanika Lukk, Villu Raja, Riin Saar ja Hanna Britt Soots, tehnilise toe

(mh domeeni ja serverite seadistamise) eest Tallinna Reaalkooli IT juht Veiko Somelari, pranglimisrakenduse nime ja logo konkursi meeskonda Ene Saar, Riin Saar ja Kaisa Tamkivi, samuti kõiki Tallinna Reaalkooli õpetajaid ning 145., 146. ja 147. lennu õpilasi, kes rakendust katsetasid.

1. Välearendus

1.1. Välearenduse kirjeldus

Välearendus (ingl k *agile development*) on tarkvaraarenduse metoodika, mille alus on iteratiivsus ja mis rõhutab pooltevahelise koostöö olulisust (Andmekaitse ja infoturbe leksikon *s.v.* agile development). Eestikeelne termin „välearendus“ on välja pakutud Andmekaitse ja infoturbe leksikonis (AKIT), mis ühtlasi nimetab „agiilset arendustsüklit“ naiivseks poolkeelseks laenuks (*ibid.*). Sel põhjusel kasutatakse ka selles uurimistöös sõna „välearendus“. Kuna välearendus sarnaneb pigem filosoofilise mõtte kui algoritmiga, on sellele täpset definitsiooni võimatu anda. See tähendab ka, et ühte kindlat välearenduse mudelit pole olemas. Selleks, et praktiseerida välearendust, tuleb kasutada agiilsed meetodid vastavalt konkreetsele projektile. (Shore, Warden 2008a)

Välearenduse tavad on sõnastatud *Manifesto for Agile Software Development*'s („Agiilse tarkvaraarenduse manifest“). Manifestis kirjeldatakse välearenduse nelja põhimõtet:

1. Hindame enam inimesi ja nende suhtlemist kui protsesse ja arendusvahendeid.

2. Hindame enam töötavat tarkvara kui täiuslikku dokumentatsiooni.

3. Hindame enam koostööd kliendiga, kui läbirääkimisi lepingute üle.

4. Hindame enam muudatustega hakkamasaamist kui algse plaani järgimist.

See ei tähenda aga, et välearenduses parempoolseid tegureid ei hinnata, vaid vasakpoolseid hinnatakse rohkem. (Beck *et al.* 2001)

Oluline on mõista, et välearendus ei ole iseenesest konkreetne arendusmeetod, vaid pigem katustermin. Selleks, et välearenduse kasutamine oleks lihtsam, on kasutusele võetud erinevad välearenduse raamistikud (ingl k *framework*). Kõige populaarsem välearenduse raamistik on Scrum, mida kasutab 87% arendajatest (State of Agile 2022). Scrum hõlmab endas projekti jagamist väikesteks osadeks ehk sprintideks, mille pikkus jääb tavaliselt

nädala ja kuu vahele. Korraga keskendub arendustiim vaid ühele sprindile. (PremierAgile) Väga sarnane raamistik on Extreme Programming (XP), mis jagab projekti samuti sprintideks, kusjuures pärast igat sprinti üritatakse meetodit vastavalt hetkeseisule võimalikult palju optimeerida. Samas, erinevalt Scrumist rõhutab XP pideva kliendiga suhtlemise vajadust. Erinevalt Scrumist, mida kasutatakse projektijuhtimises üldiselt, on XP, nagu nimigi viitab, kasutusel vaid tarkvara arendamisel. (Raeburn 2022) Kolmas populaarne raamistik on Kanban (56%), mille võlu seisneb lihtsuses. Selle meetodiga luuakse arendusseisust visuaalne tahvel, kus on näha nii tulevikus alustatavad, praegu töösolevad kui ka juba tehtud arendused. Lisaks mainitutele on välearenduse põhimõtetele ülesehitatud raamistikke veelgi. (PremierAgile; State of Agile 2022)

Tänapäeval on välearendus tarkvaraarenduse oluline osa. Välearenduse meetodeid kasutavad pea kõik suured tehnoloogiafirmad, muuhulgas Microsoft, IBM ja Apple. (Flynn 2022) 2023. aasta seisuga kasutab välearenduse meetodeid 86% professionaalsetest tarkvaraarendajatest ning 71% tarkvara arendavatest firmadest (Outworks Solutions Private Ltd. 2023). Seega tuleb arendajale välearenduse vähemalt põhiliste meetodite tundmine ja kogemus kasuks nii teiste arendajatega sujuvama koostöö loomiseks, aga ka aina konkurentsitihedamal tööturul. Kuna välearendust kasutab aina rohkem tarkvarafirmasid, on see oluline mitte ainult arendajatele, vaid ka näiteks projektijuhtidele. Samuti kasutatakse välearenduse põhimõtteid ka projektides, kus tarkvara ei arendata, seega tuleb see oskus kasuks ka väljaspool IT-sektorit. (PremierAgile 2023)

Vaatamata väga suurte firmade mõjule ja laialdasele populaarsusele arendajate seas, on oluline märkida, et empiirilised uurimused välearenduse efektiivsuse kohta on jõudnud erinevate järeldusteni. Dybå ja Dingsøyr toovad 36 välearenduse uuringut analüüsides, et nende uurimuste tulemuste väärtus on kaheldav. Valdavad veakohad on nii uurimuslikud, näiteks potentsiaalsete kalduvuste (ingl k *bias*) eiramine, aga vigu on tehtud ka andmete kogumise faasis. Nimelt leiti, et vaid vähesed uurimused olid tehtud firmades, mis olid välearendust kasutanud üle aasta. Valdav osa (76%) uurimustest uuris vaid XP kasutamist, seega ei saa nende põhjal eeldada samasugust mõju, kui kasutada mõnda muud välearenduse raamistikku, näiteks Scrumi. Järelikult ei ole nende uurimuste tulemused olukorras, kus valdav osa arendajaid kasutab teisi raamistikke, enam relevantsed. Dybå ja Dingsøyr rõhutavad just Scrumi efektiivsuse ja eeliste põhjalikuma uurimise olulisust. Seega tuleb

enne välearenduse kasutuselevõtmist kaaluda selle võimalikke eelised konkreetses projektis, mitte toetuda vaid suurfirmade eeskujule. (Dybå, Dingsøyr 2008)

1.2. Välearenduse ajalugu

Tsüklilised arendusmeetodid olid kasutusel juba 1950. aastate keskpaigast (Basili, Larman 2003). Samas oli selliste meetodite kasutamine harv. Pigem eelistati koskmudelil põhinevaid arendusmeetodeid. (Sacolick 2022) Koskmudel on nime saanud kosest läbi voolava vee tõttu - vesi liigub ühesuunaliselt edasi ja tagasi enam ei pöördu. Koskmudeli järgi peab enne uue etapi alustamist kõik eelnevad täielikult valmis olema. Näiteks ei hakata rakendust programmeerima enne, kui kõik nõuded ja kogu disainiprotsess läbitud on. (Conrad 2011) Koskmudeli eeskujuks oli Fordi konveieriliin, mis oli 20. sajandi alguses teinud revolutsiooni autotööstuses. Koskmudeli eesmärk oli kindlustada projekti vastavus kliendi algsetele soovidele. Koskmudeli üheks suuremaks puuduseks oli arenduse ajaline pikkus ja jäikus. (Sacolick 2022)

Veebirakenduste esiletõusuga hakkasid koskmudeli puudused aina enam silma paistma. Kui varasemalt oli rakenduste loomiseks erinevaid tehnoloogiaid ja uuendusi võrdlemisi vähe, siis veebi pideva ja kiire arenemise tõttu pidid ka arendajad aina paindlikumaks ja kiiremaks muutuma. Efektiivsuse suurendamiseks ei olnud kogu dokumentatsiooni ja disaini valmis tegemine enne arenduse algust enam võimalik. Aina suurenevate arendustiimide tõttu tekkis vajadus koostööd suurendavate meetodite järele. (*ibid.*)

2001. aastal kogunes Utah's Snowbirdi kuurordis grupp kogenud arendajaid, kes olid mõistnud, et arendavad tarkvara traditsioonilisest koskmudelist erinevalt. Need 17 arendajat, teiste seas XP looja Kent Beck ja Scrumi looja Jeff Sutherland, panid oma kogemustele tuginedes kirja välearenduse põhimõtted (*Manifesto for Agile Software Development*). Välearendus, mis rõhutas koostöö ja pideva muutumise olulisust ja kritiseeris jäika juhtimisstiili, oligi sündinud. (Copeland 2001; Krill 2014; Sacolick 2022)

Sellest ajast on välearendus arendusmaailmas aina levinud. 2005. aastal avaldasid Cockburn ja Highsmith dokumendi, milles avaldati välearendusliku projektijuhtimise põhimõtted (Cockburn, Highsmith 2005). 2009. aastal avaldatud *Software Craftmanship Manifesto* täiendas Beck *et al.* kirjutatud manifesti, lisades juhendi välearenduse kasutamiseks suurtes

ja professionaalsetes arendustiimides (Martin 2009). 2012. aastal lõi Laurent Bossavit avatud lähtekoodiga sõnaraamatu *Agile Glossary* (kuni 2016. aastani tuntud kui *Guide to Agile Practices*) välearenduses kasutatavate terminite ja tavade selgitamiseks (Agile Alliance 2023; McDonald 2016).

Välearenduse põhimõtteid hakkasid suurfirmad üsna kiiresti kasutama. Juba 2007. aastal oli Google mitmes projektis välearenduse põhimõtted kasutusele võtnud (Striebeck 2007). Apple on üle viieteistkümne aasta kasutanud Scrumi (Denning 2012). Forbes'i 2015. aastal avaldatud artikli kohaselt on ka Microsoft välearenduse põhimõtted kasutusele võtnud (Denning 2015). Iga-aastase välearenduse seisu analüüsiva raporti *State of Agile* 2022. aasta väljalaske andmetel kasutab välearendust mingil määral 80% vastanud firmadest (State of Agile 2022).

2. Pranglimisrakenduse loomisel kasutatud tehnoloogiad

Üks enimkasutatavaid viise veebiarenduse liigitamiseks on liigitus *front-end*'iks ja *back-end*'iks (GeeksForGeeks 2023a). Selline jaotus on kasulik, sest paljud veebiarendajad keskenduvad oma oskustest ühele nendest.

*Front-end*i all mõeldakse seda osa veebilehest, mida kasutaja näeb ja millega ta suhtleb. Selleks on veebilehe graafiline kasutajaliides (GUI, ingl k *graphical user interface*). *Front-end*i alla kuuluvad näiteks küljendus, fondid, värvid, nupud, pildid, videod, navigatsioonimenüüd. *Front-end* koosneb põhiliselt kolmest komponendist: HTML, CSS ja JavaScript. Nendele lisanduvad arendajakogemuse (DX, ingl k *developer experience*) ja koodi haldamise ja loetavuse parandamiseks erinevad raamistikud. (*ibid.*)

Back-end on veebilehe osa, mida kasutaja otseselt ei näe. See aga ei tähenda, et tegu oleks kuidagi vähem olulisema osaga veebiarendusest. *Back-end*i alla kuuluvad näiteks *routing* (vastavalt URL-ile sobiva sisu leidmine), andmete kogumine ja kasutamine, serverite haldamine jne. Erinevalt *front-end*ist pole *back-end*is välja kujunenud standardkeeli, see valitakse vastavalt vajadustele. Kõige populaarsem *back-end* keel on PHP, aga kasutatakse ka C++ ja JavaScripti. *Back-end*is kasutatakse väga tihti erinevaid raamistikke, et lihtsustada integratsioone andmebaasidega, autentimist ja *front-end* raamistike kasutust. (*ibid.*)

2.1. *Front-end* tehnoloogiad

2.1.1. *Hypertext Markup Language* (HTML)

Hypertext Markup Language (HTML) on *front-end* veebiarenduse kõige põhilisem tehnoloogia. HTML annab veebilehel olevale sisule tähenduse ja struktuuri. (MDN Web Docs 2023c) Oluline on siinjuures mõista, et HTML ei ole programmeerimiskeel, vaid märkekeel (*ibid.*). See tähendab, et kasutades HTML süntaksit saab muuta sisu väljanägemist, aga mitte anda arvutile käsklusi mingite ülesannete täitmiseks. Näiteks saab HTML'i kasutades muuta tekst rasvaseks (*bold*), aga (peale veebilehe avamist) mitte teksti sisu ennast.

HTML dokument koosneb erinevatest elementidest ehk märgenditest, mis enda sisse jääva sisu teatud moodi vormistab. Näiteks vormistab märgend *p* enda sisu tavalise tekstina, *h1* (*heading 1*, suurim pealkiri) aga suure pealkirjana (MDN Web Docs 2023d). HTML süntaksi lihtsus peitub asjaolus, et märgendeid saab ka üksteise sisse panna. Nii saab näiteks ainult ühe osa lõigust kaldkirja panna. Märgendile saab lisada erinevaid atribuute, näiteks hüperlingi (*a*) loomisel lisatakse attribuut *href*, millega määratakse URL, kuhu link suunab. Igale märgendile saab lisada klassi (*class*), mis saab olla mitmel elemendil, ja ID, mis peab igal elemendil olema unikaalne. (MDN Web Docs 2023c; MDN Web Docs 2023d)

Kuna HTML mängis suurt rolli kogu veebi loomisel, on sellel pikk ajalugu. Esimese versiooni lõi CERNis töötav füüsik sir Tim Berners-Lee juba 1991. aastal, kuid see jäi avaldamata. See versioon, nimega HTML 1.0, avaldati 1993. aastal ja pani aluse veebile endale. Sellest ajast peale on HTML pikkamisi arenenud. Tänapäeval kasutatakse 2014. aastast pärit HTML5 versiooni. See versioon lisas muuhulgas platvormipõhise toe video- ja audioformaatidele ja palju semantilisi elemente. (University of Boston 2020b) Kui HTML 1.0 oli vaid 18 elementi, siis HTML5 sisaldab endas lausa 137 elementi. (MDN Web Docs 2023d)

HTMLi võib vaadelda veebilehe selgrooga – lehtede tegemine ilma selleta on võimatu. Uurimistöö käigus arendatud pranglimisrakenduse kasutajaliides on Reacti-põhine, mis kasutab HTML-sarnast keelt JSX, mis ühendab endas JavaScripti ja HTMLi ja millest räägitakse täpsemalt osas 2.1.4.

2.1.2. *Cascading Style Sheets* (CSS)

Cascading Style Sheets (CSS) on laadilehekeel, mis määrab, kuidas HTML koodi ekraanil kuvatakse. Erinevalt HTMList ei lisa CSS veebilehele sisu, vaid kujundab seda. Väikeste eranditega võime öelda, et HTML vastutab veebilehe tekstilise sisu ja CSS selle esitamise eest. (MDN Web Docs 2023b)

CSS kasutab reeglitepõhist süntaksit. Reegel (ingl k *rule*) algab selektori (*selector*) määramisega. Selektor on struktuur, millega määratakse valitavate HTML-elementide hulk. Näiteks saab selektoriga valida kõik pealkirjad (*h1*) dokumendis, aga samuti klasse ja IDsid. Kasutades sellised lihtsaid selektoreid saab kombineerida ka tunduvalt komplekssemaid. Näiteks saab valida iga teise lõigu, millel on klass „tekst“ ja mis asuvad *div* elemendi sees.

Selektorile järgneb loend deklaratsioone (*declaration*), mis muudavad kindla omaduse (ingl k *property*) väärtust (*value*). Näiteks omadusega *font-family* saab muuta elemendi fonti ja *background-color* muudab taustavärvi. (W3Schools 2023b)

Lisaks elementide laadi muutmisele pakub CSS ka võimekat kastimudelit (ingl k *box model*) (W3Schools 2023a). CSS kohtleb kõiki elemente ristkülikutena, mis võimaldab kuvada neid ekraanile võimalikult väikese jõudlusega. Tänu kastimudelile saab elementide paigutust lihtsalt ja paindlikult muuta. (MDN Web Docs 2024) CSSi teine suur eelis on meediapäringud (ingl k *media query*), mis lisavad võimaluse muuta elementide laadi vastavalt brauseriakna suurusele (ingl k *viewport*). Seega saab CSSiga teha lehti, mis on ka mobiilisõbralikud (ingl k *responsive*). (MDN Web Docs 2023a)

Kuna esialgu oli veeb mõeldud teadlastele sisu jagamiseks, ei olnud veebilehe laad oluline. Veebi populariseerumine tõi aga kaasa vajaduse visuaalselt ilusate lehtede loomiseks. Juba 1994. aastal pakkus Håkon Wium Lie välja idee CSSist ja 1996. aastal avaldatigi CSS 1, millega sai muuta väga põhilisi omadusi, nagu fonti ja värve. Kaks aastat hiljem avaldati CSS 2 ja alustati tööd CSS 3-ga. Selle versiooniga tehti põhimõtteline uuendus: kui varem oli avaldatud versioon tervikuna, siis CSS 3 avaldati moodulitena. (University of Boston 2020a) Esimene suur moodul avaldati 2012. aastal, peale mida on keelt senimaani pidevalt uuendatud. (Raj 2023)

Kuna CSS on niivõrd palju muutunud ja arenenud, on see muutunud üsna keeruliseks ja koodi poolest pikaks. Probleemi lahendamiseks on loodud erinevaid CSS raamistikke, mille eesmärgiks on muuta põhiliste kasutajaliidese komponentide loomine võimalikult lihtsaks. Sellel lähenemisel on mitmeid eeliseid. Esiteks muudab see veebilehe disaini lihtsaks ja kiireks. Samuti on komponendid disaininud tiim professionaalseid disainereid, mistõttu sobivad komponendid (näiteks nupud, lingid jne) omavahel visuaalselt hästi kokku. Raamistike kasutamisel on ka negatiivseid pooli. Kuna komponendid on juba valmis tehtud, tähendab see, et iga leht, mis ühe raamistikuga tehtud on, näeb võrdlemisi sarnane välja. Kuna raamistikud on disainitud võimalikult erinevatele lehtedele sobivaks, võivad nendega tehtud lehed tavalised ja isegi igavad välja näha. Märksa tehnilisem probleem, eriti vanemate raamistikega, on liigne kood. Kuna raamistikud koosnevad väga paljudest komponentidest erinevateks olukordadeks, juhtub praktikas tihti, et veebileht neid kõiki ei kasutada. Seega jääb veebilehele palju koodi, mida see tegelikult ei kasuta,

mis võib lehe aeglasemaks muuta. (Ayebola 2023; Bose 2023)

Pranglimisrakenduse arendamisel kasutab autor CSSi ilma ühegi raamistikuta. Selline otsus tehti, võttes arvesse eelmises lõigus toodud puudusi. Autori jaoks on oluline rakenduse visuaalne erilisus ja ise komponentide loomine on autori jaoks oluline veebiarenduse osa. Seega lõi autor pranglimisrakenduse jaoks sisuliselt eraldi enda raamistiku, sest iga lehe puhul on ühtlane disainikeel siiski oluline. Kuna CSS on autorile varasemast kogemusest tuttav, ei olnud lehtede kujundamine CSSiga väga keeruline. Samas üritas autor arenduse käigus kirjutada võimalikult modulaarset ja muudetavat CSSi. Üks CSSi funktsioon, mida autor selle projekti käigus õppis, oli muutujate kasutamine CSSis. Kui paljudel komponentidel on näiteks sama taustavärv, ei ole mõtet igale komponendile eraldi ühte värvikoodi kopeerima, sest kui seda on hiljem muuta vaja, tuleb selleks palju tööd teha. Siis on mõistlikum salvestada värv muutujasse ja kasutada komponentides muutujat. Muutujatesse saab loomulikult salvestada igat tüüpi väärtuseid. Muutujate kasutamine CSSis seondub ka välearendusega, sest pideva tagasiside saamine muudab tõenäoliseks asjaolu, et osa disainist tuleb ümber teha, mis on muutujaid kasutades palju kiirem. Samuti võimaldas muutujate algusest peale kasutamine luua kiiresti ja võimalikult vähese vaevaga võimalus põhivärvi koheseks vahetamiseks, millest on täpsemalt juttu alapeatükis 3.1.

2.1.3. JavaScript

JavaScript on kolmas veebiarenduse põhitehnoloogia. Tegu on dünaamiliselt kirjutatud interpreteeritud kõrgetasemelise programmeerimiskeelega. (MDN Web Docs 2023e) Kuigi brauseris (ingl k *client-side*) saab veebileht teoreetiliselt jooksutada erinevaid programmeerimiskeeli, kasutavad veebiarenduse uuringufirma W3Techs hinnangul JavaScripti brauseris 98,7% veebilehtedest (W3Techs 2023). Populaarse arenduskeskkonna Stack Overflow igaaastase küsitluse tulemusena oli 2022. aastal JavaScript juba kümnendat aastat järjest kõige sagedamini kasutatud programmeerimis- ja märkekeel (Stack Overflow 2022). Seetõttu on veebilehtede arendamisel JavaScripti oskus sisuliselt kohustuslik.

JavaScripti kasutusaladeks pole aga ainult koodi jooksutamine brauseris. Just suure populaarsuse tõttu kasutatakse JavaScripti tihti ka *back-end* arenduseks (MDN Web Docs 2023e). Stack Overflow 2022. aasta küsitlusest nähtub, et nii professionaalsete arendajate kui kõigi vastajate hulgas on populaarseim veebiraamistik (ingl k *web framework*)

JavaScripti-põhine Node.js (Stack Overflow 2022).

Üheks näiteks, kus JavaScripti kasutatakse väljaspool brauserit, on Node.js. Node.js on JavaScriptil põhinev populaarne platvormist sõltumatu *back-end* raamistik (Node.js 2023). Node.js muutis JavaScripti kasutamise populaarseks väljaspool brauserit (BrainHub 2023). Node.js'i populaarsust omistatakse eeskätt lihtsale ja mõistetavale süntaksile, kõrgel kohal on ka raamistiku kasutatud JavaScript, mis on arendajatele juba tuttav ja seega muudab arenduse kiiremaks (*ibid.*).

Kolmest põhitehnoloogiast on JavaScriptil kõige keerulisem ajalugu. Veebi esimestel aastatel said veebilehed olla ainult staatilised, sest programmeerimiskeelt brauseris kasutada ei saanud. Selle probleemi lahendamiseks ühines NetScape'i (esimene populaarne veebibrauser) meeskonnaga Brendan Eich. Eichi ülesandeks sai luua uus Java-laadse süntaksiga programmeerimiskeel veebilehtede jaoks. JavaScripti esialgse versiooni suutis Eich 1995. aasta maikuus kirjutada kõigest kümne päevaga. (GeeksForGeeks 2023b; W3Schools 2023c)

Veebibrausereid tegid juba tol ajal erinevad firmad, mis kasutasid pisut erinevaid programmeerimiskeeli. Näiteks lõi Microsoft 1996. aastal Interneti Exploreri jaoks JavaScripti alusel enda keele - JScript. See põhjustas raskusi arendajatele, kes ei suutnud teha lehti mõlema brauseri jaoks. Probleemi lahendamiseks esitas Netscape 1996. aasta novembris JavaScripti standardiorganisatsioonile Ecma International (ECMA). Sellest arenes 1997. aasta juunis välja ECMAScript, standardiseeritud programmeerimiskeel, mis loodigi tagamaks koodi töötamise brauseriüleselt. (ECMA 1997; GeeksForGeeks 2023b; W3Schools 2023c)

Vahepeal saavutas Microsft Internet Exploreriga jõupositsiooni. 2000. aastate alguseks kasutas Internet Explorerit 95% internetikasutajatest. Kuigi alguses läks Microsoft ECMAScripti standardiga kaasa, lõpetasid nad hiljem organisatsiooniga koostöö. Kuna sisuliselt oli JScriptist Internet Exploreri populaarsuse tõttu saanud uus standardkeel, jättis see ECMAScripti arengu seisma. Internet Exploreri populaarsuse aegadel kestis veebiarenduses programmeerimiskeele vaatepunktist stagnatsioon. Seda muutis 2004. aastal ilmunud uus brauser, Firefox, mille lõi Netscape'st välja arenenud uus firma Mozilla. Järgmisel, 2005. aastal ühines Mozilla Ecma standardiga. Tänu Firefoxi populaarsusele tõusis taas ECMAScripti olulisus, ent Microsoft oli siiski liiga suur, et oleks võinud seda standardi arendamisel ignoreerida. (Maricheva 2023)

2008. aastal tõi Google turule Chrome'i brauseri, mis oli teistest oluliselt kiirem. Kiirus saavutati tänu *just-in-time* koodikompileerimisele (JIT), mis kompileerib koodi programmi töö ajal vahetult enne, kui seda vaja läheb, mitte kogu koodi enne programmi käivitamist (freeCodeCamp 2020). Selle uuenduse mõju oli niivõrd suur, et 2008. aastal kohtusid firmad olukorra arutamiseks ja lepiti kokku ECMAScripti kasutamises. Selle kokkuleppe tulemus, ECMAScript 5, avalikustati 2009. aasta detsembris. Sellest ajast on ECMAScript pidevalt uuenenud ja JavaScript kui ECMAScripti spetsifikatsioonil põhinev programmeerimiskeel on kindlalt veebiarenduse standard. (Maricheva 2023)

Pranglimisrakenduse *front-end* kood on kirjutatud JavaScripti-põhises raamistikus React. Seega on projektis kasutatud JavaScript väga suures osas seotud Reactiga, mistõttu saab ka JavaScripti kasutamisest lähemalt lugeda järgnevast alapeatükist.

2.1.4. React

React on vabavaraline avatud lähtekoodiga JavaScripti teek (Herbert 2023). React on *front-end* raamistik, mida kasutatakse, et luua komponendipõhiseid kasutajaliideseid. See tähendab, et Reactiga tehtud kasutajaliidesed koosnevad väikestest, modulaarsetest osadest ehk komponentidest (ingl k *component*), mida omavahel kombineerides saab luua keerulisi lehekülgi. Kuigi React oli algselt loodud vaid veebilehekülgede loomiseks, saab sellega tänapäeval luua ka mobiiliäppe (raamistikuga *React Native*) või *full-stack* äppe (raamistikuga *Next.js*). (React 2023)

Kuigi veebiarenduse põhitehnoloogiaid on ainult kolm (HTML, CSS, JavaScript), ei tehta tänapäeval sisuliselt ühtegi suuremat veebilehte vaid nendega. Kuna veebilehed on ajas vaid keerukamaks muutunud, oleks tänapäeval ainult põhitehnoloogiate kasutamine aja- ja ressursikulukas väga pika koodi tõttu. Sellist koodi on ka raskem hallata, testida ja parandada. Seetõttu kasutataksegi veebiarenduses hulgaliselt teeke (ingl k *library*). Teegid on teiste programmeerijate loodud koodikogud, mida saab programmis vajaduspõhiselt kasutada (Meltzer 2023). Heaks näiteks on kasutaja autentimine: kuna turvaline autentimine on keeruline ja nõuab palju teadmisi ka näiteks krüptograafia valdkonnas, kasutatakse üldiselt autentimisteeke, mis teevad arendaja eest suure töö ära. Niimoodi ei pea iga arendaja looma enda koodibaasi (ingl k *code base*). Kuna enamik programmeerijaid ei ole

krüptograafiaeksperid, teeksid nad enda autentimissüsteemi luues arvatavasti ka vigu, mistõttu võib teekide kasutamine olla ka turvalisem. Seega on teekide eelisteks nii turvalisus, lihtsam haldus kui ka pidev uuendamise võimalus.

Kasutajaliidese loomine on veebiarenduses väga olulisel kohal, seetõttu on selleks ülesandeks palju erinevaid teeke, millest kõige populaarsem ongi React (Stack Overflow 2022). Reacti populaarsus põhineb mitmel omadusel. Üks olulisematest on fakt, et React on JavaScripti raamistik. See tähendab, et enamik arendajatest ei pea Reacti kasutamiseks uut programmeerimiskeelt õppima, vaid saavad kasutada juba õpitud keelt. Veebiarenduses, kus olulisel kohal on nii arendusaeg kui ka koodi hooldatavus, on tuntud programmeerimiskeele, eriti JavaScripti kasutamine tihti raamistiku valiku puhul eelduseks. Veel on oluline Reacti modulaarne arhitektuur, mis võimaldab koodi korduvalt kasutada. Väga palju annab juurde ka Reacti suur kogukond, mis aitab teeki õppida, toetab üksteist ja korraldab erinevaid üritusi ja seminare, mis muudab teegi ajakohaseks ja töökindlaks. (Soomro 2023)

React on raamistikuna suhteliselt uus ja lühikese ajalooga. Reacti lõi 2011. aastal Meta tarkvaraarendaja Jordan Walke. Esimest korda kasutati seda samal aastal Facebooki veebilehel, hiljem ka Instagramis. Reacti lähtekood avati 2013. aasta maikuus. Kaks aastat hiljem avaldati ka Reacti mobiiliraamistiku React Native lähtekood. Sellest ajast saati on React olnud pidevas arenduses, näiteks uuendati 2017. aastal teegi renderdamisalgoritmi. Tänaseks on Reactist saanud ülipopulaarne teek, mida kasutavad lisaks millionitele teistele lehtedele näiteks Facebook, Instagram, Netflix ja ChatGPT. (BuiltWith 2023; Herbert 2023)

Pranglimisrakenduse *front-end* kood on kirjutatud Reactis. React paistab teiste omasuguste seast silma just väga suure populaarsuse ja koodi mugava haldamise ning taaskasutamisega. Pranglimisrakenduse jaoks Reacti õppimine nõudis küll pisut aega, ent polnud ülemäära raske. Tänu autori kogemusele JavaScriptiga juba enne arenduse algust oli ka Reacti sellevõrra kergem omandada. Lisaks on selle projekti juures väga oluline rakenduse kiirus, nii veebilehe laadimisel kui lehesiseselt, näiteks uue tehte näitamisel kasutajale. React kasutab virtuaalset DOMi (ingl k *Document Object Model*, veebilehe struktuuripuu), mis võimaldab rakendusel iga muudatust testida ja seeläbi ka riske hinnata. Siiski on Reactil kui *front-end* raamistikul ka omad puudused. Kuna tegemist on võrdlemisi mahuka

raamistikuga on ka failisuurused selle võrra suuremad.

Pranglimisrakenduse arenduse kogemuse põhjal leiab autor, et raamistik on selliseks projektiks väga sobiv. React läheb hästi kokku ka välearendusest tulenevate põhimõtetega. Reacti modulaarne ja korduvkasutatav kood võimaldab kerge vaevaga luua väga paindlikke komponente. Heaks näiteks on ekraanil kuvatav klaviatuur, millega kasutaja vastuseid sisestab. Veebiarenduse ja -disaini põhimõtete kohaselt peaksid kõik klaviatuuri nupud võimalikult ühetaolised olema. Siiski on neil teatud määral vaja ka erineda, näiteks „Kustuta“ nupp on traditsiooniliselt punast värvi. React muudab sellise koodi kirjutamise ja haldamise väga lihtsaks. Tavalise HTML lahendusega peab arendaja kõikide nuppude jaoks välja kirjutama vähemalt HTML-struktuuri ja ka JavaScriptil põhineva töötamisloogika. Võimalike muudatuste tegemiseks tuleks seega käsitsi muuta igat nuppu loovat koodi, mis võib arusaadavalt muutuda väga aeganõudvaks ja üksluiseks tegevuseks, kus võib tekkida ka vigu, eriti kui on vaja sarnast tegevust teha mitmeid kordi. Kasutades Reacti saab aga nupu jaoks luua eraldi komponendi, millega saab ühest küljest kõiki nuppe korraga muuta, aga samas anda igale nupule ka vajaduse korral erinev väljanägemine ja funktsionaalsus.

2.1.5. Figma

Figma on veebirakendus kasutajaliideste disainimiseks. Figma on loodud disainerite reaalajas koostöö edendamiseks. See on võrdlemisi uus tööriist, mis on tarkvara disainimisprotsessi täielikult muutnud. Erinevalt teistest uurimistöös esitatud tehnoloogiatest ei nõua Figma programmeerimisoskust, vaid on puhtalt disainimisplatvorm. (Kopf)

Idee veebipõhisest disainiplatvormist tuli Evan Wallace'le ja Dylan Field'le juba 2011. aastal. Nende eesmärkideks oli muuta disain kättesaadavamaks rohkematele inimestele ja võimaldada mitmetel disaineritel töötada korraga ühe projekti kallal ilma versioonihalduseta. (Figma 2024a) 2015. aasta detsembris anti välja kutsepõhine Figma beetaversioon, järgmise aasta septembris esimene Figma avalik väljalase (Feldman 2022; Kopf). Esialgne tagasiside ei olnud positiivne. Figmat kirjeldati kui huvitavat, ent ebapraktilist ideed või isegi õudusunenägu. Reaalajaline koostöö arvati olevat disainerivaenulik. (Figma 2024a) Siiski leidis enamik disaineritest, et Figma võimaldab efektiivsemat ja kiiremat tööd, võimalust töötada väljaspool kontorit ja koos teistega. Seetõttu sai Figma kiiresti väga populaarseks. Tänapäeval kasutavad Figma võimalusi paljud suured tehnoloogiafirmad,

teiste hulgas Dropbox, GitHub ja Airbnb. (Figma 2024a)

Figma on rakendusena pidevalt arenemas. 2017. aastal avaldati Figma mobiiliversioon, millega on võimalik Figma prototüüpe mobiilsetel seadmetel katsetada (Google Play 2023). Lisaks saab Figmat kasutada ja eraldi rakendusena Windowsil, MacOSil ja Linuxil. 2021. aastal avaldati FigJam, digitaalne tahvel. FigJami kasutatakse tarkvara planeerimiseks, muuhulgas ka välearenduse põhimõtetel. (Figma 2024b)

Pranglimisrakenduse kasutajaliidese disain töötati välja Figmas. Selle ülesande jaoks valiti Figma tänu selle lihtsusele, suurele populaarsusele ja võimalusele tööd lihtsasti jagada. Pranglimisrakenduse kasutajaliidese disainimise kogemuse pealt vastab Figma suuresti nendele põhjendustele. Leidsin, et enda ideede teisendamine Figmasse oli intuitiivne ja mugav. Enne Figma kasutamist olin murelik, kas suudan selle kasutamise piisaval määral õigeks ajaks ära õppida, ent alustamine on väga lihtne, sest pole vaja midagi alla laadida. Figma kasutamine oli mugav ka välearenduse kontekstist vaadatuna, sest kõik ettepanekud, mida kasutajaliidese osas tehti, sai väga kiiresti visualiseerida, ilma programmeerimata.

2.2. *Back-end* tehnoloogiad

2.2.1. *PHP: Hypertext Preprocessor*

PHP on eeskätt veebiarenduseks mõeldud programmeerimiskeel. Lühendi PHP tähendus oli algselt *Personal Home Page*, aga selle ülisuure populaarsuse tõttu ka väga suurtes projektides tõttu vahetati tähendus rekursiivseks ja on nüüd *PHP: Hypertext Preprocessor*. Erinevalt näiteks JavaScriptist on PHP mõeldud just *back-end* koodi kirjutamiseks HTMLi sees. (PHP 2024c) Uuringufirma W3Techs andmetel kasutab 2024. aasta jaanuari seisuga PHPd 76,6% veebilehtedest, mille *back-end* keel on teada. Tuntuimad neist on näiteks Facebook, Microsoft ja Wikipedia. (W3Techs 2024)

Sarnaselt JavaScriptiga on PHP dünaamiliselt kirjutatud keel, kuigi valikuliselt saab andmetüüpe määrata (PHP 2024c). PHP kood algab algussildiga *<?php*. PHP koodi võlu seisneb selles, et seda saab kasutada HTMLi sees. Näiteks saab HTMLis pealkirja elemendi sisse kirjutada PHP koodi, mis tagastab sinna vajamineva teksti. Samuti kasutatakse PHPd tihti HTML koodist väljas ja *front-end*i jaoks kasutatakse muid tehnoloogiad, nagu ka pranglimisrakenduses. Lisaks nendele võimalustele saab PHPd edukalt jooksutada

otse terminalist. Väga lai kasutamisvõimalus erinevates valdkondades ka veebiarenusest väljaspool on PHPst teinud tuntud ja laialdaselt kasutatud keele.

PHP lõi 1994. aastal Taani programmeerija Rasmus Lerdorf. PHP oli algselt loodud väga lihtsaks ülesandeks - jälgida, mitu inimest Lerdorfi CV-d vaadanud on. (PHP 2024b) Aja möödudes lisas ta sinna aina rohkem võimalusi, näiteks integratsiooni erinevate andmebaasidega, kaasa arvatud MySQL, mida kasutatakse ka pranglimisrakenduses (JetBrains 2020). Sellest hetkest oli PHPst saanud lihtsamate veebirakenduste loomiseks sobilik keel. 1995. aasta juunis avalikustaski Lerdorf PHP esimese avaliku versiooni (Lerdorf 1995). Paar aastat hiljem sai PHP endale maskoti - sinise elevandi nimega ElePHPant, kellele on peale kirjutatud „PHP“ (JetBrains 2020; PHP 2024a). Sellest ajast saati on PHP olnud pidevas arengus ja paljukasutatud keel. Muuhulgas kasutati seda 2003. aastal WordPressi, maailma populaarseima CMSi (ingl k *Content Management System*) kirjutamiseks ja 2004. aastal Facebooki veebilehe loomiseks. Tänaseks on välja antud PHP 8. versioon, mis parandas keele jõudlust, aga lisas ka palju uusi tehnilisi võimalusi (Roose 2020).

Pranglimisrakenduse *back-end* on kirjutatud PHP-põhises raamistikus Laravel (osa 2.2.2). See tähendab, et sisuliselt kogu *back-end* on kirjutatud PHPs. Valik PHP kasuks tuli põhiliselt kooli serverist, mis seda keelt toetas. Nagu eelnevalt mainitud, saaks PHPd kasutada ka *front-end* koodis. Pranglimisrakendus kasutab seda siiski eksklusiivselt *back-end* keelena. Autor leidis rakendust planeerides, et PHP ei võimalda nii modulaarset, tänapäevast ja ka välearendusega sobituvat koodi, kui vaja läheb ja see tingis *front-end* koodi kirjutamise teistes keeltes (JavaScript + React). See otsus on tagasivaates põhjendatud, React integratsiooni väike keerukus kaalub kindlasti üles eelised, mis see võimaldab. Sellest olenemata on PHP osutunud heaks *back-end* keeleks, millel on tänu pikale ajaloole suurepärane ühilduvus erinevate serverite, andmebaaside ja raamistikega. Tänu raamistikele nagu Laravel on võimalik PHP koodi aegunumad osad asendada uuematega, mis hoiab PHP asjakohasena ka tänapäeval.

2.2.2. Laravel

Laravel on PHP-põhine veebirakenduste loomiseks mõeldud raamistik (Laravel 2024c). Laravel kasutab MVC (*model-view-controller*) arhitektuuri, mis muudab veebirakenduste loomise intuitiivseks ja mugavaks (Jalli 2022). Laraveli peetakse üheks tuntuimaks PHP

raamistikuks, eeskätt tänu raamistiku väga heale arendajakogemusele (DX, ingl k *developer experience*), mille loob lühike ja kergesti mõistetav kood (Ahmed 2023). Laravelil on ka suur tugi kogukonna poolt ning hulgaliselt laiendusi, mis teevad ka keerukamate rakenduste loomise kiireks ja lihtsaks (*ibid.*).

Back-end koodi kirjutamiseks kasutatakse enamasti programmeerimiskeelte, näiteks PHP, täienduseks raamistikke. Selleks on mitmeid põhjuseid. Raamistiku kasutamise põhiliseks eeliseks on lihtsustatud intergratsioon teiste tehnoloogiatega, näiteks andmebaasid ja autentimine. (Tahiroğlu 2023) Laravel on andmebaasidega ühendumise teinud võimalikult lihtsaks. Raamistik toetab ise paljusid populaarseid andmebaase, näiteks MySQL ja MariaDB, mille konfiguratsiooniks kulub kõigest paar rida koodi (Laravel 2024a). Pärast andmebaasi ühendamist rakendusega aitab Laravel sellega suhelda läbi laienduse Eloquent ORM (*Object – relational mapping*), mis võimaldab arendajal kirjutada loetavat koodi, mis ei kasuta andmebaaside populaarseimat keel SQL (osa 2.2.3). Kõik see lihtsustab rakenduse arendamist ja seega vähendab ressursikulu nii aja kui raha poolest, kuivõrd andmebaasid on tänapäeva veebirakendustes sisuliselt asendamatud (Tahiroğlu 2023).

Juba pranglimisrakenduse arendamise algul teadsid arendajad, et soovivad rakenduse *back-end* koodi luua raamistiku abil. Laravel polnud aga esimene valik. Lihtsuse ja väikese suuruse tõttu oli esialgu kavas kasutada raamistikku CodeIgniter 4 (CodeIgniter 2024). Selle raamistikuga esines aga mitmeid probleeme, näiteks raskendatud integratsioon Reacti-põhise kasutajaliidesega ja sisseehitatud autentimisvõimaluse puudus. Raamistiku vanuse tõttu olid eelistatud vanemad tehnoloogiad, mis aga oleksid negatiivselt mõjutanud pranglimiseks vajaminevaid nõudeid nagu kiirust. (*ibid.*) Seetõttu valiti raamistikuks Laravel, mis on märksa uuem, pidevamalt arendatud ja suurema kogukonna toega (Jalli 2022; JetBrains 2020). Arenduse käigus saadud kogemus kinnitab väidet, et Laravel on arendajale väga mugav. Nii eelnevalt mainitud andmebaasi integratsioon kui näiteks autentimine on tehtud lollikindlaks. Heaks näiteks on OAuth süsteemide (sisselogimine Google, Facebook, Apple jne konto abil) integreerimine rakendusse. Selle asemel, et iga teenusepakkuja jaoks eraldi väga palju koodi kirjutada, mis võib alustavate arendajate korral ka turvaauke tekitada, pakub Laravel laiendust Socialite, mille abil saab iga teenusepakkuja jaoks piirduda paarirealise sisselogimiskoodiga (Laravel 2024b). Kasutades Inertiat on lihtne ühendada Laravelis kirjutatud *back-end* React *front-end* vaadetega (Inertia.js 2024).

Kõigest ülaltoodust johtuvalt on Laravel tänapäevane, väga mitmekesiste võimalustega ja mugav raamistik.

2011. aastal oli populaarseim PHP raamistik CodeIgniter. Kuigi seda kasutati väga palju, ei olnud sellel kõiki soovitud võimalusi, näiteks autentimist (Degni 2023). CodeIgniterile tugeva alternatiivi pakkumiseks andis Taylor Otwell 2011. aastal välja endaloodud Laravel 1 raamistiku (Degni 2023; JetBrains 2020). Otwelli eesmärgiks oli luua arendusprotsessi lihtsustav, lihtsastiloetava koodiga ja tugevate võimalustega raamistik (Degni 2023). Tuginedes raamistiku arendamisel samal ajal välja antud paketihaldurile Composer ja PHP arendajatele juba tuntud komponentide teegile Symfony, tegi Otwell Laraveli tulevikukindlaks, aga samas uudseks raamistikuks, mille süntaksi lihtsust ei andud teiste omasugustega võrreldagi (Degni 2023; JetBrains 2020). Tänaseks on Laravelist saanud üks enimkasutatud PHP raamistik, millest on tänaseks ilmunud juba kümnes suur versioon, Laravel 10 (Ahmed 2023).

2.2.3. MySQL

MySQL on avatud lähtekoodiga andmebaasihalduse süsteem (DBMS, *database management system*) (Domantas 2024). Kuna MySQLiga saab hoiustada vaid relatsioonandmeid, kasutatakse selle kohta ka terminit relatsioonbaasihalduse süsteem (*ibid.*). MySQL on arendajate seas väga populaarne ja seda kasutavad nii väikesed arendustiimid kui maailma suurimad tehnoloogiafirmad, näiteks Spotify, YouTube ja GitHub (MySQL 2024). MySQL andmebaasides on andmed hoiustatud tabelites, iga andmestruktuur eraldi veerus (ingl k *column*). MySQL pakubki raamistikku selliste andmebaaside loomiseks, haldamiseks ja kasutamiseks. (Domantas 2024; W3Schools 2024a)

Nagu nimigi ütleb, kasutab MySQL andmebaas SQL keelt. SQL (*Structured Query Language*) on omataoliste seas vaieldamatult populaarseim andmebaasikeel (GeeksForGeeks 2024). Samuti on SQL ISO (Rahvusvaheline Standardiorganisatsioon, ingl k *International Organization for Standardization*) standard juba aastast 1987 (W3Schools 2024b). SQL keel koosneb päringutest, mida kasutatakse andmebaasi loomiseks, lugemiseks, uuendamiseks ja kustutamiseks (CRUD, loo-loe-uuenda-kustuta, ingl k *create-read-update-delete*) (GeeksForGeeks 2024). Erinevus SQLi ja MySQLi vahel ongi asjaolu, et SQL on programmeerimiskeel ja MySQL on konkreetne andmebaasihalduse süsteem, mis seda keelt

kasutab. Seega saavad SQL keelt kasutada ka paljud teised haldussüsteemid, näiteks SQL Server ja PostgreSQL. (W3Schools 2024b)

MySQL esimese versiooni lõid Michael Widenius, David Axmark ja Allan Larsson ning see avalikustati 1995. aasta maikuus (Rieuf 2016). Nende eesmärgiks oli luua andmehaldussüsteem, mida saaks kasutada nii tavakasutajad kui suured firmad. Kuigi esmalt oli MySQL suletud lähtekoodiga, mida omas Rootsi firma MySQL AB, avati juba 2000. aastal lähtekood kõigile. (*ibid.*) Veidi hiljem, 2001. aasta detsembris sai MySQL ametlik maskott, delfiin, endale nimeks Sakila (Andrade 2019). Sellest ajast saati kuni tänapäevani on MySQL olnud üks populaarsemaid andmebaasihalduse süsteeme maailmas (Statista 2023).

Kuna Tallinna Reaalkooli serverid toetasid MySQLi, ei olnud haldussüsteemi valik kuigi keeruline. Et arendajatel puudus pea igasugune kogemus SQL andmebaasidega, oli võrreldes teiste kasutatud tehnoloogiatega selle õppimise protsess siiski pikem. Tegelikkuses ei nõudnud pranglimisrakenduse arendamine aga edasijõudnud arusaama SQL keelest. Laraveli Eloquent ORM abil muutus andmebaas abstrakseks taustaks, sest SQL päringuid polnud vaja kirjutada. Sellist mõju suurendas ka phpMyAdmin tarkvara kasutamine, mis MySQL andmebaasid veebirakenduses visualiseerib. Sellest hoolimata omandasid arendajad töö käigus põhilise arusaama SQL ja MySQL tööpõhimõtetest ja koodist. Kuivõrd MySQL andmebaasis on hoiustatud kõik pranglimisrakenduse kasutajad ja nende andmed, mängib see tehnoloogia rakenduses väga olulist rolli.

3. Välearenduse kasutamine pranglimisrakenduse arenduses

3.1. Välearenduse põhimõtete kasutamine pranglimisrakenduse loomisel

Uurimistöö esimeses osas on välja toodud välearenduse neli põhimõtet. Selles alapeatükis analüüsitakse pranglimisrakenduse arenduse vastavust nendele, samuti erinevate välearenduslike meetodite kasutamist.

Autori hinnangul järgib pranglimisrakenduse arendus kõiki välearenduse põhimõtteid. Esimese punkti („Hindame enam inimesi ja nende suhtlemist kui protsesse ja arendusvahendeid.“) täitmisega alustasid rakenduse arendajad kohe rakenduse arenduse alguses 2023. aasta suvel. Selle asemel, et kiiruga arendust alustada, kohtusid arendajad Tallinna Reaalkooli nii matemaatika- kui ka algklasside õpetajatega. Kohtumise eesmärgiks oli saada algset sisendit õpetajate (kliendi) ootustest, tulevikusoovidest ja funktsionaalsustest, mis neile olulised või mitteolulised on. Selle ja kõigi järgnevate kohtumiste aluseks oligi välearenduse esimene ja kolmas põhimõte, millega haakus ka neljas põhimõte. Kohtumised õpetajatega olid produktiivsed ja tingisid tagasivaates kliendi soovidele vastavama rakenduse kui ilma kohtumisteta võimalik olnuks, mis ongi ju arendajate lõppeesmärk.

Pranglimisrakenduse arendus on samuti kooskõlas välearenduse teise eesmärgiga, „[h]indame enam töötavat tarkvara kui täiuslikku dokumentatsiooni“. Pranglimisrakenduse arenduse käigus kirjutati dokumentatsiooni, näiteks loodi veebilehekülg erinevate mängutüüpide tutvustamiseks ja soovituste jagamiseks, samuti tehnilise poole kohta, näiteks piisavalt kommenteeritud koodi. Samas hoiduti edukalt ohu eest, et dokumentatsiooni koostamisele kuluv aeg pärsiks pranglimisrakenduse valmimist oluliselt. Autor leiab, et sellistes arendustöödes on oluline leida tasakaalukoht täielikult dokumenteerimata rakenduse/koodi ja ilmselge nn „üledokumenteerimise“ vahel. Vale on väita, et dokumentatsiooni koostamine on arendajatele lihtsalt aeganõudev ja tüütu kohustus, kuna dokumentatsioon aitab

iseenda ja teiste arendajate koodis paremini navigeerida ja mõista selle funktsionaalsust ja tegelikku tahet. Samuti tuleb mõelda tulevikule: on oluline, et rakenduse potentsiaalsed uued arendajad saaksid koodist kiire ülevaate ja mõistaksid, miks midagi just niimoodi lahendatud on. Samas on ilmne, et selline pranglimisrakenduse projekt, kus tähtajaks on valmis vaid põhjalik dokumentatsioon funktsionaalsusest ja disainifailid, on läbikukkunud, sest tegelikku väärtust õpilastele ja õpetajatele ei ole. Sellisesse lõksu sattumise eest hoiatabki välearenduse teine põhimõte, mida rõhutavad ka Shore ja Warden (Shore, Warden 2008b). Oluline on mõista, et välearendus ei eita dokumentatsiooni vajalikkust, vaid rõhutabki tasakaalu olulisust ja töötava tarkvara ülimuslikkust dokumentatsiooni suhtes.

Välearenduse kolmas põhimõte on „[h]indame enam koostööd kliendiga, kui läbirääkimisi lepingute üle“. Uurimistöö autor leiab, et suures osas on see põhimõte konkreetsele projektile kohaldamatu, kuivõrd lepingut autori ja kliendi vahel sõlmitud pole[1]. Saab aga analüüsida selle punkti mõtet: selmet igas väiksemaski punktis eelnevalt täpselt kokku leppida, millele kuluks palju väärtuslikku aega, anti arendajatele võrdlemisi suur iseseisvus uue funktsionaalsuse loomisel. Nii ei pidanud autorid näiteks uute mängutüüpide loomisel seda eelnevalt õpetajatega kooskõlastama, küll aga näidati neid hiljem õpetajatele, küsiti nende arvamust ja tehti vastavad muudatused. Heaks näiteks on mängutüüp „Kujundid“, mille ideele tuli üks arendjatest ja mille arendajad omaalgatuslikult valmis tegid. Õpetajatepoolne tagasiside aitas arendajatel hiljem mängu veelgi täiustada, lisades võimaluse erineva värvi ja suurusega kujundite loendamiseks. Seetõttu leiab autor, et niivõrd, kui seda on mõeldav konkreetse projekti korral, saab välearenduse kolmandat põhimõtet lugeda täidetuks.

Samuti on täidetud ka neljas välearenduse põhimõte – „[h]indame enam muudatustega hakkamasaamist kui algse plaani järgimist“. Selles punktis peitubki välearendus kui pidev, muudatusi tervitav protsess. Pranglimisrakenduse arenduse käigus muutusid paljud rakendusele vajalikud tegurid. Ilmekaimaks näiteks on rakenduse nimi ja logo. Kuna nimi ja logo valiti koolisisesel konkursil, mis toimus rakenduse arenduse üsna hilises staadiumis, vajas kasutajaliides nimele, aga eriti just logole kohatäiteid (ingl k *placeholder*). Alles võidutööde väljakuulutamisel said arendajad vastavad elemendid rakendusse lisada,

[1] Võlaõigusseaduse § 9 lg-d 1, 3

milles väljendubki üks välearenduse kasutuskoht. Osas 1.2 kirjeldatud koskmudeli kohaselt oleksid kõik disainielemendid, sh nimi ja logo, pidanud valmima juba enne rakenduse programmeerimise algust. Samas on igaühele arusaadav, et õpilaste arendatud projekti jaoks, mille õnnestumine ei olnud kindel, ei olnud mõistlik konkurss nii vara läbi viia. Lisaks ei olnud selleks hetkeks veel rakenduse vaadete disain loodud, ilma milleta puudus õpilastele igasugune inspiratsiooniallikas, seega poleks pakutud tööd suure tõenäosusega ka rakenduse välimusega kuigi hästi kokku sobinud. Sellisest käitumisest tingitud sobivamad disainelemendid võib lugeda ka välearenduse eeliste hulka, millest on täpsemalt kirjutatud järgmises alapeatükis.

Kuigi kogu välearendus baseerub neile neljale põhimõttele, ei tähenda nende järgimine tingimata välearenduse *de facto* kasutamist. Tegelik tarkvaraarenduse maailm on märksa komplekssem. Kasutatakse välearendust lihtsustavamaid meetodeid ehk raamistikke, mille põhimõtted ja näited on toodud esimeses peatükis (osas 1.1). Samuti soovitavad Shore ja Warden kasutada juba välja töötatud meetodeid, mitte üritada iseenda raamistikku luua (Shore, Warden 2008c). Järgnevalt analüüsib autor välearenduslike meetodite kasutamist pranglimisrakenduse arendusel.

Üks olulisemaid välearenduslikke võtteid, mida pranglimisrakenduse arenduses kasutati, on paarisprogrammeerimine (Andmekaitse ja infoturbe leksikon *s.v.* pair programming). Paarisprogrammeerimine on eelkõige välearenduses, ent ka näiteks õppetöös kasutatud arendusmeetod, milles kaks programmeerijat kasutavad arendamiseks ühte arvutit. Üks programmeerijatest on „juht“, kes koodi arvutisse kirjutab, teine „kaardilugeja“, kelle ülesandeks on mõelda välja uusi lahendusi ja jälgida, et koodi ei satuks vigu (Nurk 2015). Pranglimisrakenduse arenduses mängis paarisprogrammeerimine suurt rolli. Et PHP oli mõlema arendaja jaoks uus programmeerimiskeel, tuli probleemidele kahekesi lahenduse otsimine nii ajaliselt kui õppimise vaatepunktist kasuks. Lisaks saavad arendajad omavahel kiiresti mõtteid vahetada ja valida tuleviku seisukohast sobilikema. Eriti tuli see eelis välja andmebaasi struktuuri loomisel, kus arendajad leidsid kasutaja eelistuste salvestamiseks kõige efektiivsema lahenduse justnimelt koostöö tulemusena. Sarnaseid eeliseid toob enda bakalaureusetöös välja ka Nurk (*ibid.*). Tema töös välja toodud paarisprogrammeerimise negatiivsed küljed – põhiliselt isiksusetüüpide mittesobivus paarisprogrammeerimisega – õnneks pranglimsrakenduse arenduse käigus ette ei tulnud (*ibid.*).

Samuti kasutasid uurimistöö arendajad arenduse käigus kanban-meetodit. Kanban (看板, jaapani keeles märk ja tahvel) on välearenduses laialdaselt kasutatud tööhalduse mudel. (TTÜ IT kolledž 2020) Selle meetodi kasutamiseks koostatakse kanban-tahvel, kuhu saab lisada ülesandeid ehk kaarte. Kõige lihtsamad ja tavapärasemad tahvlid koosnevad kolmest tulbast – „pole alustatud“, „praegu töös“ ja „tehtud“. Tüüpiliselt alustavad kõik kaardid esimesest tulbast ja eesmärk on need läbi tulpade viia viimasesse. Välearendusega seob seda meetodit esimene ning neljas põhimõte ja välearendusele omane tsüklilisus, sest tahvlilt lisatakse ja eemaldatakse kaarte pidevalt. Lisks on sellise meetodi eeliseks töömahu visualiseerimine. (Martins 2024; TTÜ IT kolledž 2020)

Kanban ei ole loodud tarkvaraarenduseks, alguse sai see hoopis varem. Meetodi mõtles 1940. aastate lõpus välja autofirma Toyota insener Taiichi Ohno. Alguses oligi see mõeldud Toyota tootmisprotsessi lihtsustamiseks. Selle asemel, et toota autosid vastavalt eelduslikule nõudlusele, sai Ohno süsteemi abil toota autosid vastavalt tarbijate tegelikule vajadustele. Alles 21. sajandi algul sai just tänu sellele eelisele kanbanist nõutud meetod tarkvara arendamisel. (Martins 2024)

Pranglimisrakenduse arendusel kasutati selle meetodi rakendamiseks virtuaalset kanban-tahvlit veebikeskkonnas GitHub. Lisaks eeltoodud kolmele põhitulbale kasutati ka neljandat tulpa „tulevik“, mille eesmärk oli kaardistada võimalusi, mis võiksid rakenduses olla, ent mille arendus kas polnud veel võimalik või oli ajakulu tõttu äärmiselt ebaotstarbekas. Selle meetodi puhul hindasid arendajad (visuaalset) haldusvõimalust, samuti süsteemi kui motivatsiooniallikat, sest iga kaardi tahvlil edasilükkamine oli edasiminek rakenduse arengus. Allikates on eelistena veel välja toodud näiteks suurem omavaheline suhtlemine, mis samuti välearendusega haakub (esimene põhimõte), kuid tänu sama mõju avaldavatele teistele meetoditele (paarisprogrammeerimine) ei märganud arendajad seda kanbani puhul (TTÜ IT kolledž 2020). Samuti leiab autor, et kanban-tahvli pideva täitmise kohustus võib vähendada selle efektiivsust (*ibid.*).

Oluliseks tuleb lugeda ka arendajate pidevat õpetajatega kohtumist, samuti arenduse hilisemas faasis toimunud katsetamistunde õpilastega. Kuigi täpselt sellist meetodit, mis mainitud samme nõuaks, ei ole, lähevad need siiski kokku välearenduse kõigi põhimõtetega ja on seega kooskõlas välearenduse mõttega. Õpetajate ja õpilastega kohtumised olid arenduse jaoks hädavajalikud ja andis tootele ja arendajatele nii tagasisidet kui konkreetseid

ettepanekuid uute võimaluste jaoks.

Õpetajatega kohtumised andsid arendajatele ideid tulevikuarenduste jaoks, samuti uusi funktsionaalsusi ja ettepanekuid vigade lahendamiseks. Pranglimisalgoritm (programm, mis genereerib tehteid) oli arenduse alguses vigaderohke, tasemed läksid liiga kiiresti liiga raskeks. Probleemi lahendamiseks pakkusid õpetajad Riin Saar ja Helli Juurma erinevate mängutüüpide (liitlahutamine, korrujagamine) jaoks välja tabelid, kus oli kirjas tasemes lubatavate arvude miinimum- ja maksimumväärtused. See aitas vähendada „hüpete“ tekkimist tehete vahel, kus järgmine tehe on märgatavalt keerulisem. Samuti andsid õpetajad Riin Saar, Helli Juurma ja Jaanika Lukk ja Hanna Britt Soots väärtuslikku tagasisidet kasutajaliidesele. Näiteks tuli just õpetajatel idee luua võimalus paar korda mängu jooksul tehe vahele jätta, mis on rakenduses juba üsna arenduse algusest saati sees olnud. Samuti tuli õpetajatelt mängu vaates oleva „Katkesta“ nupu mõte. Funktsionaalsuse poole pealt oli õpetajate olulisim ettepanek luua nn tähega tasemed ehk tasemed, mis on mõeldud tugevamatele pranglijatele ja mis võivad tavamängijale olla üle jõu käivad. Samuti tuli Riin Saarel ja Helli Juurmal idee kujundite kokkulugemise mängutüüpi lisada võimalused eri värvi ja suurusega kujundite jaoks. (Hellat 2024)

Ka kohtumised õpilastega olid produktiivsed ja arendajad said häid mõtteid uute võimaluste loomiseks. *Front-endi* poolest oli olulisim ettepanek kahtlemata kasutaja võimalus valida endale meeldiv põhivärv (ingl k *primary color*). Algselt oli rakenduse põhivärv tumeroheline, ent pärast kohtumisi lisas autor kiiresti võimaluse kasutajal valida kümne põhivärvi vahel, mis vastaksid võimalikult paljude maitsele. Hilisemates kohtumistes õpilastega sai autor tagasisidet, milliseid värve enim soovitakse, nii lisandus näiteks punane värv. Kasutajakogemuse (UX) poolest oli märkimisväärne õpilaste soov automaatselt salvestavate seadete lisamiseks, mis on küll serverile rohkem koorvam, ent peale kohtumisi mõistsid arendajad, et kasutajamugavus kaalub selle üles. Õpilaste suurim roll oli aga vigade leidmine programmis, selle ülesande täitsid nad suurepäraselt. Esimeselt kohtumiselt avastati näiteks viga programmis, mis juhtus lehe automaatsel tõlkimisel brauseri poolt. Ilma õpilastepoolse testimiseta on ebatõenäoline, et taoline viga oleks enne rakenduse avalikku väljalaset leitud, mis näitab ilmekalt nende kohtumiste vajalikkust. (*ibid.*)

3.2. Välearenduse eelised pranglimisrakenduse arendamise näitel

Välearenduse suur populaarsus on tingitud paljudest eelistest, mis sellel meetodil teiste sarnastega võrreldes on. Järgnevas alapeatükis käsitletakse teiste arendajate poolt välja toodud peamisi välearenduse eeliseid ja analüüsitakse neid pranglimisrakenduse arenduskogemuse toel.

Välearenduse üheks olulisimaks eeliseks tuuakse välja kliendirahulolu (Kulshrestha 2019; University of Minnesota 2022). Kuna arendus toimub tsükliliselt, saab klient projekti seisu pidevalt näha. Kui pärast tsüklit kliendile näidatud projekti mingi aspekt talle ei sobi, saab vastavaid muudatusi kerge vaevaga sisse viia (Kulshrestha 2019). Pranglimisrakendust arendades viidi läbi kohtumisi Tallinna Reaalkooli matemaatikaõpetajate Riin Saare ja Helli Juurmaga, kes selles kontekstis sobituvad kliendi definitsiooni alla. Ei ole võimalik väita, et pranglimisrakenduse arendamine välearendusega oleks kliendirahulolu suurendanud, sest puudub teadmine, milline see oleks olnud ilma välearendust kasutamata. Sellegipoolest leiab autor, et kliendirahulolu selle projekti arenduse käigus oli kõrge. Kohtumised matemaatikaõpetajatega olid produktiivsed ja sisutihedad, anti tagasisidet tehtud tööle ning tehti ettepanekuid edasiseks. Seega võib väita, et kuigi puudub võrdlus teiste arendusmeetoditega, püsis kliendirahulolu selle projekti arenduse käigus välearendusele omaste meetodite kasutamise tõttu kõrge.

Forbes toob enda 2015. aasta artiklis „*The Benefits Of Using Agile Software Development*“ välearenduse üheks eeliseks välja arendusmeeskonna suurema eesmärgikindluse (Forbes Technology Council 2016). Nende sõnul loob välearendus kogu meeskonnale ühtsustunde ja soovi töötada koos ühise eesmärgi poole. Samuti leitakse, et selliselt motiveeritud meeskonnad töötavad kiiremini kui need, kellele on ette antud jäigad tähtajad. (*ibid.*). Selline mõju ilmneb ka pranglimisrakenduse arendamisest saadud kogemusest. Autor leiab, et oluliseks põhjuseks, miks pranglimisrakenduse arendus õnnestus, on arendajatevaheline tihe koostöö. See tagab, et arendajatel on projekti tulevikust ühine arusaam, ilma milleta võib projekt ebaõnnestuda, kuna keskendutakse liiga erinevatele aspektidele. Välearenduses tihti kasutatud paarisprogrammeerine võimaldas pranglimisrakenduse arendust kiirendada, leida lahendusi tekkinud komistuskividele ja saada enda koodile uut perspektiivi. Vähetähtis pole ka paarisprogrammeerimise hariduslik mõju – kahekesi ühel ajal koodi kirjutamine võimaldab üksteiselt õppida. Kuigi pranglimisrakenduse arendustiim oli väike,

vaid kaheliikmeline, oli projektiga seotud märksa rohkem inimesi ja seetõttu leiab autor, et eelnevad hüved kehtivad ka suuremate arendustiimide korral. Johtuvalt ülaltoodud põhjendustest on välearenduse kasutamine toonud kaasa kiirema, vähemate vigadega koodi ja samas andnud arendajatele võimaluse enesetäiendamiseks.

Pranglimisrakenduse arenduse jooksul oli oluline ka projekti pidev areng. Erinevalt koskmudelist, kus kõik nõuded, tehnoloogiad, liidese disainid ja dokumentatsioon peab enne rakenduse elluviimist täielikult valmis olema, ei ole välearenduses nii jäiku reegleid (Lutkevich, Lewis 2022). See tingis kokkuvõttes parema rakenduse, kuna vajalikke uuendusi sai luua töö käigus. Heaks näiteks on mängutüübid nagu jaguvusseadused, murru lihtsustamine ja kujundite loendamine, mis tulid mõttesse rakenduse arenduse käigus. Koskmudelit kasutades oleksid need rakendusest välja jäänud või oleks pidanud kogu protsessi uuesti alustama, ent tänu välearendusele oli võimalik muudatused kliendiga kooskõlastada ja ellu viia. Ühtlasi kiirendab selline metoodika arendust kui protsessi, sest ei pea kulutama aega dokumentatsiooni ja kõikide vaadete disainimisele. Projekti ebaõnnestumise korral oleks sel juhul tehtud palju tööd ilma, et sellel reaalset väärtust oleks.

Standish Groupi 2015. aasta raportis uuriti muuhulgas projektide õnnestumismäära välearenduse ja koskmudeliga (Standish Group 2015). Leiti, et õnnestunud projekti tõenäosus on välearendust kasutades enam kui kolm korda suurem, kui koskmudeliga (vastavalt 39% ja 11%). Lisaks on projekti ebaõnnestumise tõenäosus välearendusega umbes kolm korda väiksem (9% ja 29%). (*ibid.*) Kuna pranglimisrakendus on projekt, mis õnnestumise korral oleks kasutatav kõigi Tallinna Reaalkooli (potentsiaalselt ka teitse koolide) õpilaste ja õpetajate poolt ja samas seotud ka mitme uurimistööga, oli oluline minimeerida ebaõnnestumise võimalust. Uurimistöö kirjutamise ajaks on pranglimisrakendus suures osas valmis ja päriselus kasutatav. Sarnaselt kliendirahuloluga pole loomulikult võimalik ühe näite põhjal kirjutada võrdusmärki välearenduse ja õnnestunud projekti vahele. Samas on autori hinnangul pranglimisrakenduse õnnestumise taga tegurid, mis tulenevad otseselt välearenduse põhimõtetest. Pranglimisrakendus õnnestus tänu tugevale koostööle arendajate ja matemaatikaõpetajate vahel, mis ongi üks välearenduse alussammastest. Samuti on autori jaoks märkimisväärne arendajatevaheline pidev koostöö, mille tagas muuhulgas tihe suhtlus ja paarisprogrammeerimise järjepidev kasutamine. Ülaltoodust lähtuvalt võib öelda, et ehkki puuduvad ümberlükkamatud tõendid seosest välearenduse ja

projekti õnnestumise vahel, on uuringute tulemused kombineeritult pranglimisrakenduse arenduskogemusega selle teooria kõnekad pooltargumendid.

Välearenduse populaarsus on tingitud paljudest eelistest, millest enamik leidsid kinnitust ka pranglimisrakendust arendades. Põhiline kasu meetodi kasutamisel tuleb tihedast koostööst nii arendajate endi kui ka arendajate ja kliendi vahel. Kuigi kõiki allikates välja toodud eeliseid ei olnud võimalik pranglimisrakenduse arendusega kohaldada, on siiski ilmne, et vähemalt mingil määral on välearenduse kasutamine arendust kiirendanud või koodi kvaliteeti tõstnud.

Kokkuvõte

Kasutatud materjalid

Agile Alliance. (2023) Agile Glossary. Loetud: https://www.agilealliance.org/agile101/agile-glossary/, 02.01.2024.

Ahmed, M. (2023) Best PHP Frameworks in 2023. Loetud: https://medium.com/@moeezahmed.k/best-php-frameworks-in-2023-c896ce046a71, 18.02.2024.

Andmekaitse ja infoturbe leksikon. Agile development. Loetud: https://akit.cyber.ee/term/2148-agile-development, 19.10.2023.

Andmekaitse ja infoturbe leksikon. Pair programming. Loetud: https://akit.cyber.ee/term/16639-paarisprogrammeerimine, 10.03.2024.

Andrade, P. (2019) Name the MySQL Dolphin Contest! Loetud: https://www.linkedin.com/pulse/name-mysql-dolphin-contest-pedro-andrade-, 18.02.2024.

Ayebola, J. (2023) CSS Frameworks vs Custom CSS – What's the Difference? Loetud: https://www.freecodecamp.org/news/css-frameworks-vs-custom-css/, 03.01.2024.

Basili, V., Larman, C. (2003) Iterative and incremental developments. a brief history. Loetud: https://ieeexplore.ieee.org/document/1204375, 02.01.2024.

Beck, K., Beedle, M., Bennekum, A. van, Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D. (2001) Agiilse tarkvaraarenduse manifest. Loetud: http://agilemanifesto.org/iso/et/manifesto.html, 19.10.2023.

Bose, S. (2023) Top 5 CSS Frameworks for Developers and Designers. Loetud: https://www.browserstack.com/guide/top-css-frameworks, 03.01.2024.

BrainHub. (2023) Why Node JS? 7 Reasons Famous Companies Choose It. Loetud: https://brainhub.eu/library/why-node-js, 22.11.2023.

BuiltWith. (2023) React Usage Statistics. Loetud: https://trends.builtwith.com/javascript/React, 03.12.2023.

Cockburn, A., Highsmith, J. (2005) Declaration of Interdependence. Loetud: https://web.archive.org/web/20180127094805/http://www.pmdoi.org/, 02.01.2024.

CodeIgniter. (2024) CodeIgniter. Loetud: https://www.codeigniter.com/, 18.02.2024.

Conrad, E. (2011) Waterfall Model - an overview. Loetud: https://www.sciencedirect.com/topics/computer-science/waterfall-model, 06.03.2024.

Copeland, L. (2001) Extreme Programming. Loetud: https://www.computerworld.com/article/2585634/extreme-programming.html, 06.03.2024.

Degni, R. (2023) Laravel: Celebrating 12 Years of Powering PHP Development. Loetud: https://www.codemotion.com/magazine/languages/laravel-best-practices/, 18.02.2024.

Denning, S. (2012) Is Apple Truly 'Agile'? Loetud: https://www.forbes.com/sites/stevedenning/2012/02/03/is-apple-truly-agile/, 02.01.2024.

Denning, S. (2015) Surprise: Microsoft Is Agile. Loetud: https://www.forbes.com/sites/stevedenning/2015/10/27/surprise-microsoft-is-agile/, 02.01.2024.

Domantas, G. (2024) What Is MySQL and How Does It Work. Loetud: https://www.hostinger.com/tutorials/what-is-mysql, 18.02.2024.

Dybå, T., Dingsøyr, T. (2008) *Empirical studies of agile software development: A systematic review.* Loetud: https://www.researchgate.net/publication/222827396_Empirical_studies_of_agile_software_development_A_systematic_review, 01.01.2024.

ECMA. (1997) ECMAScript: A general purpose, cross-platform programming language. Loetud: https://ecma-international.org/wp-content/uploads/ECMA-262_1st_edition_june_1997.pdf, 06.03.2024.

ECMA. Organisation. Loetud: https://ecma-international.org/organisation/, 06.03.2024.

Feldman, D. (2022) How did Figma Succeed? A Brief History. Loetud: https://dfeldman.medium.com/how-did-figma-succeed-a-brief-history-b816492da146, 06.03.2024.

Figma. (2024) About Figma, the collaborative interface design tool. Loetud: https://www.figma.com/about/, 03.01.2024.

Figma. (2024) The Online Collaborative Whiteboard for Teams. Loetud: https://www.figma.com/figjam/, 03.01.2024.

Flynn, J. (2022) 16 Amazing Agile Statistics [2023]: What Companies Use Agile Methodology. Loetud: https://www.zippia.com/advice/agile-statistics/, 01.01.2024.

Forbes Technology Council. (2016) The Benefits Of Using Agile Software Development. Loetud: https://www.forbes.com/sites/forbestechcouncil/2016/05/09/the-benefits-of-using-agile-software-development/, 17.01.2024.

freeCodeCamp. (2020) Just in Time Compilation Explained. Loetud: https://www.freecodecamp.org/news/just-in-time-compilation-explained/, 06.03.2024.

GeeksForGeeks. (2023) Frontend vs Backend. Loetud: https://www.geeksforgeeks.org/frontend-vs-backend/, 10.09.2023.

GeeksForGeeks. (2023) History of JavaScript. Loetud: https://www.geeksforgeeks.org/history-of-javascript/, 06.03.2024.

GeeksForGeeks. (2024) SQL Tutorial. Loetud: https://www.geeksforgeeks.org/sql-tutorial/, 18.02.2024.

Google Play. (2023) Figma – prototype mirror share. Loetud: https://play.google.com/store/apps/details?id=com.figma.mirror, 03.01.2024.

Hellat, J. J. (2024) Pranglimisrakenduse programmeerimine ja piloteerimine [uurimistöö]. Tallinn: Tallinna Reaalkool.

Herbert, D. (2023) What is React.js? Uses, Examples, & More. Loetud: https://blog.hubspot.com/website/react-js, 06.03.2024.

Inertia.js. (2024) Inertia.js - The Modern Monolith. Loetud: https://inertiajs.com/, 18.02.2024.

Jalli, A. (2022) What Is Laravel? Loetud: https://builtin.com/software-engineering-perspectives/laravel, 18.02.2024.

JetBrains. (2020) 25 Years of PHP History. Loetud: https://www.jetbrains.com/lp/php-25/, 17.02.2024.

Kopf, B. The Power of Figma as a Design Tool. Loetud: https://www.toptal.com/designers/ui/figma-design-tool, 06.03.2024.

Krill, P. (2014) Scrum co-inventor: Agile can lower risk, but it won't tell you how to code. Loetud: https://www.infoworld.com/article/2138580/scrum-co-inventor-agile-can-lower-risk-but-it-wont-tell-you-how-to-code.html, 06.03.2024.

Kulshrestha, S. (2019) What Is Agile Methodology — Know the What and How? Loetud: https://medium.com/edureka/what-is-agile-methodology-fe8ad9f0da2f, 16.01.2024.

Laravel. (2024) Database: Getting Started. Loetud: https://laravel.com/docs/10.x/database, 18.02.2024.

Laravel. (2024) Laravel Socialite. Loetud: https://laravel.com/docs/10.x/socialite, 18.02.2024.

Laravel. (2024) Meet Laravel. Loetud: https://laravel.com/docs/10.x#meet-laravel, 18.02.2024.

Lerdorf, R. (1995) Announce: Personal Home Page Tools (PHP Tools). Loetud: https://groups.google.com/g/comp.infosystems.www.authoring.cgi/c/PyJ25gZ6z7A/m/M9FkTUVDfcwJ/, 17.02.2024.

Lutkevich, B., Lewis, S. (2022) What is the waterfall model? Loetud: https://www.techtarget.com/searchsoftwarequality/definition/waterfall-model, 17.01.2024.

Maricheva, A. (2023) A Trip Back in Time: the History of JavaScript. Loetud: https://softteco.com/blog/history-of-javascript, 06.03.2024.

Martin, R. C. (2009) Manifesto for Software Craftsmanship. Loetud: https://manifesto.softwarecraftsmanship.org/#/en, 02.01.2024.

Martins, J. (2024) What is Kanban? Free Kanban template, with examples. Loetud: https://asana.com/resources/what-is-kanban, 10.03.2024.

McDonald, K. (2016) How You Can Help Agile Alliance Help You. Loetud: https://www.agilealliance.org/how-you-can-help-the-agile-alliance-help-you/, 06.03.2024.

MDN Web Docs. (2023) At-rules. Loetud: https://developer.mozilla.org/en-US/docs/Web/CSS/At-rule, 25.09.2023.

MDN Web Docs. (2023) CSS: Cascading Style Sheets. Loetud: https://developer.mozilla.org/en-US/docs/Web/CSS, 25.09.2023.

MDN Web Docs. (2023) HTML basics. Loetud: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics, 21.09.2023.

MDN Web Docs. (2023) HTML elements reference. Loetud: https://developer.mozilla.org/en-US/docs/Web/HTML/Element, 21.09.2023.

MDN Web Docs. (2023) JavaScript. Loetud: https://developer.mozilla.org/en-US/docs/Web/JavaScript, 26.09.2023.

MDN Web Docs. (2024) The box model. Loetud: https://developer.mozilla.org/en-US/docs/Learn/CSS/Building_blocks/The_box_model, 06.03.2024.

Meltzer, R. (2023) What is a Programming Library? A Beginner' s Guide. Loetud: https://careerfoundry.com/en/blog/web-development/programming-library-guide/, 06.03.2024.

MySQL. (2024) MySQL Customers. Loetud: https://www.mysql.com/customers/, 18.02.2024.

Node.js. (2023) About Node.js® | Node.js. Loetud: https://nodejs.org/en/about, 22.11.2023.

Nurk, M. (2015) Programmeerimise õpetamise meetodid ja nende rakendamine gümnaasiumi valikkursusel [bakalaureusetöö]. Tartu: Tartu Ülikool.

Outworks Solutions Private Ltd. (2023) Agile Software Development: Real-World Impact and Tools for Agile Project Management. Loetud: https://www.linkedin.com/pulse/agile-software-development-real-world-k1iac, 01.01.2024.

PHP. (2024) ElePHPant. Loetud: https://www.php.net/elephpant.php, 17.02.2024.

PHP. (2024) History of PHP. Loetud: https://www.php.net/manual/en/history.php.php, 17.02.2024.

PHP. (2024) PHP: Introduction. Loetud: https://www.php.net/manual/en/language.types.intro.php, 12.01.2024.

PremierAgile. (2023) The Future of Agile Jobs: Trends and Predictions. Loetud: https://premieragile.com/future-agile-jobs-trends-predictions/, 01.01.2024.

PremierAgile. Top 7 Agile Frameworks. Loetud: https://premieragile.com/types-of-agile-frameworks/, 01.01.2024.

Raeburn, A. (2022) Extreme programming (XP) gets results, but is it right for you? Loetud: https://asana.com/resources/extreme-programming-xp, 01.01.2024.

Raj, A. (2023) History of CSS: The Evolution of Web Design. Loetud: https://www.almabetter.com/bytes/articles/history-of-css, 06.03.2024.

React. (2023) React. Loetud: https://react.dev/, 22.11.2023.

Rieuf, E. (2016) History of MySQL. Loetud: https://www.datasciencecentral.com/history-of-mysql/, 18.02.2024.

Roose, B. (2020) What's new in PHP 8. Loetud: https://stitcher.io/blog/new-in-php-8, 17.02.2024.

Sacolick, I. (2022) A brief history of the agile methodology. Loetud: https://www.infoworld.com/article/3655646/a-brief-history-of-the-agile-methodology.html, 02.01.2024.

Shore, J., Warden, S. (2008) The Art of Agile Development. Loetud: https://www.jamesshore.com/v2/books/aoad1, 19.09.2023.

Shore, J., Warden, S. (2008) The Art of Agile Development. lk 4. Loetud: https://www.jamesshore.com/v2/books/aoad1, 19.09.2023.

Shore, J., Warden, S. (2008) The Art of Agile Development. lk 9–12. Loetud: https://www.jamesshore.com/v2/books/aoad1, 19.09.2023.

Soomro, H. A. (2023) Why React is so Popular? Loetud: https://www.linkedin.com/pulse/why-react-so-popular-hamza-ali-soomro, 22.11.2023.

Stack Overflow. (2022) Stack Overflow Developer Survey 2022. Loetud: https://survey.stackoverflow.co/2022/, 22.11.2023.

Standish Group. (2015) Chaos Report 2015. Loetud: https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf, 17.01.2024.

State of Agile. (2022) 16th State of Agile Report. Loetud: https://digital.ai/resource-center/analyst-reports/state-of-agile-report/, 01.01.2024.

Statista. (2023) Ranking of the most popular database management systems worldwide, as of September 2023. Loetud: https://www.statista.com/statistics/809750/worldwide-popularity-ranking-database-management-systems/, 18.02.2024.

Striebeck, M. (2007) Agile Adoption at Google. Loetud: https://www.agilesociety.co.kr/news_file/google_agile.pdf, 02.01.2024.

Tahiroğlu, U. (2023) "To Use" or "Not to Use" Frameworks in Backend Development. Loetud: https://www.linkedin.com/pulse/use-frameworks-backend-development-utku-tahiro%C4%9Flu, 18.02.2024.

TTÜ IT kolledž. (2020) Väledad tarkvaraarenduse mudelid. Loetud: https://wiki.itcollege.ee/index.php/V%C3%A4ledad_tarkvaraarenduse_mudelid#Kanban, 10.03.2024.

University of Boston. (2020) History of CSS. Loetud: https://www.bu.edu/lernet/artemis/years/2020/projects/FinalPresentations/HTML/historyofcss.html, 25.09.2023.

University of Boston. (2020) History of HTML. Loetud: https://www.bu.edu/lernet/artemis/years/2020/projects/FinalPresentations/HTML/historyofhtml.html, 21.09.2023.

University of Minnesota. (2022) Agile Methodology: Advantages and Disadvantages. Loetud: https://ccaps.umn.edu/story/agile-methodology-advantages-and-disadvantages, 16.01.2024.

W3Schools. (2023) CSS Box Model. Loetud: https://www.w3schools.com/css/css_boxmodel.asp, 22.10.2023.

W3Schools. (2023) CSS Syntax. Loetud: https://www.w3schools.com/css/css_syntax.asp, 25.09.2023.

W3Schools. (2024) Introduction to MySQL. Loetud: https://www.w3schools.com/mysql/mysql_intro.asp, 18.02.2024.

W3Schools. (2024) Introduction to SQL. Loetud: https://www.w3schools.com/sql/sql_intro.asp, 18.02.2024.

W3Schools. (2023) JavaScript History. Loetud: https://www.w3schools.com/js/js_history.asp, 22.11.2023.

W3Techs. (2023) Usage statistics of JavaScript as client-side programming language on websites. Loetud: https://w3techs.com/technologies/details/cp-javascript, 26.09.2023.

W3Techs. (2024) Usage statistics of PHP for websites. Loetud: https://w3techs.com/technologies/details/pl-php, 12.01.2024.

Resümee

Pranglimisrakenduse arendamine välearenduse põhimõtetel

Peastarvutamine on eluline oskus, mis on erinevates situatsioonides kasulik igale inimesele. Pranglimine ehk aja peale peastarvutamine on üks parimaid võimalusi selle oskuse arendamiseks. Käesoleva uurimistöö eesmärgiks on luua Tallinna Reaalkooli matemaatikaõpetajate soovidele vastav pranglimisrakendus välearenduslikel põhimõtetel ja arenduse käigus analüüsida välearenduse kasutamist ja selle eeliseid tegelikus elus.

Töö teoreetilises osas tehakse kokkuvõte välearenduse olemusest ja selle ajaloost. Samuti kirjeldatakse lühidalt välearendusele eelnenud koskmudeli põhimõtteid, populaarsemaid välearenduslikke raamistikke ja välearenduse kasutamist suurtes tehnoloogiafirmades. Praktiline osa on jaotatud kaheks. Esmalt kirjeldatakse erinevaid pranglimisrakenduses kasutatud programmeerimiskeeli ja raamistikke. Tehakse ka kokkuvõte tehnoloogia ajaloost ja esitatakse põhjendused, miks valiti pranglimisrakenduse arenduseks just see tehnoloogia. Seejärel analüüsitakse välearenduse kasutamist rakenduse arenduse jooksul ja tehakse ülevaade selle eelistest. Samuti tuuakse välja kohad, kus välearendus aitas arendajaid parema rakenduse loomisel.

Analüüsist selgus, et suurem osa vastavas kirjanduses esitatud välearenduse eelistest, näiteks projekti õnnestumise suurem määr ja suurem kliendirahulolu, leidsid kinnitust ka pranglimisrakenduse arendusest saadud kogemusele tuginedes. Lisaks aitas välearendus luua rakendusele palju esialgselt mitteplaneerituid võimalusi, näiteks kasutaja valitav liidese värv, uued mängutüübid ja nupp tehte vahele jätmiseks. Samas leiti, et ühe rakenduse arendusest saadud tulemuste ekstrapoleerimisel välearenduse kasuks tuleb olla ettevaatlik, sest positiivsed mõjud võivad olla tingitud ka teistest teguritest.

Abstract

The development of a mental arithmetic application using the Agile software development methodology

Mental arithmetic is a useful skill that can, in different situations, be used by everyone. Speed-based mental arithmetic (pranglimine) is one of the best ways to develop this skill. The aim of this research paper is to, using the Agile methodology, create a mental arithmetic application which is tailored to the needs of the math teachers of Tallinn Secondary School of Science and analyse the usage and advantages of Agile based on a real-life experience.

The theoretical part of the paper provides a summary of the Agile methodology and its history. It also brielfy describes the waterfall model, a development method used before Agile, several popular Agile frameworks and the usage of Agile methodology in large technology companies. The practical part is divided into two. Firstly, it describes different programming languages and frameworks used in the development of the application. As summary of each technology's history and the reasons for choosing it is also provided. Then, the use of Agile in the development of the application is analysed and an overview of its advantages is provided. It also identifies how Agile methodology helped the developers build a better application.

The analysis showed that most of the advantages of Agile described in the literature, such as a higher project success rate and higher client satisfaction, were confirmed by the experience got from the development of the application. In addition Agile helped develop many originally unplanned features, such as user-changable primary colour, additional types of arithmetic and a button for skipping operations. However, it is important to note that extrapolating the results from the development of one application in favor of Agile has to be done carefully as the positive results could potentially have been caused by a combination of different factors.

Kinnitusleht

Uurimistöö autorina kinnitan, et

- koostasin uurimistöö iseseisvalt ning kõigile töös kasutatud teiste autorite töödele ja andmeallikatele on viidatud;
- uurimistöö vastab Tallinna Reaalkooli uurimistöö juhendile;
- olen teadlik, et uurimistööd ei edastata teistele tulu teenimise eesmärgil ega jagata teadlikult plagieerimiseks.

. . . . . .

kuupäev / nimi / allkiri

Juhendajana kinnitan, et uurimistöö vastab Tallinna Reaalkooli uurimistöö juhendile ja lubatakse kaitsmisele.

Juhendaja

. . . . . .

kuupäev / nimi / allkiri