Lappeenrannan-Lahden teknillinen yliopisto LUT
School of Engineering Science
Tietotekniikan koulutusohjelma

WEBASSEMBLYN INTEGROIMINEN SOVELLUSKEHITYSPROSESSIIN

Työn tarkastaja(t): Apulaisprofessori Antti Knutas

TIIVISTELMÄ

Lappeenrannan-Lahden teknillinen yliopisto LUT
School of Engineering Science
Tietotekniikan koulutusohjelma
Aleksi Kuznetsov

WebAssemblyn integroiminen sovelluskehitysprosessiin

Kandidaatintyö 2021

28 sivua, 1 kuva, 2 taulukkoa, 11 koodiesitystä
Työn tarkastajat: Apulaisprofessori Antti Knutas

Hakusanat: WebAssembly, internet, Node.js, internet-selain, sovelluskehitys, JavaScript
Keywords: WebAssembly, internet, Node.js, web browser, application development, JavaScript

Työssä tutustutaan WebAssemblyn mahdollisuuksiin olla webin neljäs virallinen kieli. WebAssemblya tarvitaan, sillä JavaScript ja selainten JavaScript-moottorit ovat kehitetty suorituskyvyn puolesta loppuun asti. Kielen syntaksista, semantiikasta, ja suorituskyvystä on paljon tutkimuksia, joten tässä työssä ei tutustua tarkasti niihin ominaisuuksiin, vaan työn tarkoitus on tarkastella kuinka helppoa WebAssemblyn käyttäminen eri työkaluilla on, sekä mitkä ovat tietotekniikan alan korkeakouluopiskelijoiden mielipiteet ja ennakkoluulot teknologiasta. Tutkimusmetodeina on kysely, sekä konstruktiivinen tutkimus tekemällä esimerkkiprojekteja, joita työkalut tarjoavat. Tuloksina kyselystä saatiin, että opiskelijoiden näkemykset ja mielipiteet olivat positiivisia, mutta käyttäminen nähtiin vaikeana ja ei välttämättömänä. Konstruktiivisen tutkimuksen tuloksien perusteella virallisia ja epävirallisia ohjeita ja dokumentaatiota seuraten käyttö ei ole vaikeaa.

ABSTRACT

Lappeenranta-Lahti University of Technology LUT
School of Engineering Science
Degree Programme in Software Engineering
Aleksi Kuznetsov

Integrating WebAssembly into application development process

Bachelor's Thesis 2021

28 pages, 1 figures, 2 tables, 11 code snippets
Examiners: Assistant Professor Antti Knutas

Keywords: WebAssembly, internet, Node.js, web browser, application development, JavaScript

The object of this thesis is to find the possibilities of WebAssemblyn being the fourth official language of the web. WebAssembly is needed, since the JavaScript language and the JavaScript engines of web browsers are developed to their maximum regarding their performance. There are many papers published on the syntax, semantics, and performance of WebAssembly, so this thesis doesn't explore them further. Rather the object of this thesis is to find out how easy WebAssembly is to use, and what are the preconceptions and opinions of college students on the technology. This thesis uses enquiry and constructive research as research methods. Results of the enquiry are that the students think WebAssembly is good, but hard to use and not absolutely necessary. The second method shows that following the official and unofficial instructions and documentation of certain tools is easy.

SISÄLLYSLUETTELO

SYMBOLI- JA LYHENNELUETTELO

API	Application Programming Interface
CSS	Cascading Style Sheets
DOM	Document Object Model
ES6	ECMAScript 6
HTML	Hypertext Markup Language
IoT	The Internet of things
SLOC	Source Lines of Code
TAM	Technology Acceptance Model
W3C	World Wide Web Consortium
WABT	WebAssembly Binary Toolkit

1 JOHDANTO

Javascript, HTML (Hypertext Markup Language), ja CSS (Cascading Style Sheets) ovat jo pitkään olleet verkkoselainten kolme tärkeintä ja ainoaa laajasti tuettua kieltä. Ensimmäinen verkkoselain, jolla oli tuki verkkosivun sisällön kanssa vuorovaikutuksen mahdollistavalle ohjelmointikielelle, JavaScriptille, oli NetScape Navigator 2 [22]. HTML on merkkauskieli itse sivun sisältöä varten ja CSS toimii työkaluna sisällön ulkoasun määrittämiseen. Internetin alussa verkkosivujen sisältö oli vain tekstiä, kuvia, ääniä, tai videoita. Selainten käyttökohteet ja ominaisuudet ovat kuitenkin muuttuneet siitä, mihin niitä käytettiin, joten on syytä miettiä arkkitehtuurin parantamista ja kehittämistä. Verkkoselaimet ovat olleet jo pitkään yleisimpiä ohjelmistoja, joita älypuhelimiin, tietokoneisiin, ja tabletteihin saa asennettua [11]. Selaimet ovat kasvussa myös alun perin käyttöjärjestelmätasolla pyörivien ohjelmistojen ajamisessa. Teknologiat, kuten Electron [8], erilaiset kehysympäristöt (engl. framework), Babel [3], ja Node.js [23], ovat parantaneet Javascriptin käytettävyyttä ja yleisyyttä, ja vieneet kieltä eteenpäin, mutta uusia kielen ominaisuuksia ei voi lisätä helposti, sillä taaksepäinyhteensopivuus verkkoselaimissa on tärkeää näin laajassa ekosysteemissä.

WebAssembly on tavukoodia (engl. bytecode), jonka tavoitteena on saada verkkoselaimeen suorituskykyisempää ohjelmointikoodia, joka on myös turvallista käyttää, ihmiselle luettavaa, helposti korjattavaa (engl. debuggable), vähemmän epädeterminististä (engl. non-determinism) kuin JavaScript, vähemmän muistia vievää, ja samalla taaksepäinyhteensopivaa [35–37, 45]. WebAssemblya voi käyttää myös muilla alustoilla, kuten sulautetuissa järjestelmissä, älypuhelimilla, IoT-laitteilla (The Internet of things), tai Node.js-alustalla. Tiedostokoko on huomattavasti pienempi verrattuna JavaScriptiin. [14, 15, 26] Siihen kuuluu osaksi binäärikoodi ja osaksi tekstimuotoinen Assembly-ohjelmointikielen tapainen välimuotoinen koodi [36]. Päätarkoituksena ei ole kirjoittaa WebAssembly-tiedostoa itse, vaan kääntää se sopivasta matalan tason ohjelmointikielestä. Näistä kerrotaan enemmän luvussa 2. Mozillan mainitsemat ohjelmointikielet, joista WebAssemblya voi kääntää, ovat C, C++, Rust, ja Typescript. Käännettyä WebAssemblyn .wasm-tiedostoa voi ajaa JavaScriptin avulla samassa ympäristössä kuin JavaScript ajetaan. [36] JavaScriptin rooli kuitenkin säilyy, sillä muun muassa DOM-elementtien (Document Object Model) hallintaan WebAssembly ei pysty [36], ja JavaScriptin poistaminen selaimista rikkoisi kaikki vanhat verkko-osoitteet. WebAssemblyn avulla voisi käyttää esimerkiksi kirjastoja, joita kirjoitettiin C-ohjelmointikielellä. Vuonna

2017 WebAssemblylle saatiin tuki neljälle suurimmalle verkkoselaimelle: Mozilla Firefox, Google Chrome, Microsoft Edge, ja Safari [20]. Näiden verkkoselanten kehittäjät ovat mukana WebAssemblyn kehityksessä yhdessä [35]. Syksyllä 2019 WebAssemblystä tuli virallisesti neljäs ajettava kieli tärkeimmissä verkkoselaimissa W3C:n (World Wide Web Consortium) suosituksesta [7]. Keväällä 2020 87,19 % kaikista maailman asennetuista verkkoselaimista tukee WebAssemblya [4]. Vaikka WebAssemblyssa onkin todettu tietoturvallisuusaukkoja, jotka mahdollistavat haitallisen koodin ajamisen, kuten esimerkiksi kryptovaluutan louhintaa käyttäjältä piilossa suorittava koodi [18], on se isossa roolissa web-ohjelmoinnissa tulevaisuudessa.

WebAssembly julkaistiin vuonna 2015 [10], joten se on tämän työn kirjoittamishetkellä (kevät 2020) verrattain uusi teknologia. Julkaistuista kirjoituksista suuri osa keskittyy kielen suorituskykyyn verrattuna JavaScriptiin tai perehtyy tarkemmin kielen ominaisuuksiin. Monia yrityksiä saattaisi kiinnostaa WebAssemblyn tarjoamat mahdollisuudet web-kehityksessä.
Kielen syntaksista, semantiikasta, ja suorituskyvystä on paljon tutkimuksia, joten tässä työssä ei tutustua tarkasti niihin ominaisuuksiin, vaan tässä työssä tutkimuskymyksinä mietitään aiheita kuten kenen kannattaisi käyttää WebAssemblya, kuinka vaikeaa sen käyttö on, miten sitä käytetään, mitä mielipiteitä tai ennakkoluuloja teknologiasta on, mitä WebAssemblyn käytöllä saavutetaan, ja kannattaako se. Tavoitteena on nostaa yleisön tietoisuutta teknologiasta, ja kumota ennakkoluuloja, mitä alan ihmisillä on. Työssä katsotaan esimerkiksi, kuinka helposti JavaScriptiin tottuneet web-kehittäjät sopeutuisivat muutokseen, ja jos muut kuin web-kehittävät saisivat jotain irti WebAssemblysta. Esimerkiksi C- tai C++-kehittäjille saattaisi olla tarvetta korvaamaan osa JavaScriptista WebAssemblyyn aihealueissa, missä suorituskykyä vaaditaan, kuten videopelit, 3D-grafiikka, videoiden toisto, virtuaalitodellisuus, kryptografia, konenäkö, tai raskas matematiikka [31].

Toisessa luvussa tutkitaan kirjallisuutta ja erilaisten työkalujen dokumentaatiota aiheeseen liittyen. Tutkitaan verkkoselainten historiaa lyhyesti ja niiden nykytilaa. Sitten tarkastellaan haasteita, mitä WebAssemblyn kehitys on kohdannut ja kielen käytöstä. Tutkitaan, mitä WebAssemblyn kaltaisia yrityksiä aiemmin on ollut ja eri työkaluja WebAssemblyn luomiseen. Sitten tutkitaan, kuinka vaikeaa WebAssemblyn käyttäminen on verrattuna perinteisesti käytettyyn JavaScriptiin tai eri web-kehysympäristöihin, ja kuinka WebAssemblyn käyttöönottaminen ohjelmistoprojektissa onnistuu.

Työtä varten tehdään malliksi esimerkkiprojekteja WebAssemblyn kehitysympäristön luomisesta verrattuna JavaScriptin kehitysympäristön luomiseen työkalujen ja kielien virallisten ja epävirallisten ohjeiden ja dokumentaatioiden mukaan. Toteutetaan myös kysely, jossa otetaan selvää WebAssemblyn tietämyksestä ja ennakkoluuloista, jotka kaikki vaikuttavat käyttöönottamisen laajuuteen. Näistä kerrotaan luvussa 3. Myöhemmin luvuissa 4 ja 5 kerrotaan tutkimusten tulokset ja keskustellaan tuloksista helppouden kannalta. Luvussa 6 yhteenvedossa pohditaan, mitä tulokset antoivat ilmi.

2 KIRJALLISUUDESTA JA DOKUMENTAATIOSTA

2.1 Nykytila verkkoselaimissa

Se mikä alkoi ennen dokumenttien tarkasteluun tai staattisten tiedostojen jakamiseen tarkoitettuna, on nykyään ubiikki alusta kaikelle informaatiolle, sovelluksille ja viihteelle. Web-alustan kasvamisen myötä JavaScript ei enää riitä kaiken sen sisällön esitykseen, mitä nykyään sieltä löytyy, kuten 3D-pelejä, virtuaalitodellisuutta, kryptografiaa, konenäköä, tai videoiden suoratoistoa (engl. streaming), vaikka kieli onkin kasvanut paljon. Web-sovelluksiin tarvitaan yhä enemmän tehokasta ja ennalta arvattavaa suoritusnopeutta. [12, 14, 26, 31] Verkkoselainten käyttäminen työtehtäviin on helppoa, sillä se ei vaadi mitään muuta kuin selaimen asennuksen käyttöjärjestelmään. Tämä tekee selaimelle tehtävistä ohjelmistoista erittäin siirrettäviä (engl. portable) [14]. JavaScript-moottorit (engl. engine) eivät ole syypäitä heikkoon suorituskykyyn, vaan ongelma on kielessä itsessään, joka suosii käytön helppoutta suorituskyvyn sijaan [26]. On laskettu, että JavaScript suoriutuu verkkoselaimessa parhailla JavaScript-moottoreilla - kuten Chromen V8:lla ja Firefoxin SpiderMonkey:lla - keskimäärin 2 kertaa huonommin, kuin käyttöjärjestelmätasolla ajettavalla C-kielellä, mutta tämä arvio myös vaihtelee suuresti. WebAssemblyn suorituskyky tyypillisimmissä selaimissa, sekä Node.js-ympäristössä on noin 75 % - 100 % C-kielen suorituskyvystä. [14] Neljän suurimman verkkoselaimen kehittäjät ovatkin yhdessä ryhtyneet WebAssemblyn kehitykseen, jonka tarkoituksena on tehokas, turvallinen, ja matalatasoinen tavukoodi selaimelle [12].

2.2 WebAssemblyn kehityksen haasteet

Tähän mennessä kaikki yritykset rakentaa WebAssemblyn tavoitteissakin listattujen ominaisuuksien mukainen käännöskohde ei ole onnistunut, sillä ominaisuuksien saavuttaminen ei ole helppoa. Tämän käännöskohteen pitäisi olla kompakti lähettää verkon yli, helposti siirrettävä monelle alustalle ja arkkitehtuurille, optimoitu kääntämisen yhteydessä, ja turvallinen käyttäjille, sillä verkkosivujen koodi on aina luotettamattomilta lähteiltä (engl. source). WebAssembly on ensimmäinen teknologia, jossa edellä mainitut tavoitteet on saavutettu. [12]

2.3 Käyttö

WebAssemblyn .wat-tiedostoformaatissa koodi on S-lausekkeina (engl. S-expression), mikä tekee siitä helposti ymmärrettävän ihmisille. Suositellumpi tapa on silti kääntää se toisesta

korkeammasta kielestä jonkin myöhemmin mainittavista työkaluista avulla. WebAssemblyn suora kirjoittaminen on kannattavaa lähinnä silloin, jos haluaa mielenkiinnon tai opiskelun vuoksi työskennellä binäärikoodin kanssa. Verkkoselaimessa JavaScript API (Application Programming Interface) on se työkalu, jolla WebAssemblya voi käyttää. Se lataa, kääntää, ja kutsuu WebAssembly-moduulien funktioita. Käännetty binääritiedosto voidaan tallentaa verkkoselaimen IndexedDB-tietokantaan, ja säilyttää se muistissa seuraavaa istuntoa (engl. session) varten, nopeuttaen sivuston lataamista. [12]

2.4 Yleisimmät työkalut WebAssemblyn parissa työskentelyyn

WABT (WebAssembly Binary Toolkit) on joukko työkaluja, joilla voidaan muun muassa vaihdella formaattia binääriformaatin ja tekstiformaatin välillä, sekä monia muita työkaluja WebAssembly-tiedostojen kanssa työskentelyyn ja niiden manipulointiin eri keinoin. Työkaluja nimeltä ovat: *wat2wasm*, *wasm2wat*, *wasm-objdump*, *wasm-interp*, *wasm-decompile*, *wat-desugar*, *wasm2c*, *wasm-strip*, *wasm-validate*, *wast2json*, *wasm-opcodecnt*, ja *spectest-interp*. [15, 39] Näiden käyttöä varten tarvitaan CMake-työkalu [44]. Koodiesityksessä 5 on esimerkki .wat-tiedostosta, jossa WebAssembly on sen tekstiformaatissa.

Binaryen on työkaluinfrastruktuuri ja kääntäjä, joka muuttaa WebAssembly-tiedoston tekstiformaatista binääritiedostoon *.wasm*-päätteellä ja myös toisin päin. Se myös optimoi koodia aina kun siihen pystyy käyttäen omaa sisäistä esitystänsä koodista, joka lopulta käännetään. Työkaluketjuun liittyy muita kääntäjiä, jotka perustuvat Binaryeniin, kuten *asm2wasm* (Asm.js ja WebAssembly), *wasm2js* (JavaScript ja WebAssembly), *Asterius* (Haskell ja WebAssembly), ja AssemblyScript. [1, 26, 38]

AssemblyScript on kääntäjä, joka kääntää TypeScriptillä kirjoitetun lähdekoodin koodin ennenaikaisesti WebAssemblyyn käyttäen Binaryenia. Projektin aloitukseen tarvitaan Node.js asennettuna ja npm, joka tulee Node.js:n asennuksen mukana. [28] Seuraavaksi mainitut koodiesitykset muodostavat ohjelmoijille tyypillisen ”Hello World!”-projektin, ja ovat JavaScript- ja TypeScript-ohjelmoijille lähes kokonaan tuttua koodia. Kuitenkaan kaikki TypeScript-koodi ei ole kelpaavaa koodia WebAssemblyyn. Esimerkiksi Koodiesityksessä 2 TypeScriptin tyypilliseen tapaan ei käytetä Number-muuttujaa, vaan 32-bittistä integer-muuttujaa, joka on WebAssemblyn sisään rakennettu tietotyyppi. Koodiesityksessä 2 on

esimerkki yksinkertaisesta ohjelmasta, joka käännetään *hello-world.wasm*-moduuliksi komennolla ”asc hello-world.ts -b hello-world.wasm”. Koodiesityksessä 4 on funktio *.wasm*-moduulin lataamiselle. Koodiesityksessä 3 moduuli ladataan ja kutsutaan sitä JavaScriptin kautta. Lopuksi lisätään ”<script type="module" src="./hello-world.js"></script>” .html-tiedostoon ES6-moduulina (ECMAScript 6). Toimiva ohjelma tulostaa ”Hello World! addResult: 48”. [34]

Speedy.js on kääntäjä, joka kääntää koodia TypeScriptistä ja JavaScriptistä WebAssemblyyn ja generoi tarpeellisen rajapinnan kommunikointiin WebAssemblyn ja käännetyn JavaScriptin välillä. Speedy.js asettaa muutaman rajoituksen ohjelmoijalle lähdekoodin kirjoittamiseen. Lähdekoodi, joka aiotaan kääntää WebAssemblyyn, pitää sisältää vain kielen ominaisuuksia, joille löytyy vastaava koodi kohdekielessä. Jos ohjelmasta yritetään kääntää ominaisuuksia, jotka eivät ole tuettuja, kääntäjä antaa virheilmoituksen. Koodiesityksessä 1 etsitään, että onko argumentti *value* alkuluku. ”use speedyjs”-deklaraatio *isPrime*-funktiossa kertoo, että kyseessä on Speedy.js-funktio, joten se käännetään WebAssemblyksi. Nämä funktiot kirjoitetaan aina TypeScriptillä. *Main*-funktio käännetään niin, että se sisältää tarpeellisen rajapinnan, jotta voidaan kutsua laskufunktiota. [26] Kääntämisprosessiin liittyy myös LLVM [29], Emscripten [19], ja Binaryen [38] – kahdesta jälkimmäisestä kerrotaan seuraavissa kappaleissa. Käyttämällä Speedy.js-kirjastoa saavutetaan suorituskyvyn kasvamista jopa nelinkertaiseksi verrattuna TypeScriptiin tietyissä tehtävissä ja testeissä riippuen verkkoselaimesta [26].

Rust-ohjelmointikieli on verrattain uusi ohjelmointikielten maailmassa ja siitä löytyy hyvät työkalut WebAssemblyyn kääntämiseen. Rustwasm-dokumentaatiosta löytyy esimerkkikoodi ja ohjeet ”Hello World!”-ohjelmaan. Ensin ladataan Rust-ympäristön tarvittavat osat kuten *rustup, rustc, cargo, wasm-pack, ja cargo-generate*. Tähän projektiin tarvitaan myös Node.js ja npm. Koodiesityksessä 6 on esitetty, kuinka selaimen ”window.alert”-JavaScript-funktio on käytettävissä. ”wasm-pack build”-komennolla saadaan Rust-lähdetiedostot käännetyksi .wasm-tiedostoon. Komento myös luo rajapinnan, jotta JavaScript saa kutsuttua WebAssembly-funktioita. Komennolla ”npm init wasm-app www” saadaan luotua ohjelma, joka toimii selaimella. [13]

Emscripten on työkalu, jolla voi kääntää lähinnä C- ja C++-tiedostojen lähdekoodia LLVM-binäärikoodiin ja siitä edelleen JavaScriptiin ja WebAssemblyyn, mutta toimii myös muilla kielillä, joita voi kääntää LLVM-binäärikoodiksi, kuten Python, Rust, ja Objective-C. Se on

rakennettu LLVM-ympäristön päälle, ja käyttää Clang-kääntäjää. JavaScript tai WebAssembly voidaan sitten ajaa joko verkkoselaimessa tai Node.js-alustalla. [5, 12, 14, 19, 29]

Lyhyesti mainitaan myös awesome-wasm-langs, joka on lista tämänhetkisistä ohjelmointikielistä, joista WebAssemblyyn on mahdollista kääntää eri työkaluilla [2].

Vaihtoehtoisten tapojen määrä tuottaa WebAssemblya on suuri. Tämä antaa ohjelmoijalle suurta vapautta valita omat työkalunsa. Uuden kielen tai ekosysteemin esittely ennestään JavaScript-pohjaiseen projektiin kuitenkin lisää kompleksisuutta. Se pakottaa jakamaan projektin kahtia: JavaScript-osaan ja toiseen uuteen osaan, joka keskittyy suorituskykyiseen osaan koodista. Se myös tarkoittaa, että JavaScriptin ja WebAssemblyn välillä tulee rakentaa rajapinta. [26]

Koodiesitys 1. Ohjelma tarkistaa onko jokin luku alkuluku. ”use speedyjs” mahdollistaa työkalun käyttämisen jos sellainen on ja jättää rivin huomioimatta jos sellaista ei tarvitse. [26]

```
async function isPrime(value: int) {
  "use speedyjs";

  if (value <= 2) {
    return false;
  }

  const sqrt = Math.sqrt(value)as int;
  for (let i = 2; i <= sqrt; ++i) {
    if (value % i === 0) {
      return false;
    }
  }
  return true;
}

async function main() {
  const prime = await isPrime(10000);
  console.log("isPrime 10000: " + prime);
}
```

Koodiesitys 2. TypeScript-ohjelma kahden luvun summaamiseen. Luvut otetaan 32-bittisinä kokonaislukuina WebAssemblya varten. *hello-world.ts*. [34]

```
// This exports an add function.
// It takes in two 32-bit integer values
// And returns a 32-bit integer value.
export function add(a: i32, b: i32): i32 {
  return a + b;
}
```

Koodiesitys 3. JavaScript-ohjelma, jossa kutsutaan .wasm-moduulin funktiota. [34]

```
const runWasmAdd = async() => {
  // Instantiate our wasm module
  const wasmModule = await wasmBrowserInstantiate("./hello-world.wasm");

  // Call the Add function export from wasm, save the result
  const addResult = wasmModule.instance.exports.add(24, 24);

  // Set the result onto the body
  document.body.textContent = `Hello World! addResult: ${addResult}`;
};
runWasmAdd();
```

Koodiesitys 4. JavaScript-ohjelma, jossa mukana .wasm-moduulin lataaminen. *hello-world.js*. [30, 34]

```
// https://github.com/torch2424/wasm-by-example/blob/master/demo-util/
export const wasmBrowserInstantiate = async(wasmModuleUrl, importObject) => {
  let response = undefined;

  if (!importObject) {
    importObject = {
      env: {
        abort: () => console.log("Abort!")
      }
    };
  }

  // Check if the browser supports streaming instantiation
  if (WebAssembly.instantiateStreaming) {
    // Fetch the module, and instantiate it as it is downloading
    response = await WebAssembly.instantiateStreaming(
        fetch(wasmModuleUrl),
        importObject);
  } else {
    // Fallback to using fetch to download the entire module
    // And then instantiate the module
    const fetchAndInstantiateTask = async() => {
      const wasmArrayBuffer = await fetch(wasmModuleUrl).then(response =>
          response.arrayBuffer());
      return WebAssembly.instantiate(wasmArrayBuffer, importObject);
    };
    response = await fetchAndInstantiateTask();
  }

  return response;
};
```

Koodiesitys 5. Esimerkki .wat-tiedostosta, jossa WebAssembly on tekstimuotoisena. [33]

```
(module
  (type $t0 (func (param i32 i32) (result i32)))
  (func $addTwo (export "addTwo") (type $t0) (param $p0 i32) (param $p1 i32) (result
i32)
    (i32.add
      (local.get $p0)
      (local.get $p1))))
```

Koodiesitys 6. Rust-ohjelmointikielen tiedosto, jossa käytetään WebAssemblya. src/lib.rs. [13]

```
mod utils;

use wasm_bindgen::prelude::*;

// When the `wee_alloc` feature is enabled, use `wee_alloc` as the global
// allocator.
#[cfg(feature = "wee_alloc")]
#[global_allocator]
static ALLOC: wee_alloc::WeeAlloc = wee_alloc::WeeAlloc::INIT;

#[wasm_bindgen]
extern {
    fn alert(s: &str);
}

#[wasm_bindgen]
pub fn greet() {
    alert("Hello, wasm-game-of-life!");
}
```

2.5 Aiemmat yritykset WebAssemblyn kaltaiseksi käännöskohteeksi

Asm.js on kirjasto, joka on kielenä JavaScriptin matalan tason osajoukko (engl. subset). Se käännetään matalan tason kielistä, kuten C++. Se tukee ominaisuuksia kuten ennenaikainen optimointi ja saavuttaa lähes työpöytäsovelluksen suorituskyvyn. Asm.js:stä käännetty JavaScript kuitenkin perii lähes kaikki ne huonot ominaisuudet, joita JavaScriptillä normaalistikin on. WebAssemblyn on tarkoitus syrjäyttää Asm.js. [26, 35, 42]

Enemmän tai vähemmän muita yrityksiä hyvälle suorituskyvylle verkkoselaimella ovat olleet ActiveX, Native Client [40], Java, Flash, ja monet muut [12, 41]. Aiemmin suositun Flashin käyttö on käytännössä loppunut verkkoselaimilla turvallisuusongelmien vuoksi [43]. Monet edellä mainituista pitää myös asentaa erikseen lisäosina (engl. plugin). Ja kaikki näistä eivät edes toimi kaikilla alustoilla. Esimerkiksi Java ja Flash eivät toimi iOS-alustoilla [41]. Tämän vuoksi JavaScript säilyy pääasiallisena webin kielenä. Monista ohjelmointikielistä on työkaluja JavaScriptiin kääntämiseen, kuten Javasta, Pythonista, F#, ja muistakin, mutta tietysti kaikkia tietyn kielen ominaisuuksia ei pystytä kääntämään, jos niille ei löydy vastaavaa JavaScritptistä [41].

2.6 WebAssembly ohjelmistoprojektissa

Ohjelmistoprojektin hintaa yleensä lasketaan käyttämällä kaavaa, jossa projektin miestyötunnit kerrotaan keskimääräisellä tuntipalkalla. Projektin vaivannäön ennustaminen kuitenkin on vaikeaa. Eräässä ennustamismallissa on 3 muuttujaa projektissa, jotka vaikuttavat vaivannäköön: henkilöstön kokemus, ohjelmistovaatimukset ja sen koko, ja metodit ja työkalut, joihin myös ohjelmointikielen valinta kuuluu. Eräs tapa mitata ohjelmiston kokoa on lähdekoodin rivien määrä (engl. SLOC eli source lines of code). Tämä on kutienkin epätarkka tapa mitata ohjelmistoprojektin kokoa, sillä koko riippuu valitusta ohjelmointikielestä. Ohjelmistoprojektissa tarkoitus on saada rakennettua mahdollisimman lyhyessä ajassa mahdollisimman luotettavaa ja virheetöntä ohjelmistoa. Sillä, käyttääkö kolmannen vai neljännen sukupolven ohjelmointikieltä, ei ole vaikutusta ohjelmiston koodaamisen vaivannäköön. [24] Koska WebAssembly on käännöskohde useammalle kielelle, tästä voidaan päätellä, että sen käyttäminen ei vaikuta suuresti vaivannäköön. Varsinkin jos ohjelmistoprojektin kielenä käytetään jotain, jolle on hyvät työkalut WebAssemblyyn kääntämiseen, ei tämä tuota ylimääräistä vaivaa juuri ollenkaan.

Ohjelmointikielten käyttäjät voidaan jakaa seuraavasti rooleihin tai kategorioihin: tiettyyn ohjelmointikieleen uskovat (engl. believers), tietystä ohjelmointikielestä riippuvaiset (engl. dependents), ja vapaaehtoiset (engl. volunteers) [27]. Esimerkki ensimmäisestä roolista on henkilö, joka uskoo C-kielen olevan se oikea tapa kirjoittaa koodia. Toiseen rooliin voisi kuulua esimerkiksi tietotekniikan opiskelija, joka suorittaa yliopiston kurssia jollain määrätyllä ohjelmointikielellä. Näin hän on riippuvainen kielestä. Kolmanteen kategoriaan kuuluvat ne henkilöt, jotka eivät kuulu toiseen kategoriaan. Heillä on omat syynsä käyttää jotain tiettyä

kieltä, kuten vapaa-ajan projekti. Vaikka tämä ei WebAssemblyn tapauksessa ole niin olennaista tietää, on siitä hyvä puhua. WebAssemblyyn liittyy niin monta eri tapaa tuottaa sitä, että suurelle osalle heistä, jotka sen parissa haluavat työskennellä löytyy se oma toimintatapa ja työkalut.

3 TUTKIMUSMENETELMÄ

Tässä luvussa käydään läpi kaksi tutkimusmetodia aiemmin mainittujen kysymyksien vastaamiseen. Työtä varten tehdään kysely, koska kyselytutkimus on sopiva ihmisten mielipiteiden tutkimiseen [25], ja jos halutaan selvittää teknologian käytettävyyttä tai helppoutta, tarvitaan ihmisten mielipiteitä. Työssä myös tehdään konstruktiivista tutkimusta tekemällä kolme esimerkkiprojektia eri työkaluilla ja ohjelmointikielillä jo olemassa olevien ohjeiden mukaan ikään kuin teknologian kanssa aloittelevan ohjelmoijan näkökulmasta [21]. Tekemällä esimerkit voidaan tutkia vaiheita, joita vaaditaan ohjelmistoprojektin pystyttämiseen näillä työkaluilla. Luvussa 4 ja 5 katsotaan, kuinka vaikeita vaiheet oli toteuttaa ja keskustellaan siitä, kenelle mikäkin työkalu voisi sopia.

3.1 Teknologian hyväksymismalli

Työtä varten tehdään kysely, jossa käytetään pohjana teknologian hyväksymismallia (engl. Technology Acceptance Model tai lyhyesti TAM), jossa mitataan havaittua hyötyä ja käytön helppoutta jonkin teknologian saralla. Nämä taas vaikuttavat käyttäjän aikomukseen. TAM:ista on tullut hyväksytty ja suosittu tapa mitata juuri näitä ominaisuuksia. TAM2 on ehdotettu lisäys TAM:iin, jossa vaikuttavina tekijöinä ovat teknologian imago, teknologian käyttämisen tulosten demoamismahdollisuudet, teknologian käyttämisen tulosten laatu, käyttäjän työkuvan asiaankuuluvuus, kokemus teknologiasta, subjektiiviset normit (eli kuinka käyttäjä suhtautuu johonkin asiaan riippuen siitä, miten muut hänelle vaikutukselliset henkilöt suhtautuvat samaan asiaan), ja käyttäjän vapaaehtoisuus (eli onko esimerkiksi pakotettu käyttämään työ puolesta), ja nämä vaikuttavat lopulta käyttäjän käytökseen. [32] Kuvassa 1 on tekijöiden suhteet toisiinsa.

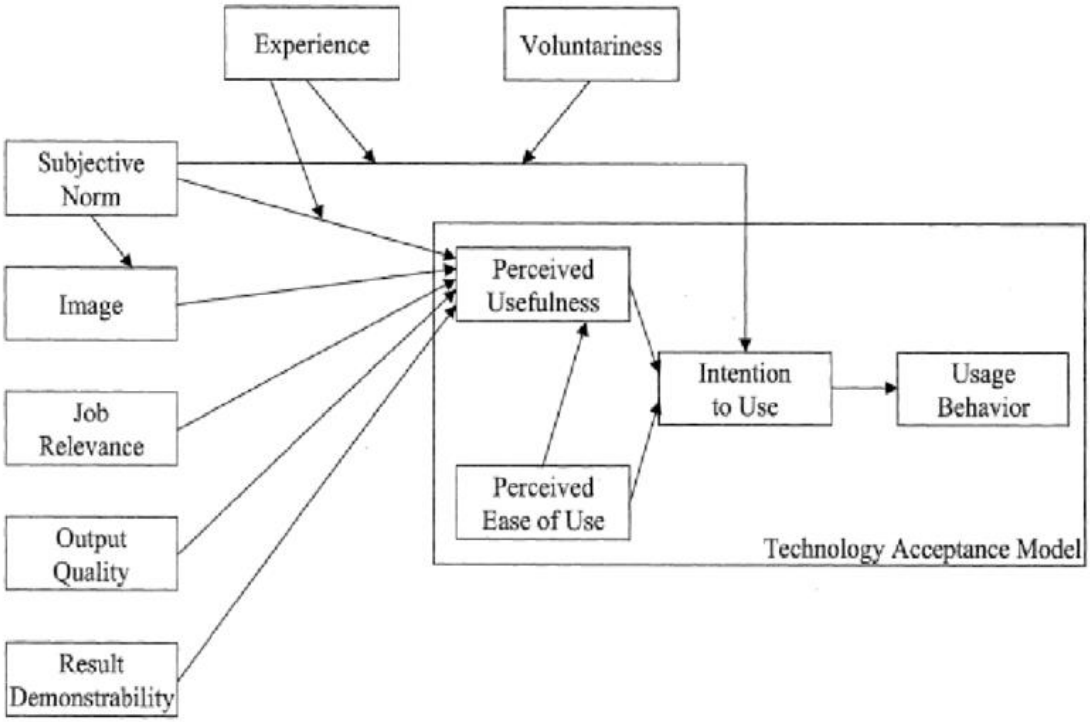
3.2 Kyselyn toteutus

Kysely tuotetaan yksinkertaisesti Google Forms-työkalulla. Kirjautuminen vaaditaan, jotta voidaan olla lähes varmoja, että jokainen vastaaja vastaa vain kerran. Kyselyyn vastataan anonyymisti. Tieto kyselystä lähetetään nykyisille tai entisille korkeakouluopiskelijoille tietotekniikan koulutusohjelmassa teknillisessä yliopistossa erään chat-sovelluksen mainittuja henkilöitä sisältävälle ryhmälle. Tästä syystä voimme olettaa vastaajilta tietämystä alalta ja käyttää tiettyä terminologiaa kyselyssä, jota voidaan olettaa vastaajien tuntevan [17]. Kohdepopulaation (engl. target population) määrittäminen on tärkeää [16]. Koska tiedetään

kohdepopulaatio, voidaan paremmin arvioida vastausprosenttia. Kyselyn tarkoitus on selvittää ennakkoluuloja ja tietämystä WebAssemblysta. Taulukosta 1 näkee kyselyn rakenteen. Riippuen siitä, miten vastasi kysymykseen 3, vastaaja ohjattiin joko osioon 2, tai kysely päätettiin. Edellisestä johtuen vastaajia on vähemmän osiossa 2, kuin osiossa 1. Kysymyksiin pitää pystyä vastaamaan niin, että vastaaja ymmärtää mitä oikeasti on vastannut ja miten vastausta tullaan tulkitsemaan [17]. Tästä syystä kysymyksissä käytetään hyväksi Likert-asteikkoa, joka on tyypillinen mielipidekysymyksissä ja psykologisissa testeissä. Kysymyksissä käytetään hyväksi TAM-malliin liittyviä kysymyksiä [32]. Osaan kysymyksistä on pakollisia vastata, jotta saadaan ideaa, mikä on WebAssemblyn yleinen tietämys ja käyttöaste. Kokemuksia kysytään sekä nykyisestä, menneestä, että tulevasta aikeesta käyttää WebAssemblya. Vastausaika pitäisi olla tarpeeksi pitkä [17]. Aikaa vastata annetaan noin viikko, joka on tarpeeksi näin lyhyelle kyselylle, johon kestää arviolta noin kahdesta kolmeen minuuttia vastata. Kysymykset on suunniteltu lyhyiksi, jotta ne olisi helppo ymmärtää [17].

3.3 Konstruktiivinen tutkimus tekemällä WebAssembly-projekteja

Esimerkeissä käytetään seuraavia käyttöjärjestelmä-, kehitysympäristö-, ja työkaluversioita: Kubuntu 20.04, Node.js 12.16.3, Express.js 4.17.1, AssemblyScript 0.9.4, Rust 1.43.0, Git 2.25.1, ja Emscripten 1.39.13. Node.js- ja Express.js-kehitysympäristöä käytettiin, jotta .wasm-tiedostojen palveleminen olisi helppoa. Git tarvitaan, jotta voidaan ladata tiettyjä työkaluja tai malliprojekteja. Kaikki nämä projektit toteutetaan virallisilla tai suosituilla epävirallisilla ohjeilla ja dokumentaatioilla. Ensimmäinen projekti on ”Hello world!”-projekti AssemblyScriptillä [30]. Toinen projekti on olemassa olevan C-kirjaston siirtäminen (engl. porting) WebAssemblyyn käytettäväksi [9]. Kolmantena tutustutaan Rust-ohjelmointikielen WebAssembly-projektiin [6]. Edelliset projektit toteutetaan järjestyksessä Wasm By Example-sivuston ohjeilla, Googlen ohjeilla, ja Mozillan ohjeilla. Mikä tahansa tekstieditori käy tämänlaatuiseen koodaamiseen, jossa tulos ei riipu mistään erillisestä kirjastosta tai ohjelmointiympäristöstä.



Kuva 1. TAM2-malli. [32]

Taulukko 1. Kyselyn rakenne.

Kysymysnumero/kysymys	**Pakollisuus vastata**	**Vastaustapa**
Osio 1		
1. Oletko opiskelija/opettaja/työssäkäyvä/yms?	Pakollinen	Vapaa sana
2. WebAssemblyn osaamistaso/tietämys	Pakollinen	Likert-asteikolla 1 (En tiedä aiheesta mitään) – 7 (Tiedän aiheesta paljon)
3. Minulla on aikomus käyttää WebAssemblya tulevaisuudessa/käytän sitä jo/olen käyttänyt sitä jossain projektissani/työssäni	Pakollinen	(Kyllä/ehkä – Ei)
Osio 2		
4. Uskon, että WebAssemblyn käyttäminen parantaa tehokkuuttani projektissani/työssäni	Ei	Likert-asteikolla 1 (En tiedä aiheesta mitään) – 7 (Tiedän aiheesta paljon)
5. WebAssembly voisi olla/on hyödyllinen projektissani/työssäni	Ei	Likert-asteikolla 1 – 7
6. WebAssemblyn käyttäminen vaikuttaa helpolta/on helppoa	Ei	Likert-asteikolla 1 – 7
7. Henkilö, jota pidän asiantuntevana tai vaikuttavana sanoo, että minun kannattaisi (vapaaehtoisesti) käyttää WebAssemblya	Ei	Likert-asteikolla 1 – 7
8. WebAssemblyn käyttö on vapaaehtoista projektissani/työssäni	Ei	Likert-asteikolla 1 – 7
9. Näen WebAssemblyn käyttäjät suuressa arvossa	Ei	Likert-asteikolla 1 – 7
10. WebAssemblyn käyttäminen on tärkeää projektissani/työssäni	Ei	Likert-asteikolla 1 – 7
11. WebAssemblylla saa laadukasta tulosta	Ei	Likert-asteikolla 1 – 7
12. WebAssemblylla saatu tulos on helppo demota muille	Ei	Likert-asteikolla 1 – 7

4 TULOKSET

Tässä luvussa esitellään lyhyesti kyselyn tulokset ja konstruktiivisen tutkimuksen tulokset.

4.1 Kyselyn tulokset

Kyselyyn vastasi 22 henkilöä, joka on pieni määrä kunnolliseen tilastolliseen tutkimukseen, mutta on hyvin suuntaa antava. Tietoisia kyselystä oli noin 160 henkilöä, mutta tämä on vain suuntaa antava luku, sillä välttämättä kaikki nämä 160 henkilöä eivät nähneet viestiä, jossa kyselystä ilmoitettiin. Vastaamisprosentin voidaan siis näin ollen sanoa olevan 10 – 20 % luokkaa. Taulukossa 2 näkyvät kyselyn vastaukset. Osioon 2 jatkoi 9 vastaajaa, jotka siis käyttävät, ovat käyttäneet, tai aikovat käyttää WebAssemblya joko töissä tai muussa projektissa. Suurin osa vastaajista oli vain korkeakouluopiskelijoita ja pieni osa käy myös töissä opiskelujen aikana. Noin puolet vastasivat käyttävänsä tai harkitsevansa WebAssemblyn käyttöä. Suurin osa vastaajista eivät tienneet mitään, tai tiesivät hyvin vähän aiheesta. Osion 2 vastaukset painottuivat suurilta osin Likert-asteikon numeroarvoon 4, eli keskivaiheille.

4.2 Konstruktiivisen tutkimuksen tulos

Kaikki kolme projektia saatiin tehtyä loppuun asti yhden päivän aikana. Kaikkia projekteja varten asennettiin Express.js-kehitysympäristö .wasm-tiedostojen palvelemista varten. Se on suunniteltu erittäin yksinkertaisesti käytettäväksi ja helposti asennettavaksi kotisivuston ohjeiden mukaan, joten se riittää tässä tapauksessa. Koodiesitys 7 näyttää minkälaisella tiedostolla .wasm-tiedostot palveltiin.

Projekti, jossa siirrettiin C-kirjasto WebAssemblyyn, vei eniten aikaa ottaen huomioon koodin kirjoittaminen, työkalujen asennus, ja ongelmien korjaaminen. C-kirjasto, jolle tämä tehtiin, oli Webp-koodekin lähdekoodi. Tässä projektissa tuli eniten ongelmia vastaan. Liian suuri kuvakoko aiheutti virheen selaimen konsoliin. Virheilmoitus ei ollut aluksi selvä, että mistä se tuli. Tähän on dokumentaation mukaan kääntämisen yhteydessä parametri, joka korjaa virheen, mutta tässä tapauksessa asia korjattiin pienentämällä kuvan kokoa. Koodiesityksessä 10 näkyy HTML-tiedosto, jossa on rajapinta käännetyn WebAssemblyn ja JavaScriptin välillä. Koodiesityksessä 11 taas näkyy C-tiedosto, jossa on kaikki tarvittavat funktiot kuvan muuntamiseen. Tiedosto on normaalia C-kieltä ja siinä on EMSCRIPTEN_KEEPALIVE myös mukana . Emscripten- ja webp-kirjastot ovat tuotu mukaan. Encode-funktiossa on ohjelman pääfunktionaalisuus.

AssemblyScript-projektin sai toimimaan lähes suoraan ohjeiden mukaan ja ilman mitään ongelmia. AssemblynScriptin asennus onnistuu helposti Npm-paketinhallintajärjestelmällä (engl. package manager). Ainoa asia, mikä pitää muistaa AssemblyScriptin käytössä on se, että kaikki TypeScriptin tai JavaScriptin ominaisuudet eivät ole olemassa WebAssemblyssa. Käytettiin samoja lähdetiedostoja kuin koodiesityksessä 2, koodiesityksessä 3, ja koodiesityksessä 4.

Kolmas projekti vaatii Rust-ohjelmointikielen asennusta sen työkalujen ohessa. Asennettaessa työkaluja kävi ilmi, että joitakin vaadittuja ohjelmia tai työkaluja ei mainittu. Muun muassa C-kääntäjä, OpenSSL ja build-essential piti asentaa ennen jatkamista. Työkalujen asennukseen meni paljon aikaa, mutta asennusten jälkeen kaikki muut vaiheet dokumentaatiossa sujuivat ongelmitta. Koodiesityksessä 6 näkyy, että minkälainen lähdetiedosto on. Koodiesityksessä 9 taas näkyy, että minkälaisella tiedostolla WebAssemblya käytetään.

Koodiesitys 7. Express.js-tiedosto, jota käytettiin .wasm-tiedostojen palvelemiseen.

```
const express = require('express')
const app = express()
app.use(express.static('./'))
app.listen(8000, () => console.log('Serving at http://localhost:8000!'))
```

Koodiesitys 8. Komento, jolla saadaan käännettyä C-tiedostot WebAssemblyyn. [9]

```
emcc -O3 -s WASM=1 -s EXTRA_EXPORTED_RUNTIME_METHODS='["cwrap"]' \
    -I libwebp \
    webp.c \
           libwebp/src/{dec,dsp,demux,enc,mux,utils}/*.c
```

Koodiesitys 9. JavaScript-tiedosto, jolla voidaan käyttää WebAssemblya.

```
const js = import("./hello-wasm/pkg/hello_wasm.js");
js.then(js => {
  js.greet("WebAssembly");
      });
```

Taulukko 2. Saadut kyselyn vastaukset.

Osio 1								
	Vastaajien lukumäärä	**Vapaa kuvailu vastaajasta**						
1. kysymys	22	18 korkeakouluopiskelijaa, 4 työssäkäyvää korkeakouluopiskelijaa						
	Vastaajien lukumäärä	**Likert-asteikolla annettu arvosana (1 (En tiedä aiheesta mitään) – 7 (Tiedän aiheesta paljon))**						
		1	**2**	**3**	**4**	**5**	**6**	**7**
2. kysymys	22	12	8	1	-	-	-	1
	Vastaajien lukumäärä	**Vastattiin kyllä/ehkä**			**Vastattiin ei**			
3. kysymys	22	9			13			
Osio 2								
	Vastaajien lukumäärä	**Likert-asteikolla annettu arvosana (1 (En tiedä aiheesta mitään) – 7 (Tiedän aiheesta paljon))**						
		1	**2**	**3**	**4**	**5**	**6**	**7**
4. kysymys	9	-	1	-	6	1	1	-
5. kysymys	9	-	-	1	4	2	1	1
6. kysymys	9	-	1	3	5	-	-	-
7. kysymys	9	2	-	1	2	1	2	-
8. kysymys	9	-	1	-	4	-	-	3
9. kysymys	9	-	2	-	3	3	1	-
10. kysymys	9	2	2	2	3	-	-	-
11. kysymys	9	-	1	1	5	1	1	-
12. kysymys	9	-	1	1	5	1	1	-

Koodiesitys 10. HTML-tiedoston osuus, josta näkee kuinka WebAssembly-moduulit ladataan ja kuinka niitä käytetään. [9]

```
<body>
  <script>
    Module.onRuntimeInitialized = async _ => {
      const api = {
        version: Module.cwrap('version', 'number', []),
        create_buffer: Module.cwrap('create_buffer', 'number', ['number',
'number']),
        destroy_buffer: Module.cwrap('destroy_buffer', '', ['number']),
        encode: Module.cwrap('encode', '', ['number', 'number', 'number',
'number']),
        get_result_pointer: Module.cwrap('get_result_pointer', 'number', ''),
        get_result_size: Module.cwrap('get_result_size', 'number', ''),
        free_result: Module.cwrap('free_result', '', ['number']),
      };

      const image = await loadImage('/image.jpg');
      const p = api.create_buffer(image.width, image.height);
      Module.HEAP8.set(image.data, p);

      api.encode(p, image.width, image.height, 100);
      const resultPointer = api.get_result_pointer();
      const resultSize = api.get_result_size();
      const resultView = new Uint8Array(Module.HEAP8.buffer, resultPointer,
resultSize);
      const result = new Uint8Array(resultView);
      api.free_result(resultPointer);

      const blob = new Blob([result], { type: 'image/webp' });
      const blobURL = URL.createObjectURL(blob);
      const img = document.createElement('img');
      img.src = blobURL;
      document.body.appendChild(img)

      api.destroy_buffer(p);
    };

    async function loadImage(src) {
      // Load image
      const imgBlob = await fetch(src).then(resp => resp.blob());
      const img = await createImageBitmap(imgBlob);
      // Make canvas same size as image
      const canvas = document.createElement('canvas');
      canvas.width = img.width;
      canvas.height = img.height;
      // Draw image onto canvas
      const ctx = canvas.getContext('2d');
      ctx.drawImage(img, 0, 0);
      return ctx.getImageData(0, 0, img.width, img.height);
    }
  </script>
</body>
```

Koodiesitys 11. C-tiedosto, jossa on kaikki tarvittavat funktiot kuvan muuntamiseen. Tiedosto on normaalia C-kieltä ja siinä on EMSCRIPTEN_KEEPALIVE myös mukana . Emscripten- ja webp-kirjastot ovat tuotu mukaan. Encode-funktiossa on ohjelman pääfunktionaalisuus. [9]

```
#include "emscripten.h"
#include "src/webp/encode.h"
#include <stdlib.h>

EMSCRIPTEN_KEEPALIVE
int version() {
  return WebPGetEncoderVersion();
}

EMSCRIPTEN_KEEPALIVE
uint8_t* create_buffer(int width, int height) {
  return malloc(width * height * 4 * sizeof(uint8_t));
}

EMSCRIPTEN_KEEPALIVE
void destroy_buffer(uint8_t* p) {
  free(p);
}

int result[2];
EMSCRIPTEN_KEEPALIVE
void encode(uint8_t* img_in, int width, int height, float quality) {
  uint8_t* img_out;
  size_t size;

  size = WebPEncodeRGBA(img_in, width, height, width * 4, quality, &img_out);

  result[0] = (int)img_out;
  result[1] = size;
}

EMSCRIPTEN_KEEPALIVE
void free_result(uint8_t* result) {
  WebPFree(result);
}

EMSCRIPTEN_KEEPALIVE
int get_result_pointer() {
  return result[0];
}

EMSCRIPTEN_KEEPALIVE
int get_result_size() {
  return result[1];
}
```

5 KESKUSTELU TULOKSISTA

Tässä luvussa analysoidaan kyselyn tuloksia ja pohditaan konstruktiivisen tutkimuksen tuotoksia, sekä tiivistetään tulokset.

5.1 Kyselyn tulosten arviointi

Kyselyä jaettiin kanavilla, jotka olivat tietotekniikan korkeakouluopiskelijoiden käytössä, joten oli odotettavissa, että kaikki vastaajat myös olivat heitä. 22 vastaajaa on pieni määrä, jotta saisi tilastollista merkittävyyttä, mutta tulokset antavat silti hyvin suuntaa. Kyselyä olisi voinut jakaa muillakin kanavilla, jotta otoskokoa olisi saatu isommaksi. Hieman alle puolet vastanneista kertoi ettei ole käyttänyt, eikä aio käyttää WebAssemblya. Tämäkin oli odotettavissa, sillä teknologian uutuudesta johtuen moni ei siitä tiedä, ja sitä ei ole ehditty tietotekniikan alan koulutusohjelmiinkaan lisätä opetettavaksi. Yksi henkilö vastasi osaavansa WebAssemblya suurimmalla tasolla, eli hän luultavasti työskentelee teknologian kanssa työssään tai on muuten vain erittäin perehtynyt aiheeseen. Osion 2 vastausten perusteella voisi sanoa, että:

- Työn tehokkuus ei lisäänny monen vastaajan mielestä.
- Näkemykset WebAssemblyn hyödyllisyydestä olivat keskitasolla.
- Monen mielestä WebAssemblyn käyttäminen ei ole helppoa tai ei vaikuta helpolta. Tämä eriää työn kirjoittajan mielipiteestä, joka on, että WebAssemblyn käyttö pienessä projektissa ei ole vaikeaa. Toisaalta suuremmissa projekteissa se saattaa olla vaikeaa. Tämä on tapauskohtaista.
- Toiset ovat kuulleet joltakin arvostuksen kohteena olevalta taholta kehotuksia käyttää WebAssemblya – toiset taas eivät.
- WebAssemblyn käyttö on suurimmilta osin vapaaehtoista.
- WebAssemblyn käyttäjiä ei nähdä suuremmassa arvossa, kuin jos he eivät käyttäisi teknologiaa.
- WebAssemblyn käyttäminen ei ole tärkeää tai on yhdentekevää.
- Kysymysten 11 ja 12 tulokset olivat molemmat keskimääräisiä, ja näyttäisi siltä, että kyseessä olivat oletusvastaukset heille, joilla ei ollut oikeaa mielipidettä.

Siitä on käyty keskustelua, että pitäisikö Likert-asteikolla olla neutraali tai välinpitämätön vaihtoehto keskellä [17]. Tässä työssä päädyttiin antamaan neutraali vaihtoehdoksi. Kaiken kaikkiaan näkemykset ja mielipiteet olivat positiivisia, mutta käyttäminen nähtiin vaikeana ja ei välttämättömänä. Käyttämisen vaikeus eriää työn kirjoittajan mielipiteestä, joka on, että

WebAssemblyn käyttö pienessä projektissa ei ole vaikeaa. Toisaalta suuremmissa projekteissa se saattaa olla vaikeaa. Tämä on tapauskohtaista.

5.2 Konstruktiivisen tutkimuksen tuloksista keskustelu

AssemblyScript-projekti oli luultavasti helpoin kolmesta esimerkkiprojektista. Node.js on luultavasti asennettu ohjelmistokehittäjillä jos tekee työtä web-kehityksen parissa ja varsinkin silloin jos tekee TypeScriptin kanssa työtä, joten web-kehittäjät varmasti tuntevat olevansa tutun teknologian äärellä. C-kieli taas on erittäin käytetty kieli kun halutaan suorituskykyä, joten erilaisia kirjastoja voi löytyä paljonkin WebAssemblyyn käännettäväksi ja näin ollen hyödynnettäväksi muuallakin kuin työpöytäympäristössä, missä ne usein toimivat. C-kielen osaajia myös löytyy suuri määrä kielen yleisyyden takia. Emscripten-työkalun asennus oli suhteellisen helppoa ohjeiden mukaan. Emscripten antaa tarvittavat C-kielen standardikirjastot WebAssembly-moduulin käytettäväksi. Rust-projekti oli helppo tehdä, mutta Rust ei ole vielä niin yleinen kieli kuin C tai TypeScript, joten käyttäjiä ei oletettavasti tule olemaan suurta määrää ainakaan lähitulevaisuudessa. AssemblyScript sopii web-kehittäjille, Rust uuden teknologian adoptoijille, ja Emscripten C-kielen osaajille. Kaikki esimerkkiprojektit olivat helppo tehdä ohjeita seuraten ja dokumentaatiota lukien.

5.3 Tuloksista tiivistetysti

Kyselyn tulosten perusteella WebAssemblyn käyttäminen on helpompaa, kuin mitä alan opiskelijat luulevat. Se nähtiin kuitenkin hyödyllisenä – vaikkakaan ei kovin tarpeellisena tai välttämättömänä työkaluna. WebAssemblyyn kääntyviä ohjelmointikieliä ja työkaluja löytyy varmasti jokaiselle jotain. Ohjeita ja dokumentaatiota, joita voi seurata ja opiskella löytyy eri tahoilta paljon. Emscripten, Rust, ja AssemblyScript ovat erittäin kypsiä ja paljon käytettyjä ekosysteemejä WebAssemblyn kanssa työskentelyyn. Konstruktiivista tutkimusta tekemällä opittiin, että käyttäminen ei ole vaikeaa myöskään vaikeammissa projekteissa, joissa pitää esimerkiksi siirtää olemassa oleva C-kirjasto WebAssemblyyn ja JavaScriptin kanssa käytettäväksi. Jos siis tarvitsee determinististä ajonaikaista suorituskykyä, on WebAssembly hyvä valinta.

6 YHTEENVETO

Työssä tutkittiin WebAssemblyn käyttöä erilaisissa projekteissa, kuinka vaikeaa se on, kenelle se sopii, ja mitä mielipiteitä ja ennakkoluuloja alan opiskelijoilla on aiheesta. Todettiin jo alussa, että WebAssemblyn käyttäminen lisää ohjelman suorituskykyä ja ennalta-arvattavuutta. Parannukset ovat suurimpia kaikenlaisessa raskaassa videopeleissä, 3D-grafiikassa, videoiden toistossa, virtuaalitodellisuudessa, kryptografiassa, ja konenäössä. JavaScript on tiensä päässä sen suorituskykynsä suhteen, ja selainten JavaScript-moottoritkin ovat erittäin kehittyneitä nykyään. WebAssembly on tapa saada kaivattua suoritustehoa ja on jo nyt suuressa käytössä ja varmistanut tulevaisuutensa, sillä se on jo nyt osa suurinta osaa verkkoselaimista. Nykyhetkellä sen poistaminen selaimista saattaisi aiheuttaa enemmän ongelmia, kuin sen jättäminen nykytilaansa, joten se on tullut jäädäkseen. Teknologian tulevaisuudennäkymät ovat siis hyvät. Työssä tehtiin 3 esimerkkiprojektia ja kysely, joilla selvitettiin käyttöön ottamisen helppoutta ja teknologian imagoa.

Käyttöön ottamisen rima on matala, sillä suurelle osalle teknologiasta kiinnostuneille löytyy omat työtavat ja työkalut sen parissa työskentelyyn. Monet ohjelmoijat saattavat olla paljon tuottavampia toisella ohjelmointikielellä tai vain pitävät siitä paljon. Myös vanhan koodin uudelleenkäytettävyys vähentää vanhan koodin uudelleenkirjoittamista tai siirtämistä toiselle kielelle. AssemblyScript sopii web-kehittäjille, Rust uuden teknologian adoptoijille, ja Emscripten C-kielen osaajille. Kaikki esimerkkiprojektit olivat helppo tehdä ohjeita seuraten ja dokumentaatiota lukien. Kyselyn tulosten perusteella WebAssemblyn käyttäminen on helpompaa, kuin mitä alan opiskelijat luulevat. Se nähtiin kuitenkin hyödyllisenä – vaikkakaan ei kovin tarpeellisena tai välttämättömänä työkaluna.

WebAssemblyn käytettävyydestä ei ole kovin tutkimuksia. Ainoastaan sen syntaksista, semantiikasta, ja suorituskyvystä löytyy paljon tietoa. Työtä varten ei löytynyt juurikaan mitään teknologian käyttämisestä vaikka omassa projektissa. Jos tätä työtä haluaisi laajentaa, niin kyselyä voisi laajentaa isommalle kohdepopulaatiolle ja ottaa mukaan eri yrityksiä, jotka jo WebAssemblya käyttävät. Työn esimerkkiprojekteja voisi hyödyntää omissa projekteissaan tai työssä.

LÄHTEET

[1] Advanced Tools - WebAssembly: *https://webassembly.org/getting-started/advanced-tools/*. Accessed: 2020-03-16.

[2] appcypher/awesome-wasm-langs: 2020. *https://github.com/appcypher/awesome-wasm-langs*. Accessed: 2020-03-16.

[3] Babel · The compiler for next generation JavaScript: *https://babeljs.io/*. Accessed: 2020-03-15.

[4] Can I use... Support tables for HTML5, CSS3, etc: *https://caniuse.com/#feat=wasm*. Accessed: 2020-02-28.

[5] Clang C Language Family Frontend for LLVM: *https://clang.llvm.org/*. Accessed: 2020-03-16.

[6] Compiling from Rust to WebAssembly: *https://developer.mozilla.org/en-US/docs/WebAssembly/Rust_to_wasm*. Accessed: 2020-05-05.

[7] Couriol, B. 2019. WebAssembly 1.0 Becomes a W3C Recommendation and the Fourth Language to Run Natively in Browsers. *InfoQ*.

[8] Electron | Build cross-platform desktop apps with JavaScript, HTML, and CSS.: *https://www.electronjs.org/*. Accessed: 2020-03-15.

[9] Emscripting a C library to Wasm | Web: *https://developers.google.com/web/updates/2018/03/emscripting-a-c-library*. Accessed: 2020-05-05.

[10] Going public launch bug · Issue #150 · WebAssembly/design: *https://github.com/WebAssembly/design/issues/150*. Accessed: 2020-02-28.

[11] Grosskurth, A. and Godfrey, M.W. Architecture and evolution of the modern web browser. 24.

[12] Haas, A., Rossberg, A., Schuff, D.L., Titzer, B.L., Holman, M., Gohman, D., Wagner, L., Zakai, A. and Bastien, J. 2017. Bringing the web up to speed with WebAssembly. *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation* (Barcelona, Spain, Jun. 2017), 185–200.

[13] Hello, World! - Rust and WebAssembly: *https://rustwasm.github.io/docs/book/game-of-life/hello-world.html*. Accessed: 2020-03-17.

[14] Herrera, D., Chen, H., Lavoie, E. and Hendren, L. WebAssembly and JavaScript Challenge: Numerical program performance using modern browser technologies and devices. 26.

[15] Jacobsson, M. and Wåhslén, J. 2018. Virtual machine execution for wearables based on WebAssembly. (2018).

[16] Kitchenham, B. and Pfleeger, S.L. 2002. Principles of survey research: part 5: populations and samples. *ACM SIGSOFT Software Engineering Notes*. 27, 5 (Sep. 2002), 17–20. DOI:https://doi.org/10.1145/571681.571686.

[17] Kitchenham, B.A. and Pfleeger, S.L. 2002. Principles of survey research: part 3: constructing a survey instrument. *ACM SIGSOFT Software Engineering Notes*. 27, 2 (Mar. 2002), 20–24. DOI:https://doi.org/10.1145/511152.511155.

[18] Lonkar, A. and Chandrayan, S. 2018. The dark side of WebAssembly. *Virus Bulletin*.

[19] Main — Emscripten 1.39.8 documentation: *https://emscripten.org/*. Accessed: 2020-03-14.

[20] McConnell, J. 2017. WebAssembly support now shipping in all major browsers. *The Mozilla Blog*.

[21] McGregor, C. 2018. Using Constructive Research to Structure the Path to Transdisciplinary Innovation and Its Application for Precision Public Health with Big Data Analytics. *Technology Innovation Management Review*. 8, 8 (2018), 7–15. DOI:https://doi.org/10.22215/timreview/1174.

[22] Netscape Now Program: 1996. *https://web.archive.org/web/19961026040131/http:/www3.netscape.com/comprod/mirror/netscape_now_program.html*. Accessed: 2020-02-28.

[23] Node.js: *https://nodejs.org/en/*. Accessed: 2020-03-15.

[24] Pendharkar, P.C. and Rodger, J.A. 2007. An empirical study of the impact of team size on software development effort. *Information Technology and Management*. 8, 4 (Dec. 2007), 253–262. DOI:https://doi.org/10.1007/s10799-006-0005-3.

[25] Pfleeger, S.L. and Kitchenham, B.A. 2001. Principles of survey research: part 1: turning lemons into lemonade. *ACM SIGSOFT Software Engineering Notes*. 26, 6 (Nov. 2001), 16–18. DOI:https://doi.org/10.1145/505532.505535.
[26] Reiser, M. and Bläser, L. 2017. Accelerate JavaScript applications by cross-compiling to WebAssembly. *Proceedings of the 9th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages* (Vancouver, BC, Canada, Oct. 2017), 10–17.
[27] Stefik, A. and Hanenberg, S. 2014. The Programming Language Wars: Questions and Responsibilities for the Programming Language Community. *Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software* (Portland, Oregon, USA, Oct. 2014), 283–299.
[28] The AssemblyScript Book: *https://docs.assemblyscript.org/*. Accessed: 2020-03-16.
[29] The LLVM Compiler Infrastructure Project: *https://llvm.org/*. Accessed: 2020-03-11.
[30] torch2424/wasm-by-example: *https://github.com/torch2424/wasm-by-example*. Accessed: 2020-03-16.
[31] Use Cases - WebAssembly: *https://webassembly.org/docs/use-cases/*. Accessed: 2020-02-28.
[32] Venkatesh, V. and Davis, F.D. 2000. A Theoretical Extension of the Technology Acceptance Model: Four Longitudinal Field Studies. *Management Science*. 46, 2 (Feb. 2000), 186–204. DOI:https://doi.org/10.1287/mnsc.46.2.186.11926.
[33] wabt demos: *https://webassembly.github.io/wabt/demo/*. Accessed: 2020-03-16.
[34] Wasm By Example - Examples Listing: *https://wasmbyexample.dev/all-examples-list.html*. Accessed: 2020-03-16.
[35] Watt, C. 2018. Mechanising and verifying the WebAssembly specification. *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs* (Los Angeles, CA, USA, Jan. 2018), 53–65.
[36] WebAssembly Concepts: *https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts*. Accessed: 2020-02-27.
[37] WebAssembly High-Level Goals - WebAssembly: *https://webassembly.org/docs/high-level-goals/*. Accessed: 2020-02-28.
[38] WebAssembly/binaryen: 2020. *https://github.com/WebAssembly/binaryen*. Accessed: 2020-03-11.
[39] WebAssembly/wabt: 2020. *https://github.com/WebAssembly/wabt*. Accessed: 2020-03-16.
[40] Welcome to Native Client - Google Chrome: *https://developer.chrome.com/native-client*. Accessed: 2020-03-14.
[41] Zakai, A. 2011. Emscripten: an LLVM-to-JavaScript compiler. *Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion* (Portland, Oregon, USA, Oct. 2011), 301–312.
[42] *asm.js*.
[43] 2017. Flash & The Future of Interactive Content. *Adobe Blog*.
[44] Overview | CMake.
[45] 2019. *WebAssembly Core Specification*. The World Wide Web Consortium (W3C).