

WebAssembly: An Emerging and Future Trend in Web Development Technology

Gabriel Avbenake (1902228)

Course: L7 - Emerging and Future Trends
(ACCA7016)

University of Wales Trinity Saint David

2020

Table of Content

Table of Content	1
Table of Figure	2
Introduction	3

What is WebAssembly?	3
Why WebAssembly?	5
Features of WebAssembly	6
Efficient and fast	6
Safe	6
Readable, Open and Debug-able	7
Portability	7
Part of the open web platform	7
Off-the-Web Availability	7
Current Limitations and Future Directions	8
WebAssembly Concurrency	8
DOM Access	8
SIMD Support	8
Security Implications	9
Conclusion	9
References	10

Table of Figure

Figure 1: Illustration of WebAssembly's place in Web Development	5
Figure 2: WebAssembly: A joint effort aimed at providing a cross-browser compiler target	5
Figure 3: How to Implement WebAssembly on the Web	6

Introduction

The Web has evolved to a real application platform from its nascent simple document browsing and distribution environment. With the advancement of the Web, there are more increasingly demands and possibilities. It has become a popular target for varied myriad of applications. With this, there is the demand for performance-critical web code that can run compute-intensive applications.

For a long time, JavaScript has been the only programming language natively used in a web browser. JavaScript performance is insufficient for performance-intense applications like graphic manipulation, simulations, games or cryptography. The deficiency is inherent in the language itself [1]. Undoubtedly successful and convenient, JavaScript ran in any browser and enabled some kind of interactivity on web page. Nevertheless, it was a high-level language, not designed with computeintensive workloads in mind [2]. This limitation stifles performance. The web programming is usually limited by JavaScript performance and functionalities. In the past, additional functionalities could only be achieved using third-party plug-ins. There were attempts such as Flash, ActiveX and Java. But they all faded out of popularity. This is why most of the popular games and powerful applications are only available as a native desktop app [3].

All this is about to change with the arrival of WebAssembly (abbreviated as WASM).

What is WebAssembly?

WebAssembly is a portable, fast, small and load-time-efficient language which can be compiled to execute at native speed by taking advantage of common hardware capabilities available on a wide range of platforms, including mobile and IoT [4]. In other words, it is a platform-independent, lowlevel assembly-like language with a compact binary format that runs with near-native performance and provides languages such as C/C++ and Rust with a compilation target so that they can run on the web [5]. It is designed to run alongside JavaScript, allowing both to work together.

The open standards for WebAssembly are developed by the World Wide Web Consortium (W3C) in a W3C Community Group [6]. This includes a W3C Working Group as well as representatives from Google, Microsoft, Apple, and Mozilla (the owners of the four major browsers) [7].

Currently, WebAssembly is available in all the four major browser engines: Google Chrome, Mozilla Firefox, Apple Safari and Microsoft Edge [8]. WebAssembly is bringing native-like performance to the browsers, which hitherto was difficult. What is more, it is allowing developers to build these highspeed web apps in any language of their choice. Performance-intensive applications such as music streaming, games, video editing, photo editing, CAD applications etc can be delivered directly as web apps with the aid of WebAssembly [9].

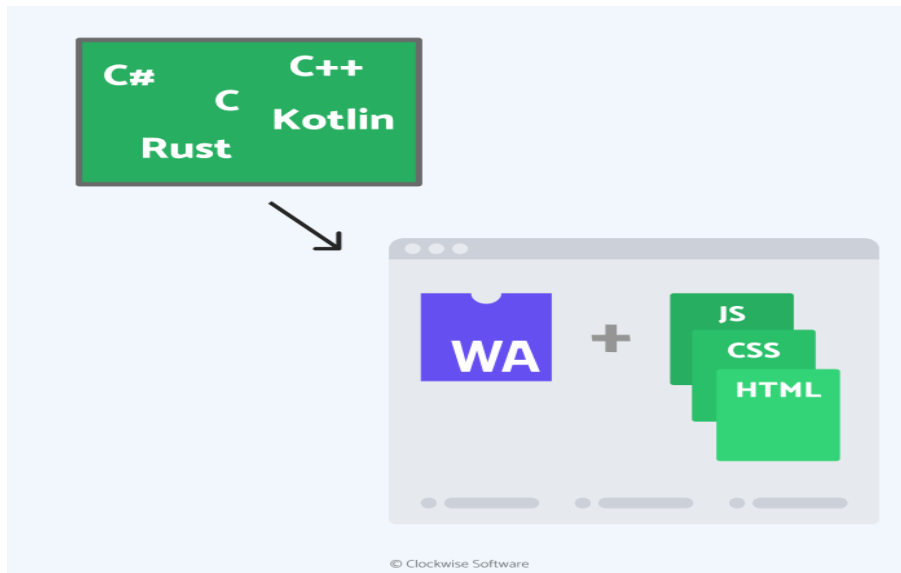


Figure 1: Illustration of WebAssembly's place in Web Development

WebAssembly is a compilation target for any language. It is not meant to be written directly. Developers need not write WebAssembly directly; they could write in any of their choice language, which is then compiled into WebAssembly bytecode. This is then run on the browser where it is converted into native machine code and executed at high speed [9].

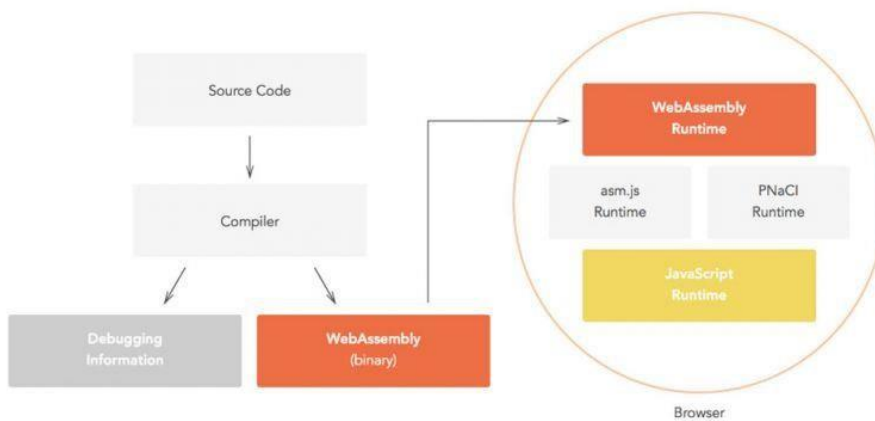


Figure 2: WebAssembly: A joint effort aimed at providing a cross-browser ~~compiler~~ target

Commented [GD1]: Ref figures in figure title/label please?

Why WebAssembly?

As stated earlier, with WebAssembly, graphics-heavy applications and games, heavy scientific simulation as well as other compute-intensive applications can run in a browser without plug-ins. Attempts in the past such as Flash, ActiveX and Java failed because, among other things, they needed plug-ins. WebAssembly also has non-web applications such as the Internet of Things, mobile apps and JavaScript virtual machines [10].

Another beautiful thing about WebAssembly is that it allows developers to put their large applications already written with C++/C, Java, Rust, Python, etc on the web without having to rewrite them from the scratch. Such applications just need to be compiled into WebAssembly, and WebAssembly would allow them to execute on modern web browsers at near-native speeds.

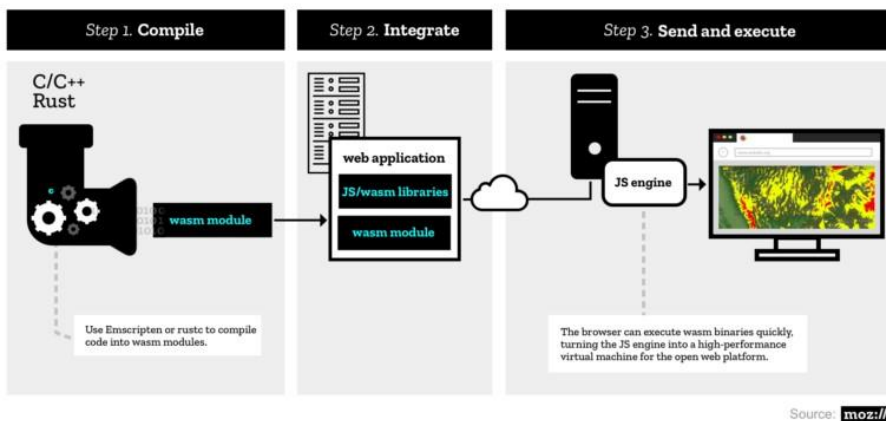


Figure 3: How to Implement WebAssembly on the Web

Apart from Flash, ActiveX and Java, other attempts were made, such as Google Native Client and Mozilla asm.js Project. Asm.js approach was to compile C/C++ codes to a low-level subset of JavaScript. However, JavaScript's inconsistent performance was still a limiting factor [8]. Native Client (NaCl) approach was to run compiled C/C++ codes in a secured compartment inside Google Chrome [11]. NaCl is not inherently portable. To address that, Portable Native Client (PNaCl) was developed. Its compactness was still an issue [8]. More so, it was exclusively in Chrome. Microsoft's ActiveX had security issues because it achieved safety through a trust model, not through technical construction [8]. Flash and Java, on the web, had security and performance issues [8].

All these (NaCl, PNaCl, asm.js, ActiveX, Flash and Java) failed because they did not meet all the requirements of fast, safe and portable semantics as well as efficient representation [8]. Here is where WebAssembly excels. As Haas, et al. describes it succinctly by fast, safe and portable semantics is meant safe and fast to execute, language-independent, hardware-independent and platform-independent, deterministic and easy to reason about as well as having simple interoperability with the Web platform [8]. By efficient representation is meant compact and easy to decode, easy to validate and compile, easy to generate for producers as well as parallelizable [8]. These were the goals for WebAssembly and it is achieving it. This is why it is replacing all the

previous attempts. More so, it is worth remembering that it is a joint effort by W3C, Microsoft, Google, Apple and Mozilla.

Features of WebAssembly

Efficient and fast

WebAssembly code is executed faster than JavaScript. This is because it is encoded in a load-time efficient binary format that is smaller in size than the JavaScript. Since WASM is smaller in size than JavaScript, the time to download its code would be faster than that of downloading JavaScript code. This is especially useful over slow networks. Of course, the size of a JavaScript bundle could be reduced significantly by compaction. Nevertheless, a compressed binary representation of WASM is still smaller [12].

Unlike JavaScript, WASM code is load-time efficient also because it does not go through any parsing transformation phase. WASM code is already an intermediate representation (bytecode) that only needs to be decoded and validated to ensure there are no errors [12].

For the compilation and optimization phase, WebAssembly starts off much closer to machine code than JavaScript. Furthermore, for WebAssembly, more optimizations have already been done ahead of time at the server-side. Hence, less work is needed to compile and optimize it [12].

Since WebAssembly was designed as a compiler target, it was designed for compilers to generate, and not for human programmers to write. Since human programmers don't need to program it directly, WebAssembly can provide a set of instructions that are more ideal for machines [12]. Taking advantage of common hardware capabilities available on a wide range of platforms, WebAssembly aims to execute at native speed [6].

Safe

Security is one of the key objectives of WebAssembly. From the documentation of WebAssembly, two important goals are integral to the security model of WASM. The first is to protect users from malicious programmes. The second is to provide developers with useful primitives and mitigations for developing safe applications [13].

With these goals, WASM codes executes inside a memory-safe, sandboxed execution environment separated from the host execution using fault isolation techniques. It is subject to the security policies of its embedding. For example, on the browser, it will be subject to the same permission security policies of the browser. Generally, applications in WASM execute deterministically with limited exceptions. They can only access outside environment using the appropriate APIs of the host. Furthermore, they execute independently of each other [13].

In WASM design, dangerous features were eliminated from its execution semantics. For example, at load time, all accessible functions and their associated types must be declared by modules. With this, control-flow integrity is enforced. Furthermore, hijacking attacks on control flow is prevented with WASM immutable runtime compiled code [13].

Commented [GD2]: You need to explain some terms to your reader. Include a glossary if there are many specialised term that you rely on in your discussions.

Readable, Open and Debug-able

WebAssembly is a low-level assembly language but has a human-readable text format. Its human-readable text format is convertible to and from the binary format, using View Source functionality. The textual format is for debugging, testing, teaching, learning, experimenting, optimizing and writing the modules by hand. It is the textual format that would show when viewing the source of WebAssembly modules on the browsers [6].

Portability

Portability here refers to hardware and platform independence. With its binary format, WebAssembly is designed to be efficiently executable on various operating systems and architectures, on the Web and off the Web [14]. This is paramount because the web is accessed by a great variety of devices, with different configurations and classes. There are diverse operating systems, machine architecture and browsers navigating the web. If WebAssembly is to succeed, it must allow applications to run across browsers with the same behaviour irrespective of the hardware and software configurations of the host device [8].

Part of the open web platform

WebAssembly is designed to execute and integrate well within the existing Web platform. This means maintaining the backwards-compatible nature of the web. Its modules will be able to call in and out of the JavaScript as well as access browser functionality through the same Web APIs accessible to JavaScript [4].

On the web platform, WebAssembly is not meant to replace JavaScript, but to accompany it and provide more functionality and performance. WebAssembly modules can be loaded into a JavaScript app and functionality can be shared between the two. With this, both advantages of the performance and power of WebAssembly as well as the expressiveness and flexibility of JavaScript can be harnessed in the same application [5].

Off-the-Web Availability

WebAssembly also supports non-web embeddings. Non-Web environments may provide different APIs than Web environments. Whereas web embeddings use Web APIs, non-web embedding might use POSIX or SDL or JavaScript VMs etc. WASI (WebAssembly System Interface) is a project by Mozilla to standardize how WASM code interacts with operating systems outside of a browser context [2].

With its web and off web availability, WASM compiled binaries can run both within and without browsers, across different devices and operating systems, at near-native speeds. Given this portability advantage, native apps developers might favour WebAssembly, in the future. A wide range of languages can be compiled to WebAssembly and it can then run on a wide range of platforms and hardware [2].

Current Limitations and Future Directions

One of the goals of WebAssembly design was *Specify and Implement* incrementally. The idea was to specify a Minimum Viable Product (MVP) for the standard. This becomes like the minimum for all browsers to work with and implement [4]. Then, gradually additional features could be added postMVP. Some of the current issues and possible future directions of WebAssembly include:

WebAssembly Concurrency

The MVP version of WebAssembly focused on single-threaded computation. Making different parts of a code run in parallel, at the same time, is one way to speed up code. Since WebAssembly is about performance-intensive applications, support for multiple threads is one of the proposals for future WebAssembly updates. However, multiple threading can sometimes backfire, since the overhead of communication between threads can sometimes take up more time than the task would have in the first place. This overhead can be reduced by sharing memory between threads. Thus, there is a proposal for synchronised access to shared memory side by side with multiple threading. One of the many future directions for WebAssembly is Shared memory concurrency. A suitable specification of a memory model is necessary for the future mechanisation of WebAssembly concurrency [15].

For now, WebAssembly Concurrency was somewhat coupled to JavaScript SharedArrayBuffer, which suffers from Spectre-style set of vulnerabilities. SharedArrayBuffer is a Standard built-in object in JavaScript that enables the allocating and sharing memory as well as the updating and synchronizing shared memory with atomic operations [16]. Spectre attacks can read arbitrary memory by exploiting speculative executions [17] [18] [19].

All major browsers have disabled SharedArrayBuffer by default due to the Spectre vulnerabilities [20]. Only Google Chrome has re-enabled it [21]. Until the vulnerabilities are resolved, WASM concurrency might not see widespread use.

DOM Access

Currently, there is no way for WebAssembly to interact with the Document Object Model (DOM) [22]. DOM is the handle that applications can use to change webpage content and style. WebAssembly would have to go through JavaScript to access it. Certainly, direct access would have been faster than going through JavaScript. This would certainly affect some applications of WebAssembly.

SIMD Support

WebAssembly does not have SIMD (Single Instruction, Multiple Data) support for now [22]. With SIMD, the same instruction can be applied to different parts of a larger data structure simultaneously. This can help speed things up especially in complex computations as in games or Virtual Reality.

Security Implications

With WebAssembly features and performance, it was used in browser-based miners by malware authors to mine cryptocurrency using the CPU resources of the victim. The mining process occurred, mostly unknown to the victim [23]. By leveraging WASM, these authors increased their illicit mining efficiency as crypto-mining requires large computing power [24].

Lonkar & Chandrayan, 2018 illustrated that WebAssembly can be used in Tech Support Scam to defraud people and in phishing campaigns (using keyloggers) to capture confidential information without leaving many traces for detection purposes. They worry that WebAssembly might be exploited in browser exploits and malicious redirections [23].

In the same vein, Bergbom, 2018 opines that there are some potential security issues with WebAssembly. For now, there is no way to verify that a WebAssembly application has not been tampered with. An official documentation for Integrity Checking of WebAssembly programmes is lacking [25]. Furthermore, WebAssembly applications are potentially exposed to some of the vulnerability classes present in native applications. Many of these applications are going to be simply compiled from pre-existing codebases with only minor modifications to allow them to work within a browser framework [25].

Certainly, it is going to be more laborious to analyse WebAssembly than JavaScript because it is a binary format whereas JavaScript is clear text. Analysts fear that, with WebAssembly binary code, malicious authors (such as scammers, malware writers and phishing attackers) might get new ways to hide the intentions of their code, since string-based detection is almost impossible [25] [23].

It is usual for new technologies to have some potential security issues which need to be considered. Security is one of the objectives of WebAssembly. A lot has already been done in the area of security. There are still ongoing improvements in its general implementation and security standards. Hopefully, the existing potential security issues would be considered in the Post-MVP.

Conclusion

WebAssembly is a new technology to come to the web platform in the past decade. As a new, lowlevel, assembly-like language that runs efficiently on the existing web platform, with backwardcompatible, it is a compiler target, enabling developers to translate their applications into web-ready modules that can run on browsers at near-native speeds [10].

The full impact of WebAssembly is yet to be felt on web development. The WebAssembly technology is still evolving with lots of potentialities. There have been many interesting developments. Some of these are highlighted in Colin Eberhardt's Article: "*WebAssembly 2019 Year In Review*" [26]. There is the online demonstration of running Doom 3 Demo on a web browser using WebAssembly [27]. This is available on <http://wasm.continuation-labs.com/d3demo/>. Of course, this is still work in progress. There is also the ongoing project to bring the full scientific Python stack to run entirely on the browser using WebAssembly [28]. There is also .NET 5 that can now run on WebAssembly. It is a unified .NET platform/framework that targets Windows, Linux, Android, iOS, macOS, tvOS and WebAssembly etc. [29].

Nonetheless, the impact of WebAssembly is still small; not many developers have keyed into it yet. Many are stuck on JavaScript and other components. It might take some time before WebAssembly would have a landscape conversion. It would be interesting to monitor and juxtapose its adoption

Commented [GD3]: Don't place new work in conclusions.

Commented [GD4]: Ditto above. Also this point needs a greater discussion in main body of your report.

rate and impact on web and off web to see how the trend develops. There is no doubt, however, that WebAssembly has plenty of potentials. It is going to be very influential in the area of Progressive Web Apps (PWAs) which is a growing trend to develop web apps that combine the best of websites and mobile apps [30]. Of course, PWA can be achieved without WebAssembly.

WebAssembly is an emerging web development trend that has the potential to be a game-changing factor for the web and beyond. Certainly, some of its present challenges would need to be addressed, especially security vulnerability issues.

References

- [1] M. Reiser and L. Bläser, "Accelerate JavaScript Applications by Cross-Compiling to WebAssembly," New York, 2017.
- [2] M. Bursell and G. Haff, "Why everyone is talking about WebAssembly," 2020. [Online]. Available: <https://opensource.com/article/20/1/webassembly>. [Accessed 16 May 2020].
- [3] S. Merenych and A. Popovych, "Web Development Trends and the Latest Web Technology Stacks in 2020," 2020. [Online]. Available: <https://clockwise.software/blog/web-developmenttrends/>. [Accessed 14 May 2020].
- [4] webassembly.org, "WebAssembly High-Level Goals," [Online]. Available: <https://webassembly.org/docs/high-level-goals/>. [Accessed 16 May 2020].
- [5] MDN contributors, "WebAssembly: In a Nutshell," 2020. [Online]. Available: <https://developer.mozilla.org/en-US/docs/WebAssembly>. [Accessed 16 May 2020].
- [6] webassembly.org, "WebAssembly," [Online]. Available: <https://webassembly.org/>. [Accessed 14 May 2020].
- [7] C. Watt, "Mechanising and Verifying the WebAssembly Specification," New York, 2018.
- [8] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai and J. F. Bastien, "Bringing the web up to speed with WebAssembly," *ACM SIGPLAN Notices*, vol. 52, no. 6, 2017.
- [9] S. Yegulalp, "What is WebAssembly? The next-generation web platform explained," 2018. [Online]. Available: <https://www.infoworld.com/article/3291780/what-is-webassembly-the-next-generation-web-platform-explained.html>. [Accessed 14 May 2020].
- [10] Mozilla Research Expanding the Foundations of the Open Web, "WebAssembly format," 2018. [Online]. Available: <https://research.mozilla.org/webassembly/>. [Accessed 16 May 2020].

- [11] google-nativeclient, "Welcome to Native Client," [Online]. Available: <https://developer.chrome.com/native-client>. [Accessed 17 May 2020].
- [12] L. Clark, "What makes WebAssembly fast?," 2017. [Online]. Available: <https://hacks.mozilla.org/2017/02/what-makes-webassembly-fast/>. [Accessed 15 May 2020].
- [13] webassembly.org, "WebAssembly Documentation: Security," [Online]. Available: <https://webassembly.org/docs/security/>. [Accessed 16 May 2020].
- [14] webassembly.org, "WebAssembly Documentation: Portability," [Online]. Available: <https://webassembly.org/docs/portability/>. [Accessed 16 May 2020].
- [15] C. Watt, U. Rossberg and J. Pichon-Pharabod, "Weakening WebAssembly," *Proceedings of the ACM on Programming Languages*, vol. 3, no. OOPSLA, p. Article 133, October 2019.
- [16] MDN contributors, "SharedArrayBuffer," 2020. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/SharedArrayBuffer. [Accessed 15 May 2020].
- [17] Graz University of Technology, "Meltdown and Spectre: Vulnerabilities in modern computers leak passwords and sensitive data," 2018. [Online]. Available: <https://meltdownattack.com/>. [Accessed 15 May 2020].
- [18] B. L. Titzer and J. Sevcik, "A year with Spectre: a V8 perspective," 2019. [Online]. Available: <https://v8.dev/blog/spectre>. [Accessed 15 May 2020].
- [19] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwartz and Y. Yarom, "Spectre Attacks: Exploiting Speculative Execution," San Francisco, 2019.
- [20] C. Eberhardt, "Faster Fractals with Multi-Threaded WebAssembly," 2019. [Online]. Available: <https://blog.scottlogic.com/2019/07/15/multithreaded-webassembly.html>. [Accessed 15 May 2020].
- [21] A. Danilo, "WebAssembly Threads ready to try in Chrome 70," 2018. [Online]. Available: <https://developers.google.com/web/updates/2018/10/wasm-threads>. [Accessed 15 May 2020].
- [22] L. Clark, "Where is WebAssembly now and what's next?," 2017. [Online]. Available: <https://hacks.mozilla.org/2017/02/where-is-webassembly-now-and-whats-next/>. [Accessed 14 May 2020].
- [23] A. Lonkar and S. Chandrayan, "The dark side of WebAssembly," 2018. [Online]. Available: <https://www.virusbulletin.com/virusbulletin/2018/10/dark-side-webassembly/>. [Accessed 16 May 2020].
- [24] G DATA Security Lab, "Breaking New Ground: WebAssembly's Rising Role in Web Development and Its Security Concerns," 2019. [Online]. Available: <https://www.gdatasoftware.com/blog/2019/11/35494-webassemblys-rising-role-in-webdevelopment-and-its-security-concerns>. [Accessed 17 May 2020].

- [25] J. Bergbom, "WebAssembly security: potentials and pitfalls," 2018. [Online]. Available: <https://www.forcepoint.com/blog/x-labs/webassembly-potentials-and-pitfalls>. [Accessed 17 May 2020].
- [26] C. Eberhardt, "WebAssembly 2019 Year In Review," 24 December 2019. [Online]. Available: <https://blog.scottlogic.com/2019/12/24/webassembly-2019.html>. [Accessed 16 May 2020].
- [27] G. Cuvillier, "D3WASM: A Port of ID Tech 4 / Doom 3 Engine to WebAssembly," 2019. [Online]. Available: <http://www.continuation-labs.com/projects/d3wasm/>. [Accessed 16 May 2020].
- [28] M. Droettboom, "Pyodide: Bringing the scientific Python stack to the browser," 2019. [Online]. Available: <https://hacks.mozilla.org/2019/04/pyodide-bringing-the-scientific-python-stack-to-the-browser/>. [Accessed 16 May 2020].
- [29] R. Lander, "Introducing .NET 5," 2019. [Online]. Available: <https://devblogs.microsoft.com/dotnet/introducing-net-5/>. [Accessed 16 May 2020].
- [30] D. Ramel, "Blazor WebAssembly Tackles Progressive Web Apps," 2020. [Online]. Available: <https://visualstudiomagazine.com/articles/2020/03/11/blazor-pwa.aspx>. [Accessed 16 May 2020].
- [31] wasmweekly.news, "WebAssembly Weekly," 2017. [Online]. Available: <https://wasmweekly.news/>. [Accessed 16 May 2020].

MARKING CRITERIA AND STUDENT FEEDBACK

This section details the assessment criteria. The *extent* to which these are demonstrated by you determines your mark. The marks available for each criterion are shown. Lecturers will use the space provided to comment on the achievement of the task(s), including those areas in which you have performed well and areas that would benefit from development/improvement.

Common Assessment Criteria Applied (Assignment 1)	Marks available	Marks
1. Engagement with Literature Skills Extent of research and/or own reading, selection of credible sources, application of appropriate referencing conventions.		
You have engaged with relevant and credible sources. The appropriated referencing format used. Perhaps an over-reliance on web refs.	20	12.5
2. Knowledge and Understanding Skills Extent of knowledge and understanding of concepts and underlying principles associated with the discipline.		
You have explained and discuss the selected topic well. Some ideas and terms would benefit from a brief introduction or definition.	20	14
3. Cognitive and Intellectual Skills Analysis, evaluation and synthesis; logic, argument and judgement; analytical reflection; organisation of ideas and evidence		
A good introduction into the research topic area. While your report explores WebAssembly it would be good to have included a case study here - perhaps expanding on your points/references you included in the conclusion or used in an experimental way. Use diagrams that can help to support your discussions. Re. Fig 2 the right hand side did not add an awful lot here and key relationships needed identifying/describing. No new discussion/ topics/ideas should be in your conclusion, you should be summarising the salient/important points made earlier in the report. (Grade mark: 70.00%)	50	34.5
4. Practical Application Skills Deployment of methods, materials, tools and techniques; application of concepts; formulation of innovative and creative solutions to solve problems.		
5. Transferable Skills for Life and Professional Practice Attributes in professional practice: individual and collaborative working; deployment of appropriate media; presentation and organisation.		

Your report flow well, was generally well structured. AS mentioned earlier, no new discussion/ topics/ideas should be in your conclusion. You should be summarising the salient/important points made earlier in the report.	10	7
--	----	---

Assignment Mark (Assessment marks are subject to ratification at the Exam Board. These comments and marks are to give feedback on module work and are for guidance only until they are confirmed.)	Late Submission Penalties (tick if appropriate)		68%
	Up to four weeks late		
	First Attempt?		