console.log("The Web that Won't Hold"); Software Inefficiency and Electronic Waste

Juni Kim

January 29, 2022

Since its inception in the 1990s, the world wide web has seen exponential growth. Although at first it consisted only of static documents, JavaScript was soon developed in order to make web pages more interactive to the user. The language has now become ubiquitous, powering programs of all scales from simple blogs to applications that handle millions of users a day. However, the encroachment of progressively larger amounts of JavaScript into even non-web applications hastens the obsolescence of functional hardware, fueling an unsustainable consumerist cycle in the process. Unusable hardware is dumped as electronic waste (or "e-waste"), which has devastating consequences for the environment and its many inhabitants. The current ecosystem of software development, and particularly the widening presence of web technologies, encourages wasteful and environmentally harmful practices.

1 let title = 2; title = "Fundamental Inefficiencies in JavaScript"

JavaScript was invented in the days of Netscape by Brendan Eich. In just 10 days, Eich created Mocha, the direct ancestor to JavaScript. This initial implementation allowed developers to make their websites interactive by adding simple animations and effects. It was thus not intended nor designed as a platform in which desktop-like applications could run. This can be seen with the case of Google Docs, which has replaced traditional word processors like Microsoft Word. Since such mainstream applications are written in an inherently unoptimal language, they prevent older hardware from efficiently running them, making functional computers unusable. Although there are several independently written runtimes that are capable of executing JavaScript, they all share several core features that make them more inefficient.

Runtimes in Javascript have to determine the type of a variable, or what category of data the variable represents, when executing code. For instance, numbers and characters are different types and thus have different properties and behaviors; you can add two numbers but not two characters together. In some programming languages, such as C, variable types never change after assignment. The compiler, a program which converts C to machine code, can guarantee that when the program is executed, behaviors of all variables will be valid.

JavaScript, on the other hand, allows variables to change types while code is being executed, which means that the runtime must continuously check as to whether a variable can do actions that it was able to do before. Although this system, known as dynamic typing, allows developers to rapidly write programs without double-checking their work, it

creates more overhead on the part of the runtime because it has to re-evaluate types that other programming languages do not.

JavaScript also requires a process called garbage collection, which abstracts memory management away from the developer but increases evaluation time. In C, which is not garbage collected, the developer must manually allocate and free memory for objects. This makes the program more efficient but can also lead to memory leaks if certain objects are not freed when a program ends. JavaScript runtimes automatically perform this process via reference counting, where they keep track of the number of variables that exist that currently point to an object. When this number reaches zero, the object is deleted (the garbage collection step). This functionality makes the application more secure but requires JavaScript execution to be continuously supervised. The developer is free to create virtually unlimited waste, knowing that the runtime will clean it up.

2 return <Frameworks language="JavaScript" />

As JavaScript continued its meteoric rise, applications were forced to keep their increasingly complex state in sync with what was being displayed to the user. To illustrate, if a developer needs to write a simple to-do list, they need to keep a store of exactly what to-do's are in the list as well as displaying this information in a manner legible to the user. When the user adds a new to-do, the program must both update the code that stores the list and display the new to-do to the user. When writing in plain JavaScript, these updates are written separately, which leaves the codebase fragile. If even a single update isn't properly done, the entire system will fail to properly display data.

JavaScript frameworks, such as React, remedy this problem by updating the user interface when any change to the state is made, making sure that the internal and external updates always happen together. Although they make it easier for the developer to write complex applications, they can also mask the amount of JavaScript that must be executed in order for the program to work properly. In particular, frameworks have motivated the rise of single-page applications, where instead of fetching a new document on every page load, the browser first loads all of the necessary JavaScript, known as a bundle, and creates the document on the spot. Notable examples of single-page applications include Instagram and Gmail. Although these applications minimize page loads (because no document is ever loaded after the first download), they create considerable overhead because the JavaScript must not only make incremental updates but also rewrite entire sections of the webpage. It is also inefficient and therefore wasteful because every reload of the website requires the code for the entire app to be re-downloaded. These practices result in the pointless execution of megabytes of code.

3 window.loadFile("The Web on the Modern Desktop.html")

Due to the particularly large number of JavaScript developers, Electron, a desktop framework, was created to allow them to write desktop applications while only using JavaScript.

Electron tends to be economically advantageous for companies because they do not have to hire separate teams of developers with distinct skills to create applications that run both on the browser and on the desktop. Web developers can also use this framework when they

want to create applications that need privileges beyond what the browser gives them, such as accessing the filesystem.

However, Electron ships a modified version of the Chrome browser because it is the only way by which JavaScript can be executed. Due to its dependence on JavaScript, Chrome is already inefficient and resource-hungry. This effect is compounded when users run multiple Chrome-based applications simultaneously.

For instance, a typical user may have Spotify, Slack, Skype, and Chrome running while they are working, which essentially means that they are running 4 separate instances of Chrome. This creates considerable strain on the resources of any computer that isn't brand new, rendering it incapable of performing everyday tasks. Electron, by bringing the web browser to the desktop, incentivizes developers to write inefficient software and thus hastens the obsolescence of computers.

4 while(true) console.log("Environmental Consequences")

E-waste created via deprecated hardware creates long-term consequences for the environment when its constituent substances proliferate.

A 2020 report by the Global E-waste Monitor indicated that approximately 53 million tons of e-waste was generated in a single year. Just two years prior, according to Ohio State University, this number was at 40 million tons, representing a 33% increase. At this rate, by 2030, we will be producing over 200 million tons of e-waste every year. OSU also indicates that e-waste, despite contributing to 2% of landfill trash, contributes 70% of toxic heavy metals, such as lead, cadmium, and mercury. This issue is compounded by the fact that less

than 20% of all e-waste is properly handled and recycled.

Improper methods of e-waste disposal create dire environmental consequences. When e-waste is burned, the resulting products are incorporated into the air, which contributes to air pollution and respiratory problems in humans and other animals. Heavy metals from e-waste can contaminate the soil and groundwater, with long-term ramifications for local agriculture and water quality. Bioamplification then increases the amount of toxins that humans ingest by orders of magnitude, further accentuating the many hazards that they pose. Lead, for instance, has severe effects on animal nervous systems; it also bioaccumulates, meaning that it is not removed by the excretory system. As a result, people do not know that they are lead poisoned until their levels are dangerously high.

E-waste can also have catastrophic effects on the biodiversity of local ecosystems because only certain organisms and microorganisms can survive in toxic environments. Ecosystems then become more vulnerable to disruptions that may eliminate entire species.

Although many of these consequences stem from the hazardous substances in the hardware, poor software practices indirectly cause these effects. Since older computers are unable to run new programs, consumers are forced to prematurely discard their electronics and create millions of tons of pointless waste.

5 console.error("Consumerist Underpinnings")

Driving the proliferation of web-based applications is a cultural need to use the latest technologies and abstractions without regard for the usability of older solutions. These practices are indicative of a short-sighted and consumerism-based economy that prioritizes material

and technological progress over sustainability.

Modern consumers often obtain products with the expectation that they will soon be replaced with an upgraded version. Products tend to be updated with superficial features that do not inherently increase their usefulness nor efficiency, but companies use these updates as a way of inducing consumers to discard previous versions, whether or not they are functional. There is a primary focus on gaining new improvements rather than repairing minor issues.

Tech corporations, such as Apple, take advantage of this need to upgrade hardware by making their devices difficult to repair. For instance, screen repairs by non-partner shops will disable Apple's biometric identification. This forces consumers to repair their iPhones via Apple's official service, where they usually pay an amount near-equivalent to the price of the next phone. This is by design; the parts of a screen simply cannot cost remotely the same as that of the whole phone. As a result, the user will buy a replacement phone while discarding an otherwise perfectly repairable one.

Our focus on hardware also comes from the way in which we evaluate new technology. Reviewers typically focus on the fact that a computer is powerful because it can run many resource-hungry applications at the same time. There is minimal focus as to whether it is even necessary for these programs to have such hardware requirements. Perhaps we should place greater value on applications that are able to run on the lowest powered hardware.

An adage called Wirth's law states that software is becoming slower at the same rate that hardware is becoming faster. Even developers acknowledge that they write inefficient and bloated code that requires the capacities of newer hardware.

As an example of such bloat, node_modules is a folder in JavaScript-based projects that

contains the code for all of the libraries that are used. The size of this folder has faced ridicule because of its tendency to be accidentally committed to the cloud, causing gigabytes of JavaScript to be uploaded for no reason. For reference, a common meme in the developer community compares the size of node_modules to being heavier than a black hole. This brings into scope both the unreasonably large dependencies for JavaScript projects along with the vast amount of code that users execute.

6 printf("Solutions")

Although the web permeates the world of software, we can mitigate its inefficiencies by preferring applications that computers can run natively. Although near-native web technologies have been developed, the vast majority of applications continue to be written using JavaScript and its many front-end frameworks.

WebAssembly is a well-known example of a near-native web technology. WebAssembly files are binaries that are optimized for the browser, cutting time that would have been spent on parsing out JavaScript. However, the runtime is still in its early stages compared to JavaScript, which has had decades to optimize its performance. JavaScript also has millions of libraries available for creating applications; if a similarly-scaled ecosystem can be created in WebAssembly, it could potentially become a more appealing choice for developers.

Although it is possible to make web applications less bloated, one can also refer back to the UNIX Philosophy as a more economical way of using a computer. UNIX is the ancestor to both MacOS and Linux, and its original implementation did not contain monoliths, but rather a set of small programs that could each perform a single task. Users could chain

up these programs to perform more complex actions, using far fewer resources than a giant JavaScript application ever could. As consumers move their computing to the cloud, it may be reasonable to use basic utilities of the computer in order to accomplish the same tasks.

There have been international efforts to properly recycle e-waste and to mitigate some of its destructive effects. However, these efforts are merely treating the symptoms of the underlying problem, as this paradigm simply enables users to maintain an unsustainable cycle of replacing their hardware as developers create more resource-intensive applications. There has been minimal discussion on the role that software plays in the creation of e-waste. Software, and the attitude that developers have towards its creation, must prioritize efficiency in order to make technology more sustainable.

7 process.exit(0)

JavaScript is spearheading a vicious cycle in software development where applications are becoming increasingly bloated, making them unusable on older computers that do not have today's specifications. The language is not optimal because it was designed for small-scale purposes and contains runtime features that slow down code execution. JavaScript is usually paired with a web framework, which further multiplies the runtime requirements for a website. Technologies such as Electron permit JavaScript applications to be run on the desktop, which strains computers whose users wish to run multiple apps concurrently. This inefficiency forces consumers to unsustainably replace their computers, with severe environmental repercussions. We must reconsider how the browser has encroached into our lives and continues to fuel our consumerist attitudes toward technology.

Developers and users alike consistently embrace the latest developments without considering how they might affect existing infrastructure. Consumers assume that the hardware should serve the software and should therefore be discarded when it cannot run the inefficient monoliths that developers create. However, software, by its nature, is expendable and upgradable, whereas every hardware upgrade creates physical waste. We need to create a paradigm in which software is catered to the hardware, where programs do not need to make computers obsolete so that they are able to run. The web, as it is currently structured, will not hold; it is time we rethink software.