Chapter 3: Evolution and Revolution

> *There are two ages of the Internet – before Mosaic, and after. The combination of Tim Berners-Lee's Web protocols, which provided connectivity, and Marc Andreesen's browser, which provided a great interface, proved explosive. In twenty-four months, the Web has gone from being unknown to absolutely ubiquitous.* – A Brief History of Cyberspace, Mark Pesce, ZDNet, October 15, 1995

In 1994, two graduate students at Stanford, Jerry Yang and David Filo, published a webpage they called "Jerry and David's Guide to the World Wide Web."[1] At first the page was only a collection of Jerry's golf scores and some links to content the two enjoyed on the Web. But as they added more links, almost overnight, the guide become a massive hit. According to Mental Floss magazine, "[Jerry and David] quickly realized they might need a name that took less than three minutes to say, so they switched to a word they liked from the dictionary – one that described someone who was 'rude, unsophisticated, and uncouth.'"[2] Yahoo! was born.

Yahoo! was the first example of a Web portal: a website that provided hierarchical lists of links, allowing users to browse the Web from a central hub. When the Web was small enough, people could remember off the top of their heads which webpages they might want to visit and where they were located. As the Web grew however, the need for link aggregators like Yahoo! to provide content organization as a service became increasing important. For a while, this model was extremely successful; Yahoo! captured the public's imagination. Unfortunately for the portal sites however, the growth of the Web was unrelenting. Quickly the idea that one group of people could understand the entire Web, and manually manage a set of links to the best content became unrealistic. Before long, to keep up with the pace of growth, Yahoo! would need to add hundreds of links to its hierarchy every day.

In 1996, a different pair of Stanford graduate students saw the problem differently. "Each computer was a node, and each link on a Web page was a connection between nodes - a classic graph structure,"[3] Larry Page, one of these students, told Wired Magazine in 2008. The Web, Page and his partner Sergey Brin theorized, was the largest graph ever created.[4] And it was growing at a breakneck pace. In order to organize this graph so that it could be reasonably understood by Internet users, they needed a heuristic for a webpage's importance. Their key insight was that the importance of a page was built into the graph. Chances were that the nodes with the most incoming connections were the most important. For instance, if all the best webpages on dogs linked back to a certain page, that page likely contained some high value information on dogs. Page and Brin wrote an algorithm that assigned scores to webpages using this heuristic, and applied it to the whole Web and allowed people to use it to search. Their results blew every other search engine out of the water; Google, the company founded around the algorithm, quickly became the Web standard for search.

The explosion of the Web in the late 90s demanded smarter search and machine learning techniques to map it, but the Web was not finished growing. In 1998, when Google's search engine first indexed the Web, there were 28 million unique web pages. In 2000, there were a billion and in 2008, Google's announced that their index has recorded over a trillion unique pages.[5] User counts moved in

---

1 http://www.theguardian.com/business/2008/feb/01/microsoft.technology
2 http://blogs.static.mentalfloss.com/blogs/archives/22707.html
3 http://www.wired.com/wired/archive/13.08/battelle.html
4 http://www.wired.com/wired/archive/13.08/battelle.html
5 http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html

tandem with more content. In December 1995, the month Netscape launched Navigator 2.0, there were estimated to be 16 million internet users in the world. In 2013, there are 2.7 billion.[6] Such dramatic changes in the size of the Internet caused major changes in the nature of the technology. Once a provence of static documents for scientists at large research universities, the Web became a multifaceted platform for almost everything. Amazon proved that the Web could be used as a storefront. Online forums and blogs proved that the Internet was a place for hobbies and politics. MySpace and Facebook proved that the Web could be a place to socialize. Throughout the late 90s and early 2000s, massive markets were opened up on the Web for watching movies, reading books, meeting strangers, and buying and selling just about anything. Not only did the graph grow exponentially, but its cultural impact on the world grew as well. Today the Web is a cultural phenomenon. And under the hood of this phenomenon, powering its growth and feeding off its successes, is a funny little language called JavaScript.

JavaScript Evolution

As the Web became more mainstream, users demanded more and more web page interaction. Flashing banners, buttons that responded when clicked, and embedded games became popular attractions that drove traffic (and ad revenue). There were a few common ways to achieve these effects if you developed for the Web over the last ten years. Java was one option

Both explicitly and implicitly, JavaScript has become a better language since Eich first wrote it in May 1995. First of all, many of the early complaints against JavaScript aren't actually the language's fault. These complaints fall into two categories: complaints about the DOM and HTML. And complaints about JScript and JavaScript being slightly different.

Then there were complaints that were actually against the language itself. Some of these got fixed by ECMA.

Then there were complaints about speed.

- JavaScript gets faster, stronger, better in the browser
  - Browser implementations converge to common standard
    - In early days, scripts in Internet Explorer vs. other browsers weren't 100% compatible
  - ECMA takes over specification of JavaScript
    - Netscape hands off JavaScript to ECMA for standardization in 1996
    - ECMAScript 3
      - First changes to spec made by ECMASCRIPT
      - Adds regular expressions and try, catch error handling
    - ECMAScript 4 → never released due to political fallout
    - ECMAScript 5 → current version supported in all major browsers
      - Adds 'strict mode' and support for JSON
    - ECMAScript 6 (codename: Harmony)
      - “const” statements like C++
      - “let” statements like ML

6 http://www.internetworldstats.com/emarketing.htm

- Not yet supported in all browsers – only some features in some browsers

Need for Speed.

- Improvements to JavaScript engines → leaps and bounds increases in language performance
  - SpiderMonkey
    - Originally developed by Brendan Eich at Netscape – the first JavaScript engine
      - Now open source. Development lead by Mozilla.
    - Compiles JavaScript to an intermediate language (C++)
    - Major (JIT) improvements to keep up with V8, others
  - Rhino
    - Developed at Mozilla, but runs on the JVM – compiles JavaScript to Java classes
  - V8
    - Developed by Google (Lars Bak) for use in Chrome in 2008
    - Made a number of improvements on SpiderMonkey performance
      - Compiles all JavaScript to machine code and then runs as if compiled C
      - Forces SpiderMonkey to make big improvements
      - Explicitly built to "raise the bar" of JavaScript performance
  - Because many of the best engines are open source and built into other applications, browsers are not the only beneficiaries of fast JavaScript (example: Node)

JavaScript Revolution

- AJAX and the invention of the web app
  - HTML was designed as a tool for describing documents, not tools
  - HTTP protocol was not designed for building rich applications
    - It was originally designed to serve purely static content
      - Browser makes a request to a server
      - → server returns file
    - But developers extended this model in order to build applications
      - Browser makes a request to a server
      - → server makes a request to backend application code
      - → app code queries database, which returns data
      - → app code constructs HTML file with data
      - → server returns file
    - Even the extended model is very limited compared with desktop applications
      - Page is static, only really good for viewing data
      - Page must be refreshed to show any changes in data
      - JavaScript can be used to tie together HTML elements, but it is sandboxed to the browser
        - No data JavaScript handles can get back to the application layer
        - JavaScript can only receive data passed to it – one way channel

    - The 'XMLHttpRequest' method
        - Originally developed for Microsoft's Outlook Web Access (2000)
            - Developer: Alex Hoppman
            - Outlook team convinces Internet Explorer team to implement it in IE5
                - 2 weeks before launch
        - Implemented by other browsers
        - Allows for JavaScript code to make requests to servers for data without page refresh
            - JavaScript can send data back to the server
            - Changes in the front-end can cause new data to be loaded
    - Google proves the concept with GMail and
        - XMLHttpRequest hardly used for many years
        - Potential was not understood
    - HTML5 and CSS3
- Popular libraries make JavaScript more expressive in the browser
    - JQuery
- JavaScript as a compilation target
    - GWT (Google Web Toolkit, 2006)
        - Toolkit for writing managing complex JavaScript applications in Java
    - CoffeeScript (2009)
        - Transcompiles to JavaScript
        - Better functional concepts / uses only 'good' parts of JavaScript
            - Steals functional concepts ideas from Ruby, Python and Haskell
            - Allows for array comprehensions, gets rid of semicolons, no global variables
    - Many others: Ruby, Python, Java, Haskell, Scala → JS

Great example for compilation!
The JavaScript Problem

A 2012 post on HaskellWiki, a community edited website for developers working in the Haskell programming language, defines “The JavaScript Problem.” The problem definition is two-fold. First, “JavaScript sucks.” According to the page, “The depths to which JavaScript sucks is well-documented and well-understood.”[7] According to HaskellWiki, these include JavaScript's weak type system, its verbose syntax, its lack of a module system, and its strange equality behavior. Those grievances might be reason enough not to write the language, especially for those on HaskellWiki—who presumably are happy writing Haskell. However, the second part of the problem makes it clear “We need JavaScript.”[8]

The fact is that JavaScript is the only language you can write for the browser.

- JavaScript leaves the browser
    - NodeJS
        - Ryan Dahl
        - Why use JavaScript?

7 http://www.haskell.org/haskellwiki/The_JavaScript_Problem
8http://www.haskell.org/haskellwiki/The_JavaScript_Problem