Sean McQueen

Politics and JavaScript:
an Odd Language and the Future of the Web

Chapter 1: Introduction

- JavaScript is a big deal:
  - Debated: is JavaScript the world's most popular programming language?
  - Likely running in more places than any other language:
    - More Internet connected devices = more JavaScript runtime environments
    - Currently 2.5 billion Internet users, 10 billion Internet devices → projected to continue growing at a rapid pace
  - JavaScript rode the wave of the web: "capitalized on an emerging frontier in computing"
    - Similar to how C capitalized on Unix, Java and PHP on web 1.0, etc.
- Java Script is defined by dichotomy:
  - Amateur vs. pro users of JavaScript
  - Is it a toy language or the language of large software projects?
    - Copy & Paste culture of amateur web development
    - Google Maps / Gmail / Outlook → professional software engineering
  - Ubiquitous adoption vs. known technical drawbacks
    - JS was written in 10 days by 1 person, truly the opposite of Java's origin story
    - JavaScript is everywhere, yet so many have criticized the language
    - Language is hated and loved
- Central questions for exploration:
  - How did JavaScript become so popular?
  - What forces have changed the language (is it the same language today?)
  - What is the future of JavaScript and who will be shaping it?
- Core argument:
  - An unlikely choice for the language of the web, JavaScript has ridden a perfect storm of political forces to become the most important language of the modern era. The future of the web as a software platform rests on the future of JavaScript.

Chapter 2: the origins and early history of JavaScript

The creation story of Javascript explains its strange nature and much of its popularity.

- Origin story at Netscape
  - A product of the browser wars
    - Fierce competition between Netscape's Navigator and Microsoft's Internet Explorer
    - An unmet need for a web scripting language
      - This was not necessarily obvious, but visionaries at the time saw the need for more than static content
  - Brendan Eich – creator
    - Background: operating systems and networking engineer at Silicon Graphics – 7 years
      - Came to Netscape to write functional languages (Scheme)
    - Wanted to use a functional language for the browser – Scheme
    - Writes 'Mocha' in 10 days, is renamed 'LiveScript', later 'JavaScript'

        - Where we find Eich later: CTO of Mozilla
            - Still an outspoken leader in the JavaScript community
    - Java and JavaScript: relationship between Sun Microsystems and Netscape
        - Java in secret development from 1990 – 1995 as the *Green Project*
            - Well planned, built by a team. Very much the opposite of JavaScript.
        - First public release of Java by Sun is accompanied by a surprise announcement from Netscape that Java will be incorporated into Navigator.
            - Executives at Netscape and Sun both want to beat Microsoft
        - Java was going to be for server programming → top Netscape execs wanted JavaScript to look/feel similar
            - In the end, JavaScript is sort of an afterthought
                - Thought is that Java might be end up being the browser language anyway
                - JavaScript can be lightweight, temporary. “Just make it look like Java.”
            - Bill Joy (Sun co-founder) was influential over development
            - Eich tries to make Scheme look like Java – perhaps explains JavaScript syntax
- Microsoft implements JavaScript for Internet Explorer
    - Called JScript to avoid trademark conflicts
    - Originally modeled off of Netscape's Javascript – copied to compete
        - Still implemented in Internet Explorer, now based on ECMA standards
        - Marginal differences between JScript and JavaScript today
    - Key point: Internet Explorer effectively wins the browser wars
        - Dominates market share for a decade (70 – 90% of Internet users)
        - For many years, all other browsers must follow its lead
            - For later browsers (Firefox, Safari, Chrome, Opera) supporting JavaScript is a no-brainier
            - Seems inconceivable that another web scripting language could gain traction
- 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
        - Not yet supported in all browsers – only some features in some browsers
- Problems with JavaScript from the beginning
    - Bad object oriented concepts
        - Prototyping OO rather than classes (debate on whether this is good or bad)
        - The 'this' keyword is used to mean different things in different contexts
    - Bad functional concepts
        - Scope is per function, not block

- Poor design:
  - No built-in module system
    - No 'import' concept – relies on HTML for dependency imports
  - Implied global variables, if 'var' is left out
  - Auto type conversation between strings and numbers leads to confusing behavior
  - Takes a lot of C's bad ideas (++, - -, switch statements, blockless statements)
- Type system:
  - No static typing → difficult to reason about JS programs formally
  - No Array type, just objects acting as hashmaps
  - '==' vs '===' : comparison is hard to understand – odd behavior
  - Only one 'Number' type
    - Implemented as IEEE floating point number, no ints
    - No 'bigints' – overflows of floating points a major issue
- Security problems, for a language with big security needs
- Note: these issues were not considered much of a problem until later
  - At its creation, it would have been crazy to compare language features of JavaScript to those of Java, Python or C.

Chapter 3: the JavaScript Revolution

Popularity of the web → many large, semi-related positive changes to JS → JS is suddenly attractive for big complex projects in many enviroments

- Backdrop: the explosive growth of the Internet
  - Sheer numbers of nodes and edges of the web explodes
    - 1995, 16 million Internet users → 2005, 1 billion Internet users
    - 1995, ~100,000 websites → 2005, hundreds of millions of websites
      - Number of web*pages* likely much greater
  - Search engines organize the web → Internet becomes more usable, enables growth
  - Users demand more and more interaction on webpages
    - No longer just for static academic documents
    - People want flashing banners, browser games, pages that react to user input
    - Adobe's Flash is a popular solution
      - Introduced in 1996 by Macromedia (later bought by Adobe)
      - Uses ActionScript, a superset of EMCAScript with different OO model (so even this is JavaScript)
      - Apple does not support Flash with hugely popular iOS → damages Flash popularity
  - Result of all this → JavaScript use grows very rapidly
- 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
  - EMCA improvements are significant
    -
  - 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)
- AJAX and the invention of the web app
  - Limitations with JavaScript data handling
  - XMLHttpRequest implemented in Internet Explorer
  - GMail proves the concept
  - 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
- JavaScript leaves the browser
  - NodeJS
    - Ryan Dahl
    - Why use JavaScript?
  - MongoDB
    -

Chapter 4: case study on the current state of JavaScript application development
My work with David Ordal and the Exavault Team provides a good study of the benefits and challenges of modern JavaScript application development

- Technologies
  - NodeJS
  - Express web framework for Node
  - AngularJS
  - MongoDB

    - Grunt
- Successes
- Challenges
- Observations

Chapter 5: the future of JavaScript

The future of the web platform rests on JS, which rests on the actions of Google, Microsoft and Mozilla

- The future of the web as a software platform (even operating system?)
    - Desktop applications have always been more powerful
        - Can you build something off the quality of Excel or Photoshop with the web standard?
    - Open vs closed platforms
        - iOS → the future of the Internet applications?
- Remaining problems with JavaScript today
    - Still no static types
        - Makes tooling, static analysis difficult
        - Static analysis of JavaScript is an active area of research
    -
- The major political forces today: the browser makers
- Google's vision
    - Dart
    - NaCl
    - Angular
    - Continued support of ECMA
- Microsoft's vision
    - TypeScript
    - Web standards built into Windows 8
- Mozilla's vision
    - Emscripten + asm.js
        - Emscripten: LLVM to asm.js compiler
        - asm.js: strict subset of JavaScript
            - "its just JavaScript" → its compatible with all existing browsers
            - For browsers that support it, allows for Ahead-of-Time compilation (AOT)
        - Entire chain: C++ → LLVM → asm.js → browser → SpiderMonkey → machine code
            - For developers already writing C++, web becomes just another compile target