CISC 322/326 Fall 2018: A1 Report

Google Chrome Conceptual Architecture Analysis

TEAM 25: CHROME CASTAWAYS

Erik Koning

Roman Sokolovski

Ryan Rossiter

Sammy Chan

Sarah Nowlan

Scott Reed

TABLE OF CONTENTS

ABSTRACT

In this report we will be analyzing the conceptual architecture of the Google Chrome web browser and its subsystems. We start by looking at Chrome and Chromium as a whole and identifying the tasks it preforms, which we then will categorized to be implemented as subsystems. While looking at the dependency of these subsystems we settled on object-oriented style to be most fitting for our conceptual architecture. This report will also go further into detail regarding each of the 6 subsystems and the architecture style they implement. With the use of sequence diagrams, we demonstrate our understanding of how the subsystems work together to complete tasks within our proposed architecture. Next, we evaluate how our architecture allows for concurrent execution and how chrome uses this capability to create a reliable, responsive experience. Lastly, this report will cover possible issues the chromium development team would have implementing our conceptual architecture and the lessons we have learned throughout the process of creating this report.

INTRODUCTION

Google released two browsers in 2008, Google Chrome and Chromium. Google chrome is their proprietary browser that is powered by chromium at the lower level but builds on top of it with Google proprietary features. Google utilizes Chromium's open source systems and flexible design to build on top of it and make it more user centric. Some of the features google adds to Chromium include automatic updates, bug tracking, increased google suite app support, and restrictions for certain extensions that can be added to improve security.

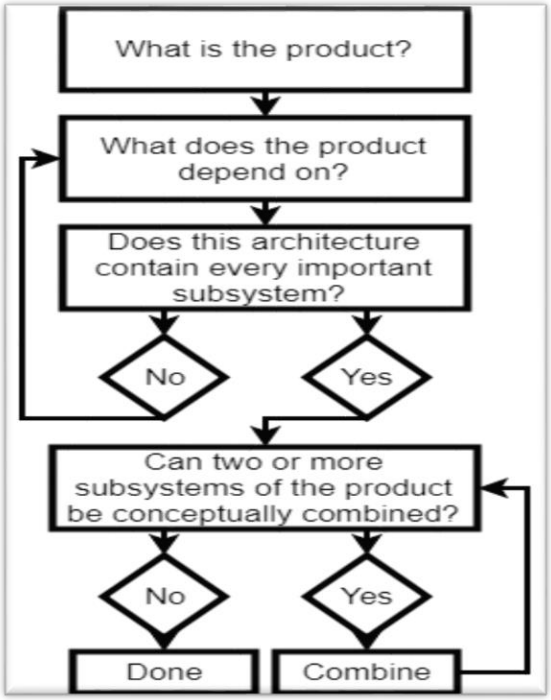
The Chromium browser was designed to reinvent the web interface with a browser that was both minimal in its user interface and responsive to the user even when multiple tabs are opened. Since 2008 chrome has gone through many iterations in its design and has expanded its feature set. The notable additions to chromium since its inception include, support for Linux in 2009, improved graphics support engines such as Skia, and the Blink engine by 2013. 2012 brought tab synchronization over multiple devices by connecting a browser user with a google account. 2014 brought improved JavaScript integration, and then in 2015 support for “Ok Google” voice commands.

The Chromium browser architecture will be the focus of this report since Google’s Chrome has a much broader feature coverage, and more involved architecture which is not in scope for this report. The Chromium browser will be investigated for its architecture, and subsystems will be further investigated for their functional requirements, and dependencies. Reasoning for the architecture chosen, use case diagrams, and development issues the Chromium team would have faced will also be discussed.

DECISION PROCESS

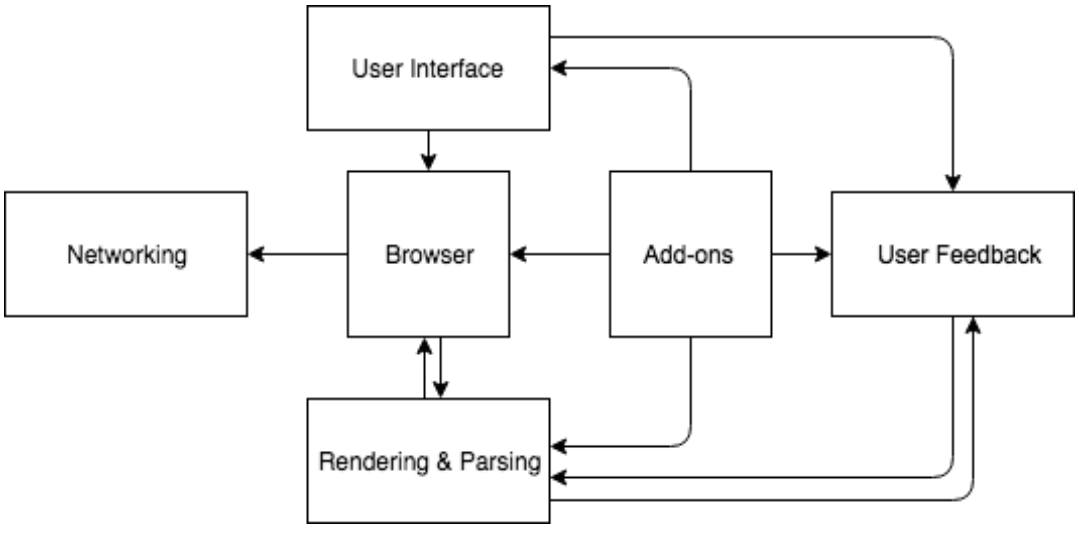
After our team sat down to develop the first attempt architecture, we had some difficulty sticking with a consistent process for developing and improving the architecture. Our initial architecture had some contradictions with our findings of how Google Chrome is designed that

caused our initial architecture to be invalidated and decided to re-work our architecture with a new derivation process.



We devised this process around an initial product which we believe is the UI, because it is what the user is directly working with and all inputs and outputs are served on. All the other subsystems in Chrome are thought of as helping the UI do its job. We then listed all the subsystems the UI depends on to make it function correctly, this can include the render engine, IPC's, plugins, and user data. This is similar to a divide and conquer algorithm because we are expanding all the possible system blocks to be listed and will simplify them down to around 6 later. Once all relevant subsystems were listed and dependency arrows drawn, we considered if subsystems could be combined conceptually based on their dependencies. Once the architecture could no longer be simplified, we were happy with it and did a final sequence diagram example to make sure it made sense to us.

CONCEPTUAL ARCHITECTURE



Our team explored several different architecture styles including layered and pipe and filter. We decided to use the object-oriented style and focus on the individual subsystems to emulate how chrome operates on a macro level. The subsystem Browser handles all of the requests and operations of the browser which relying on the other subsystems for core elements of the browser such as user interface and extensions/theme to be separately implemented. As a concept this allows for the development team to focus on each component independently. As different display drivers (such as DX12) or security standards change submodules can be updates as needed without having to inspect the whole architecture. However, this requires extensive testing to make sure direct and indirect interactions between the six subsystems do not cause unintended results.

COMPONENTS

USER INTERFACE

The User Interface subsystem controls the base elements of the browser such as tab handling as well as settings menus. User Interface controls the visualization of these features and is dependent on the Browser subsystem which controls each of the specific features of chrome. User Interface is also dependent on User Feedback as many UI elements such as volume indicator and mute functions is dependent on the user feedback subsystem.

Themes and visual changes to chrome (such as password managers) are handled by User Interface which Browser and thus Rendering and Parsing handles for all visual changes to the browser.

BROWSER

The Browser subsystem would be the central part of the Chrome browser, where the browser engine and data persistence are present. The browser is responsible for performing tasks such and operations going back and forward a page, reloading a page and opening new tabs. It also manages data such as cache, bookmarks, browsing history, browser account, storing login information, managing tabs and information about the current session.

The Browser acts as an interface between Networking and Rendering. When the user enters a website into the address bar, that information is used by the browser which sends it to Networking in order to retrieve the website's data. Once the browser has retrieved the page info from Networking, it can than send it along with any relevant data such as cached information. Rendering takes the data from the Browser, so it can be rendered and parsed in order to be displayed. As the user continues interact with the webpage, so too does the continual back and forth between these subsystems.

NETWORKING

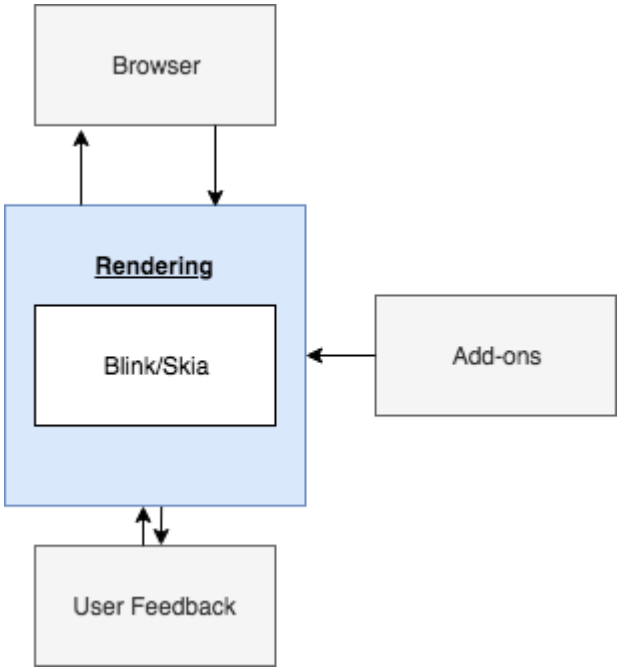
The Networking subsystem is responsible for handling all the Chrome browser's communication with the internet. It behaves as an asynchronous API that parses URLs and can retrieve content via several different protocols. Additionally, it can open and manage WebSockets upon request. It is solely used by the Browser subsystem, which relays URLs to Networking when there is content requested by a page's contents.

Networking also provides a cache for frequently requested content; this could be any document-based content (excluding WebSockets). Upon receiving a URL to fetch the Networking subsystem will first check if it has that document in the cache. If the cached document is found in the cache it is returned to the browser, otherwise the document is requested from the URL's domain.

RENDERING

The majority of the browser's function lies in the Rendering subsystem. This subsystem produces the rendered content for tabs and add-ons and it executes Javascript within that content. At its core

are Blink and Skia; Blink is a rendering engine for HTML and CSS, and Skia is a graphics library used for other graphical features.



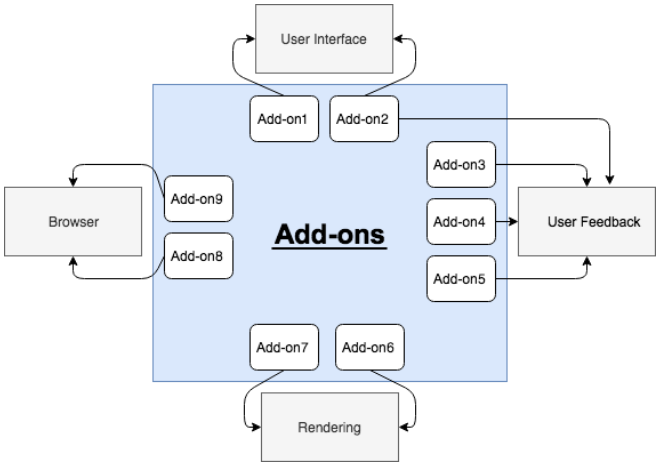
The rendering engine Blink is a fork of Apple's WebKit; after originally making heavy modifications for WebKit to work in Chromium, the team decided to fork it and make optimizations from within to better suit the requirements of the browser. Blink also embeds the Javascript engine v8, which was developed by The Chromium Project, to execute Javascript on the rendered page.

When the browser receives the main HTML document for a website, it sends the contents of that document to the Rendering subsystem. Rendering then parses and renders the content and requests any other content that is referenced on the page from the Browser subsystem (which relays the request to Networking). Once the page has been fully rendered, the resulting image is sent to User Feedback to be displayed. Input events generated by the User Feedback subsystem (click, key press, etc) are sent to the Rendering subsystem to be handled by the page's content. Any other forms of user feedback (sound, notifications) generated by the content or scripts running on the page are sent to User Feedback.

Add-ons work similarly to a tab in the browser, they can request the rendering of content. Add-ons can also inject Javascript into tabs, providing additional functionality such as ad blocking.

ADD-ONS

In many of the reference architectures observed, plugins were usually included as a subsystem. Plugins allowed for additional functionality not already contained within the browser. Websites accessed may request access to or require installation of some plugins to make use of their features, Adobe Flash Player being the one of the most common examples. Overtime, additional methods of customization and modification have become available. Extensions allow the user to modify their browser experience, from simple appearance changes, to changing how sites are rendered, thus affecting more subsystems of the browser. The concept of plugins did



not extend to such capabilities. For this reason, the Add-ons subsystem was derived to encompass plugins, extensions, themes and other such features.

Add-ons do not affect one another and typically serve a single purpose, only affecting on or two other subsystems. Extension such as add-blockers typically provide additional browser widgets for ease of access in addition to blocking out adds. From this point of view, add-ons resembled a repository style architecture but with the other subsystems acting as repositories for the individual add-ons. Since add-ons are optional, no other subsystems are dependent on it while it is dependent on others. Using this style of architecture, add-ons could be added, removed and replaced without affecting other add-ons or subsystems. This is reflected in the actual Chrome browser where extensions are able to be added, disabled and removed without affecting other systems. Also reflected in the real Chrome browser, updates to the browser or specific subsystems may require updates to the plugins that depend on it, just as updates to Chrome may break functionality of some extensions, requiring the developers to update them.

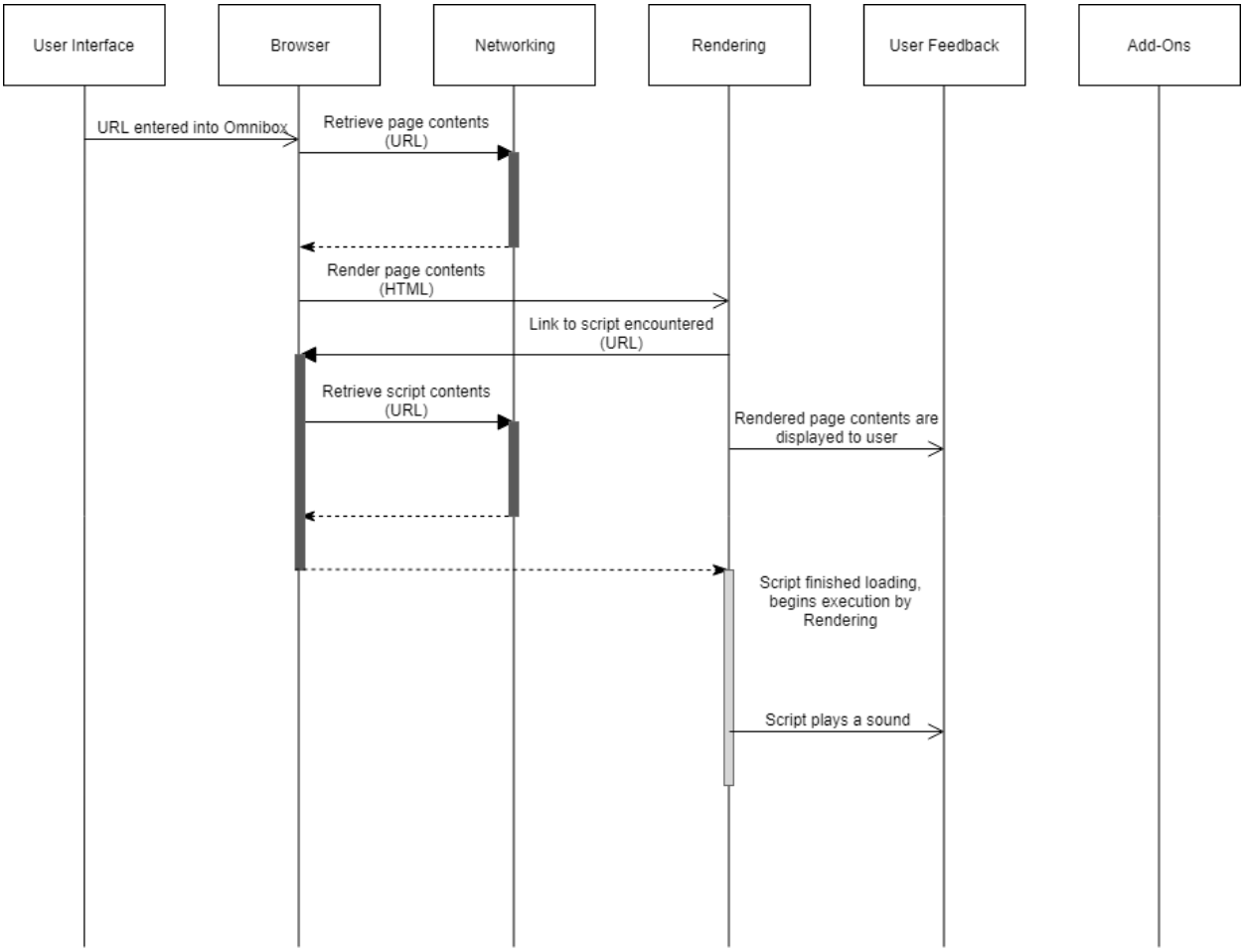
User Feedback

The term “User Feedback” is used to encompass both visual feedback and auditory feedback. Reference architectures used featured “Display Backend” modules as their primary method of feedback. When taking into account concurrency, many users commonly have tabs with music playing in the background while they use another tab. A subsystem for display only would not be able to account for this, thus User Feedback was derived to account for both display and audio.

In terms of display, User Feedback is responsible for the appearance of the browser and displaying web pages. The User Interface is dependent on User Feedback as it gives visuals to the widgets and other items the user would interact with along with providing visual feedback for actions such as hovering and clicking. User feedback itself is dependent on the rendering subsystem to provide the information it is meant to display for the website.

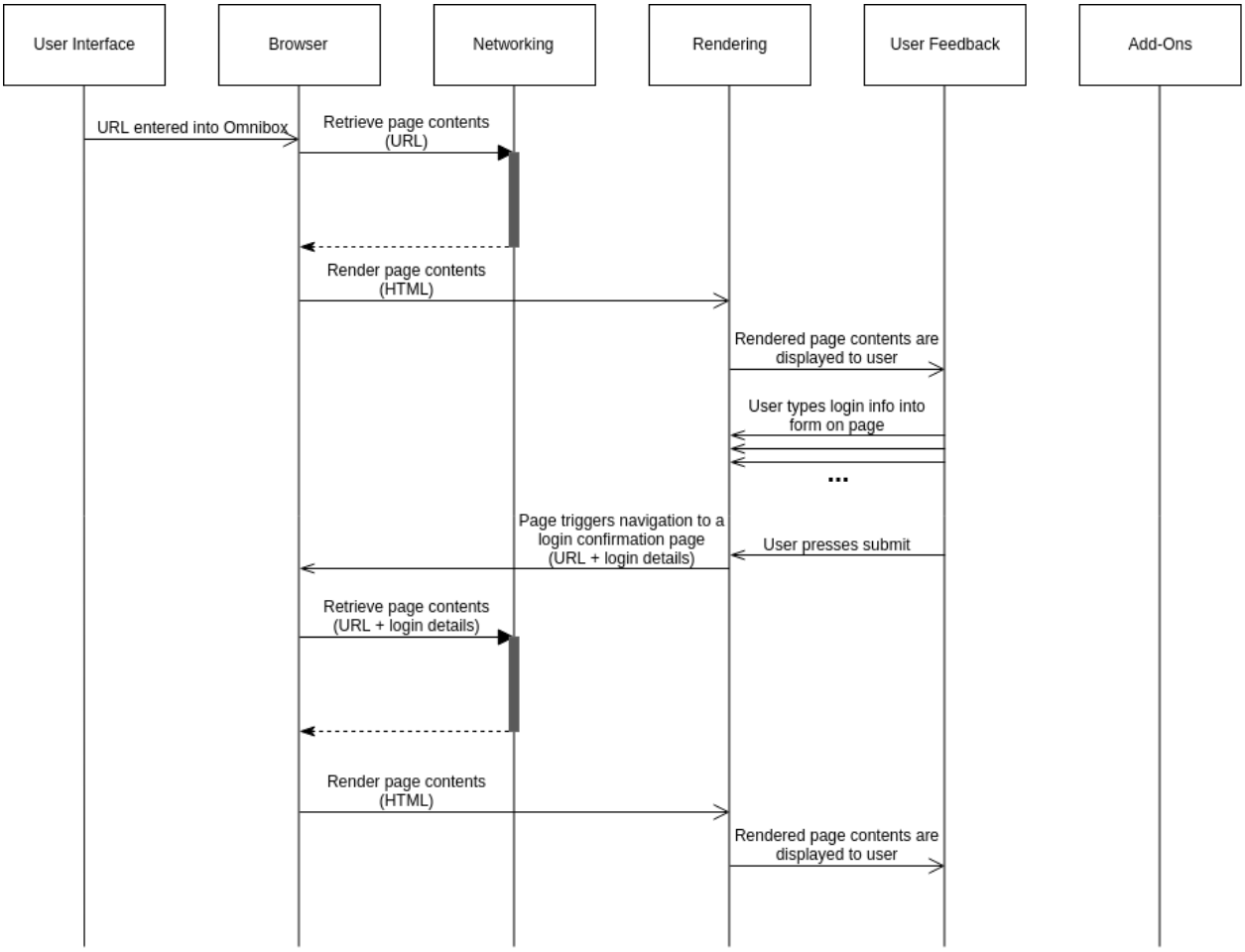
USE CASE SEQUENCE DIAGRAM

CHROMIUM RENDERING A PAGE WITH JAVASCRIPT



This sequence diagram illustrates the flow of execution within the browser when it is loading a page that contains a script to be run. After the URL is entered into the Omnibox by the user, the browser loads the page's main content (HTML) and sends it to Rendering. During the parsing of the document, the Rendering subsystem encounters a script tag with a URL source, so it requests that script from the Browser which triggers another document load from Networking, and ultimately returns the script contents to Rendering. Rendering then executes the script, and the script plays a sound that is sent to User Feedback to be heard by the user.

LOGGING INTO A WEBSITE AND CHROMIUM SAVES THE PASSWORD



Taking into account feedback received, the following use case diagram differs than what was presented. This diagram contains the sequence of events executed by the browser when a user visits and logs into a website. After the initial page load that is started by the user entering a URL into the Omnibox, the rendered page is displayed to the user when Rendering sends the rendered contents to User Feedback. The user then types their login information into the website, illustrated as a series of independent input events fired from User Feedback to Rendering. Rendering accepts these key inputs and passes them to the page content, in this situation the destination for the inputs would be an input field. Upon a click event fired from User Feedback to Rendering (In this case the assumed destination for the event is a "Submit" button), the contents of the page trigger a navigation away from the current page to submit the login details by notifying the Browser subsystem of this navigation. This triggers another page load similar to the initial load, except the request also contains the login details entered by the user. When the response of the request is received, the page contents are rendered by Rendering and displayed to the user.

DATA DICTIONARY

Term	Description
User Interface	The visual components located on the front end of the browser application. These components consist of input and output text boxes, buttons, and other elements the human user interacts with to complete the objects said user has while using the browser
Browser	The browser engine handles the processing of input and output data that is taken and given, from and to the user respectively, and provides a high-level reference to the under-laying rendering engine.
Networking	The networking engine that handles URL lookup and forwarding of the network requests over file transfer protocols such as HTTP and FTP.
Rendering	The rendering engine which produces the visual representation of the data elements, such as the HTML and XML documents which can be optionally typed with CSS.
User Feedback	A new term we created to describe both auditory and visual feedback the browser handles.
Add-ons	Customizable features chromium can take on that modify or extend the experience for the user. This can include plugins for file type's, or extensions to the user interface & browser such as ad blockers.

TEAM ISSUES

Although our architecture was specifically designed to minimize issues. When trying to implement it, the chromium development team may encounter a few possible problems. They may find that our subsystems are quite reliant on one another, meaning there are multiple two-way dependencies. The chromium development team would have to make sure that each component is working properly, and that nothing is broken. In the case of a subsystem being un-operational, it may negatively affect other components in the system leading to potential failure.

Another issue that the chromium development team may find is that since there are always new features and new add-ons being implemented into the browser, and due to this they may find that it is quite difficult to maintain the system. Since browsers have become more capable over the years so have the threats against them. Occasional security updates would be beneficial to our architecture and Ongoing security maintenance would be required to make sure that all components are up to date and that no malicious activity has occurred.

LESSONS LEARNED

In our first assignment, developing the proper conceptual architecture for chrome was the main goal. Throughout Assignment 1 there were plenty of lessons we've learned when it comes to the development of a conceptual architecture ranging from team decisions to Architectural design.

When we first started reading the outline for this assignment, we quite honestly thought it would be a pretty clear straight forward assignment, but we quickly learnt this wasn't the case. We first found that decisions don't come easy. We had our first taste of this when we were deriving different architecture styles that can perhaps fit the chrome architecture. Many members in our team brought up quality points relating to different styles whether it be a layered, OO or even pipe and filter style, so we had to thoroughly discuss all pros and cons before we could all decide on one. We found that this was the case with a lot of the aspects in assignment 1, there are always a lot of things to take under consideration when trying to decide on things, whether it be designs, diagrams, approaches or even the simplest things such as who's doing what part in the project.

As we continued the assignment, we found that looking through the chromium documentation helped the team get a better understanding of the whole system. With all the info in the docs it was then easy for us to conceptualize what subsystems would be required in the chromium architecture. However, the issue we came across was it was quite difficult to determine the dependencies of the subsystems. We found that determining the correct dependencies of each subsystem is a crucial role in an architecture and that it is not as cut and fold as it may seem. Many subsystems rely on one another and we found that after brainstorming and discussing why each component relies on the other we were able to agree on conceptual dependencies.

Furthermore, as the team progressed with the assignment we found that since we are developing only a conceptual architecture, it is quite difficult to think of all the things that will be needed in a real functioning system. We found that since the browsers are becoming more and more capable everyday they would be able to support an ever-increasing variety of tasks and because of this we found that it should be expected for the architecture to be flexible and allow subtle additions to its design.

The team eventually concluded that by incorporating an easy to use style such as Object Oriented, would allow for easy implementation and maintenance of an architectural system by therefore minimizing the amount of issues that come up during the building process.

CONCLUSION

Chromium is quite a large system and took a while for us to full break it down and create the conceptual architecture. By looking at the different task the system preforms we were able to divide the system into subsystems and from there which subsystems depend on each other. Finally, we decided that me most fitting architecture style was object oriented for our conceptual architecture of Chrome. We divided the system up into six subsystems with each having dependencies on one another and architecture styles most fitting to their processes. The subsystems we divided the system up into are: User Interface, Browser, Networking, Rendering and Parsing, and User Feedback.

We presented our conceptual design with diagrams of the overall system architecture, diagrams of important subsystem, sequence diagrams for rendering a page and logging into a website, and a

detailed look into each of the subsystems. We acknowledge that the architecture isn't faultless and recognize possible issues the Chrome development team could encounter when trying to implement our design.

Overall, as a team we learned a lot about system architecture by trying to create this one for Google Chrome and look forward to furthering our knowledge by attempting the concrete architecture next assignment.

REFERENCES

[1] Paul, Ryan (September 2008). *Google unveils Chrome source code and Linux port*. Retrieved 09 October, 2018. from https://arstechnica.com/information-technology/2008/09/google-unveils-chrome-source-code-and-linux-port/

[2] Grosskurth, A. & Godfrey, M. W. (2005). *A Case Study in Architectural Analysis: The Evolution of the Modern Web Browser*. Retrieved 09 October, 2018 from http://cs.queensu.ca/~ahmed/home/teaching/CISC322/F18/files/emse-browserRefArch.pdf.

[3]The Chromium Project (2018). *Graphics and Skia*. Retrieved 09 October, 2018 from *https://www.chromium.org/developers/design-documents/graphics-and-skia*

[4]haraken (14 August, 2018). *How Blink Works*. Retrieved 09 October, 2018 from https://docs.google.com/document/d/1aitSOucL0VHZa9Z2vbRJSyAIsAz24kX8LFByQ5xQnUg/edit#heading=h.v5plba74lfde