Does Asynchronous Programming in JavaScript

affect Software Quality?

A Repository Mining Study on GitHub

**An empirical study of Software Quality attributes in Asynchronous Programming constructs in JavaScript**

**Mining Software Repositories on GitHub to study software quality attributes in asynchronous programming constructs in JavaScript**

How are different Asynchronous Programming Constructs in JavaScript related to Software Quality? A Repository Mining Study on GitHub

BSc Thesis

**Context & Motivation**

In 1995, JavaScript [1] was developed and introduced for small client-side tasks in the browser. JavaScript spread rapidly with the growth of the internet and is now the most widely used programming language according to a Stackoverflow survey [2]. Reasons for this include its versatility, flexibility, and ease of use. TypeScript [3], a superset of JavaScript, is also becoming more popular. The main difference is its static type system and its class-based object orientation [4]. However, there is a lack of sufficient empirical evidence to support the claim that TypeScript leads to better software quality than JavaScript. The difference between dynamically typed and statically typed languages has been studied occasionally, but results are not conclusive. While some studies claim that statically typed languages have a positive impact on code quality [5], others claim the opposite [6]. There is a lack of studies that directly compare JavaScript and TypeScript projects in terms of software quality on a larger scale.

However, there is still a lack of sufficient empirical evidence for the claim that using functional programming has significant impact on software quality in comparison to other paradigms like procedural or object-oriented programming [6], especially since many other language prejudices could not be supported by studies in the past [7].

The evolution of Web 2.0 technologies makes web applications prevalent in various platforms including mobile devices and smart TVs. While one of the driving technologies of web applications is JavaScript, the extremely dynamic features of JavaScript make it very difficult to define and detect errors in JavaScript applications. The problem becomes more important and complicated for JavaScript web applications which may lead to severe security vulnerabilities.

JavaScript is one of the most popular programming languages. However, understanding the dynamic behaviour of JavaScript apps is challenging in practice. There are many factors that hinder JavaScript comprehension, such as its dynamic, asynchronous, and eventdriven nature, the dynamic interplay between JavaScript and the Document Object Model, and the asynchronous communication between client and server.

* Understanding Behavioural Patterns in JavaScript

JavaScript has become the lingua franca of web development. It is voted as the most popular language [26] and is the most used language on GitHub [13]. Comprehending JavaScript applications entails a set of challenges for developers. JavaScript is singlethreaded and thus callbacks are often exercised to simulate concurrency. Nested and asynchronous callbacks are used regularly to provide capabilities such as non-blocking I/O and concurrent request handling. This use of callbacks, however, can gravely complicate program comprehension – a problem coined as callback hell.

* Understanding Behavioural Patterns in JavaScript

JavaScript [1] has become the lingua franca of web development over time, since it was developed in 1995 and introduced for small client-side tasks in the browser. It is voted as the most popular language [26] and is the most used language on GitHub [13].

Comprehending JavaScript applications entails a set of challenges for developers. JavaScript is singlethreaded and thus callbacks are often exercised to simulate concurrency. Nested and asynchronous callbacks are used regularly to provide capabilities such as non-blocking I/O and concurrent request handling. This use of callbacks, however, can gravely complicate program comprehension – a problem coined as callback hell.

Understanding JavaScript applications involves a wide range of challenges for developers. JavaScript is single-threaded, and so callbacks are usually made to simulate concurrency. Nested and asynchronous callbacks are used frequently to provide features such as non-blocking I/O and concurrent request handling. However, using this callback can seriously complicate program understanding - a problem known as callback hell.

Understanding JavaScript applications involves a wide range of challenges for developers. JavaScript is single-threaded, and so callbacks are usually made to simulate concurrency.

Understanding JavaScript apps poses a number of difficulties for programmers. Because JavaScript is single-threaded, callbacks are frequently used to simulate concurrency. Callbacks, both nested and asynchronous, are frequently used to offer non-blocking I/O and concurrent request handling. However, using callbacks can make it difficult to understand a program, a phenomenon known as callback hell.

There are many factors that hinder JavaScript comprehension, such as its dynamic, asynchronous, and eventdriven nature and the asynchronous communication between client and server.

Many aspects, such as JavaScript's dynamic, asynchronous, and event-driven nature, the dynamic interplay between JavaScript and the Document Object Model, and the asynchronous connection between client and server, make it difficult to comprehend.

However, understanding JavaScript applications involves a wide range of challenges for developers. Many aspects make it hinder to comprehend, such as its dynamic, asynchronous and event-driven nature, the dynamic interplay between JavaScript and the Document Object Model, and the asynchronous communication between client and server. Because JavaScript is single-threaded, callbacks are frequently used to simulate concurrency.

Nested and asynchronous callbacks are used regularly to provide capabilities such as non-blocking I/O and concurrent request handling. This use of callbacks, however, can gravely complicate program comprehension – a problem coined as callback hell.

Nested and asynchronous callbacks are used frequently to provide features such as non-blocking I/O and concurrent request handling. However, using this callback can seriously complicate program understanding - a problem known as callback hell.

To provide capabilities such as non-blocking I/O and concurrent request handling, nested and asynchronous callbacks are used regularly. Using callbacks can seriously complicate program comprehension – a problem coined as callback hell.

However, understanding JavaScript applications involves a wide range of challenges for developers. There are potential factors of JavaScript, such as its dynamic, asynchronous and event-driven nature, the dynamic interplay between JavaScript and the Document Object Model, and the asynchronous communication between client and server, which hinder comprehension. Because JavaScript is single-threaded, callback, async/await and promise functions are frequently used to simulate concurrency. Nested, anonymous and asynchronous callback scheduling is used regularly to provide capabilities such as non-blocking I/O and concurrent request handling. Non-trivial callback-oriented programming tends to result in nested hierarchies of callback functions which make following the program flow hard and Fmain - a problem coined as “callback hell”. Long term maintenance of large applications may become close to impossible due to tight coupling of the callbacks and fragility of the structure. Error handling and coordinating asynchronous tasks can get messy quickly if programming discipline is not enforced and proper patterns not followed. Furthermore, a program consisting of callback hell has the risk of also introducing security vulnerabilities. However, there is still a lack of sufficient empirical evidence for the claim that using asynchronous programming constructs has significant impact on software quality.

These software quality metrics, such as code quality, maintainability and security, are almost always ignored.

There is another source of potential problems in asynchronous server side JavaScript programming. Non-trivial callback oriented programming tends to result in nested hierarchies of callback functions which make following the program flow hard. Long term maintenance of large applications may become close to impossible due to tight coupling of the callbacks and fragility of the structure. Error handling and coordinating asynchronous tasks can get messy quickly if programming discipline is not enforced and proper patterns not followed. The phenomenon is known as callback soup. A program consisting of callback soup has the risk of also introducing security vulnerabilities.

However, developers are frequently frustrated by “callback hell” — the comprehension and maintainability challenges associated with nested, anonymous callbacks and asynchronous callback scheduling

Before choosing among JFs, take into consideration their quality and maintainability metrics. These metrics are almost always ignored, but if you are about to invest in a technology you should not. A software that is hard to maintain, it is hard to evolve and the possibility to be abandoned in the near future is not negligible.