**Abstract**

Since its introduction for small client-side tasks in the browser in 1995, JavaScript has become the lingua franca of web development.

It was voted the most popular language and was the most used language on GitHub until the last quarter of 2021. However, understanding JavaScript applications involves 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 may 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 makes following the program flow hard - a problem described as "callback hell".

Long term maintenance of large applications may be severely impacted due to tight coupling of callbacks and structural fragility.

Handling errors and coordinating asynchronous tasks can quickly get messy if programming discipline is not enforced and proper patterns are not followed.

Furthermore, a “callback hell” program comes with increased risk of introducing security vulnerabilities.

However, there is still a lack of empirical evidence how different asynchronous programming constructs in JavaScript impact software quality.