**Final Presentation**

1. Context & Motivation

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 the 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 an increased risk of introducing security vulnerabilities. However, there is still a lack of empirical evidence on how different asynchronous programming constructs in JavaScript impact software quality.

1. Objectives & Tasks (RQs)

The goal of this study is therefore to empirically analyze a large set of JavaScript projects that use asynchronous programming constructs. The data collection should provide insights into the potential influence of different asynchronous programming constructs like callbacks, promises, etc. on software qualities, e.g., functional correctness, performance efficiency, or maintainability. The concrete quality aspects to be analyzed as well as more detailed research questions should be defined by the student.

1. Study Design

Software-Repositories wurden traditionell für Archivierungszwecke verwendet. Die MSR-Forschung hat gezeigt, dass diese Repositories ausgewertet werden könnten, um nützliche Muster und verwertbare Informationen über Softwaresysteme und -projekte aufzudecken. MSR-Forscher haben Techniken vorgeschlagen, die traditionelle Software-Engineering-Daten, -Techniken und -Werkzeuge ergänzen, um wichtige und herausfordernde Probleme zu lösen, wie z. B. die Identifizierung von Fehlern und die Wiederverwendung von Code, mit denen sich Praktiker täglich konfrontiert sind und sie lösen müssen. (The road ahead for MSR)

The research should be conducted as a mining software repository (MSR) study using a large number of open-source projects on GitHub. Data collection should be conducted with appropriate tools (e.g., using static analysis tools or the GitHub API) and automated as much as possible to achieve the best possible reproducibility. For the analysis, suitable techniques could be hypothesis testing, correlation, or regression. The detailed study design will be created by the student.

1. Schedule of Thesis