DWA_01.3 Knowledge Check_DWA1

---

1. Why is it important to manage complexity in Software?

As software addresses become larger there will be larger problems, interactions between entities become more complex, thus increasing the difficulty of creating new software solutions. Managing complexity in software is crucial for several reasons like maintenance and debugging, adaptability to changes, and reducing the rate of errors. It contributes to the long-term success of a software project by making it easier to understand, maintain, and adapt to changing requirements and technologies.

---

2. What are the factors that create complexity in Software?

Several factors contribute to the complexity of software, and it's often a combination of these elements that makes software development challenging. Some of the factors are: The size and scale of the codebase, Requirements ambiguity or changes, Poor architecture and design, and Security considerations.

---

3. What are ways in which complexity can be managed in JavaScript?

Managing complexity in JavaScript is crucial for developing maintainable, scalable, and error-free code. JavaScript applications can become complex due to the dynamic nature of the language, asynchronous operations, and various browser environments. Some ways to manage complexity are: Use ES6+ Features, Unit testing, Clear documentation, Design Patterns, and Error handling.

---

4. Are there implications of not managing complexity on a small scale?

Yes, even on a small scale, not managing complexity in software development can have several implications and negative consequences. While the impact may be less pronounced compared to larger projects, it can still affect the efficiency, maintainability, and overall success of a software project. Here are some implications of not managing complexity on a small scale: Reduced readability and understanding, Increased debugging time, Increased probability of errors, Resistance to change and Difficulty in collaboration.

---

5. List a couple of codified style guide rules, and explain them in detail.

Rule 1: Descriptive Variable Names - Use descriptive variable names that convey the purpose or meaning of the variable.

Explanation: Descriptive variable names enhance code readability and maintainability. When a variable is named in a way that clearly communicates its purpose, developers can quickly understand the role of that variable within the code without needing to delve into its implementation. This practice is part of the broader concept of self-documenting code.

Rule 2: Consistent Code Indentation - Use consistent indentation throughout the codebase. Typically, spaces or tabs are used to create uniform indentation levels within blocks of code.

Explanation: Consistent indentation improves code readability and helps developers understand the structure of the code. It visually represents the hierarchy of code blocks, making it easier to identify where functions, loops, and conditional statements begin and end. Consistent indentation is crucial for maintaining a clean and professional-looking codebase.

Rule 3: Camel Case for Naming Variables and Functions - Use camel case for naming variables and functions. In camel case, the first word is lowercase, and subsequent words are capitalized without spaces or punctuation.

Explanation: Camel case is a widely adopted convention in JavaScript for naming variables and functions. It improves readability and consistency in code. By adhering to this rule, developers can quickly identify whether an identifier refers to a variable, function, or another entity. Consistent naming conventions across the codebase contribute to a clean and professional appearance.

Rule 4: Use Strict Equality Checks (===) - Prefer strict equality checks (=== and !==) over loose equality checks (== and !=) to avoid type coercion.

Explanation: Strict equality checks compare both value and type, ensuring that the operands are of the same type without any implicit type conversion. This helps prevent unexpected behavior that can arise from type coercion in loose equality checks. Style guides often recommend using strict equality checks to enhance code reliability and avoid subtle bugs.

---

6. To date, what bug has taken you the longest to fix - why did it take so long?

That would be in the last project I did when I was working on the book connecting. The search function was the bug I was struggling with, it took me 9 hours to research and implement the different codes to search the books name, the genre and the author. I had to implement the codes to make it got through the list of books and pick the ones I was searching for only.
