DWA_01.3 Knowledge Check_DWA1

---

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

Managing complexity in software is crucial for several reasons.

- Maintainability

It improves maintainability by ensuring that the software remains understandable and modifiable as it grows.

- Reliability

It enhances reliability by reducing the likelihood of bugs and errors. Third, it promotes scalability, allowing the software to adapt and handle increased user demands.

- Efficient Collaboration

It enables efficient collaboration among developers, as well-structured and manageable code is easier to work with.

- User experience

Lastly, it enhances user experience by ensuring that the software remains intuitive and user-friendly. Overall, managing complexity leads to more robust, maintainable, scalable, and user-centric software systems.

---

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

Several factors contribute to the creation of complexity in software.

- The size of the software system, including the number of components, modules, and lines of code, can increase complexity.
- Interdependencies among various components and modules can create complexity when changes in one area affect others.
- Poorly designed or unclear requirements can lead to complexity during development.
- Integration with external systems or third-party libraries can introduce complexity.

Fifth, **the presence of legacy code**, **outdated technology**, or undocumented systems can make software complex to understand and modify. Lastly, evolving business or user requirements can introduce complexity if not properly managed.

---

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

Managing complexity in JavaScript is crucial for writing clean, maintainable, and efficient code. Here are a few ways to achieve that in simple terms:

Modularity - Breaking down your code into smaller, reusable modules can make it easier to understand and maintain. Each module focuses on a specific task, and they can interact with each other through well-defined interfaces.

Functions and Abstraction - Functions allow you to encapsulate a set of instructions and execute them when needed. By organizing your code into functions, you can abstract away complex logic, making it easier to read and reason about. This way, you can reuse the same code in multiple places without duplicating it.

Proper Naming and Comments - Using meaningful and descriptive names for variables, functions, and classes helps to improve code readability. Additionally, adding comments to explain the purpose and behavior of specific sections of code can assist other developers (and your future self) in understanding it.

Error Handling - Handling errors gracefully is important to prevent your code from breaking and ensure smooth execution. Implementing error handling mechanisms, such as try-catch blocks, enables you to anticipate and handle potential errors, making your code more robust.

Code Organization - Structuring your code in a logical and consistent manner is essential for managing complexity. Group related functions and variables together, separate concerns, and organize code files into directories. This approach makes it easier to locate specific pieces of code and understand the overall structure of your project.

Documentation and Testing - Documenting your code, such as providing API references and usage examples, can greatly assist other developers in understanding and utilizing

your code. Additionally, writing automated tests to verify the correctness of your code helps catch bugs early on and provides confidence when making changes.

---

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

Yes, there are implications of not managing complexity on a small scale in JavaScript. If complexity is not properly managed, it can lead to difficulties in understanding and maintaining the code. It may become harder to find and fix bugs, make changes or add new features. The code might become tangled and confusing, making it difficult for other developers to collaborate. It can also result in slower performance and increased memory usage. Ultimately, not managing complexity can hinder productivity, increase the chances of errors, and make it harder to build reliable and efficient software.

---

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

- Use meaningful variable and function names.

The rule suggests using descriptive names that clearly convey the purpose and functionality of variables and functions. It helps improve code readability and makes it easier for other developers (including your future self) to understand the code. For example, instead of using single-letter variable names like "x" or "i", use names that describe the data they represent, such as "totalCount" or "index".

- Indent code consistently using spaces or tabs.

This rule focuses on consistent indentation of code to enhance readability and maintain a clean visual structure. Whether you choose spaces or tabs, it's important to stick to a consistent indentation style throughout the codebase. Indentation helps visually indicate code blocks and their hierarchy, making it easier to understand the flow and nesting of statements. Consistency in indentation also promotes code collaboration as different developers can work on the codebase without conflicting formatting styles.

- Avoid using magic numbers or hard-coded values.

This rule encourages avoiding the use of arbitrary numeric values directly in the code without providing a clear explanation of their purpose. Instead, assign such values to well-named constants or variables to enhance code maintainability and readability. Magic numbers can be confusing and make the code harder to understand and modify

in the future. By using constants or variables, the intent of the value becomes clear, and if the value needs to be changed later, it can be done at a single location.

- Always use braces for control flow statements, even for single-line blocks.

This rule advises using braces, such as curly brackets ({ and }), around control flow statements like if statements and loops, even if the block contains only a single line of code. Using braces consistently improves code clarity and reduces the chances of introducing bugs when additional statements are added to the block later. It helps prevent common mistakes caused by unintentional line indentation or confusion about whether the block should be extended.

---

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

It was on Interactive Web Applications 19 (IWA 19)
The bug that took me too long to figure out was the search button preview of the book, I did manage to fix where I could select the book from the dropdown list of ‘All authors’ and ‘All genres’ but the submit button was not working for the book to show. The reason it took me too long was because I could not find a way to make it work but had to learn it from other peers that managed to make it work.
