DWA_01.3 Knowledge Check_DWA1

---

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

Managing complexity in Software is primarily essential in avoiding inevitable bugs in one's code and reducing the risk of code crashing. One must always maintain better readability overall of one's code for a better understanding of the code and easier maintenance of the intended functionality of software regardless of how much software growth is accumulated or team collaboration occurs over time. Complexity management also alleviates the processes of debugging and testing the software, thus reducing the development time and improving the software quality.

---

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

Software growth in size, scale and functionality tends to breed complexity in the Software. Poor readability and understandability of the code can also hinder the process of identifying and resolving bugs in the system which will make for an extremely complex environment. Complex and ambiguous requirements will inevitably cause complexity in software, requirements that are unclear, unrealistic, or inconsistent are a very high risk. Poor design and architectural choices with inadequate modularization, excessive dependencies, and lack of abstraction in one's code can make the software harder to understand and maintain. Tight deadlines and/or limited resources can often pressurize developers to take shortcuts in their code and bypass ethical software practices.

---

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

Strive to improve readability, understandability, and simplistic functionality. Simplistic non-ambiguous project requirements, stray as much as possible from unclear contradictory requirements that are constantly changing to avoid complexity in software. Making better well-informed decisions regarding the design and architecture of the software. Continuously striving for improvement in code clarity, readability and maintainability by making good use of abstraction and refactoring, code comments and JSDoc, and adhering and keeping up with best coding practices. It helps to break down

complex code into smaller, simpler manageable parts and group code by relativity and common functionality.

---

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

Code maintainability becomes heavily affected by complexity even on a small scale where codebase can easily become convoluted and difficult to understand resulting in a higher risk of introducing new bugs and potentially a higher risk of system failure due to a slower development process and a higher technical debt, making the code non-reusable in a short space of time and difficult for team collaboration.

---

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

Indentation and formatting which entails the basic layout of the codebase, making the code less congested and more readable and understandable. Knowing when to use white space in the code for separating operands and operators, alignment of code, and line breaks. Using line continuation for lines of code that exceed the length limit to be broken down into multiple lines. Having consistent spacing at the beginning of each line in a code block visually represents hierarchy structure which also helps to visually separate code blocks and nested structures like loops, conditionals, and functions.

Naming conventions which entail making use of descriptive and meaningful names for variables/code elements and using a consistent camelCase in all naming conventions with the exception of constants and enumerations which are written in uppercase snake_case(separated by underscores - essentially looking like “MAX_LENGTH”), and always avoid ambiguity and redundancy in naming conventions(always make sure names are easy to understand, clear and not confusing, do not reuse the same name with slight differences to avoid confusion and misunderstandings, and potential errors).

Commenting and documentation: Use inline comments to briefly explain specific lines, sections or any complex logic that may be difficult to understand for the next person. Strive to be concise and relevant, and avoid stating the obvious or over-commenting unnecessarily. Use block comments to explain larger sections of code(like functions) by providing an overview of the purpose, inputs, outputs, and intended functionality of the code block. Comments should be grammatically correct and use proper punctuation and sentence structure for improved readability. Maintain a consistent style of commenting throughout codebase and always updates comments as the code evolves

over time. Documentation comments provide structured information about code elements, such as functions, classes, or modules and serve as a reference for other developers reading the code. These comments can further be processed by documentation generation tools like JSDoc. They should include a concise and meaningful description of code elements (highlighting their purpose, behavior and usage), input parameters of functions/methods(specifying their types, names, etc.), return value/output(type of return and possible values to be returned), exceptions and error conditions(cases where the code element will raise errors and how they should be dealt with should they arise), and a usage example demonstration(highlighting how to use the code element correctly).

---

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

To date, I had a bug in my 'BookConnect' codebase that restricted my code from functioning completely, where the theme toggle refused to apply the new settings, the search button dropdown did not function, and the book previews code snippet wasn't even running. It took me the longest to figure out that a single word that was repeatedly used throughout the codebase was indeed the reason behind this code crash due to the minor spelling being easily overlooked and I had to go through the entire codebase line by line trying to figure out where the bug could have been despite the rest of the code proving to be functional to the eyes. Minor spelling errors can be easily missed, and in JavaScript, using the wrong case in a querySelector can cause problems. For example, using a singular instead of a plural ('data-key-author' instead of 'data-key-authors') can be a big issue that is easy to overlook.
