DWA_01.3 Knowledge Check_DWA1

---

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

- When software breaks, it is said to be catastrophic. In this sense software is like an arrangement of dominos - a small bug can cause a cascading effect, leading to crashes, data loss, and even physical harm in critical systems. By managing complexity, you reduce the chances of these domino effects.
- Software crashes a lot. Imagine a complex maze of wires. Complexity makes it more likely for wires to cross and cause malfunctions. Similarly, intricate code is prone to errors that can lead to crashes and unexpected behavior. Simpler, well-organized code makes it easier to avoid these issues.
- To improve maintainability and teamwork. With many developers working on large codebases, complex software becomes like a conversation with too many people talking at once. It's hard to understand, modify, or add new features. Managing complexity makes the code clearer, allowing different teams to collaborate more effectively.
- Software requires a high level of precision, even the smallest of miscalculations, or errors can cause the entire system to fail. Complex code makes it harder to achieve and maintain that precision. By simplifying the code, you make it more accurate and reliable.

---

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

- Poor code structure can add unnecessary complexity to software.
- Software is developed by a large number of different people and different teams, all of whom have different roles, skill levels and areas/industries of expertise.
- There is also inherent complexity that arises from the problem that the software is trying to solve.
- The idea that software needs to be built in such a manner that it works correctly under all possible scenarios, also adds complexity.
- Always changing and evolving requirements also make software complex because we need to build software that is flexible to changing requirements.

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

- **Effective Use of Coding Styles:** Adopting a consistent coding style is crucial for managing complexity in JavaScript. This includes consistent indentation, use of semicolons, and placement of braces. It helps in maintaining readability and understanding the flow of the code.
- **Avoid Vague and Ambiguous Variable Names:** Variables should be named in a way that indicates their purpose or the kind of value it holds. For example, instead of `x`, use `totalScore`. This makes the code self-explanatory and reduces the need for excessive comments.
- **Code Readability:** The code should be as readable as English. This can be achieved by using meaningful function and variable names, keeping functions small and focused on a single task, and avoiding deep nesting of functions or loops.
- **Good Documentation and Commenting:** Comments should be used to explain the 'why' (rationale or purpose) rather than the 'what' (which should be clear from the code itself). Additionally, maintaining up-to-date documentation, including function descriptions, parameter types, and return values, is essential for understanding the larger picture and how different parts of the codebase interact.
- **Use of Abstraction:** Abstraction can be used to hide the complexity of the code. This can be achieved by encapsulating complex code into functions or classes, which can then be reused throughout the codebase. This not only makes the code more manageable but also promotes code reuse and modularity.
- **Strict Checking:** Use features like `let` and `const` instead of `var` for stricter variable scoping, preventing accidental variable reassignments.
- **Testing:** Write unit tests to verify individual parts of your code work as expected. This helps catch errors early and prevents regressions as the codebase grows.

---

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

In the broader scheme of things it may seem like there are no major implications of not managing complexity at a small scale, however even at a small scale there are still some things to be wary of.

- **Increased Difficulty in Understanding:** Unmanaged complexity can make the software difficult to understand. This can slow down the development process as developers spend more time trying to understand the code.
- **Higher Maintenance Costs:** Complex software is often harder to maintain. Bugs are harder to find and fix, and adding new features can be challenging without introducing new bugs.
- **Reduced Scalability:** If the software complexity is not managed properly, it may not scale well when the need arises to add more features or handle more users.
- **Poor Performance:** Unnecessary complexity can lead to poor performance. This could be in terms of speed, memory usage, or other resources.
- **Increased Risk:** With increased complexity, there's a higher risk of failure. It's harder to predict how complex systems will behave, especially under unusual conditions or in the event of a failure.

---

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

1. **Use object destructuring when accessing and using multiple properties of an object.**
    - Destructuring saves you from creating temporary references for those properties, and from repetitive access of the object. Repeating object access creates more repetitive code, requires more reading, and creates more opportunities for mistakes. Destructuring objects also provides a single site of definition of the object structure that is used in the block, rather than requiring reading the entire block to determine what is used.
2. **Don't use iterators. Prefer JavaScript's higher-order functions instead of loops like `for-in` or `for-of`.**
    - Higher-order functions like `map()`, `filter()`, and `reduce()` are more expressive and easier to understand at a glance. They clearly convey the operation being performed on the array.

    - These functions return a new array and do not mutate the original array. This is important in functional programming and can prevent bugs related to data mutation.
    - They abstract away the process of iteration, allowing you to focus on the operation being performed on each element.

---

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

In the previous module, we worked on recursion tasks from FreeCodeCamp exercises, during which I encountered a bug. This bug proved challenging to resolve due to the inherent complexity of recursion and the unpredictability of its outcomes. I found it difficult to monitor the fluctuation of values with each function call, to understand the passed values, and to comprehend the impact of these elements when the function reverts to its initial instance.
