DWA_01.3 Knowledge Check_DWA1

---

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

Managing complexity in JavaScript is an important aspect of software development. According to Steve McConnell, author of the classic book “Code Complete,” managing complexity is the most important technical topic in software development, making it the software’s primary technical essential.

JavaScript is a powerful language that can be used to create complex applications. However, as the size and complexity of a JavaScript application grows, it becomes increasingly difficult to maintain and debug. Poorly managed complexity can lead to code that is difficult to understand, modify, and extend.

By managing complexity in JavaScript software, developers can create code that is easier to understand and maintain. This can lead to faster development times, fewer bugs, and more efficient use of resources.

---

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

There are several factors that can contribute to the complexity of software development, including the size of the codebase, the number of features, the number of developers working on the project, and the level of interdependence between different parts of the code.

In JavaScript, some common factors that can contribute to complexity include the use of global variables, callback functions, and closures.

1. **Global variables** can make it difficult to keep track of the state of an application, especially as the size of the codebase grows.
2. **Callback functions** can be difficult to reason about because they are executed asynchronously and can lead to complex control flow.
3. **Closures** can be powerful tools for creating private variables and methods, but they can also lead to code that is difficult to understand and debug.

---

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

To manage complexity in JavaScript software, developers can use techniques such as minimizing essential complexity by organising code, isolating code, dividing, and conquering problems into sub-problems, using abstractions, simplifying designs, and designing for reusability. Here are some ways to manage complexity in JavaScript:

1. **Modularization:** Breaking down a large codebase into smaller, more manageable modules can help reduce complexity and make code easier to maintain.
2. **Abstraction:** Abstraction is the process of hiding implementation details while showing only the necessary information to the user. This can be achieved using classes and functions.
3. **Code organisation:** Proper code organisation can help reduce complexity and make code easier to read and maintain.
4. **Documentation:** Proper documentation can help developers understand complex codebases and reduce the time required for debugging and maintenance.
5. **Testing:** Writing tests for your code can help identify issues early in the development process, reducing the time required for debugging and maintenance.
6. **Refactoring:** Refactoring is the process of improving existing code without changing its functionality. This can help reduce complexity and improve code quality.

---

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

Yes, there are implications of not managing complexity in software development, even on a small scale. Unnecessary complexity can increase the effort required for every feature delivery, reduce quality as the number of bugs increases, and reduce functionality that can be delivered in the same time frame.

In the worst case, it might become the major source of project risks and even start to influence task sequencing.

---

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

Codified style guide rules are a set of guidelines that help developers write clean, readable, and maintainable code. These rules are designed to ensure that code is consistent and easy to understand, which makes it easier to maintain and debug.

Here are some examples of qualified style guide rules for JavaScript.

1. ***Use modern JavaScript features:*** *You can use new features on every major browser — Chrome, Edge, Firefox, and Safari — supports them. For example, you can use:*
    a. ***let*** and ***const*** instead of ***var***,
    b. ***arrow functions*** instead of ***function keyword***,
    c. destructuring assignment.
2. **Array creation:** For creating arrays, use literals and not constructors, by creating arrays like this:

    ```
    const visitedCities = [ ];
    ```

    Instead of:

    ```
    const visitedCities = new Array(length);
    ```

    Item addition: When adding items to an array, use push() and not direct assignment such as the following array:

    ```
    const pets = [ ];
    ```

    Add items to the array like this:

    ```
    pets.push("dog");
    ```

    Don't add items to the array like this:

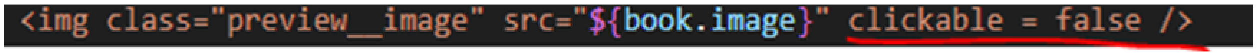
    ```
    pets[pets.length] = "dog";
    ```

3. **Comments:** Comments are critical to writing good code examples. They clarify the intent of the code while helping current or upcoming developers across different teams to understand the code and its intended behavior.

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

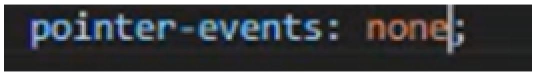
The biggest challenge amongst others, was in the “Book Connect” website for IWA18, where all the elements (title, subtitle, and the image) of a returned card were clickable instead of the whole card object being the clickable item.

I added an attribute (clickable = false) to all elements (title, subtitle, and the image) of each returned card in the view.js which was created for anything related to the DOM being all the content to be viewed by users on the website page.



```
<img class="preview__image" src="${book.image}" clickable = false />
```

I also had to edit the stylesheet by setting the pointer-events of the title, subtitle, and the image elements to none which disabled the mouse for the title, subtitle, and the image elements.



```
pointer-events: none;
```

All this was done because the code was leading to unpredictable behavior which was not the intended behavior.
