DWA_01.3 Knowledge Check_DWA1

---

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

As noted by Marijn Haverbeke, in *Eloquent Javascript*, keeping programmes under control is the main problem of programming. Getting code to work and be quicker is not the primary concern of a developer, but rather structuring languages such as JavaScript, as it is inevitable that a programme will grow out of control. “The art of programming,” Haverbeke writes, “is the skill of controlling complexity.” In the most extreme contexts, if structure is not well-considered, it can have catastrophic consequences, as evidenced by historical examples such as the Arianne flight V88, one of the most infamous and expensive software bugs. Ultimately code that works with bugs is more dangerous than code that doesn’t work at all. Yet in smaller, daily contexts, structuring remains equally crucial in order to ensure that a project is economical in terms of time and financial costs; to avoid lack of readability so that multiple people can work on a project with ease; to decrease the bugs and errors that come with technical debt; and to ensure the future longevity of a project and its code.

---

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

Evolving requirements and technology – projects evolve, clients needs shift, technical advancements occur frequently. All of these things mean that the code on a project is changing regularly and therefore increasing in complexity.
The build-up of technical debt – sometimes things need to happen quickly due to project/client needs, this can create a situation where code is not well-structured and thought through, but is just made to work in order to meet requirements on time.
Scaling – code needs to be fluid and adapt to client/company growth. As the company grows, so does the code, the team, and each project. As mentioned by Haverbeke, a programme will inevitably grow out of control, even if only one person is working on it.

---

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

- Provide documentation using JSDoc to provide additional context

- Incorporating a coding style, ensure readability in naming of variables, functions and object literals, using local variables to avoid reassignment
- Add checks to prevent code from running in circumstances that would be detrimental.
- Implement abstraction, only exposing the essential features to the reader wherever possible using elements such as modules and destructuring. Abstraction essentially hides complexity in order to make the code more comprehensible.

---

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

Primarily, time management is the biggest implication of not managing complexity, and wasted time can also translate into wasted finances. A great deal of time can be wasted attempting to understand something that hasn't been properly documented, lack of readability, for example variables that have been badly or obscurely named, or code that has become overly complex and unreadable not utilizing approaches such as abstraction.

---

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

Use only **const** or **let** instead of **var** for all variables, ensuring that you don't create global variables, as this results in less bugs and errors.

When it comes to naming, avoid single letter names or obscure acronyms, and be descriptive with your naming, rather than brief. This allows the reader to gain a better understanding of your code without unnecessarily obvious commenting. If using a singular and a plural, rather utilize longer names for clarity and to avoid confusion using the wrong element in the wrong place such use **singleItem** and **multipleItems** rather than **item** and **items**. With functions use verbs to better explain its functionality such as “get” and “create”.

Variables, classes and functions should be assigned before they are used. If not, this can harm readability and become confusing as the reader won’t know what is being referenced.

Use // for single line comments and use JSDoc /** */ for multiline comments and start all comments with a space to make it easier to read. JSDoc allows for the use of tags that can provide more information, which will then appear later on in the code via tooltip, which is especially useful for projects that contain very lengthy code.

---

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

The project that took me the longest was IWA Final Capestone. The element of the project that took me the longest was resolving the bugs in the search functionality. One that stands out in my mind was the filter element of the search that allows the user to search for partial matches of a title. I recall this element because there was one small issue, the && operator was used instead of the || operator. Everything else was functioning, there were no errors, and this was a case of really focussing and understanding the logic of the code.
