DWA_01.3 Knowledge Check_DWA1

---

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

Managing complexity in software is crucial for readability, maintainability, and bug prevention.
Without clear organisation and documentation, code becomes difficult to understand, increasing the risk of errors and hindering collaboration.
Additionally, effective complexity management facilitates long-term code and project management, enabling easier issue resolution and adaptability to future requirements.
This has a tangible effect on the cost of the project as well as the long term sustainability.
Complexity management allows for software that is better suited for scalability.

---

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

- Programming is inherently complex,
- The requirements of a brief are constantly evolving and the understanding of users needs is form dynamically during the process, resulting in different expectations and potential communication problems.
- ‘Technical debt’ is accrued when bugs are fixed quickly resulting in an accumulation of sloppy code’
- Scaling introduces inherent complexity as the effects of efficiency, and scope of consequences scale up.

---

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

- Using code style guides to have uniform, standardised, clear, easy to read code. Consistent naming and formatting structure.
- Documentation and commenting to provide clarity and context.
- Modular design that keeps functional units distinct and reusable. Bugs are therefore also localised and problems affect only parts of the whole, allowing the program to be more resilient to errors and easier to debug.

- Abstraction allows you to build smaller programs that are composed into more complex structures. Along with proper documentation this can also hide the internal functionality from the next user. Allowing a coder to know what a function/program needs and what it does, but not needing to know how it works. This process is referred to as encapsulation.

---

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

Because code is highly precise, managing complexity is something that is done at every scale, especially including small scale. Tiny errors can snowball and cause larger issues so having a low tolerance for small errors is a good philosophy when creating code. Important components of small scale complexity handling is in creating tests that allow you to detect or prevent errors quickly.
This could include keeping information like units of measurement with the integer value, enclosed in an object instead of keeping values as freestanding variables. Then having the function require that information and report errors if data is mishandled (eg. conversion errors) this allows you to easily implement tests and gives the code context.

These tiny errors have have dramatic effects on development time and can even bankrupt companies if they are not caught early and result in the delivery of subpar final product, not meeting client expectations and increased cost of production

---

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

Use always use let and const when declaring variables
Use correct case for global and local variables
Use 2 spaces for indentation
Use single quotes for strings except to avoid escaping
No unused variables
Add a space after keywords
Add a space before a function declaration's parentheses
Always use === instead of == (where possible)
Infix operators must be spaced
Commas should have a space after them

Here is a table comparing the different style guide rules:

Rule(Rule Name)	Google	AirBnB	Standard
Semicolons (semi)	Required	Required	No
Trailing Commas (comma-dangle)	Required	Required	Not Allowed
Template Strings (prefer-template)	No Stance	Prefered	No Stance
Space Before Function Parentheses (space-before-function-paren)	No Space	No Space	Space Required
Import Extensions (import/extensions)	Allowed	Not Allowed	Allowed
Object Curly Spacing (object-curly-spacing)	No Space Allowed	Space Required	Space Required
Console Statements (no-console)	No Stance	None	No Stance
Arrow Functions Return Assignment (no-return-assign)	No Stance	No	No
React Prop Ordering (react/sort-prop-types)	N/A	No Stance	No Stance
React Prop Validation (react/prop-types)	N/A	Required	Not Required

...
...

---

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

A challenge in the IWA_05 section, the code ran but didn't give me the correct answer and I sat with it for two weeks, only to find that I forgot to add in 'toys' to a line of code needed to calculate everything that was ordered. It took me a long time to figure out what the problem was because every time I got my code to run, something new would come up and I would need to figure out what that was. It felt like nothing that I was learning was sticking.

IWA17 used a looping function to fill in the values of the calendar. The way this information was handled was deeply counterintuitive and breaking the problem up into

smaller functional units would have helped make this process easier.
Having the week 1, week2, … column to appear on the left was particularly challenging.
