# DWA_01.3 Knowledge Check_DWA1

---

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

Managing complexity in software development is of paramount importance for creating software that is robust, sustainable and adaptable.
Debugging becomes less time consuming and more efficient when a codebase is well structured.
Modular code makes it easy for developers to reuse the modules that are well-isolated in other parts of the application and even in other projects, this can save time and effort in future development.

---

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

There are a number of factors that create complexity in software, some of which are:

a) Inefficient algorithms and data structure.
b) Poor Documentation.
c) Lack of abstraction and encapsulation principles, which basically leads to exposing unnecessary code and increases complexity in a software.
d) Software that depends on API's, external libraries and frameworks (softwares that rely on these become more complex when changes or updates are required).
e) Softwares that needs to interact with external systems such as databases, web-services or third party API's (these can result in complexity related to data formats, protocols and error handling).

---

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

Developers can implement certain practices to effectively manage complexity in JavaScripts: some of which are:

a) Use proper documentation
b) Make use of functional programming (FP) principles such as immutability, pure functions and higher-order functions.
c) Practice modular programming, this approach makes it easy for developers to test individual pieces of code effectively.
d) Consistent coding styles and formatting.

e) Writing unit tests and test integration to ensure that the code works as expected.

---

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

Yes, even though complexity may not be popular in small scale projects as in large projects, not managing complexity on small scale projects can lead to a number of issues and challenges, such as:

a) Limit reusability of the code in future.
b) Spending more time debugging (locating and fixing your code).
c) Increase technical debt, which is basically the cost of fixing the issues and improving the codebase in future.

---

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

a) Use consistent indentation, this basically refers to choosing a specific number of spaces for indentation and sticking to it. This makes the code easier to read.
b) Use of descriptive variables and function names.
c) Consistent name convention. Choosing a specific naming convention such as camelCase, PascalCase or snake_case and using it consistently in variables, functions or other identifiers in your codes makes it more understandable.
d) Use “const” and “let” and avoid “var” when declaring variables. “Const” and “var” have a block-level scope which basically allows them to be safer and predictable than “var”. Use “const” for variables that will be reassigned after initialization and “let” for variables that may change their value at a later stage.
e) Use template literals for string concatenation. Template literals (backticks) offer a better syntax compared to the traditional string concatenation “+”, and they also allow us to add expressions in strings.

---

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

While I was working on IWA18, everything was working smoothly until I got to the part where I had to click on a certain order and edit it. The algorithm for achieving the desired functionality was working, but every time a different order was clicked, it still displayed the same incorrect data. This took me about 3 to 4 days to fix even though,

only to find out that the code was only “targeting” the same order ID, instead of “targeting”  the closest clicked order.

---