DWA_01.3 Knowledge Check_DWA1

---

1. Why is it important to manage complexity in Software?
    1) Complexity increases the effort required for a feature or the work that is required to be delivered.
    2) It reduces the quality of your work as the amount of bugs is also increased.
    3) It also reduces functionality we can deliver in the same time frame.

---

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

In order to gain the correct understanding of how to deal with complexity in software, it's important to understand three basic types of software complexity: 1) Essential (unavoidable), 2) accidental and 3) incidental complexity.

1) This type of software complexity is related to the complexity of the domain we are trying to model. In more specific terms, we are dealing with business policy rules that govern business processes. When trying to implement and automate the processing of those policy rules, we encounter various levels of complexity. Essential complexity is unavoidable, and is basically the real reason why we are gainfully employed as software engineers
2) Accidental complexity manifests itself in poor architecture, poor design, poor code as well as poor software engineering processes. Because accidental complexity is caused by some or all of the above factors, the only way to get rid of it is by removing its causes. The most efficient way to remove the causes of accidental complexity is by education, training, mentoring and coaching. There is always room for improvement, and the more we focus our daily activities on learning, the quicker we'll be able to minimize accidental complexity.
3) This type of complexity is the toughest one to deal with. And it is the most counter-productive type of complexity.

---

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

The ways in which complexity can be managed in JS is by looking at code style and style guides to make the code more readable (Ensuring that style guide is kept to what you have agreed on). Secondly, add documentation and ensure that code is commented correctly describing what the code does, the types and what the code returns. Thirdly, to build the code to be much more modular so that code is easy to reuse, keep related code next to one another and things such as functional programming, and object oriented programming. Lastly, abstraction and this deals with what keeps the code manageable and the interface. (Making sure that you are not redeclaring code or functions and making the

---

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

The main implication of not managing complexity is the fact that one will not be able to not let the complexity grow out of control (i.e bugs) and that is the main problem of programming. So the code being buggy, being able to understand what you have written, the ability to have a good final product for the business and user.

---

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

Create Descriptive Names

Use long descriptive names, like complementSpanLength, to help yourself, now and in the future, as well as your colleagues to understand what the code does. The only exception to this rule concerns the few key variables used within a method's body, such as a loop index, a parameter, an intermediate result, or a return value.

Comment and Document

Start every routine you write (function or method) with a comment outlining what the routine does, its parameters, and what it returns, as well as possible errors and exceptions. Summarize in a comment the role of each file and class, the contents of each class field, and the major steps of complex code. Write the comments as you develop the code.

In addition, ensure that your code as a whole (for example, an application or library) comes with at least a guide explaining what it does; indicating its dependencies; and providing instructions on building, testing, installation, and use. This document should be short and sweet; a single README file is often enough.

Check for Errors and Respond to Them

Routines can return with an error indication, or they can raise an exception. Deal with it. Don't assume that a disk will never fill up, your configuration file will always be there, your application will run with the required permissions, memory-allocation requests will always succeed, or that a connection will never time out. Yes, good error-handling is hard to write, and it makes the code longer and less readable. But ignoring errors and exceptions simply sweeps the problem under the carpet, where an unsuspecting end user will inevitably find it one day.

Split Your Code into Short, Focused Units

Every method, function, or logical code block should fit on a reasonably-sized screen window (25–50 lines long). If it's longer, split it into shorter pieces. An exception can be made for simple repetitive code sequences. However, in such cases, consider whether you could drive that code through a data table. Even within a routine, divide long code sequences into blocks whose function you can describe with a comment at the beginning of each block.

Furthermore, each class, module, file, or process should concern one single thing. If a code unit undertakes diverse responsibilities, split it accordingly.

---

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

To date, the longest a bug has taken for me to fix was when I was working on the IWA Capstone project because it took a while to understand and formulate the moving parts of the code in a way that was useful in making the application and its different parts functional. Once I had figured out how to logically construct the theme's settings, for example, it was a light bulb moment in coding and troubleshooting the other issues that were required for me to fix. It also taught me the importance of working through code/ problem solving methodically and also in a structured meaningful way

---