DWA_01.3 Knowledge Check_DWA1

---

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

It is important to manage complexity in Software because it allows software developers to build software systems that are strong, adaptable and sustainable. This can be achieved when software is improved to be easily maintained, debugged, made easy to collaborate on, scalable, use code components that can be reused and has great performance and efficiency.

Maintainability: Software often evolves overtime and its complexity increases, making it hard to to understand, refactor and fix. A well-managed software can keep the codebase clean, modular, and well organised. This makes software easier to maintain and therefore enhances the software.
Example: Software developers can ensure a codebase that is well-organised, has consistent code style and is well documented to improve maintainability.

Debugging: Errors are prone to occur in complex software and it can be challenging t o identify and fix the root cause of these errors. A well-managed code with a clear structure and consistent code style for readability makes troubleshooting and debugging easier, saving software developers time and effort.
Example: If a codebase is well-managed, with clear separation of concerns and modular components, developers can quickly narrow down the problematic area and debug the issue more efficiently, saving time and effort.

Collaboration: Software is often developed in team effort. Developers, designers, project managers, etc. may be involved in building and maintaining software. A well-managed complexity in software can enhance collaboration. When the codebase is understandable it reduces the chances of miscommunication and facilitates better teamwork.
Example: If a team of developers are working on different modules of a software and the codebase is properly managed, with well-defined interfaces and clear documentation, developers can collaborate effectively. They can understand each other's code, integrate their work, and avoid conflicts, resulting in good collaborative effort.

Scalability: Complex software often needs to be scaled up to accommodate increasing demands and growing user capacity. A poorly managed software codebase can hinder scalability with performance issues. Software Developers can design and implement scalable code solutions that can handle growing demands effectively by managing complexity.
Example: A web application can experience a sudden surge in traffic. If the codebase is poorly managed and lacks scalability considerations, the system may struggle to handle the increased load, leading to slow response times or even crashes. By managing complexity and employing scalable code components, developers can ensure the system can handle higher loads efficiently and improve scalability,

Reusability: Codebases that are well-managed are more modular, making it easier to reuse components of code and or extend the existing functionality. It also reduces the development time for new software features and the quality of software.
Example: If the codebase is well-managed and follows modular design principles, developers can identify reusable components and extend existing functionality rather than starting from scratch. This promotes code reusability and reduces development time for new features.

Reliability: Complex software systems are more prone to bugs, errors and unexpected behaviours and compromises the reliability of the software. A well-managed software complexity can improve the robustness of and reliability of the software and enhance user satisfaction.
Example: Imagine a complex software system where a critical bug causes data corruption. If the codebase is properly managed for complexity and follows best practices, such as defensive coding and comprehensive testing (as seen in Part 2 pre recorded lecture), developers can identify and fix the bug, ensuring the system is more robust and reliable, minimising the occurrence of similar issues in the future.

Performance: A poorly managed software complexity can negatively impact the performance of software applications. It can lead to inefficient algorithms, excessive resource usage and slow execution time. On the contrary, managing complexity can optimise the software for better performance, ensuring expected response time and resource usage.
Example: Consider a resource-intensive application that performs complex calculations such as a Mars Orbiter system. If the codebase is not well-managed and contains inefficient algorithms or redundant operations, the application may suffer from poor performance. By managing complexity and optimising the codebase, developers can

enhance performance, ensuring the application meets the expected response times and resource utilisation. Resulting in no financial losses.

---

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

Factors that can create complexity in software include:

1. Changing requirements.
2. Poor or overly complicated software.
3. Interdependencies of software that relies heavily on external libraries, frameworks and other components.
4. When abstraction layers are not well designed and encapsulation boundaries are unclear, it becomes harder to reason about the behaviour and interactions of different components.
5. Large scale software naturally tends to have more complexity due to size of codebase, modules and data volume involved.
6. The choice of technology stack (programming languages, frameworks and tools) can impact software complexity.
7. Time constraint and tight deadlines may create pressure to take shortcuts, neglect proper documentation or skip through testing.
8. Legacy software codebase can cause complexity due to outdated technology, poor documentation or lack of understanding the original implementation of original code.
9. External factors such as changing business requirements, evolving technologies, security concerns and compliance regulations.

---

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

Complexity can be managed using the following strategies:

Modularisation: breaking down code into smaller, self contained modules promotes better organisation and encapsulation. Abstraction and Encapsulation of software using JavaScript's Object-Oriented Programming features to promote code reusability, maintainability and modularity and encapsulates related data and behaviour.

Design patterns: Be familiar with common design patterns in JavaScript, such as the Module, Singleton, Factory or Observer patterns that provide proven solutions to common problems and can help structure complex code in a manageable way.

Functional Programming: Embracing functional programming concepts like higher-order functions, pure functions, and immutable data can lead to more modular, testable, and maintainable code.

Code style and organisation: Group related functions and variables together, follow naming conventions, and maintain a consistent file structure. This makes it easier for developers to locate and understand different parts of the codebase.

Documentation and comments: Use comments to explain complex logic, document function parameters and return values, and provide overall code structure documentation. Clear documentation can greatly reduce confusion and make the codebase more approachable.

Reusability: Testing: Implement unit tests to verify the correctness of your code and ensure that changes do not introduce unintended side effects. Automated testing helps catch potential issues early, promotes code stability, and provides confidence when refactoring or extending functionality.

Testing: Implement unit tests in your code to verify correctness of code and that changes do not introduce unintended problems. These tests can be automated to catch potential issues early, enable code stability and provide confidence when refactoring or extending functionality.

Refactority: Regular code reviews and refactoring can help improve design, eliminate redundancy and simplify logic, optimise performance, enhance readability and reduce overall complexity of the codebase.

Using Libraries and Frameworks: Leverage established JavaScript libraries and frameworks that provide abstractions and tools to manage complexity. These Frameworks and Libraries offer structured approaches to build and manage complex applications.

---

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

Even on a small scale, software complexity can lead to various challenges and problems that can hinder development, maintenance, and overall software quality.The following are some potential implications:

Problems with readability and maintainability can lead to increased development time, higher risk of introducing bugs and difficulty in implementing changes and adding new features.

Complex code can introduce bugs and makes it challenging to identify the root cause of issues and to debug them effectively, resulting in longer debugging cycles and increased frustration.

Complex code tends to be tightly coupled and less modular which reduces chances of code to be extracted and reused. This means code cannot be leveraged for efficiency when building new features of software.

Scalability issues arise when software grows and its complexity increases due to lack of consideration for complexity management. Performance issues, decreased responsiveness and limitations in handling large workloads.

The implication of overtly complex code is that it becomes hard to understand and collaborate on the codebase impacting knowledge transfer within the team leading to miscommunications and misunderstandings among team members.

Another implication of not managing complexity is the accumulation of technical debt where additional work and cost are incurred due to taking shortcuts or making suboptimal design decisions during development. This results in difficulty making changes and maintaining software quality.

---

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

Indentation and Formatting: Consistent indentation and formatting are important for readability. For example, using tabs or spaces for indentation, choosing a specific number of spaces for each level of indentation, and defining rules for line breaks, braces, and parentheses.

.

Naming Conventions: Establishing consistent naming conventions for variables, functions, classes, and other identifiers is crucial. This may include using camel case or snake case, prefixing or suffixing certain types of identifiers, and avoiding reserved keywords.

Commenting and Documentation: Guidelines for writing comments and documentation help improve code understanding. This includes documenting function signatures, explaining complex algorithms or logic, and adding comments to clarify intent or highlight potential issues.

Error Handling: Consistent error handling practices ensure code robustness and maintainability. This involves deciding when to throw exceptions, how to handle error conditions, and adhering to specific error handling patterns like try-catch blocks.

Function Length and Complexity: Establishing guidelines for function length and complexity can enhance code readability and maintainability. This may include specifying maximum line lengths, avoiding deeply nested structures, and defining limits on cyclomatic complexity or nesting depth.

Code Organization: Defining rules for structuring code files and directories improves code maintainability. This could include organising files by feature or module, establishing a consistent import order, and outlining guidelines for file and directory naming.

Version Control: Establishing guidelines for version control practices ensures consistency across developers. This includes conventions for commit messages, branching strategies, and rules for merging or rebasing code.

Testing and Quality Assurance: Guidelines for writing tests and ensuring code quality contribute to a robust software development process. This may involve specifying naming conventions for test cases, establishing code coverage requirements, and defining rules for linting and static analysis.

---

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

Working on IWA challenges where the challenges were mostly to debug and fix code, i often took longer than expected to fix because of the following reasons:

1. Limited resources: The time I am able to spend on learning and understanding the concepts to find solutions for the challenges is constrained.
2. Communication and collaboration: I get a chance to work on my content materials after the students have gone home and I often tackle my work alone. I sometimes make calls and ask for assistance. This has made fixing bugs in my code take longer.
