Collaborative Discussion Initial Post: Factors Which Influence Reusability

In evaluating the factors influencing the reusability of object-oriented software, as identified by Padhy et al. (2018), prioritization depends on the practical implications of each factor within the software development lifecycle (SDLC).

1. Architecture-Driven Approach (ADP): The architectural software framework dictates how components interact and integrate. An architecture that supports modularity and loose coupling can facilitate the reuse of components across different systems. Well-defined architecture can isolate and reuse components, where changes in one module have minimal impact on others, and different system components can be easily reused for new use cases, simplifying the development process (Padhy et al., 2018). On the other hand, a poorly defined tech architecture could hinder its reusability as the software scales.

2. Design Patterns (DP): Utilizing design patterns is critical in creating reusable software components. Creating a template for similar use cases will avoid designing new architecture for each new project. This provides a big value, as creating a new solution design can be very difficult and time-consuming.

3. Documentation for the Project (DIP): Documentation encapsulates all the knowledge about the system, including specifications, design choices, use cases, requirements, and development comments. Well-documented components are more accessible to reuse because they can be understood and integrated into new projects with significantly less effort and fewer errors. It is essential for the initial development team and future teams who may take over the project to ensure that the software's purpose and solution design are clearly understood (Padhy et al., 2018). Without comprehensive documentation, the potential for effective reuse is reduced, especially if the team working on the project changes.

4. Knowledge Requirement (KR): Knowledge about the software development process and the system itself is an asset. When acquired and shared, it can guide the reuse of software components by providing insights into the functionality and interdependencies of different components.

5. Requirement Analysis (RA): Standardizing a process for collecting and analyzing requirements will substantially speed up the specification and refinement of the development process.

6. Used in Data (UD): Reusing data schemas and structures across different projects can save significant time and effort, given that the data models are well-designed and generalized enough to support various applications.

7. Modules in the Program (MIP): Modular programming inherently supports reusability by decomposing software into discrete modules that can be independently used and tested. This ensures that various system components can be directly

reused, without complex modifications, for various use cases. This speeds up the development cycle for new projects and ensures that if a specific module fails, the whole system is not necessarily affected. Additionally, troubleshooting individual components can be easier than the whole system.

8. Algorithms Used in the Program (AP): Algorithms represent the logical steps or formulas for solving problems. Reusing algorithms can lead to consistency and reliability in system behaviour across different applications.

9. Service Contracts (SC): Service contracts define the interfaces between various parties on how to reuse the software. Clearly defined contracts allow for smooth handover for reusing the software for a new project.

10. Test Cases/Test Design (TCTD): Reusable test cases and designs allow for the validation of components within new systems, ensuring that reused components still meet the required quality standards in the new context. As testing and validation are essential parts of software development, creating reusable TCTD makes the development process more efficient.

11. Models in the Project (MP): Standards for models such as UML diagrams or data flow diagrams, which are abstract representations of the software, can be reused to maintain a consistent approach to solving similar problems in different projects. While this can make communication between involved parties and the design process smoother, as it standardizes the work process and solution representation, it's not as essential as the above assets.

While 6 - 8 assets can be important for reusability in many contexts, that is not necessarily true for all projects. 1 - 5 are foundational assets essential for reusability in most scenarios, while 9 - 11 are less critical in most cases.

Prioritizing these factors requires considering each factor's impact on the ease of reuse, the quality of the resulting software, and the overall development process. Documentation, architecture, and design patterns rank high due to their broad impact on software development's reusability. While all factors listed by Padhy et al. (2018) play vital roles in enhancing software components' reusability, prioritization helps focus efforts where they give the most significant impact.

References:

Padhy, N., Satapathy, S., & Singh, R.P. (2018) 'State-of-the-Art Object-Oriented Metrics and Its Reusability: A Decade Review', in: Satapathy S., Bhateja V., Das S. (eds) Smart Computing and Informatics. Smart Innovation, Systems and Technologies. 77. Springer.