Initial Post on Discussion Topic:
Factors which Influence Reusability

The concept of software reuse as a distinct area in software engineering was initially proposed in the late 1960s by Douglas McIlroy (1968), who recognized the necessity of reusable components in industrial software development. However, a form of code reuse was improvised as early as the beginnings of programming. Today, code reuse is widely practiced in the software industry, along with the repurposing of software design skeletons and even entire design processes (Padhy et al., 2017). The use of software reuse has been shown to reduce costs, enhance the speed and reliability of development, as reported by Frakes and Kang (2005) and Haefliger et al. (2008).

Several factors affect code reusability, with complexity having a negative impact and modularity having a positive impact. Modularity is considered almost a prerequisite for reusability. Other factors, such as understandability or readability (Buse & Weimer, 2010), can also enhance the quality of source code and potentially improve its reusability. Techniques such as using naming conventions, writing useful code comments, and ensuring code readability can enhance understandability and overall software quality.

In the world of object-oriented software development, prioritizing reusable properties is crucial for boosting efficiency, maintainability, and overall software quality across a variety of projects. Padhy et al. (2018) outline critical factors that influence reusability

and advocate for a systematic approach to prioritize them. Let's dive deeper into this prioritization, explaining the reasoning behind each factor's significance:

1. Architecture-driven approach (ADP): A well-designed architecture serves as the foundation of software systems. Its modularity, scalability, and maintainability underpin reusability by facilitating seamless integration and component reuse across projects. ADP fosters a framework that promotes modular design, enhancing adaptability and ease of maintenance.
2. Design patterns (DP): DP represents refined solutions to recurring design issues, eliminating the need for reinvention. Their incorporation cultivates flexible, extensible, and modular code, increasing reusability. Employing DP expedites development while ensuring consistency and dependability across implementations.
3. Modules in the program (MIP): Modularization breaks down software into cohesive units, fostering reusability by reducing redundancy and enhancing consistency. MIP facilitates streamlined development, comprehension, and reuse of components across projects, thereby fortifying the software's adaptability.
4. Algorithm used in the program (AP): The reuse of algorithms contributes to efficiency and uniformity across projects, playing a pivotal role in software development. By encapsulating algorithms, developers streamline efforts and guarantee consistent functionality, thus boosting reusability.
5. Requirement analysis (RA): A comprehensive analysis of requirements lays the groundwork for identifying reusable patterns and opportunities. RA creates flexible and adaptable software systems, prepared to accommodate future reuse scenarios, thus enhancing reusability.

6. Service contracts (SC) establish clear communication standards and channels, promoting seamless integration and interoperability. Their emphasis fosters modular, extensible architectures that facilitate component reuse, thereby enhancing reusability.
7. Used in the data project (UD): Reusing data expedites development while improving software quality and consistency. Utilizing existing datasets reduces redundancy, encourages data-driven decision-making, and nurtures collaboration, thereby bolstering reusability.
8. Documentation in project (DIP): Comprehensive documentation elucidates design decisions, implementation details, and usage instructions, simplifying comprehension, maintenance, and reuse of software components. Detailed documentation fosters knowledge transfer and minimizes ambiguity, thereby enhancing reusability.
9. Knowledge requirement (KR): The dissemination of knowledge gleaned from past projects cultivates a culture of learning and continuous improvement, which is vital for fostering reusability. Emphasizing knowledge requirements ensures that insights and best practices are leveraged to fortify future reusability endeavors.
10. Models in the project (MP): While models serve as blueprints, streamlining development efforts and ensuring alignment with project requirements, their impact on reusability may be limited compared to factors such as architecture and design patterns.
11. Test cases/test design (TCTD): Although test cases are crucial for validating software correctness and robustness, their impact on overall reusability may be

limited. Their reuse potential across projects may be constrained by project-specific requirements.

By prioritizing these factors, developers can channel their efforts towards areas with the utmost impact on software quality, efficiency, and maintainability across diverse projects. Adhering to this structured approach enables developers to navigate the complexities of software development while harnessing the full potential of reusability.

References:

Buse, R.P. & Weimer, W.R. (2010) 'Learning a metric for code readability', *IEEE Transactions on Software Engineering*, 36(4), pp. 546–558. doi:10.1109/tse.2009.70.

Frakes, W.B. & Kang K., (2005) 'Software reuse research: Status and future', *IEEE Transactions on Software Engineering*, 31(7), pp. 529–536. doi:10.1109/tse.2005.85.

Haefliger, S., von Krogh, G. & Spaeth, S. (2008) 'Code reuse in open source software', *Management Science*, 54(1), pp. 180–193. doi:10.1287/mnsc.1070.0748.

Mcllroy, D. (1968) 'Mass Produced Software Components'. *Proceedings of NATO Software Engineering Conference*, Garmisch, Germany, October 1968, 138-155.

Padhy, N., Satapathy, S. and Singh, R.P. (2017) 'Utility of an object oriented reusability metrics and estimation complexity', *Indian Journal of Science and Technology*, 10(3). doi:10.17485/ijst/2017/v10i3/107289.

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.