Collaborative Discussion 1 - Factors which Influence Reusability

The concept of software reuse, a significant area in software engineering since its proposition by McIlroy (1968), has been shown to reduce costs and enhance development speed (Frakes and Kang, 2005; Haefliger et al., 2008). In object-oriented development, strategically prioritizing factors that influence reusability is essential. The provided text, drawing from the work of Padhy et al. (2018), offers a comprehensive list of such factors. While all are contributory, a logical hierarchy can be established by considering which factors are foundational and enable others.

My proposed prioritisation

At the absolute top of the hierarchy are Requirement Analysis and an Architecture-Driven Approach. Before reusable code can be written, a team must understand what to build. A thorough analysis of requirements allows developers to identify commonalities and variabilities across potential use cases, which is the very foundation of designing for reuse. Following this, a well-designed architecture, which underpins modularity and maintainability, serves as the system's blueprint (Padhy et al., 2018). Without this solid architectural foundation, attempts at creating reusable components will be difficult and unlikely to succeed on a large scale.
The next logical tier includes Modules in the Program and Design Patterns. These factors are the tangible implementation of the architectural vision. Modularity, which involves breaking a system into independent and cohesive units, is considered a prerequisite for reusability. A system that is not modular cannot have its parts reused effectively. Design patterns represent refined, reusable solutions to common problems (Padhy et al., 2018). Employing these patterns allows developers to build flexible and modular code that adheres to the established architecture without reinventing solutions.

Once well-structured components exist, their effective use is enabled by Service Contracts and Documentation in Project. A service contract, or a well-defined interface, establishes clear communication standards that allow components to interact without being tightly coupled. Comprehensive documentation is equally crucial; a component's utility is severely limited if others cannot understand how to integrate it. Clear documentation enhances the understandability and overall quality of the source code, which in turn improves its reusability (Buse and Weimer, 2010).

The subsequent factors, including the specific Algorithm used in the program, organisational Knowledge Requirement, and reuse of data, models and test cases, while important, have a more limited or indirect impact on the inherent reusability of a software

component compared to the foundational design principles. They are often consequences of a well-designed, modular and documented component rather than the primary drivers of its reusability.

References

Buse, R.P. and 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. and 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. and Spaeth, S. (2008) ‘Code reuse in open source software’, Management Science, 54(1), pp. 180–193. doi:10.1287/mnsc.1070.0748.

McIlroy, D. (1968) ‘Mass Produced Software Components’, in Proceedings of NATO Software Engineering Conference, Garmisch, Germany, October 1968, pp. 138-155.

Padhy, N., Satapathy, S. and 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. vol 77. Springer.