Initial Post

Initial Post

by Ruben Marques - Tuesday, 5 August 2025, 3:46 PM

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.

Permalink Reply

Re: Initial Post

by Lauren Pechey - Thursday, 7 August 2025, 11:46 AM

Hello Ruben,

Thank you for your post! Your structured prioritization of reusability factors in object-oriented development clearly highlights the foundational role of requirement analysis and architecture-driven design. This aligns well with the emphasis by Koti et al. (2024), who stress the importance of a multifaceted approach starting from comprehensive data acquisition and processing frameworks, which parallels your focus on understanding requirements before coding reusable components.

Moreover, your inclusion of modularity and design patterns as core enablers echoes findings from Mehboob et al. (2021), who identify modular design and well-defined interfaces as critical for maximizing software reuse. I appreciate your emphasis on service contracts and documentation, which, as noted by Padhy et al. (2018), significantly improve code understandability—a key metric for reusability.

Your hierarchy thoughtfully distinguishes between primary and secondary factors, reflecting the consensus in software engineering literature that while algorithms and organizational knowledge are important, they depend heavily on solid architectural foundations and modular design. Overall, your approach presents a practical roadmap for teams aiming to boost reuse effectively and sustainably.

References:

Koti, A., Koti, S.L., Khare, A., & Khare, P. (2024) Multifaceted approaches for data acquisition, processing & communication. 1st ed. CRC Press. Available at: https://doi.org/10.1201/9781003470939 [Accessed 7 Aug. 2025].

Mehboob, B., Chong, C., Lee, S., & Lim, J. (2021) Reusability affecting factors and software metrics for reusability: A systematic literature review. Software: Practice and Experience, 51(6): 1259–1286. DOI: https://doi.org/10.1002/spe.2961

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.C., Bhateja, V. and Das, S. (eds.) Smart Computing and Informatics. Springer, Singapore, pp. 431–441. https://doi.org/10.1007/978-981-10-5544-7_42

Maximum rating: -

Permalink Show parent Reply

Re: Initial Post

by Stelios Sotiriadis - Tuesday, 12 August 2025, 10:48 AM

Hi Ruben,

I really enjoyed reading your post and analysis—great points, and I particularly liked the choice of references. Well done.

Stelios

Permalink Show parent Reply

Re: Initial Post

by Victor Angelier - Saturday, 23 August 2025, 12:14 PM

* Peer Response *

Hi Ruben,

Your prioritisation of reusability factors in Python OOP projects, emphasizing requirement analysis and architecture-driven approaches, provides a robust foundation. To address potential oversights, such as incomplete requirement analysis or inconsistent modularity, I propose two preventive measures grounded in object-oriented metrics and reusable assets.
First, to mitigate risks of incomplete requirement analysis, systematic domain analysis ensures accurate identification of commonalities and variabilities. Frakes and Kang (2005) note that early domain analysis prevents rework by aligning components with reusable patterns, strengthening your top-tier focus. Tools like use-case modeling can proactively address gaps, ensuring robust component design.
Second, while you prioritize modularity and design patterns, inconsistent application risks undermining reuse. Frameworks like Django, identified as reusable assets (Frakes and Terry, 1996), enforce modular architectures (e.g., Model-View-Controller), reducing ad-hoc designs. Regular code reviews using object-oriented metrics such as cohesion and coupling, as supported by Mehboob et al. (2021), minimize errors and validate modularity, aligning with design patterns like Factory (Gamma et al., 1994). These measures address your lower-tier factors (e.g., algorithms) by embedding validation, complementing documentation and service contracts (Padhy et al., 2018).
Your hierarchy therefore offers a strong framework, and these preventive measures strengthen its applicability in real-world development contexts.

References

Frakes, W.B. and Kang, K., 2005. 'Software reuse research: Status and future'. IEEE Transactions on Software Engineering, 31(7), pp.529–536. Available at: https://doi.org/10.1109/TSE.2005.85.
Frakes, W.B. and Terry, C., 1996. 'Software reuse: Metrics and models'. ACM Computing Surveys, 28(2), pp.415–435. Available at: https://doi.org/10.1145/234528.234531.
Gamma, E., Helm, R., Johnson, R. and Vlissides, J., 1994. Design Patterns: Elements of Reusable Object-Oriented Software. Reading, MA: Addison-Wesley.
Mehboob, B., Chong, C., Lee, S. and Lim, J., 2021. 'Reusability affecting factors and software metrics for reusability: A systematic literature review'. Software: Practice and Experience, 51(6), pp.1259–1286. Available at: https://doi.org/10.1002/spe.2961.
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.C., Bhateja, V. and Das, S. (eds.) Smart Computing and Informatics. Singapore: Springer, pp.431–441. Available at: https://doi.org/10.1007/978-981-10-5544-7_42.

Permalink **Show parent** **Reply**

◀ **Initial Post**

Initial Post ▶