Initial Post

Initial Post

by Victor Angelier - Thursday, 31 July 2025, 6:36 PM

Initial Post: Prioritising Factors Influencing Software Reusability

Padhy et al. (2018) identify twelve factors in Table 1 that influence the reusability of object-oriented software, emphasising assets beyond code. I propose a four-tier prioritisation, grouping these factors based on their impact on creating reusable, modular, and maintainable Python-based OOP systems. This framework aligns with design principles like encapsulation, inheritance, and low coupling (Chidamber & Kemerer, 1994).

Tier 1 – Foundational OO Design Properties (Highest Priority)

Design Patterns (DP): Reusable solutions (e.g., Factory, Decorator) enhance flexibility and cross-project adaptability (Padhy et al., 2018, p. 434).

Modules in the Program (MIP): Python's modular structure (e.g., imports) promotes encapsulation and reuse by reducing interdependencies.

Architecture Driven Approach (ADP): A robust architecture (e.g., MVC in Django) ensures component interoperability and scalability.

Tier 2 – Structural Enablers (High Priority)

Inheritance: Reusability often "occurs through inheritance" (Padhy et al., 2018, p. 433). However, deep inheritance (high DIT) can hinder maintainability and performance.

Requirement Analysis (RA): Early design generalisation and abstraction enables future reuse (p. 435).

Service Contracts (SC): Well-defined interfaces (e.g., FastAPI endpoints) enable interaction and reuse across systems.

Tier 3 – Quality and Usability Factors (Medium Priority)

Documentation in Project (DIP): Python docstrings and type hints improve usability and reduce errors (Boehm, 1988).

Test Cases/Test Design (TCTD): Reusable test suites (e.g., pytest) strengthen quality assurance and prevent regressions.

Algorithms Used in the Program (AP): Reusable algorithmic components can be encapsulated as classes or functions.

Tier 4 – Supporting Factors (Lower Priority)

Models in the Project (MP): UML and diagrams are helpful but less actionable than code-level reuse.

Knowledge Requirement (KR): Tacit knowledge supports reuse but lacks automation and consistency.

Planning Stage (PS): Indirectly facilitates reuse through better project structuring.

Used Data (UD): Context-specific; valuable in data-centric systems, less so in general OOP.

This prioritisation reflects Padhy et al.'s (2018) emphasis on "frequently researched reusable assets" (Figure 3, p. 435) and aligns with Python's strengths in dynamic typing, modularity, and design flexibility. While priorities may shift in specific contexts (e.g. real-time or legacy systems), this structure ensures a solid foundation for reusability in Python OOP projects.

References
Boehm, B.W. (1988) ‘A spiral model of software development and enhancement’, Computer, 21(5), pp. 61–72.
Chidamber, S.R. and Kemerer, C.F. (1994) ‘A metrics suite for object-oriented design’, IEEE Transactions on Software Engineering, 20(6), pp. 476–493.
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

Permalink **Reply**

Re: Initial Post

by Lauren Pechey - Thursday, 7 August 2025, 11:41 AM

Hi Victor,

Thank you for your well-structured and thoughtful post. Your four-tier prioritisation effectively highlights the critical elements that influence reusability in object-oriented software, especially within Python-based systems. Emphasising design patterns, modularity, and architecture as foundational aligns well with Koti et al.’s (2024) observations about Python’s strengths in modularity and encapsulation.

I also appreciate your focus on structural enablers such as inheritance and service contracts, which Padhy et al. (2018) identify as key to enabling reuse beyond code-level components. Your inclusion of documentation and test design in the medium priority tier resonates with Mehboob et al. (2021), who stress that clear documentation and robust testing frameworks are essential for practical reuse, improving maintainability and reducing errors.

One suggestion might be to consider elevating test design slightly, as automated test suites directly support reliable reuse by ensuring software correctness over time. Overall, your framework thoughtfully integrates both theoretical and practical perspectives, providing a useful roadmap for enhancing reusability in Python OOP projects.

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, 10:47 AM

thanks for your post and analysis. You’ve raised some excellent points, and your reference selection adds real value to the discussion.

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

Re: Initial Post

by Ruben Marques - Friday, 8 August 2025, 4:17 PM

Hey Victor ;)

Your list gives a practical way to look at what matters most for reusability in object-oriented software, particularly in Python. Putting design patterns, modularity and architecture at the top makes sense and is supported by Koti et al. (2024), who note that Python's modular structure, ability to hide complexity through encapsulation and ease of maintenance are all important for reuse.

Structural features such as inheritance, interface design and service contracts fits with Padhy et al. (2018), who explain that these are not only useful for code reuse but also for making different systems work together. Including documentation and test design in your medium-priority tier also matches Mehboob et al. (2021), who found that clear documentation and reliable testing are key for reducing errors and keeping software reusable in the long term, yet aren't the highest requirements for software reusability.

I also agree with your inclusion of requirement analysis as a higher-tier factor. As Frakes and Kang (2005) highlight, identifying common requirements early allows developers to design components with reuse in mind, avoiding the need for significant rework later. This early abstraction is often overlooked but can save considerable time and effort in larger projects.

Overall, your prioritisation offers a balanced view that combines technical design with process improvements, giving a solid approach to improving reusability in Python OOP projects.

References

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.

Koti, A., Koti, S.L., Khare, A. and Khare, P. (2024) *Multifaceted approaches for data acquisition, processing & communication*. 1st edn. CRC Press. doi:10.1201/9781003470939.

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. doi: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. doi:10.1007/978-981-10-5544-7_42.

Re: Initial Post

by Victor Angelier - Saturday, 23 August 2025, 12:08 PM

Peer Response

Hi Ruben,

Your analysis of my prioritisation of reusability factors in Python OOP projects is insightful, particularly your emphasis on design patterns and modularity. To address potential gaps, such as unvalidated designs or inconsistent implementation, I propose two preventive measures grounded in object-oriented metrics and reusable assets.
First, while Padhy et al. (2018) highlight inheritance and low coupling for modularity, Frakes and Terry (1996) stress that structural choices require validation. Implementing systematic object-oriented metrics—such as cohesion and coupling—during code reviews prevents rigid designs that hinder reuse. This aligns with design patterns such as Factory, which enhance modularity (Gamma et al., 1994), ensuring scalable components.
Second, frameworks like Django provide reusable assets, such as authentication modules, and enforce architectural consistency across projects (Frakes and Terry, 1996). This reduces inconsistent designs, complementing your modularity focus. Adopting frameworks early, combined with validation methods (Frakes and Terry, 1996), ensures components meet reusability standards. Mehboob et al. (2021) further note that such practices reduce errors, enhancing long-term maintainability. These measures extend your emphasis on requirement analysis (Frakes and Kang, 2005), embedding reusability throughout the development lifecycle.
In this way, your focus on modularity is strengthened by preventive measures at both the structural and framework levels, ensuring that design principles and reusable assets together support long-term reusability in Python OOP projects.
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 ▶