Initial Post

Initial Post

by Anda Ziemele - Sunday, 24 March 2024, 11:42 PM

In their review paper, Padhy et al. (2018) outline a set of reusable assets that have been identified in multiple papers across a time period of research in software reusability. Software reusability is defined as capability of a product to be used as assets in more than one system, or in building other assets (ISO25010, 2023). It is crucial to define what are (a) assets, (b) factors and (c) metrics when discussing software reusability. Assets in software reusability can simply be defined as “reusable products or by-products of the software development” (Pinto, 1998) and in the context of this discussion, factors here are factors which affect software reusability. Finally, metrics are used to quantitively measure the degree of reusability, yet it is a challenging area to model given reusability is a dimension of quality, which is context-dependent (Pfleeger & Kitchenham, 1996, as cited in Papamichail et al., 2019).

The following factors can be used to estimate reusability of the listed software assets, as summarised by Mehboob et al. (2021): (1) Coupling, (2) Cohesion, (3) Complexity (4) Inheritance and (5) Size (code base). Additionally, Papamichail et al. (2019) also consider (6) Documentation as a factor. Assets listed in Padhy et al. (2018) can be prioritised in the following way:

1. Architecture-Driven Approach (ADP): ADP is highest in value and priority, as whole architecture system designs can be used as templates and reused to build new software systems (Ning et al., 1994). Given architecture defines all structural components and their interconnections, it scores high in inheritance, cohesion and low in coupling. Complexity and size may be a concern, depending on the chosen architecture.
2. Design Patterns (DPs): Design patterns are a set of pre-defined solutions to common design problems in software development and are inherently built for reusability, yet some patterns are more reusable than others (Ampatzoglou et al., 2011).
3. Modules in the Program (MIP): MIPs go hand-in-hand with ADP, however additional research, design and work may be required to define the coupling between modules thus increasing complexity.
4. Test cases (TCTD): Test suites consist of tests for software modules and are an integral part of software development. Test suites may often consist of tests low in complexity and coupling, which cover base functionalities of data and software testing
5. Requirement Analysis (RA): Requirements may be reused, being high in cohesion as requirements are grouped into areas of similar functionality that the software must meet.
6. Knowledge Requirements (KR): Knowledge has high value for reuse especially in the context of design and requirements, however it may be argued that knowledge management and reuse is an area of itself independent from software reusability and hence needs to be managed and measured as such. Knowledge may also be clubbed together with documentation as an asset.
7. Algorithms (AP): Reusable algorithms have the risk of high complexity and coupling to specialist systems, however this area can be explored further.
8. Data (UD): Data can change and be unpredictable, hence why using as a base for reusability may over time deviate from real-life scenarios. Additionally, data would be better suited to measure reusability, rather than be re-used in software development - test cases are more appropriate, with certain cases tested for in the data.

Based on existing research (Papamichail et al., 2019), Documentation (DIP) has not been considered as an asset, but rather a factor in software reusability with its own set of metrics. Models (MP) as a reusable asset are not clearly defined in the paper, but based on existing description, may be considered as part of ADP. On Service Contracts (SC) - service contracts for reusability, also defined as reuse contracts - are “the interaction between designers and users of reusable assets” (Lucas et al., 1997), and are not reusable assets in themselves, rather an extension of software development.

References:

Ampatzoglou, A., Kritikos, A., Kakarontzas, G. & Stamelos, I. (2011) An empirical investigation on the reusability of design patterns and software packages. *Journal of Systems and Software*, *84*(12), 2265-2283.

ISO25010 (2023) ISO/IEC 25010:2023(en). *ISO Online Browsing Platform (OBP)*. Available at: **https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-2:v1:en** [Accessed 24 March 2024].

Lucas, C., Steyaert, P., Mens, K. (1997) Managing software evolution through reuse contracts. In *Proceedings. First Euromicro Conference on Software Maintenance and Reengineering,* 165-168. IEEE.

Mehboob, B., Chong, C.Y., Lee, S.P., Lim, J.M.Y., 2021. Reusability affecting factors and software metrics for reusability: A systematic literature review. *Software: Practice and Experience*, 51(6), 1416-1458.

Ning, J.Q., Miriyala, K., Kozaczynski, W., (1994) November. An architecture-driven, business-specific, and component-based approach to software engineering. In *Proceedings of 1994 3rd International Conference on Software Reuse,* 84-93, IEEE.

Padhy, N., Satapathy, S., Singh, R.P. (2018) State-of-the-art object-oriented metrics and its reusability: a decade review. In *Smart Computing and Informatics: Proceedings of the First International Conference on SCI 2016, Volume 1,* 431-441. Springer Singapore.

Papamichail, M.D., Diamantopoulos, T., Symeonidis, A.L. (2019) Measuring the reusability of software components using static analysis metrics and reuse rate information. *Journal of Systems and Software*, 158, 110423.

Pfleeger, S. & Kitchenham, B. (1996) Software quality: The elusive target. *IEEE Software*, 12-21.

Pinto, P. E. (1998) Promoting Software Reuse in a Corporate Setting. *CMU School of Computer Science*. Available at: **https://www.cs.cmu.edu/afs/cs/usr/ppinto/www/reuse.html** [Accessed 24 March 2024]

Permalink **Reply**

Re: Initial Post

by Jordel Davidson-Swann - Tuesday, 26 March 2024, 12:55 AM

Hi Anda,

Your post was very informative.
The only thing that I believe this post could have needed were some case studies.

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

Re: Initial Post

by Oi Lam Siu - Wednesday, 3 April 2024, 4:22 AM

Hi Anda,

Your detailed analysis and extensive reading on the topic of software reusability are highly commendable. Your explanation of the definitions of assets, factors, and metrics is crucial for a comprehensive understanding of software reusability.

Regarding your ranking of the Architecture-Driven Approach (ADP) as the first factor and my ranking of ADP as the second, with

Requirements Analysis (RA) as the first, I recognize the significant importance of both factors for reusability. RA involves gathering project requirements at the beginning, laying the foundation for the project, while ADP is used to design the overall structure, ensuring it meets the project's conditions and capabilities.

From my perspective, both ADP and RA are crucial in their respective ways. RA sets the direction and goals of the project by identifying requirements, while ADP provides the means to translate those requirements into an architectural design that enables reusability. The close ranking of these factors reflects their interdependency and the critical role they play during project initiation.

Your insights further reinforce the significance of considering multiple factors and their synergistic relationship when aiming for reusable and efficient software development.

Thank you for your valuable contribution to the field.

Maximum rating: -

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

◄ Initial Post **Summary Post ►**