Peer Response 2: Factors which Influence Reusability

In reply to John Heart Ojabo

Re: Initial Post

by Andrius Busilas - Thursday, 21 March 2024, 8:36 PM

Hi John,

Your review on reusing software effectively emphasizes its historical importance and current relevance in reducing development costs and time. By citing Frakes & Kang (2005) and Mäkitalo et al. (2020), the discussion relies on established literature to bolster its credibility and strengthen its arguments.

The identification of 11 factors that contribute to the reusability of object-oriented software, as listed by Padhy et al. (2018), provides a structured framework for comprehending the intricacies of software reuse. The ranking of these factors highlights a thoughtful examination of their respective impacts on reusability.

Of particular note is the emphasis on knowledge requirement (KR) as the most critical factor, which underscores the significance of leveraging insights and experiences from past projects to enhance efficiency and inform decision-making. This highlights an in-depth comprehension of the value of knowledge management in software development.

Additionally, the discussion appropriately addresses the roles of modularization (MIP), architecture-driven approach (ADP), and requirement analysis (RA) in fostering reusability, reflecting a thorough understanding of software engineering concepts.

However, the discussion could benefit from further elaboration on factors such as documentation in projects (DIP) and test cases/test design (TCTD). While these factors are mentioned, providing more detailed explanations of their impacts on

reusability and why they rank lower in priority would enhance the overall argument and offer a more holistic perspective on software reuse.

In summary, the discussion presents a well-structured and insightful analysis of the factors affecting software reusability, offering valuable insights for software developers and practitioners. With additional elaboration on certain factors, the discussion would provide an even more comprehensive understanding of the complexities of software reuse.

Initial Post

by John Heart Ojabo - Tuesday, 19 March 2024, 9:50 AM

Number of replies: 1

Software reuse is not new, it has always been there since the beginning of programming (Frakes & Kang, 2005). The idea is to use parts of software programs written at one time in the construction of other programs later. It is essential in reducing development costs in terms of money and time.

Today, software reuse takes place in many ways including the approach of developing new software systems by reusing components that were not meant to be used together (Mäkitalo et al, 2020).

When it comes to object-oriented programming, software is generally written in a way that makes it modular and even easier to reuse. 11 factors which contribute to the reusability of an object-oriented software have been identified (Padhy et al, 2018). These factors are hereby listed in order of priority by considering their significance on the overall effectiveness on reusability:

1. **Knowledge Requirement (KR):**
   Knowledge is a fundamental aspect of software development. Reusing knowledge generated during the SDLC can significantly enhance the efficiency of subsequent projects and ensures that valuable insights and experiences are leveraged effectively, leading to better decision-making and problem-solving.

2. **Reusable Properties Reusability Factor (UD):**
   The ability to reuse data from previous projects can expedite the development process and improve accuracy. Data reuse minimizes redundant efforts and facilitates informed decision-making based on past experiences.

3. **Modules in the Program (MIP):**
   Modularization is a key principle in software engineering, promoting reusability, maintainability, and scalability. This factor also contributes to better organization and understanding of the software architecture.

4. **Architecture-Driven Approach (ADP):**

A well-defined architecture lays the foundation for scalable, maintainable, and reusable software. This factor ensures that reusability considerations are integrated into the software design from the outset, leading to more efficient reuse of components.

5. **Requirement Analysis (RA):**
   Effective requirement analysis is crucial for understanding the needs and constraints of stakeholders and designing solutions that meet their expectations. This helps developers to identify opportunities for reusability early in the development process.

6. **Design Patterns (DP):**
   Design patterns provide proven solutions to common design problems, promoting reusability, maintainability, and flexibility.

7. **Models in the Project (MP):**
   Models serve as blueprints for software systems, capturing essential aspects of their structure, behavior, and interactions thus giving developers the clarity to facilitate the analysis and reuse of design artifacts.

8. **An Algorithm Used in the Program (AP):**
   Reusable algorithms can simplify the development process and improve the efficiency and performance of software systems. While important, the priority of this factor may vary depending on the specific requirements and nature of the software being developed.

9. **Documentation in Project (DIP):**
   Documentation is very important for facilitating understanding, maintenance, and reuse of software components, but, documentation may be considered less critical than other factors in terms of directly influencing reusability.

10. **Service Contracts (SC):**
    Service contracts facilitate effective communication between developers and users, thereby assisting in the reuse of software products.

11. **Test Cases/Test Design (TCTD):**
    Test cases and test design are necessary for ensuring the quality of software systems but ranks low in priority when it comes to directly influencing the reusability of a software.

REFERENCES:

W. B. Frakes and Kyo Kang, "Software reuse research: status and future," in IEEE Transactions on Software Engineering, vol. 31, no. 7, pp. 529-536, July 2005, doi: 10.1109/TSE.2005.85

Mäkitalo, N., Taivalsaari, A., Kiviluoto, A., Mikkonen, T. and Capilla, R., 2020. On opportunistic software reuse. Computing, 102, pp.2385-2408.

Padhy, Neelamadhab, Suresh Satapathy, and R. P. Singh. "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, pp. 431-441. Springer Singapore, 2018.

In reply to Thomas Gray

Re: Initial Post

by Andrius Busilas - Sunday, 24 March 2024, 9:31 PM

Hey Thomas,

Your discussion offers a comprehensive analysis of the advantages of reusability in software design, supported by insights from Padhy et al. (2018) regarding 11 key aspects of software design that can be reused across projects. The prioritization of these aspects is effectively justified by considering their foundational nature and potential impact on reusability.

The emphasis on the Architecture-Driven Approach (ADP) and Knowledge Requirement (KR) as the most crucial factors underscores their fundamental role in laying the groundwork for scalable, maintainable, and efficient software development. This demonstrates a deep understanding of the significance of architectural design and knowledge management in facilitating reusability.

Furthermore, the discussion appropriately highlights the importance of modularization (MIP), reusable properties (UD), and requirement analysis (RA) in promoting reusability, reflecting a comprehensive understanding of software engineering principles.

The rationale provided for the ranking of each aspect, considering factors such as scope, timing, and potential future variations, enhances the credibility of the discussion. For instance, the acknowledgement that Documentation in Project (DIP) and Test Cases/Test Design (TCTD) rank lower owing to their indirect impact on reusability adds nuance to the analysis.

However, the discussion could benefit from further elaboration on certain aspects, particularly regarding their practical implications and examples of their application in real-world software projects. Additionally, addressing the potential challenges or limitations associated with reusability in software design would enrich the discussion further.

Overall, your discussion presents a thorough examination of the reusability of software design, offering valuable insights for software developers and stakeholders. With additional elaboration on practical applications and potential challenges, the discussion will provide an even more comprehensive understanding of the complexities surrounding software reuse.

Initial Post

by Thomas Gray - Friday, 22 March 2024, 12:12 PM

Number of replies: 2

Reusability in software design provides for a number of benefits to both designers, stakeholders and users. For designers, it allows for consistency and time saving efficiencies, all of which are key to a successful software design enterprise. That same consistency and efficiency also provides benefits for users and stakeholders, who can be guaranteed of the best product and service due to this. Reflecting on the work of Padhy et al. (2018), who have developed a list of 11 aspects of software design that can be reused in different projects. In order of the most important to the least important, they are hereby listed below.

1. Architecture-Driven Approach (ADP):

A well-defined architecture lays the foundation for scalable, maintainable, and reusable software. This factor ensures that reusability considerations are integrated into the software design from the outset, leading to more efficient reuse of components.

2. Knowledge Requirement (KR):

Knowledge is a fundamental aspect of software development. Reusing knowledge generated during the SDLC can significantly enhance the efficiency of subsequent projects and ensures that valuable insights and experiences are leveraged effectively, leading to better decision-making and problem-solving.

3. Modules in the Program (MIP):

Modularization is a key principle in software engineering, promoting reusability, maintainability, and scalability. This factor also contributes to better organisation and understanding of the software architecture.

4. Reusable Properties Reusability Factor (UD):

The ability to reuse data from previous projects can expedite the development process and improve accuracy. Data reuse minimises redundant efforts and facilitates informed decision-making based on past experiences.

5. Models in the Project (MP):

Models serve as blueprints for software systems, capturing essential aspects of their structure, behaviour, and interactions thus giving developers the clarity to facilitate the analysis and reuse of design artefacts.

6. An Algorithm Used in the Program (AP):

Reusable algorithms can simplify the development process and improve the efficiency and performance of software systems. While important, the priority of this factor may vary depending on the specific requirements and nature of the software being developed.

7. Requirement Analysis (RA):

Effective requirement analysis is crucial for understanding the needs and constraints of stakeholders and designing solutions that meet their expectations. This helps developers to identify opportunities for reusability early in the development process.

8. Design Patterns (DP):

Design patterns provide proven solutions to common design problems, promoting reusability, maintainability, and flexibility.

9. Service Contracts (SC):

Service contracts facilitate effective communication between developers and users, thereby assisting in the reuse of software products.

10. Documentation in Project (DIP):

Documentation is very important for facilitating understanding, maintenance, and reuse of software components, but, documentation may be considered less critical than other factors in terms of directly influencing reusability.

11. Test Cases / Test Design (TCTD):

Test cases and test design are necessary for ensuring the quality of software systems but ranks low in priority when it comes to directly influencing the reusability of a software.

The rationale for the rankings above takes into consideration the scope of each part and it’s timing within the project, as well as the prospect of variations that could impact it in the future. For example, Architecture-Driven Approach is listed as the most important due to how foundational it is for the tasks that follow it. The same thinking applies to Knowledge Requirement, as without the applicable knowledge being reused and shared, tasks would become very labour intensive and inefficient. As the authors themselves have noted, Documentation In Project and Test Cases / Test Design rank lower due to their lack of direct impact on the reusability of software.

References

Padhy, Neelamadhab, Suresh Satapathy, and R. P. Singh. "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, pp. 431-441. Springer Singapore, 2018.