Discussion Topic: Which of the 11 assets identified by Padhy et al (2017), should be prioritised to maximise the reusability of software?

My Initial Post

"Out of the 11 reusability properties of object-oriented software given by Padhy et al., (2018); I agree that the following factors contribute to the reusability of software by a user:

- quality of data for reliability
- quality of algorithms for performance reliability
- quality of documentation for accessibility
- quality of ontological model for affirmation of users own world view
- requirements for accessibility
- quality of tests for performance reliability
- quality of service for reliability
- design patterns for performance reliability

I agree the following factors contribute to the reusability of code by other developers:

- accessible modules for content
- accessible architecture for adaptation
- factors that affect the reusability of software to users are also important to developers, as they are either providing software to users, or are a user themselves.

Furthermore, I agree the following factors contribute to the reusability of experience that's gained while developing software:

- procedural knowledge to solve repeating problems

For the task of prioritising factors to maximise the reusability of software, I've chosen to prioritise the reusability of software by software users over software developers, because there are more potential users to reuse software than developers, thus maximising reusability as I understand it.

Therefore my priority list would be something like this...

1. **Highest Priority** - service, requirements, model, documentation,
2. **High Priority** - data, algorithms, tests, design patterns
3. **Low Priority** - modules, architecture,
4. **Least Priority** - knowledge

The first priority roughly centres on user accessibility. The second priority roughly centres on software reliability. The third priority roughly centres on software extensionality, and the fourth priority roughly centres on developer mastery.

How is the reusability of software a useful metric?

References:

Padhy, N., Satapathy, S., & 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*. 77. Springer.”

My Initial Post / Peer A Response

“This was a really interesting outlook on software reusability, I find it interesting that you broke down the priorities into 4 important groups.

- user accessibility.
- software reliability
- software extensionality
- developer mastery

With regards to *software extensionality,* does this refer to how *easily* the architecture can be modified to for example add more functionality down the line? If so what kind of Impact do you think it would have on the reusability of the software? “

My Initial Post / Peer A Response / My Response

“Thanks for the question

Yeah, by software extensionality, I'm referring to how the architecture/modules/functions/components/etc can be modified/reused for other projects, to add more functionality, or to even change the functionality of the original project. I'm not sure software extensionality really impacts how reusable the software is to a user. But for developers looking to save time on other projects, using premade code can save a lot of time. So if a project has a lot of code that's useful in other projects, then in some sense that project is reusable.

In my current job as a game developer, we regularly recycle code that was used in our other games, into new games. That's possible because my company sticks to one genre of game. Most games we create introduce a variation on one or two mechanics of the genre, and if it's popular, we add the implementation to our underlying framework, so that we can easily reuse it in other games. The games we create are reusable to ourselves as developers because they provide a method of saving time developing new games, which ends up being valuable to the business.”

My Initial Post / Peer A Response / My Response / Peer A Response

“Thank you for the response, it is interesting to see how software reusability can be valuable in the real world.

Another question I had is, Does testing code play a big part in your development cycle? And how do you mitigate any potential issues/bugs that may arise from reused code? (A *potential* scenario, code was not tested *properly* and has now found its way into your underlying framework.)”

My Initial Post / Peer A Response / My Response / Peer A Response / My Response

“Testing does play a big role in the development cycle. As developers, we mainly make use of big-bang integration testing. We have a team that does black-box testing too. To mitigate risks, we use a statically typed language and code linters. We use git version control too, which is valuable if used correctly It helps prevent issues through peer review, and helps fix bugs in the worse case scenarios by giving the chance to revert code changes. Usually though, debugging tools and experience is good enough to eradicate issues once they are found. Tools like Jira are great for tracking issues that people have found.”

My Initial Post / Peer B Response

“This a really well-considered post. I think that the groupings you have provided are a nice way to look at it.

What do you think about the importance of reusability? Can you consider the relative strengths and weaknesses?”

My Initial Post / Peer B Response / My Response

“Reusability as a Capability

I think we can acknowledge that 'reusability' as a concept is an important software capability, when the software helps solve a reoccurring problem and recreation of the software is timely; a common scenario. Then we could say 'reusability' is important because the time taken to create reusable software is amortized across every time the problem occurs. This time-efficiency allows people to do other important things with the time saved. Overall, there is a net positive gain in time, relative to the community containing the developers and users. On the other hand, non-reusable software requires additional time for it to be created every time it's needed, which isn't efficient for a community. So 'reusability' as a capability is important in that scenario.

On the flipside, if developers and users are part of different communities, with users being from a dangerous community, then sabotage might be reasoned to be important. Developers not providing reusability as a capability of their software, might be a viable form of sabotage against the dangerous community. This scenario is niche, and shouldn't be relevant to general software development.

Reusability as a Motive

In considering 'reusability' as a motive behind software development, it makes sense to consider software that is pay-per-use. Maximising the rate-of-pay generated by this type of software service is akin to maximising the reusability of the software. An example of this type of software service is gambling software; such a slot machines. Less obvious, is software that benefits from large amounts of users that create advertisement revenue.

A simple scenario to analysis the importance of reusability is when the software developers are from one community, but the users are from a dangerous community. If sabotage is justifiable, then the extraction of wealth from the dangerous community might be a viable form of sabotage. The motive of reusability behind software development is debatably important in that case as it boosts the effectiveness of the sabotage.

In a similar but more abstract scenario, payments may be made via information transfer instead of wealth transfer. The extraction of information boosts intelligence. The motive of reusability behind software development is debatably important in that case as it provides security intelligence. Tik Tok is purportedly an example of a software service that is believed to be an attempt of one country to boost it's intelligence about another.

In a more peaceful context where developers and users are from the same community, I think it's a bit more complicated to identify the scenarios where maximising reusability is an important motive. Any software that provides gratification on some level has the potential to become addictive and therefore harmful to the community. Any software that is popular on some level has the potential to increase the fragility of the community through bottlenecks that large portions of the community depend upon. Both these negatives are advantageous to businesses though. Businesses naturally focus on reusability as it's a desirable trait of software, and it provides a way to gain larger profits.

The Importance of Allocation

As a response to my observations above, I believe that the concept of 'allocation' is another important concept for software development businesses to consider. The allocation of important things to people without them is important for communities. For private businesses, I suspect this would require a governmental incentive.”

Peer A Initial Post

“The following is how I would rank the list of elements that impact the reusability of a piece of object-oriented software described by Padhy et al. (2018) in order of importance, starting with the most critical component.

First, Used in the Data Project should be at the top of the list. In object-oriented software development, it is essential to reuse data. Reusing data can save time and money because it eliminates the need to make new data for every project. Reusing data can also improve the project's quality and ensure the results are correct.

Second, Documentation in a Project should be the second most important thing on the list. Documentation is an integral part of software development because it helps ensure that the software is being built to meet the project's needs. Also, having Documentation that can be used again can save time and money because it means that new Documentation doesn't have to be made for every project.

Third, Architecture Driven Approach should be the third most important thing on the list. In object-oriented software development, it is essential to take an architecture-driven approach. This helps make sure that the software is designed in a way that meets the needs of the project. Using an architecture-driven approach can also help improve the software's quality and reliability and make it easier to use in the future.

The fourth most important thing on the list should be Modules in the Program . In object-oriented software development, modules are essential because they help ensure the code is organized in a way that makes it easy to reuse (Padhy et al., 2018, pp. 431-441). Modules can also make the software more efficient by breaking up the code into pieces that can be used in different projects.

Conclusion

• Used in the Data Project (UD) should be at the top of the list.

• Documentation in the Project (DIP) should be the second most important.

• Architecture Driven Approach (ADP) should be the third most important.

• Modules in the Program (MIP) should be the fourth most important.

• Followed by Test Cases/Test Design, Algorithm Used in the Program, Design Patterns, Knowledge Requirement, Models in the Project, Requirement Analysis and Service Contracts.

References:

Padhy, N., Satapathy, S. and 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 (pp. 431-441). Springer Singapore."

Peer A Initial Post / My Response

"Great post thank you,

It looks to me like you've prioritised the factors by how often they are reused within a *single* project. I really like this perspective, because it provides a guidance for good practice in very-large, open-source projects. If software projects get large enough, and enough people work on them, it's extremely important to make everything reusable, because nobody could ever know the entire codebase of the project anymore.

So, I 100% agree that data is likely to be reused within a project, especially if it's made to be public to all areas of the code. Likewise, documentation is key for large projects, as developers may need to maintain someone else's section of the code.

I'm wondering what your thoughts are on the following question: **which of Padhy et al's factors are easiest/most difficult to document during a large project**? I guess we can exclude documentation about documentation though.."

Peer A Initial Post / My Response / Peer A Response

"Thank you for your response.

That is a good question. I do not have much experience in a **large** project, what is the definition of a large project? (Is it a large team, or the complexity/time it takes to create said project? etc). However, If I were to rank the factors from easiest to most difficult, this would be the order:
Easiest: architecture, requirements, modules
Most Difficult: Test Cases, algorithms
I would rate the rest of the factors somewhere in the middle.

I would like to know your thoughts on my ranking and possibly how you would rank these factors. "

Peer A Initial Post / My Response / Peer A Response / My Response

"Thanks for the reply,

It's going to be arbitrary, but for discursive purposes, we can consider a large project to be one with 10,000+ lines of code. I could be wrong, but I believe that's enough lines of code to make an assumption that one person will not memorise the entire codebase. According to one of the ex-managers at Google, Rachel Potvin (2015), Google's entire codebase is more than 2 billion lines of code, so projects can get pretty big. I think for those larger projects, documentation may be the only way to quickly get the information you need from other sections of the code.

I've never worked on a large project, so I can only draw from experience on smaller projects too, but I suspect that requirements are quite easy to document too, thanks to use-case diagrams at the very least. I definitely agree architecture is easy to document, if there is a well-structured architecture such as a Model-View-Controller architecture that's not too granular on the details. The contents of modules shouldn't be too difficult to document either, as long as software developers proactively add comments to the header of functions/objects of the module to explain their intent, parameters and results. Then the modules documentation can just be automated as the collection of these comments.

For algorithms and test cases, its possible to encapsulate collections of them in modules, but deciding which modules should be created for a project in the first place, I don't think is trivial. Especially if you want to maximise reusability of the modules, and those algorithms/test cases across projects. I also don't think it's easy to label algorithms and test cases in a way that makes them obvious to find within documentation. Maybe there's some tricks for that too.

Do you think it's worth documenting the lesson's learnt from a project? I think it has personal value, but I'm not convinced it's valuable to anyone else. It's a good way of forming hypotheses for yourself though I guess?

References

Potvin, R. (2015) Why Google Stores Billions of Lines of Code in a Single Repository. Available from: https://www.youtube.com/watch?v=W71BTkUbdqE&ab_channel=%40Scale [Accessed 1 February 2023]."

Peer A Initial Post / My Response / Peer A Response / My Response / Peer A Response

This is quite interesting, with regards to documenting lesson's learnt from a project, my thoughts are that it has both personal value and value to the team/organization. I believe in programming we tend to encounter similar issues over and over and so being able to see an individuals thought process as to how they navigated their way through those issues but not only that it could be about how they found more easier/efficient and or robust ways of doing something.

I think this could be greatly beneficial especially to Juniors and Interns (and everyone else) who will be working on the same project later on down the line. However I would say, the documentation would have to be easily accessible and intelligible by relevant teams.

Peer C Initial Post

“Padhy et.al (2018) present a detailed approach and research methods that they have used to identify re-usable assets in software delivery. Based on professional experience and from implementing software I would categorise the assets below in terms of priority. It is fair to say that this is not set, and the selection and priority will vary between projects.

1. RA: Gathering requirements at the start of a project will capture and document the functional and non-functional needs of the projects which will then drive the decision making.

2. ADP: Architectural design alongside requirements will help drive the selection of existing/re-usable algorithms or new development.

3. KR: Requirements and architectural review will be considered alongside the knowledge of a re-usable algorithm and if it is usable in the scope of the architecture and answers the requirements.

4. DP: Existing design patterns can be evaluated against the requirements or if there is an over-arching DP for the customer it should be considered to ensure the algorithm fits into any agreed standards or protocols.

5. UDP: The data from previous implementations should be considered. While it may not be the same data for previous implementations it can impact design decisions.

5. AP: By developing a new algorithm based on requirements or design will add to a catalogue of re-usable assets and may also answer common issues. Or using an existing algorithm can allow for additions or enhancements.

6. MIP: By developing a modular design (A single or several artefacts e.g. a DLL) will allow for easy distribution of code and executables.

7. DIP: Documentation like design or specification documents should map to requirements to support developers in how to build the algorithm and documentation for existing algorithms should be maintained or updated if required.

8. SC: Once design & documentation is complete the contract should be set with the customer. Will decide testing approaches, collaboration between developer and customer in issues and delivery schedules.

9. TCTD: The steps up to and including the SC facilitate the creation of testing scripts and test cycles against the system/development.

10. MP: Models while still important are lower priority. The model for the algorithm or more importantly the solution can be fulfilled once all the design is agreed, how it fits into the architectural design, how it is tested etc. There may be iterations or changes to be considered in the earlier stages and where the aim is to reduce development time a model would be a final output alongside the solution/design artefacts.

It is worth noting this is very subjective and software delivery can vary from project to project. I have approached this based on my role as a solution architect so a role considering the wider aspects of a project. The company I work in is a COTS software vendor and I act as an overall technical lead and often these assets form deliverables or milestones in a project for how we deliver our implementation projects in the order I have listed. Internally we have our development teams working on our product but then our customers who work across many industries will often mandate their own priorities as well.

Really interested to hear any feedback from the rest of the group and to discuss further.

References:

Padhy, N., Satapathy, S., & 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. 77. Springer.”

Peer C Initial Post / My Response

“Thanks for the post.

Reflecting on priority lists, It seems a priority list is especially useful if you have a budget, and you need to allocate resources most efficiently, but it's difficult to be objective when making one. They are circumstantial, and each has it's own assumptions. So I agree with what you say.

So what I would like to consider, as it's less opinionated; is **what specific steps can be taken to make each asset more reusable?**

For example;

I believe data is clearly more reusable across projects if it's public, and it's structured well. Machine-readable formats allows data to be easily reused, and presentations of data as infographics allow the data to be reused in support of future developments. If the data is sourced-well, then it's also more reliable to be used in future projects. It's not always possible to reuse data though, so data reusability is conditional on it's relevance to future projects.

Have you got any tips on how to make any of these assets more reusable across projects?"

Peer C Initial Post / My Response / Peer C Response

"Thanks for the response Bradley.

As you said it is very objective and i think many people will have different views. In response to your question 'what specific steps can be taken to make each asset more reusable?' I would say that having well defined best practice in place not only for the development of these assets but also a well documented process that details the use of the asset, where it has been used before, what is it's purpose or what use case is it answering.

I think with well presented data, documentation, source control, processes and the asset users understanding of the domain are all key to making these assets more reusable. For example my organisation has a formal process in place where if a specific algorithm or modular piece of code is identified as been reusable it is formally documented, built, released and becomes part of our assessment process on projects. By becoming a formal release all of the assets is assessed i.e. how does it architecturally fit? Does it consider functional and non-functional requirements? Has the code been seen on previous projects? Does it fit existing design patterns? Then it is stored internally so it can be matched alongside requirements for the customer project and implemented if it meets one or possibly many customer requirements.

I think we have very similar views on this around structure, presentation, data so I hope I have answered your question."

Peer D Initial Post

"As stated in Padhy et.al (2018), it is difficult to produce a high-quality product in the software industry, therefore considering priority factors for reusability can be a deceiving process. That is because different types of industries have different types of requirements, hence priority factors may differ among software stakeholders.

Based on the finding above, and as a rule of thumb, I would prioritize the following factors from the most to the least important, when considering about software reusability.

1. **Requirement analysis**: At the beginning of the project, it is important to get familiar with the project and gather as most information as possible, as latter factor impacts everything that follows.

2. **Architecture driven approach**: It is crucial to hit the nail on the head, as a software, that does not meet the requirements is waste of time, cost and effort in the first place.

3. **Knowledge requirements**: As a baseline, professionals are required to foresee possible underlying mistakes/errors that may arise during implementation of the algorithms. Furthermore, as consequence, it takes knowledge and practice to carry out appropriate steps.

4. **An algorithm used in the program**: Being flexible is mandatory in today's world, therefore properly designed algorithms ready for reuse are an indispensable matter of each software.

5. **Modules in the program**: For the company that develops software for different industries, it is desirable to have a set of DLL files, that can be used by other developers where possible, in order to minimize the effort required to develop new features.

6. **Documentation in project:** Based on the above two factors it is important to document all the new features, so that the developers that take over the project are familiar with the design.

7. **Service contracts**: Being in a constant contact with the end customer is important, especially when considering usability and financial aspects of the software, as the project may evolve into a cost-inefficient burden bounded by the contract.

8. **Used in the data project**: Having experiences with the previous projects is important, however not every piece of software is the same as previous one, therefore it is important not to rely solely on previous experiences, as one might spiral down into unexpected issues, during software implementation.

9. **Design pattern:** Latter factor contributes to reusability through the well established project guidelines, so basically it reflects overall architectural design.

10. **Models in the projects:** Meaningful code in the project tasks is important, yet it mostly differs from software to software, therefore latter factor should not appear as most crucial when considering reusability.

11. **Test cases/test design**: As the main testing has to be already done during the implementation, it is at least important to conduct re-testing. Besides that, testing differs, depending on the type of the software, so it is hard to provide a common testing strategy for the entire software development company. Therefore this should not impact the reusability of an individual piece of software.

References:

Padhy, N., Satapathy, S., & 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*. 77. Springer."

Peer D Initial Post / My Response

"Interesting analysis, you ended up with quite a different list to me, but you've given a lot of good reasons for the value of each of factor. It looks like you've prioritised factors that make the developers jobs easier, so they can then make reusable software. I think this is a smart approach because as you say: "priority factors may differ among software stakeholders" and " different types of industries have different types of requirements", so perhaps there is no single set of factors that increase reusability for all users...

I'm curious what you think about the following question: **Is it possible to of make each factor reusable across projects?**
Algorithms of course can be reusable, if they are time-efficient, or space-efficient and relevant to other projects, but what about a requirement analysis? Is it possible to make it reusable from one project to another?"

Peer D Initial Post / Peer E Response (ref. My Response)

"Hi Ales,

Thank you for your post- I enjoyed seeing how you rank the importance of these factors in terms of reusability because you and I have similarly ranked our lists. I agree that Requirement Analysis deserves high priority because being familiar with the project requirements is crucial in any project; I even ranked Requirement Analysis as the most important factor.

However, I think Bradley raises a good question: is every factor in this list truly reusable? Are abstract, broad factors able to be used across projects? To answer Bradley's question, no, I do not think that each factor listed above can truly be made to be reusable in a useful, efficient way. Using Requirement Analysis as an example, familiarizing yourself with the requirements of one project will hardly ever be applicable to other projects because not every project has the same requirements. After considering this, my ranking of these factors would definitely change.

Thanks!"

Peer D Initial Post / Peer E Response (ref. My Response) / Peer D Response (ref. My Response)

“Hi,

I can see that the Interesting debate has started, so I will try to answer in one replay, not spamming and repeating the answers...

As a matter of reusability of individual factors, I would agree with you, that each factor can't be reused to the same extent. I think that you have very well summarised my thoughts to Bradley's question when considering whether it is possible to make each factor reusable across projects. Regarding both questions-and consequent answers I would like to add, that it probably also depends on the people responsible to conduct specific project, as one might prioritize one factor more than the same factor as his/her college. To sum up, I would say that it is not possible to make each factor reusable across all projects (at least not to the same extent/priority). This is simply because different people and different requirements are needed.

thank you both!”