**[Homework 15-16](https://github.com/hendraanggrian/IIT-ITM511/blob/assets/assignments/hw15-16.pdf): Software reuse and component-based SWE**

**Problem 1**

*What major technical and nontechnical factors hinder software reuse? Do you personally reuse much software and, if not, why not?*

Reusing software is a common technique to improve maintainability by modularization. The factors hindering software reuse are:

1. Technical

* **Overall complexity:** Modularizing a software project for reuse complicates the initial setup.
* **Incompatible components:** Software reuse might raise compatibility issues when the smaller parts are integrated.
* **Limited information:** Reusing a lesser-known or poorly documented solution slows down progress during technical roadblocks.

1. Non-technical

* **Cultural preference:** For example, a software team incentivizes original creativity in favor of default implementation.
* **Rights to reuse:** The licensing to use proprietary software components needs to align with the company's business model.

I, for one, use and encourage software reuse as much as allowed. The problems we face are likely already solved by other parties. Reusing software lets me focus on the business logic instead of reinventing the wheel. In my opinion, the only argument against reusing software is when the software components have duplicate resources or conflict with each other.

**Problem 2**

*List the benefits of software reuse and explain why the expected lifetime of the software should be considered when planning reuse.*

The benefits of reusing software are as follows:

* **Manageable approach:** Reusing software saves time by not creating everything from scratch.
* **Consistent result:** Well-maintained libraries or other software artifacts are rigorous as they have been used in real-world conditions.
* **Efficient development:** The combination of decreased time spent and consistent quality leads to efficient development planning, and in turn, project budget.

We should look after software capability, compatibility and lifetime expectancy when delivering or utilizing software reuse. Using an end-of-life product exposes us to known vulnerabilities because each software is periodically updated to introduce features and fix bugs. In another perspective, we should balance the support duration with the development cost of reusable software.

**Problem 3**

*Most desktop software, such as word processing software, can be configured in a number of different ways. Examine software that you regularly use and list the configuration options for that software. Suggest difficulties that users might have in configuring the software. Microsoft Office (or one of its open-source alternatives) is a good example to use for this exercise.*

* **Editing preferences:** Users modify text properties through configurable UI elements such as font dropdown, bold formatting toggle button, and a group of paragraph alignment radio buttons.
* **Difficulty:** Lack of basic understanding of graphical controls can make it difficult for new users to use software processing applications.
* **Dictionary and bibliography:** Software processors store user-defined words and citations in a centralized panel.
* **Difficulty:** Users need to be able to distinguish between document-specific and application-wide entries.
* **Style management:** One advanced feature of software processors is to retain consistency in text formatting by declaring and reusing custom text styles.
* **Difficulty:** Managing text styles and which paragraphs use them adds another layer of complexity to user experience.
* **Application compatibility:** A software processing document can be attached with raw files (image and audio), and exportable to other file types (PDF and RTF).
* **Difficulty:** Users have to acknowledge input types that are supported and which output types are expected.
* **Volume licensing:** Given administrative permission, ownership of proprietary products like Microsoft Word can be extended, terminated or transferred to another licensee holder.
* **Difficulty:** Choosing the right product can be overwhelming considering there are multiple versions of Microsoft Word with varied pricing. Users may also opt to use open-source alternatives like LibreOffice Writer but then would have to relearn how to navigate a new office suite.

**Problem 4**

*What are the significant benefits offered by the application system reuse approach when compared with the custom software development approach?*

* **Scalability in mind:** A software team is overseeing a smaller and more manageable sub-project with software modularization.
* **Shared stability:** The development and testing of reusable software benefits every other project integrating them.
* **Lean codebase:** Better readability because the software project size is reduced by integrating available solutions.
* **Standardized approach:** An argument to reuse software is endorsed by the IEEE board in their publication of software reusing guidelines (IEEE, 2005).

**Problem 5**

*Explain why adaptors are usually needed when systems are constructed by integrating application systems. Suggest three practical problems that might arise in writing adaptor software to link two application systems.*

A software project may be composed of smaller software components that use different technology. The differences in communication protocol or data compatibility impede their ability to integrate. To form a connection among unsupported components, an adaptor is necessary to translate data into a readable format.

The challenges of writing an adaptor software are:

* **Performance slowdown:** Converting data to a non-native format is an extra execution that negatively impacts hardware performance.
* **Corrupted conversion:** The system needs to establish an error-handling mechanism to deal with data mishandling in case of unpredictable input.
* **Dependency update:** A downstream patch is required for every upstream change that breaks the adaptor.

**Problem 6**

*What are the design principles underlying the CBSE that support the construction of understandable and maintainable software?*

1. **Components' independence:** A utilized software component should operate individually so that changes may apply without system disturbance (Sommerville, 2016, p. 466).
2. **Components' connectivity:** The ability to swap software maintenance given sufficient maintenance to the adaptor in charge of communication.
3. **Components' redundancy:** Discourages duplication of implementation or resources while a pre-built solution already exists.

**Problem 7**

*The principle of component independence means that it ought to be possible to replace one component with another that is implemented in a completely different way. Using an example, explain how such component replacement could have undesired consequences and may lead to system failure.*

The design principle of component independence is concerned with project flexibility by allowing a replacement within the existing software composition. However, migrating to an alternative solution with an unfamiliar approach has its risks. In one of the most severe cases, an integration error fails the system and destroys consumer data in the process.

For instance, a website development team decides to replace its legacy web framework with a modern counterpart. However, the obsolete web framework offers an all-in-one solution that cannot be replaced by a single component such as account authentication, database management and content caching. Since this is such a major change within the application fundamentals, it has a higher probability of behaving randomly or producing unreliable results.

**Problem 8**

*In a reusable component, what are the critical characteristics that are emphasized when the component is viewed as a service?*

1. **Functions independently:** No prior knowledge of reusable software internal procedure is required because the components interchange through a public interface (Sommerville, 2016, p. 468).
2. **Parameterized connectivity:** In this secure interface, reusable software shall communicate without leaking sensitive information potentially damaging itself or the host system.

**Problem 9**

*Why is it important that components should be based on a standard component model?*

Software products come in various forms depending on the problem they intend to solve, this raises an issue when they are unable to have a meaningful connection due to differing technologies. The standard component model as described in component-based software engineering (CBSE) acts as architectural guidelines for communication interface and deployment. The unified development approach should result in seamless integration with each other.

**Problem 10**

*What are the essential differences between CBSE with reuse and software processes for original software development?*

1. Requirements gathering
2. **CBSE with reuse:** Planning for requirements gathering starts with abstract ideas that are iteratively improved (Sommerville, 2016, p. 477).
3. **Original development:** Strict planning analyzing the entire collection of reusable software components before utilization.
4. Development flexibility
5. **CBSE with reuse:** A software team may substitute a component if it no longer fulfills user requirements.
6. **Original development:** A change to the technological stack is to be discussed with the management board.
7. Further refinement
8. **CBSE with reuse:** Keep searching for alternatives to ensure that the current composition is robust.
9. **Original development:** Feature changes are delegated into the next releases to avoid disruption in development unless it is a critical bug demanding immediate attention.
10. Integration plan
11. **CBSE with reuse:** The general development plan mainly deals with how software components associate with one another. The layer of extra functionality is added after the integration process.
12. **Original development:** In incremental development, each new software component is deployed and tested before fully assembling them into a final product.

# **Bibliography**

IEEE. (2005, September 12). *Software reuse guidelines*. Retrieved from IEEE Xplore: https://ieeexplore.ieee.org/document/1506485/

Sommerville, I. (2016). Software Engineering. In *Software engineering diversity* (10 ed., pp. 466, 477). Pearson Education.