Software Workshop / Worksheet 4, Exercise 5
Name: Vasileios Manolis, **Student ID:** 1772373 (vmx773)

Overview: Programmers have developed a number of paradigms when using computers to solve problems such as the imperative, logic, function and object-oriented paradigms. In this executive summary, the rationale of using object-oriented programming and its limitations will be analysed.

1. The Rationale of Using of Object Oriented Programming: *Problems* which are being addressed and *ways* these problems are addressed by OOP.

1.1. OOP Encourages Software Reuse

One great problem that OOP solves is related to the reuse of software. With OOP there is no need for programmers to rewrite the same part of code multiple times for the same object as they can make use of the same class to build multiple objects.
Example: If a programmer that works in a school has to develop profiles for 300 students, he can create a "Student" class once, and then he can reuse this class to build 300 objects, in our case, 300 students.

1.2. OOP Reduces Maintenance Cost & Effort

The reuse of software can lead to significantly reduced maintenance cost when programmers use OOP. In addition to that, programmers do not save time and effort only from the code writing phase but also while testing and debugging code written in OOP. Considering the fact that the cost of maintaining code is usually greater than the cost of writing code and code that cannot be maintained is not useful, OOP was a solution for lower costs.
Example: If the programmer of the previous example finds a bug in his "Student" class he has to solve the problem only once by debugging this particular class and not the entire population of objects.

1.3. The Reduced Maintenance Cost & Effort Leads to Further Improvements

The reduced maintenance cost means that OOP allows programmers to invest more time to improve other aspects of their software for example, the functional and non-functional parts of their code.
Example: The school programmer can focus to improve the speed and security of his code with OOP. Theoretically, he would not be able to do that if he had to debug the same piece of code multiple times.

1.4. OOP Allows Programmers to Easily Develop Software for Complex Problems

Once an object is built, programmers do not need to have extensive knowledge of how it works. Encapsulation, one of the fundamentals of OOP, allows users to quickly develop software for complex problems by using smaller pieces code that already work. That solved the problem of older paradigms, where programmers had to understand details about a piece of code before using it. Furthermore, data encapsulation provides the ability to hide certain parts of the Objects from other programmers. Thus, bugs and logical errors can be avoided and more stable and safer programs can be built as data access can be restricted leading to better security. Thus, OOP contributed to safer and more understandable programs.
Example: The school programmer hired an assistant to help him build an additional project. The assistant can quickly use classes created by the school programmer without knowing all the details about them or fully understand how they work. At the same time, with OOP the school programmer can prevent his assistant from accidentally giving wrong values to certain important variables.

1.5. OOP Created Common Ground Among Developers

The rationale of using OOP can also be explained by the great number of high quality sources that exists for object-oriented programming languages. Students all around the globe, can easily find plenty of material online and offline about OO languages. OOP solved the problem of communication

between developers as it is regarded today as common ground for the majority of developers no matter their background, ethnicity etc.
Example: Students from developing countries or students growing in rural areas can easily find plenty of material for OOP and high quality free classes in massive open online courses websites. It is also easier to find answers to their questions in forums (e.g. StackOverflow) for OO languages.

2. Limitations of Object-Oriented Programming

2.1. OOP Requires Intensive Testing

Intensive testing procedures are required in OOP programs. Programmers have to extensively test at the very least their code 100%.
Example: The school programmer has to write tests that cover at least the 100% of his code.

2.2. OOP Has a Steep Learning Curve

Some people find the thought process unnatural when creating an object-oriented program. That might lead to logical errors, code of bad quality and project delays.
Example: The school programmer's assistant, who is a newly hired software engineer, might find inheritance and polymorphism difficult to understand. Because of that, he made several errors while designing his code.

2.3. OOP Required More Effort & Time-Solving Problems

Many researchers believe that OOP requires initially more time to solve some problems compared to using another paradigm such as a structured programming approach. OOP also might require initially more effort while planning how to structure a piece of code.
Example: The school programmer and his assistant has to spend much time to design a complete system for the school he works for.

2.4. OOP Programs can be Larger and Slower

According to Deshpande (2016, p.524), OOP programs are usually larger. That was a serious problem especially a few years ago when size limitations were stricter. OOP programs might also be slower, and that partially because of their larger size.

Conclusion: Despite its limitations, object-oriented programming helped programmers to solve major issues they experienced with previous paradigms. These benefits led to object-oriented programming to be one of the most dominant paradigms of software today.

References

Deitel, P. and Deitel, H. (2014). *Java How to Program (Early Objects), Tenth Edition*. 10th ed. Prentice Hall, pp.52-55.

Deshpande, S. (2016). Collaboration of Object Oriented Programming and Software Development. *International Research Journal of Engineering and Technology (IRJET)*, 03(09), pp.524-525.

R. Doong and P. Frankl. (1994). The ASTOOT approach to testing object-oriented programs. *ACM Transactions on Software Engineering and Methodology*, 03(02), pp.101-130.

Arran Stewart, Rachel Cardell-Oliver & Rowan Davies (2017). A fine-grained framework for quantifying secure management of state in object-oriented programs. *International Journal of Computers and Applications*, 01(02), pp. 5-17.