Write an essay on your favorite software development paradigm and make a case for why you would choose it over the others.

A software development paradigm guides the way that developers view a given problem and organize the solution.

Software developers break the overall development process of a project up into smaller, more manageable steps or phases. The choice of software development process has a significant influence on any project’s success.

The appropriate process can lead to faster completion, reduced cost, improved quality, and lower risk, while wrong process can lead to duplicated work efforts and schedule slips, and create continual management problems.

Software developers have experimented three major software development paradigms: procedural, data driven, and object-oriented. Each of these paradigms attempts to solve a real-world problem with a software solution. Each paradigm is also able to do this but that some are able to do so more quickly than others.

1. Procedural

The procedural paradigm focuses on the algorithms or steps necessary to solve a problem. It breaks the problem into more simple sub-problems represented by procedures, functions, or methods. The procedural decomposition is depicted as a hierarchy, either a list or a tree.

The procedural model is appropriate for small, simple programs, which will carry over to the study of the object-oriented paradigm.

The disadvantage is that there are no well-defined rules for performing a decomposition or for determining when the functions were sufficiently simple to allow programming to begin. This means that two different but experienced practitioners could create two quite different designs.

1. Data Driven

Data Driven follows the data as it enters the system, passes through the processes that transforms the data, and continues to follow the data until it leaves the system. The end result is mostly a web of data and processes that described a given problem.

An analysis based on any of the data driven paradigms resulted in better problem domain understanding than an analysis based on a procedural decomposition.

Data flow-based models provides better support for software testing, validation, and documentation, and the data flow diagrams are easier for customers familiar with a problem to read than are the procedural decompositions. Unfortunately,

However, there is a large gap between the data flow diagrams and the programs during the implementation phase. This large gap makes programming difficult and reduces the data driven paradigm's usefulness.

1. Object-Oriented

The object-oriented paradigm is favorite paradigm because it encapsulates data and the procedures or operations that may access and operate on that data into an entity called an object. A group of similar objects are described by a class. Classes are typically documented with a graphical notation

The class determines what data and operations each object will support but the values stored for each data element of one object is independent of the other objects.

During the design phase, detail is added to the classes, new classes may be added, and some classes may be eliminated, but for the most part, the classes discovered during analysis are carried over to design. The refined classes are carried over into the implementation phase, then into an appropriate programming language.

Fortunately, the translation process is well-defined and one-to-one (meaning that aside from cosmetic nuances there is only one way to do the translation). More fundamentally, the *concepts* of classes and objects do not change from one phase to another. This conceptual consistency forms bridges between the real-world, analysis and design, and implementation.

The ability of the object-oriented paradigm to bridge the gaps between the phases and to smooth the development process is only one of its many strengths. It retains the best characteristics of the procedural and data driven paradigms while overcoming or minimizing their worst characteristics.

Furthermore, as each class defines a new, intermediate scope , the object-oriented paradigm also allows some but not all of the procedures in a program to access the data. Controlling data access reduces the functional coupling that ultimately set an upper limit to the size and complexity of software systems that could be practically created based on the procedural paradigm. The many strengths of the object-oriented paradigm make it the current best practice for creating large, complex software systems.