Volume 6, Issue 8, August 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Comparative Study of Structured and OOP Paradigms

Fergus U. Onu
Computer Science Department
Ebonyi State University, Abaikiliki, Nigeria

Michael Ogu Ezeji
Computer Science Department
Ebonyi State University, Abaikiliki, Nigeria

Abstract- The principles of object oriented technology problem solving is the vogue now that many problems are best approached by constructing models of real-world situations. The basis of these models are interacting objects with well-defined properties and behaviours. Solving a problem using the object-oriented approach thus consists of identifying appropriate objects and describing the functions that they must be able to perform and the information that they must hold. A computer application can then be constructed by converting such a description into a programming language. But the conventional languages have been there and have lived their bills. But what's the hype for OOP? What's the strengths and weakness of structured programming? What of the OOP? This work set out on deep-rooted comparative study to unearth the differences between the programming paradigms – object oriented and structured paradigms.

Keywords– OOP Paradigm; Structured Programming Paradigm; Real Life Models; Comparative study of OOP and structured programming; problem solving techniques;, Features of OOP; Features of Structured Programming; Applications of OOP

I. INTRODUCTION

1.1 Background of Study

In the world of computer programming, the object oriented paradigm has become a new way of thinking: the en thing. It isn't as old as the non-object programming is, but have gained a tremendous attention. It is more like a natural way of solving problem i.e. solving problem from the problem domain. Conversely, other way of solving problem that have featured from the birth of programming in 1950s had been conventional programming; made up of structural and procedural programming.

By the way, a *problem* is a functional **SPECIFICATION** of desired activities to generate the intended output. A *solution* is the **METHOD** of achieving the desired **OUTPUT** (result). Let's see an example: submitting an assignment to a lecturer is a ***problem*** statement, while boarding a vehicle, emailing or couriering the assignment to him is a ***solution***. The lecturer's receipt of it is the ***output*** (result). So, there is a problem, there is a solution, and there is a result (output); so are their respective domains. The domain or the sector to which the problem (that which specifies the requirements in a particular knowledge domain and the domain expert associated with the task of explaining the requirements) belong to defines the PROBLEM domain. Similarly, the subject matter that is of concern to the computer (in solving a specific task [i.e. programming] and the person associated with the task of devising solution) defines and belongs to the solution domain. So, problem domain specifies functional *requirements* in knowledge, *domain expert* and *scope of problems* while solution domain specifies *subject matter*, *developer* and *procedures/techniques*. [Note, domain expert is one with deep knowledge of the domain. Developers are those developing the programs – the suppliers, programmers or implementers. Users are those using the solution – the clients, customers or the end-users.] Hence, problem solving is a mapping of problem domain from start state to solution domain [i.e. problem to end/goal state]. See it;

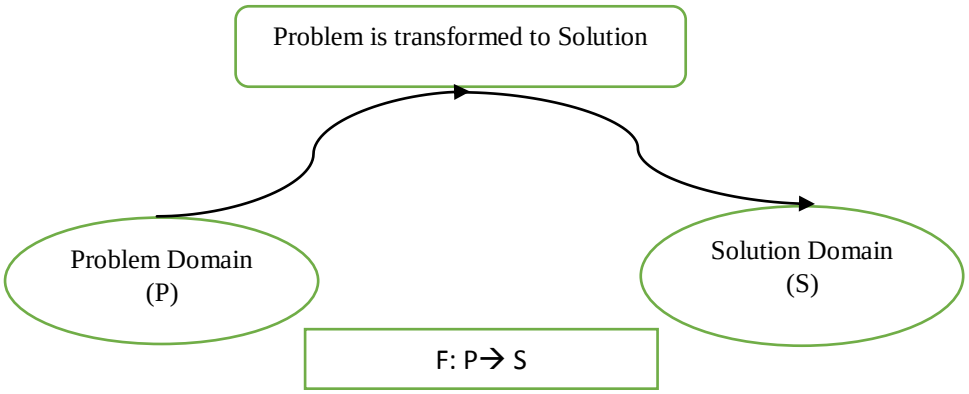


Fig. 1.1. Problem Solving

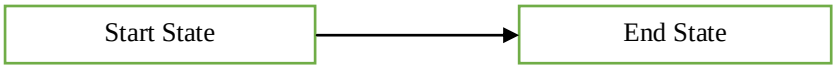


Fig. 1.2. Problem to Solutions Mapμιg

The solution is represented in a *program* which is instructions to follow by computer to generate output.

1.2 The Problem Statement

Different problems called for different expertise and approach. More to this is; computing needs have become pervasive, ubiquitous and complex. Data is countless in a big-data era that we are in already, yet data are needed in the program and will be required to take different forms beyond data structure. A typical programmer is confused sometimes on which programming tool/language can solve his/her problem or one to learn since nobody can know all. These needs exist as to how do we know the capabilities, applications areas, limitations and learning curve, hence the contrasts of each of these programming families: procedural, structural and object-oriented [problem statement]. It is this problem that this study is set out to provide answer cum solution to.

1.3 Aim & Objectives of the Study

The **AIM** of this study which is *Comparative Study Of Structural and Object-Oriented Programming* paradigms is to *study*, *compare* and *contrast* the *features*, *capabilities [hence application areas]* and *limitations* of Object-Oriented Programming paradigm as against the structured programing paradigms by examining in specific **OBJECTIVES** as;

1. To justify the recent publicity of the ***new*** thinking – Object-Oriented Paradigm in programming.
2. To find the features [principles], capabilities [applications] and limitations of the ***structural*** programming paradigm
3. To find the features [principles], capabilities [applications] and limitations of the ***Object-Oriented*** programming paradigm
4. To contrast the two [#2 and #3].

1.4 Significance [Importance] of the Study

This study contributes in its importance vis-à-vis significance to learning, practice and knowledge. Programming paradigm or choice of programming tool is not a fanfare. It is determine primarily by the ability of the programming language or paradigm to solve the problem. The realization of the set objectives will contribute to;

- ≈ Researchers/scholars/students – in granting them in-depth understanding of different programming paradigms from evolution to date and in enlarging their skills, versatility and flexibility in developing of programmes and by extension software
- ≈ Programmers – in knowing the best tool to use base on the nature and volume of data, maintainability or scope of the programme
- ≈ Other Interested Persons – in empowering him/her with the knowledge of programming that puts such on cutting-edge technology for the generation-next.

Having establish the foundation so far, we move to explore the works of other researchers here.

II. LITERATURE REVIEW

2.1 Introduction

In this section, we'll look at the concepts, theories and literatures behind these paradigms – OOP, procedural and structural.

2.2 Conceptual Framework

Programming is a sequence of instruction for the computer to perform a specific task. It can be procedural, structural, object-based, object-oriented and so forth. Procedural programming is a list or set of instructions telling a computer what to do step by step and how to perform from the first code to the second code. Procedural programming languages include C, Go, FORTRAN, Pascal, and BASIC. [1]

But, what is a procedure in a program? This is same as routine, subroutine, and function. A procedure is a section of a program that performs a specific task. It is an *ordered* set of tasks for performing some action. [4]

The core of the pure object-oriented programming is to create an object, in code, that has certain properties and methods [2]. Someone said, 'The prime purpose of C++ programming was to add object orientation to the C programming language, which is in itself one of the most powerful programming languages.'

Object-oriented programming (OOP) is a programming language model organized around ***objects*** rather than "*actions*" and ***data*** rather than *logic*. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data [3]. Object-Oriented Programming (OOP) is the term used to describe a programming approach based on *objects* and *classes*. The object-oriented paradigm allows us to organize software as a collection of objects that consist of both data and behaviour. This is in contrast to conventional functional programming practice that only loosely connects data and behaviour [5]. Since the 1980s the word *'object'* has appeared in relation to

programming languages, with almost all languages developed since 1990 having object-oriented features. Some languages have even had object-oriented features retrofitted.

In computing, the solution to a problem is written in the form of a *program*, while a computer is used to solve the problem. We know [and have said it] that a program is a set of instructions for computer to perform a specific task, written in a *programming language.* A programming language provides the platform, the medium for conveying the instructions to the computer. There are many programming languages such as BASIC, FORTRAN, Pascal, C, C++, etc. Once the steps to be followed for solving a problem are once identified, it can be sequenced, flowcharted and coded. This is to say, it is easier to convert these steps to a program via a programming language.

The formulation of a solution is important before writing a program. It requires logical thinking, careful planning and a systematic approach. This can be achieved through the proper combination of domain experts, system analysts/system designers, and developers. The program takes the input from the user and generates the desired output.

1) Applications Of Object-Oriented Programming

If there is complexity in software development, object-oriented programming is the best paradigm to deploy and solve the problem. The following areas make use of Object Oriented Program:

i. Image processing
ii. Pattern recognition
iii. Computer assisted concurrent engineering
iv. Computer aided design and manufacturing
v. Computer aided teaching
vi. Intelligent systems
vii. Data base management systems
viii. Web based applications
ix. Distributed computing and applications
x. Component based applications
xi. Business process reengineering
xii. Enterprise resource planning
xiii. Data security and management
xiv. Mobile computing
xv. Data warehousing and data mining
xvi. Parallel computing

Object concept helps to translate our thoughts to a program. It provides a way of solving a problem in the same way as a human being perceives a real world problem and finds out the solution. It is possible to construct large reusable components using object-oriented techniques. Development of reusable components is rapidly growing in commercial software industries.

2.3 Theoretical Framework

There are foundational theories and principles in which a research is based on. Programming paradigms unarguable are built on its evolution of programming languages, features and principles of problem solving.

1) Evolution of Programming Languages

A program is written IN a *programming* language. Programming language provides a medium for conveying the instruction to the computer while the computer is used to solve the problem. Few of the programming languages in their respective generations are:

FIRST generation programming languages (1954–1958) such as FORTRAN 1, ALGOL 58, IPL V and FLOWMATIC were used for numeric computations. They explained basic language and implementation concepts. Computation of numbers was the sole essence of programming. Every program makes use of *data*. Data is represented by a variable or a constant in a program. Data are declared before used. To perform an action, an operator [arithmetic and logical] acts on the data (operand – variables and constants). Operands and operators are combined to form *expressions*. Each instruction is written as a statement with the help of expressions. The structure of first generation languages is shown in Fig. 1.3.

F ig. 1.3 Structure of the first generation languages

There is no emphasis or support for subprograms because subprograms neither were considered as issue nor it entering their thoughts then. Such programming thinking is known as *monolithic programming*. The data is globally available here and hence there is no chance of *data hiding* (concept of denying the access of data). First generation languages were used only for *simple applications*. The program is closer to the solution domain by representing the operations/operators in the programming language that can be performed in the computer.

SECOND generation programming languages (1959–1961) with FORTRAN II, ALGOL 60, COBOL and LISP introduced subprograms (i.e. functions, procedures or subroutines) which was missing in first generation as shown in Fig. 1.4. Inclusion of subprograms avoids repetition of coding. Such programming is known as *procedural programming*. Second generation languages are suitable for applications that require *medium-sized* programs. FORTRAN II, COBOL, and ALGOL 60 are second generation languages survivors to-date especially the first two. The second generation languages provided the possibility of *information hiding* (i.e. hiding the implementation details of a subprogram). However, the *fact* of sharing the same data by many subprograms *breaks* and *weakens* the data-hiding principle. Hence, data hiding only partially succeeded. Here also in this second generation, the program is closer to the solution domain where *concentration* is on operations/operators using functions.

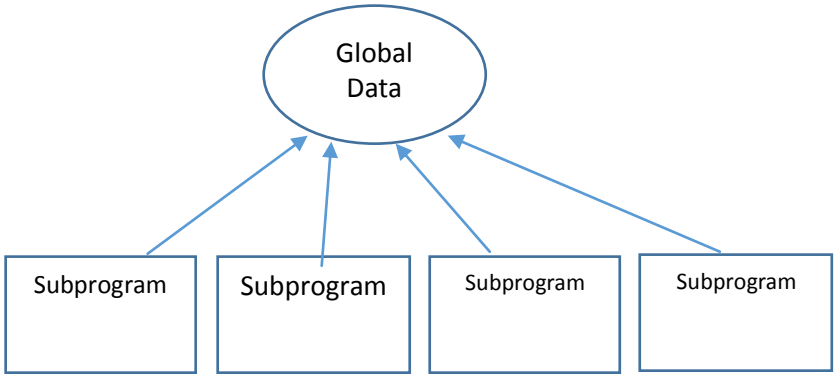


Fig. 1.4. Structure of the second generation languages

THIRD generation programming languages (1962–1969) with PL/1, ALGOL 68, Pascal, Simula and C were not as successful as second generation languages. They use sequential code, global data, local data, and subprograms as shown in Fig. 1.5. [Pascal and C were their major survivors]. They follow structured programming, which supports *modular* programming. The program is divided into a number of modules, concept called *modularity*; each module consists of a number of subprograms. Importance was given to developing an algorithm and hence this approach is *also* known as *algorithmic oriented* programming. In structural programming approach, data and subprograms exist separately as;

Algorithms + Data Structures = Programs,

where a main program calls the subprograms.

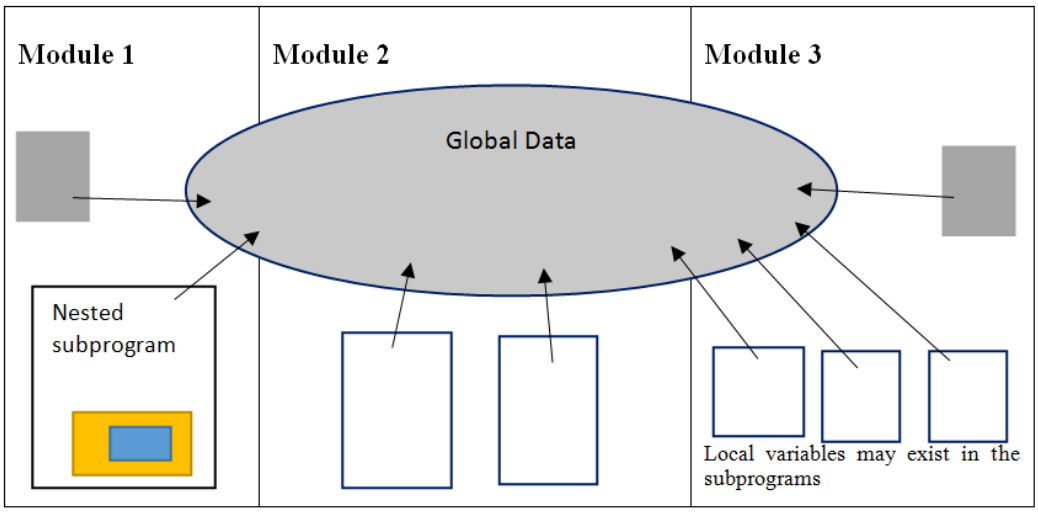
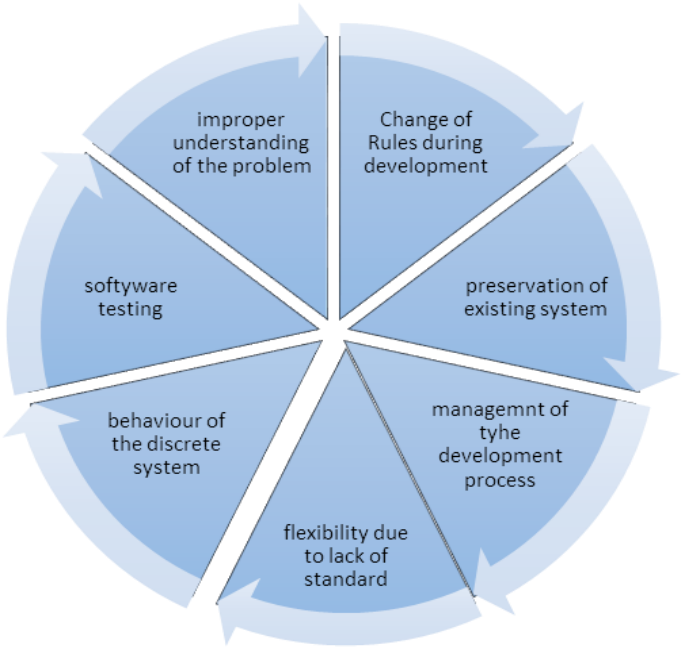


Fig. 5. Data in third generation programming language

Prior to initial evolutionary phases of computing technologies (period before 1990) simple problems were solved using computers. These days, computers are utilize in solving many mission-critical problems and they are playing a vital role in the *fields of space, defense, research, engineering, medicine, industry, business* even in *music* and *painting*. For example, fishing in Oil-well, simulations in manufacturing, Inter-Continental Ballistic Missiles (ICBM) in defense and launching of satellites in spacecraft cannot be done or controlled without computers. Such applications cannot be even ***imagined*** without computers – in fact with terrible computer programs that procedural programming cannot capture. Influence of computers in various activities leads to the establishment of many software companies engaged in the development of various types of applications.

1) The Software "Thing" Becoming Complex

Large projects involve many highly qualified persons in the software development process. Software industries face a lot of problems in the process of software development. These following factors according to Cunz (2012) influence the complexity of software development, as shown in Fig. 1.6.



Source: Object-Oriented Programming Using Java (Cunz, 2012)

Fig. 1.6. Factors influencing Software Development Complexity

1. Improper Understanding Of The Problem.. The users of a software system express their needs to the software professionals to develop solutions for them. The requirement specification is not precisely conveyed by the users in a form understandable by the software professionals. There is a lacuna in communication. This is known as impedance mismatch between the users and software professionals.

2. Change Of Rules During Development. During the software development process duration, because of some government policy or any other industrial constraints realized, the users may request the developer to change certain rules of the problem already stated.

3. Preservation Of Existing Software. In reality, the existing software is modified or extended to suit the current requirement. If a system had been partially automated, the remaining automation process is done by considering the existing one. It is expensive to preserve the existing software because of the non-availability of experts in that field all the time. Also, it results in complexity while integrating newly developed software with the existing one.

4. Management Of Development Process. Since the size of the software becomes larger and larger in the course of time it is difficult to manage, coordinate and integrate the modules of the software.

5. Flexibility Due To Lack Of Standards. There is no single approach to develop software for solving a problem. Only standards can bring out uniformity. Since only a few standards exist in the software industries, software development is a laborious task resulting in complexity.

6. Behaviour Of Discrete Systems. The behaviour of a continuous system can be predicted by using the existing laws and theorems. For example, the landing of a satellite can be predicted exactly using some theory even though it is a complex system. But, computers have systems with discrete states during execution of the software. The behaviour of the software may not be predicted exactly because of its discrete nature. Even though the software is divided into smaller parts, the phase transition cannot be modeled to predict the output. Sometimes an external event may corrupt the whole system. Such events make the software extremely complex.

7. Software Testing. The number of variables, control structures and functions used in the software are enormous. The discrete nature of the software execution modifies a variable and it may be unnoticed. This may result in unpredictable output. Hence, vigorous testing is essential. It is impossible to test each and every aspect of the software in a complex software development system. So only important aspects are subjected to testing and the user must be satisfied with it. But the reliability of the software depends on rigorous testing. But testing processes make software development more and more complex.

Uniquely, each programming language enforces a particular style of programming. The way of organizing information is influenced by its style of programming and it is known as *programming paradigm – programming thinking.*

2) Foundational Theories OOP is Built On

These are the fundamental features of object-oriented programming, just as the theories that form the foundation and features of Object-Oriented Programming. They are as follows:

i. Encapsulation
ii. Data Abstraction
iii. Inheritance
iv. Polymorphism
v. Extensibility

vi. Persistence
vii. Delegation
viii. Genericity
ix. Object Concurrency
x. Event Handling
xi. Multiple Inheritance
xii. Message Passing

1 Encapsulation. *Encapsulation* is the process, or mechanism, by which you *combine* code and the data it manipulates into a single unit [capsule]. It provides a layer of security around manipulated data, protecting it from external interference and misuse. In Java, this is supported by *classes* and *objects*.

2 Data Abstraction. Real-world objects are very complex and it is very difficult to capture the complete details, hence, OOP uses the concepts of abstraction and encapsulation. Understand that abstraction is a design technique that focuses on the essential attributes and behaviour. Data abstraction is a named collection of essential attributes and behaviour relevant to programming a given entity for a specific problem domain, relative to the perspective of the user. Closely related to encapsulation, data abstraction provides the ability to create user-defined data types.

Data abstraction is the process of *abstracting* common features from objects and procedures, and creating a single interface to complete multiple tasks e.g., a programmer may note that a function that prints a document exists in many classes, and may *abstract* that function, creating a separate class that handles any kind of printing. It also allows user-defined data types that, while having the properties of built-in data types, it also allows a set of permissible operators that may not be available in the initial data type.

3 Inheritance. This allows the extension and ***reuse*** of existing code, without having to repeat or rewrite the code from scratch. Inheritance involves the creation of new classes, also called *derived* classes, from existing classes (*base* classes). Allowing the creation of new classes enables the existence of a hierarchy of classes that simulates the class and subclass concept of the real world. The new derived class inherits the members of the base class and also adds its own. See this example, a banking system of course have customers, it keeps information such as name, address, etc. A subclass of customers could be customers who are students, where it not only we keep their names and addresses, but also track the educational institution where they are schooling. This is a class and subclass example.

Inheritance is mostly useful for ***two*** key programming strategies: extension and specialization.

For extension, extension uses inheritance to develop new classes from existing ones by adding new features.

For specialization, specialization makes use of inheritance to refine the behavior of a general class.

4 Multiple Inheritance. When a class is derived through inheriting one or more base classes, it is being supported by *multiple inheritance*. Instances of classes using multiple inheritance have instance variables for each of the inherited base classes.

5 Polymorphism. It allows an object to be processed differently by data types and/or data classes. More precisely, it is the ability for different objects to respond to the same message in different ways. Polymorphism allows a single name or operator to be associated with different operations, depending on the type of data it has passed, and gives the ability to redefine a method within a derived class. For example, given the *student* and *business* subclasses of customer in a banking system, a programmer would be able to define different getInterestRate() methods in student and business to override the default interest getInterestRate()that is held in the customer class.

6 Delegation. Delegation is an alternative to class inheritance. Delegation allows an object composition to be as powerful as inheritance. In delegation, two objects are involved in handling a request: methods can be delegated by one object to another, but the *receiver* stays bound to the object doing the delegating, rather than the object being delegated to. This is analogous to child classes sending requests to parent classes.

7 Genericity. This is a technique for defining software components that have more than one interpretation depending on the data type of parameters. Thus, genericity allows the abstraction of data items without specifying their exact type. These unknown (generic) data types are resolved at the time of their usage (e.g., through a function call), and are based on the data type of parameters. For example, a *sort* function can be parameterized by the type of elements it sorts. To invoke the parameterized sort(), just supply the required data type parameters to it and the compiler will take care of issues such as creation of actual functions and invoking that transparently.

8 Persistence. This is the concept by which an object (a set of data) outlives the life of the program, existing between executions. All database systems support persistence, but, persistence is not supported in Java. But, persistence can be simulated through use of *file streams* that are stored on the file system.

9 Concurrency. Using concurrency allows additional complexity to the development of applications, allowing more flexibility in software applications. In Java, it is represented through threading, synchronization, and scheduling.

10 Events. An event can be considered a kind of interrupt (a software or hardware that sends an alert to the Operating System to respond to certain system call and resume): they interrupt your program and allow your program to respond appropriately. In a conventional, nonobject-oriented language, processing proceeds literally through the code: code is executed in a ‘top-down’ manner. The flow of code in a conventional language can only be interrupted by loops, functions, or iterative conditional statements. In an object-oriented language such as Java [handled through the EventHandler class], events interrupt the normal flow of program execution. Objects can pass information and control from themselves to another object, which in turn can pass control to other objects, and so on.

3) Theoretical Foundation of Conventional [Procedural & structured] Programming

The structured program theorem provides the theoretical basis of structured programming. It states that three ways of combining programs and subprograms are sequencing, selection and iteration and that these are able to express any computable functions. This observation did not originate with the structured programming movement. These structures are sufficient to describe the instruction cycle of a Central Processing Unit as well as the operation of a Turing machine. Therefore a processor is always executing a "structured program". In this sense, even if the instructions it reads from memory are not part of a structured program. The structured program theorem does not address how to write and analyze a usefully structured program. These issues were addressed during the late 1960s and early 1970s with major contributions by Dijkstra, Robert Floyd W, Tony Hoare and David Gries.

III. SYSTEM ANALYSIS & METHODOLOGY

Research methodology is the process used to collect data cum information for the purpose of making research decisions. The methodology may include publications, interviews, surveys, observation, experimentation, and other research techniques, and could include both historical and present information. A research will typically include how data will be collected, what instruments will be used and the intended means of analyzing data collected [Business Dictionary].

This research explored various texts and publications for data on structured, procedural and object-oriented programming languages from FORTRAN, PASCAL, COBOL, GW-BASIC, Q-BASIC, C, DBASE III/IV, OO-COBOL, OO-PASCAL, VB and C++ to JAVA.

IV. SYSTEM DESIGN & IMPLEMENTATION

4.1 Natural Principles: The O-O Programming Paradigms

Object-oriented programming languages are developed based on object-oriented technology.
The natural way of solving software problem follows the following basic principles:

- ≈ message passing
- ≈ abstraction
- ≈ encapsulation

The importance of data is realized through O-O technology, which follows the natural way of solving problems. The object-oriented approach to programming is an easy way to master the management and complexity in developing software systems that take advantage of the strengths of data abstraction. Data abstraction and data encapsulation help to make the abstract view of the solution with information hiding. Data is given the proper importance and action is initiated by message passing.

Data-driven methods of programming provide a disciplined approach to the problems of data abstraction, resulting in the development of object-based languages that support only data abstraction. Depending on the object features supported, there are two categories of object languages:

1. Object-Based Programming Languages
2. Object-Oriented Programming Languages

Data and functionalities are put together resulting in objects and a collection of interacting objects are used to solve the problem. These object-based languages do not support the features of the object-oriented paradigm, such as inheritance or polymorphism.

Object-based programming languages support encapsulation and object identity (unique property to differentiate it from other objects) without supporting important features of OOP languages such as polymorphism, inheritance, and message based communication, although these features may be emulated to some extent. Three examples of typical object-based programming languages are Ada, C and Haskell. See the make-up;

Object-based language = Encapsulation + Object Identity

Object-oriented languages incorporate all the features of object-based programming languages, along with inheritance, hence an object-oriented language is defined 'OBJECTLY' by:

Object-oriented language = Object-based features + Inheritance + Polymorphism

Object-oriented programming languages for projects of any size use *modules* to represent the physical building blocks of these languages. [A module here in OOP is a logical grouping of related declarations, such as objects or procedures, and replaces the traditional concept of (modules in) *subprograms* that existed in structured (3rd generation) programming language].

Clear understanding of classes and objects are essential for ***learning*** object-oriented development. The concepts of classes and objects help in the understanding of object model and realizing its importance in solving complex problems. *Object model is defined by means of classes and objects*. The development of programs using object model is known as object-oriented development [OOD].

Object-oriented technology is built upon *object models*. So, we see an ***object*** as anything having crisply defined conceptual boundaries. Stationery, vehicle, staff, patient, staff, customer, equipment, etc., are examples of objects. But the entities that do not have crisply defined boundaries are not objects. Beauty, river, sky, etc., are not objects. Also, ***model*** is the description of a **specific view of a real-world** *problem domain* showing those aspects, which are considered to be important to the user of the problem domain.

Object-oriented programming language ***directly influences the way in which we view the world***, addresses the solution closer to the problem domain. To learn object-oriented programming concepts, it is very important to view the problem from the ***user's perspective*** and *model the solution using object* model.

4.2 Conventional Principles: Procedural and Structured Paradigms

The conventional [Procedural and Structured] programming follows the following principles:

≈ operator-operand concept
≈ function abstraction
≈ separation of data and functions.

The development of the ***algorithm*** is given prime importance in this conventional programming. The importance of data is not considered and hence, sometimes critical data having global access may result in miserable output. The abstraction followed is *function abstraction* and *not data abstraction*. Data and functionalities are considered as two separate parts.

Conversely, in the natural way of solving real-world problems, the responsibility is delegated to an agent. The solution is ***proposed*** in OOP instead of developing an algorithm as in the conventional. The problem is solved by having a number of agents (interfaces). The interface part is the user's viewpoint, and hence the solution is not closer to the coding of the program. The real-world problem is solved using a *responsibility-driven* approach.

In OOP approach, the relationship between the user and the programmer is emphasized. Here, solution is problem domain-specific, while in structured programming, the relationship between programmer and program is given prime importance as it *depend on the solution domain and not on the problem domain*.

O-O Program, relationship between the user and the program → emphasized [problem domain]
Structured, relationship between programmer and the program → emphasized [solution domain]

The data is not given importance regarding access permission in structured.

The possibility of reuse of software modules is minimized in structured programming, but is at the heart of OOP.

Structured programming starts with high-level descriptions of the problem representing global functionality. It successively refines the global functionality by decomposing it into subprograms using lower level descriptions, always maintaining correctness at each level. At each step, either a control or a data structure is refined. Thus the top-down approach is followed in structured programming. This is a ***fairly*** successful approach because it will cause problems *only* when there is a revision of design phase. Such revisions may result in *massive changes in the program*.

In the structured programming approach, functions are defined according to the algorithm to solve the problem. Here, function abstractions are concentrated. A function is applied to some data to perform the actions on data. This approach may be called a *data-driven* approach, which involves ***operator/operand*** concept. It depends on the *solution* domain because the algorithm (solution) is closer to the coding of the program. The relationship between the programmer and the program is emphasized in the data-driven approach. The solution is solution-domain specific.

4.3 Discussions

1) Structured Programming Features

Structured programming approach supports the following features:

i. Each procedure has its own local data and algorithm.
ii. Each procedure is independent of other procedures.
iii. Parameter-passing mechanisms are evolved.
iv. It is possible to create user-defined data types.
v. A rich set of control structures is introduced.
vi. Scope and visibility of data are introduced.
vii. Nesting of subprograms is supported.
viii. Procedural abstractions or function abstractions are achieved, yielding abstract operations.
ix. Subprograms are the basic physical building blocks supporting modular programming.

2) Object Oriented Programming Features

The following are important features in object-oriented programming and design:

i. Improvement over the structured programming paradigm.
ii. Emphasis on data rather than algorithms.
iii. Procedural abstraction is complemented by data abstraction.
iv. Data and associated operations are unified, grouping objects with common attributes, operations, and semantics.

These are in line with the foundational theories and features in which OOP is built on as in literature review. Note these features facts;

Programs are designed ***around the data*** on which it is being operated, than ***the operations themselves***. ***Decomposition*** is a mechanism in software maintenance, rather than being ***algorithmic*** as in structured paradigm. Decomposition is data-centric. Data-centric is key in OOP. Recall, to solve a complex problem using the top-down approach, first the complex problem is decomposed into smaller problems. Further, these smaller problems are decomposed and finally a collection of small problems are left out. Each problem is solved one at a time.

4.4 Comparison Between OOP & Structural Programming [Tabularized]

Understanding basic differences between structured programming and OOP concepts is shown in Table 1.

Table 1. Difference between Structured and OO Programming

SN	Structured Programming	Object-Oriented Programming
1	Top-down approach is followed	Bottom-up approach is followed.
2	Focus is on algorithm and control flow	Focus is on object model.
3	Program is divided into a number of submodules, or functions, or procedures.	Program is organized by having a number of classes and objects.
4	Functions are independent of each other	Each class is related in a hierarchical manner.
5	No designated receiver in the function call.	There is a designated receiver for each message passing.
6	Views data and functions as 2 separate entities.	Views data and function as a single entity.
7	Maintenance is costly.	Maintenance is relatively cheaper.
8	Software reuse is not possible.	Helps in software reuse
9	Function call is used	Message passing is used.
10	Function abstraction is used.	Data abstraction is used.
11	Algorithm is given importance	Data is given importance.
12	Solution is solution-domain specific.	Solution is problem-domain specific.
13	No encapsulation. Data & functions are separate	Encapsulation packages code and data altogether. Data and functionalities are put together in a single entity.
14	Relationship between programmer and program is emphasized.	Relationship between programmer and user is emphasized.
15	Data-driven technique is used.	Driven by delegation of responsibilities.

4.5 Requirements of Using OOP Approach

The method of solving complex problems using Object Oriented Program approach requires:

i. Change in mindset of programmers, who are familiar with structured programming.
ii. Closer interaction between program developers and end-users.
iii. Much concentration on requirement, analysis, and design.
iv. More attention for system development than just programming.
v. Intensive testing procedures.

4.6 Advantages & Disadvantages of OOP

1) Strengths/Advantages Of Object-Oriented Programming

The following are the advantages or strengths of software developed using OOP:

i. Software reuse is enhanced.
ii. Software maintenance cost can be reduced.
iii. Data access is restricted providing better data security.
iv. Software is easily developed for complex problems.
v. Software may be developed meeting the requirements on time, on the estimated budget.
vi. Software has improved performance.
vii. Software quality is improved.
viii. Class hierarchies are helpful in the design process allowing increased extensibility.
ix. Modularity is achieved.
x. Data abstraction is possible.

2) Weaknesses/ Limitations Of Object-Oriented Programming

The following are the limitations or weaknesses of software developed using OOP:

i. The benefits of OOP may be realized after a long period.
ii. Requires intensive testing procedures.
iii. Solving a problem using OOP approach consumes more time than the time taken by structured programming approach.

V. CONCLUSION

OO thinking has come to stay in a journey of over two decades; but it is still evolving. New models are been discovered. We cannot predict the future, but going by trends we know that the future of programming will be unpredictable. Why? The programming thinking that uses *object* and *models* cannot be boxed [concluded], because any thought can be modelled and any crispy thing can be an object.

REFERENCES

[1] Wikipedia, the free encyclopedia en.wikipedia.org/wiki/Procedural_programming
[2] TutorialsPoint www.tutorialspoint.com/cplusplus/cpp_object_oriented.htm
[3] Defi searchsoa.techtarget.com/definition/object-oriented-programming
[4] A Webopedia Definition www.webopedia.com/TERM/P/procedure.html
[5] Introduction-to-object-oriented-programming http://ee402.eeng.dcu.ie/introduction/chapter-1
[6] K. Cunz, , Software Development and Object Oriented Paradigms, *Object Oriented Programming with Java*, 2012
[7] Edsger Dijkstra, Notes On Structured Programming, p.6
[8] C. Bohm, and G. Jacopini,., Flow Diagrams, Touring Machines and Languages with only tywo formation rules, CACM 9(5), 1966
[9] Peter Wegner. Concepts and Paradigms of Object-Oriented Programming, Expansion of Oct 4 OOPSLA-89 Keynote Talk, Brown University, 1989

