Evaluating Program Design Methodologies: 
A Qualitative Survey Approach

P. O. Asagba
Department of Computer Science, 
University of Port Harcourt 
Port Harcourt, Rivers State, Nigeria
pasagba@yahoo.com

ABSTRACT
Program design is just an aspect of software development life cycle. Design is seen as the foundation for coding, debugging and testing, maintenance activities, and it also portrays how the final product will meet the requirement. In this paper, I formulated a data encryption problem and its algorithm, and I represented the algorithm using Flowchart, Data Flow Diagram (DFD), Trees, Structure Chart, and Hierarchy Diagram (H-Diagram) design methods. Questionnaire and these design methods (to compute data encryption problem) were administered to Computer Science experts only to ascertain program design quality from which analysis were made on the measures (text attributes / intuitive attributes) for each of the design method, using the program quality rating as Excellent = 5, Very Good = 4, Good = 3, Fair = 2, and Poor = 1). The average of the program quality rating of each measure becomes the metric. The result obtained from my research shows that the quality of the program design depends on the amount of the characteristics of evaluation measures desired and strict top-down sequencing - structured design and programming. The mean total of the different design methods examined, Flowchart has the lowest mean value of 31.85, while Structure Chart has the highest mean value of 38.4, with respect to text attributes. Again, Flowchart has the lowest mean value of 45.05, while Structure Chart has the highest mean value of 58.10, with respect to intuitive attributes as reported in the bar charts, which clearly shows the dependency nature relating intuitive attributes and text attributes.

Keywords: Software design, Design methods, Evaluation methods, Program quality, Structured design

African Journal of Computing & ICT Reference Format:

1. INTRODUCTION
The design of complex systems involves many people exercising multiple skills and carrying out a variety of activities. If we surveyed a large sample of the software that are available for Computer Systems and asked, "how was it designed?" We would find many methodologies used in the software design process [1]. A number of methodologies are currently in use and new ones are continually being developed.

Developing a high quality software product in an economical way is one of the fundamental goals of any software engineering activity [16]. As software systems have grown more sophisticated and complex, software developers have sought new methods for their development. Software engineering is a response to that need [2]. Software engineering has emerged as a field to help manage the process of developing today's complex software. The field of software engineering encompasses all the activities involved in the solution of problems in the development of computer systems - with the goal of carrying out successful software development projects, to produce an acceptable product.

Correct prediction of the software quality or maintain a software system is one of the most critical activities in managing software project [17]. Program design is just an aspect of software development life cycle. Looking at the framework of the system development life cycle, computer programming form a part of implementation steps in the building of a new system. Programs are designed, coded, and integrated with other system resources to produce a workable computer information system.

Design activity is the first step in moving from problem domain to solution domain. The goal of the design is to produce the model of the system which can be later used to build up the system [12]. Different methods and tools are used to perform the different software development activities [9]. Software development life cycle involves series of phases: Requirement analysis and definition, specification, design, implementation, testing and debugging, and maintenance. My interest in this paper is on program structures.
1.1 Program Structures
The program design activity determines the structure of the program. The structure of a program is the way, the computer processing function (identified during problem analysis) are to be organized and presented to program [3]. An effective method of structuring is to focus on the three main functional components that are virtually obtainable in all programs. The following, outlined the three main program sections and the types of processing activities that occur in each case:

a) Initiation activities which occur once only at the beginning of processing.
b) Main processing activities which occur repeatedly, usually once for each record to be processed.
c) Termination activities occur only at the end of processing.

This structure represents a computer problem-solving model that can be adapted for any program. It gives a framework for organizing processing functions and how they are relayed to the computer as an application program.

1.2 Logical Control Structures
Processing modules as found in a program are related in some special ways to one another. The order of processing activities are controlled by three logical control structures, these are: sequence, repetition, and selection.

1.3. Statement of The Problem
Perhaps the biggest problem facing programming today is the extreme difficulty and cost of creating and maintaining large programming systems. An overused term, modularity is often given as the answer to this problem. Project managers get more nervous in the design stages of a project than in the coding and testing stages because program design is not normally a visible process (managers can see and touch code and test cases) because design is not normally testable (managers can see the results of testing a pieces of code) [6]. The use of structured design can help in making the design process more visible. Design is the foundation of coding. It is a determining factor on the final quality of the product. Selecting / using efficient and effective program design in solving a problem will reduce costs, save coding time, increase product delivery, and improve quality of product.

Experience has shown that, in the development of large software system most errors occur in the specification and design stages [11]. Experience has shown, that especially for large systems, perfection of any quality attribute is impossible, and that the cost attaining successive increment of quality increase as the system approaches perfection [7].

2. METHODOLOGY
Methodology is an open set of procedure which provided the means for solving problems [4]. Software methodology is a pre-defined sequence of events that must be executed, followed or carried out in order to produce a well structured and robust software product that meets user’s requirement and produce good scalable tendencies [18]. A lot of methodologies abound. Design techniques only provide a guide for working through a design procedure. They cannot ensure the adequacy of a design. Only the designer can ensure the adequacy by having in mind the larger issues as the design is worked through.

Every software development methodology acts as a basis for applying specific approaches to development and maintaining software. Several software development approaches have been used since the origin of information technology [13]. For modern, professional computing, methodical design is an essential part of everyone’s approach to implementation [5]. This calls for the need for the adoption of a methodical approach. The main sources of software analysis and design methodologies are the fields of human computer interaction and software engineering. Human-computer methodologies generally focus on the user’s viewpoint of the software system, while software engineering methodologies focus on the designer’s view of the software system [1]. In this paper, my emphasis is geared towards the software engineering methodologies. This approach, provide a strict top-down sequencing in an effort to control completeness and consistency. This is a step to structure design and programming. In this paper, I adopted the qualitative survey approach.

3. PROGRAM DESIGN METHODOLOGIES
I examined the following program design methodologies: flowcharts, data flow diagram, structure charts, trees, and hierarchy diagram (H Diagram).

3.1 Flowcharts
A flowchart is another form in which the algorithm for a program can be described. It is a pictorial representation structure which implies that the program is coded and tested as a whole. Flowchart is a traditional way of representing a program. Although, flowcharts work adequately for designing small, simple programs with few processing operations, they do not represent the best method when the problem is complex. It does not conform to any programming language. Certain symbols are used to describe flow of the logical structure. These symbols are not considered in this paper.
3.2 Data Flow Diagram
Data-flow-based design methodologies have been proposed for business processing systems and for other kinds of systems [7]. Data flow techniques establishes the data flow diagram, which reviews the important data blocks, data transformations, and movement of data of a design method. Data flow diagrams focus on the data flows and their processing. As valuable as they are, for representing system, they are not suitable for portraying a system down to the level of detail such that programs can be written from them. A hierarchy is somewhat forced on the data flow diagram by expanding process blocks into other diagrams. Data flow is different from the flow of control of a program; it does not convey program sequence information of any manner.

3.3 Structure Charts
Structure chart form a tree with the root at the apex. Hierarch is more natural in a tree structure compared to data flow diagrams. Also, tree structures focus more on control. Large modules are expanded into smaller modules as one descends down the chart. They are used basically to analyze how information is passed by the module cells. Usually, the circled arrow is drawn parallel to the arrow connecting the rectangles in the chart.

3.4 Trees
The way trees are built, using a top-down principle helps to encourage sound construction from the general to the specific [2]. It has a broad definition of what the program does beginning from the left. Program parts are broken down into more details as the branches of the tree moves to the right. Trees could be represented from left to right on the page to enable one write the various descriptions horizontally - which implies that the root appear on the left side of the page and is considered as the top of the tree, with a brief description of what the program does. The tree comprised horizontal lines called branches and vertical lines called limbs. Subtrees are subprograms within the program. A branch could produce either a leaf or a limb. A leaf represents the end of a branch where no further branching occurs. Limbs have branches off them to the right. This structure is a complete description of the body of the program. The program can be compiled and run from the graphic outlay and the data description.

3.0.5 Hierarchy Diagram (H-diagram)
The H-Diagram shows in block form, the relationship among major functions, minor functions and modules in a program. The boxes in H-Diagram defines module. A module represents a program function. This shows a specific task that will be done by the computer, with each box containing the name of the function and a number to reflect its hierarchical relationship with other modules. H-diagram also contains notations indicating the program’s logical structure, or the order in which the processing functions will be performed.

This approach is based on top-to-bottom, left-to-right manner of the modules. Thus, the initiation activities, starts first; following by the main processing activities, following by the terminal activities. Meanwhile, the detailed processing modules will be carried out in the general left-to-right order. Each module will be activated once, unless specified otherwise. The modules subordinate to main processing module will be activated more than once. The notation REPEAT indicates that this set of processing function has to be repeated until there are no more input records to process. The SELECT notation indicates that only two or more available module will be activated. During processing, the computer is faced with the selection of one or the order of these processing alternatives.

4. EVALUATION METHODS
There are three distinct types of evaluation methods: quantitative, qualitative, and hybrid.

(a) Quantitative Evaluation Methods
Quantitative evaluation focuses on the assumption that certain measurable property (or properties) of your software product or process that may require changes can be identified by the application of the evaluation method/tools of interest. This type of evaluation requires objective assessment. Quantitative evaluations are organized as follows: formal experiment, case studies, and surveys.

(b) Qualitative Evaluation Methods
Feature analysis is based on identifying the requirements that users have for a particular tastes/activities and mapping those requirements to features that a method/tool aimed at supporting that tastes/activity [8]. Feature analysis may be used to describe a qualitative evaluation. The users in this context, refers to the software managers, developers or maintainers. Qualitative or feature analysis requires a subjective assessment of the relative benefits of different features and how well a feature is implemented. Feature analysis can be performed by a single person who recognizes the requirements, maps them feature by feature and assesses the extent which another method/tool will support these features by trying out the method/tool or by consulting literature. This method is best used when screening a large number of method and tools. Qualitative evaluation are also organized as follows; formal experiment, case study, and survey

(c) Hybrid Evaluation Method
Hybrid evaluation method is an evaluation method/tool that combined the feature of both quantitative and qualitative methods. Two types of hybrid evaluation method include: Qualitative effect analysis and benchmarking.
4.1 Qualitative Surveys
This research focuses on qualitative approach. In this paper, I used the qualitative survey to conduct evaluation on program design methods. Qualitative survey involves a qualitative evaluation performed by people who have the inking of the methods/tools of interest (but the evaluation is organized as a survey).

5. PROGRAM CONSTRUCTION

Formulation of Problem and Algorithms
I presented a data encryption problem to handle encoding and decoding of texts and its algorithm as well as the various methodologies.

5.1 Algorithm
The algorithm of the data encryption problem is shown in Figure 1.

1. Print program title
2. Repeat main menu program
   a) Print main menu title
   b) Input any number (1-6) from main menu list
   c) Display the current method title
3. Submenu program
   i) Print submenu title
   ii) Read one character (E,G,D,Q) from submenu list
   iii) If character equals ‘E’
       Create file name and save it
       Input encipher key (one character)
       Input text to encipher
       Encode text and store
   Else if character = ‘G’
       Read (F) to retrieve encoded text
       Search for and path and assign to FF such that FF = F search (F, path)
       If FF is equal to null string
       Display error message
       Otherwise
       Display encoded text
   Else if character =’D’
       Read (F) to retrieve encoded text
       Search for F and path and assign to FF such that FF=F search (F, path)
       If FF is equal to null string
       Display error message
       Otherwise
       Read decipher key (one character)
       Decode text
       Display deciphered text
   Until character =’Q’
   iv) Display the end of the current method until number =’6’
4. Stop

Fig. 1: Algorithm for data encryption
5.1 Flowchart
The flowchart for the data encryption problem is shown in Figure 2.

Fig. 2 Flowchart for data encryption
Print 'Please Save the file you are about to create'.

Read 'Encipher key (one character)', k

Read 'Text to Encipher', t

Encode Text

Print 'Which Text Do You Want To Retrieve?'

Print 'Please Include Path e.g. C:\Text'

Read 'Saved File ', F

FF: FSearch (F, Path)

Is FF = ' '?

Yes

Print 'You Have Entered a Wrong Choice'

No

Print 'Text After Encoding', F

Read 'Saved File ', F

FF: FSearch (F, Path)

Is FF = ' '?

Yes

Print 'You Have Entered a Wrong Choice'

No

Print 'Text After Decoding', F

Key:=Readkey

Print 'Text After Decoding', F

Key:=Readkey

Reset (Info)

EE
5.3 Data Flow Diagram
The data flow diagram for the data encryption problem is shown in Figure 3.
5.4 H Diagram
The H diagram for the data encryption problem is shown in Figure 4.

Fig 4: H-Diagram for data encryption
5.5 Trees
The Tree for the data encryption problem is shown in Figure 5.

![Tree Structure for Data Encryption](image-url)

**Fig. 5** Tree structure for data encryption
Clrscr
Readln(F)
Writeln('Please Include Path e.g C:\Text')
FF: FSearch(F, Path)

IF FF = ''
Write('You Have Entered a Wrong Choice')
Continue
Writeln('Press Enter Key To Continue')
Readln
ELSE
DisplayText
Clrscr
Writeln('Text After Encoding')
Writeln(F)
Continue
Writeln('Press Enter Key To Continue')
Readln

Clrscr
Readln(F)
Writeln('Please Include Path e.g C:\Text')

IF FF = ''
Write('You Have Entered a Wrong Choice')
Continue
Writeln('Press Enter Key To Continue')
Readln
ELSE
Decode
Clrscr
Write('Key = Readkey')
DisplayText
Writeln('Press Enter Key To Continue')
Readln
Assign (Info, F)
Reset (Info)
Continue
Writeln('Press Any Key To Continue')
Readln
5.0.6 Structure Chart
The structure chart for the data encryption problem is shown in Figure 6.

Fig. 6: Structure chart for data encryption
6. PROGRAM QUALITY
When the expression “Software Quality” is used, we usually think in terms of an excellent software product that fulfills our expectations. These expectations are based on the intended use [14]. Quality of software is increasingly important and testing related issues are becoming crucial for software [15]. It is our hope and desire that our program works correctly and efficiently. It is rather unfortunately that even, a correct and efficient program, delivered on time and within budget, may be unsatisfactory if it is poorly designed. In a study of software quality, Boehm and other [10] concluded that software utility depends on seven intuitive components:

- Portability
- Reliability
- Efficiency
- Human Engineering
- Testability
- Understandability
- Modifiability

These intuitive quality components cannot be directly measured by the examination of a program test. However, they depend strongly on some intuitive and text attributes shown in Figure 7.

*Fig. 7: Dependency structure relating intuitive attributes and text attributes
Source: (Boehm, 1978)*
6.1 Result

My evaluation was based on the data encryption problem and algorithm that was presented with reference to the respective design methods. However, in this research work, I have employed the use of the qualitative survey approach to evaluate program design methodologies. Questionnaire and the various design methods presented earlier (to compute data encryption problem) were administered to computer science experts only to ascertain program design quality from which analysis were made on the measures/text attributes for each of the design method (using the program quality rating as Excellent = 5, Very Good = 4, Good = 3, Fair = 2 and Poor = 1). The average of the program quality rating of each measure becomes the metric. Table 1 shows program text attributes measures and designs. Figure 8 shows the bar chart of mean text attributes rating against design.

Table 1: Program Text Attributes and Design

<table>
<thead>
<tr>
<th>Text Attributes</th>
<th>Flowchart</th>
<th>Trees</th>
<th>Data Flow Diagram</th>
<th>Structure Chart</th>
<th>H Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structuredness</td>
<td>2.0</td>
<td>3.6</td>
<td>3.6</td>
<td>4.4</td>
<td>3.2</td>
</tr>
<tr>
<td>Communicativeness</td>
<td>3.4</td>
<td>3.4</td>
<td>3.6</td>
<td>3.8</td>
<td>3.8</td>
</tr>
<tr>
<td>Self-Descriptiveness</td>
<td>3.8</td>
<td>3.8</td>
<td>4.0</td>
<td>3.8</td>
<td>3.4</td>
</tr>
<tr>
<td>Accessibility</td>
<td>2.6</td>
<td>3.75</td>
<td>3.75</td>
<td>3.5</td>
<td>3.25</td>
</tr>
<tr>
<td>Legibility</td>
<td>3.6</td>
<td>3.8</td>
<td>3.6</td>
<td>3.8</td>
<td>3.6</td>
</tr>
<tr>
<td>Augmentability</td>
<td>2.8</td>
<td>3.8</td>
<td>3.2</td>
<td>4.2</td>
<td>3.0</td>
</tr>
<tr>
<td>Conciseness</td>
<td>2.8</td>
<td>3.6</td>
<td>3.2</td>
<td>3.6</td>
<td>3.4</td>
</tr>
</tbody>
</table>

Fig.8: The bar chart of mean text attributes rating against design
Table 2 shows intuitive attributes measures and designs. Figure 9 shows the bar chart of mean intuitive attributes rating against design.

Table 2: Program Intuitive Attributes and Design

<table>
<thead>
<tr>
<th>Intuitive Attributes</th>
<th>Flowchart</th>
<th>Trees</th>
<th>Data Flow Diagram</th>
<th>Structure Chart</th>
<th>H Diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>Human Engineering</td>
<td>6.0</td>
<td>7.15</td>
<td>7.35</td>
<td>7.3</td>
<td>7.05</td>
</tr>
<tr>
<td>Testability</td>
<td>8.4</td>
<td>11.15</td>
<td>11.35</td>
<td>11.7</td>
<td>9.85</td>
</tr>
<tr>
<td>Understandability</td>
<td>15.0</td>
<td>18.4</td>
<td>17.6</td>
<td>19.2</td>
<td>17.0</td>
</tr>
</tbody>
</table>

Fig.9: bar chart of mean intuitive attributes rating against design.
6.2 Discussion of Result

In Table 1, mean total of the different design methods examined are specified. Flowchart has the lowest mean value of 31.85 while structure chart has the highest mean value of 38.4. H-Diagram is ranked next after flowchart with mean value of 51.20, followed by data flow diagram with mean value of 54.05, followed by trees with mean value of 54.75. In Figure 8, the bar chart clearly portrays the various means total.

In Table 2, I examined some of the intuitive attributes of the dependency structure of Figure 7 with regard to program design methods. Again, flowchart has the lowest mean value of 45.05 while structure chart has the highest mean value of 58.10. H-Diagram, Data flow diagram, and trees maintained their earlier positions. In Figure 9, the bar chart clearly shows the various means total. This clearly shows the dependency nature relating intuitive attributes and text attributes.

From the evaluation process presented so far, I could deduce that the quality of the program design depends on the amount of the characteristics of evaluation measures desired and strict top-down sequencing - structured design and programming. All designs are valuable in their own respect, but structuredness, have an immense edge over the unstructured one. For example, flowchart design is not structured. This perhaps explains while flowchart design method consistently appeared to be at the bottom / least compared to other design methodologies examined. The use of structured design helps make the design process more visible. From my research, it is obvious that modularity has a positive effect on program quality. Performance is increasingly becoming an issue of design. Good design is no longer an aspiration, it is a necessity. Good planning can reduce costs. The foundation of coding is design.

7. CONCLUSION

Program design is just an aspect of software development life cycle. Design is seen as the foundation for coding, debugging and testing, maintenance activities, and it also portrays how the final product will meet the requirement. Methodologies are used to represent designs. There are evaluation methods for selecting an appropriate evaluation method. The quality of program design methodologies depends on the amount of characteristics of evaluation measures desired and a strict top-down sequencing - structured design and programming. Structured design has an edge over the unstructured ones because modularity has a positive effect on program quality. Structured design and programming does not portray a particular design methodologies to be the best, however, the trend depends on the aspect of quality being examined. Often, some qualities must be traded off against other qualities to obtain a program that is versatile and maintainable.

REFERENCES


