Object-oriented Paradigm as Evolution of Structured approach in Programming

Galina Istatkova

Abstract: *A viewpoint is given about structured programming (SP) as a fundamental approach for complex software system development independent from procedural paradigm. The multi-interface program module form – class, entered in object-oriented paradigm (OOP) is considered as a long evolutionary process in informatics aiming to improve SP efficiency. A new form of structured algorithm is proposed as a collection of functional threads corresponding to universal function $F_{generic}$. Four possible implementations of $F_{generic}$ in procedural and object-oriented paradigms are given and compared. Functional library with hidden global data structure is considered as a procedural form of class entered in software engineering to improve SP efficiency for complex system development using procedural programming languages. The second software crisis in procedural SP is explained by defining the task to find all functional threads in complex system needed for testing as NP-hard problem. Statistic data about extremely reduce of programmer's productivity in large software projects are discussed and a function extrapolated statistic data is used for the forecast of programmer's productivity as a function of system size and complexity. Using the object-oriented interpretations of $F_{generic}$ some of the concepts of OOP such as static and dynamic polymorphism are discussed.*

Key words: *Structured programming, algebraic model of algorithms, procedural and object-oriented paradigm, NP-hard problem.*

INTRODUCTION

Majors in computer science accept SP as a subset of procedural paradigm. In this report SP is considered as a fundamental approach independent from procedural paradigm and intended for complex software system design and development. In brief SP [1, 2] may be defined as a methodology including two basic principles:

1. The hierarchical module decomposition of system (top-down, bottom-up or the combination of two approaches);
2. Using only structured algorithms in program modules at all layers.

The first principle provides the minimal connection between modules needed for parallel work of programmers which results in significant development time reduction. The second one ensures the possibility of static verification (at least as the theoretical possibility) and dynamic test of algorithms. The main goal to apply these principles for design and development the complex software system is to create reliable systems for short time. In fact, the hierarchical module decomposition could be considered as a profound method of complex system design which was mapped into informatics from the system control theory by Dijkstra, Wirth, and a lot of others computer scientists.

The intent of this report is to prove that object-oriented programming developed as the dominant programming methodology during the mid-1980s, isn't a revolution in informatics but rather a long evolutionary process aiming to improve the SP efficiency for complex system development. This process is described at the view point of program module form transformation – from the single-interface procedure to multi-interface class in OOP.

FIRST SOFTWARE CRISIS

Italian mathematicians Boehm and Jacopini proved that every algorithm can be transformed into structured form - a sequence of control constructs C_i with one input and one output - $\mathbf{C_1} \rightarrow \mathbf{C_2} \rightarrow \ldots \rightarrow \mathbf{C_n}$. In years when structured theorem was published the programming language was widely used for scientific applications was Fortran and applying **go to** statement was a programming style. Using **go to** statement results very easily in producing inconsistent, incomplete and generally unmaintainable so-called spaghetti code. In a large program with a lot of **go to** statements finding the number of paths from the first node to the final one in algorithm graph model are needed for testing

should be considered as NP-hard problem because it is similar to the shortest path problem in graph theory[3]. It's possible to say that this kind of programming style was the main reason for the first software crisis. In brief it might be described as the problem to develop complex software projects. The difficulty to test programs developed with spaghetti style was the reason for total failure of many big software projects including system catastrophes in NASA caused due to programmer's errors. Switching to SP was the only possible way to overcome the first software crisis in procedural programming and had been understood by outstanding computer scientists as well as by managers in software industry. IBM researcher Mills and scientist Niklaus Wirth developed SP concepts for practical use and tested them in a 1969 project to automate the New York Times morgue index. The engineering and management success of this project led to the spread of structured programming through IBM and the rest of computer industry [2].

FUNCTIONAL MULTI-THREAD MODEL OF STRUCTURED ALGORITHM

Obviously Boehm-Jacopini model is a procedural-oriented one and can be implemented only as a single-interface program module - procedure, subroutine or function. Let's assume the other form of structured algorithm – a collection of parallel functional threads one of which can be selected during execution. This model corresponds to universal function $F_{generic}$. $F_{generic}$ is a mathematical model of universal program that theoretically can include all other programs [5]:

$F_{generic}(F_1, F_2,\ldots F_{n,i}) = F(_i)$, where

F_i is a functional thread $\{ f_1,f_2,\ldots f_k\}$, $1 \le i \le n$.

The main advantage of the functional model of structured algorithm is its independence of program module form. As it is shown on fig.1 $F_{generic}$ can be implemented as a procedure, or as a functional library, or as a class and hierarchy of classes. This property allows SP efficiency to be compared in procedural and object-oriented programming as well as to be used to see evolutionary process of program module form transformation.

Accepting that functional model isn't a traditional form of algorithm one could ask whether is it possible to convert procedural model of structured algorithm to $F_{generic}$?

The answer to this question is given by author in earlier report [6], where using algebraic model of structured algorithms in Boolean algebra of algorithms has been proved that procedural form $\mathbf{C_1 \rightarrow C_2 \rightarrow \ldots \rightarrow C_n}$ can be transformed to $F_{generic}$:

$\mathbf{F_{generic} = P_1F_1(X_1) \vee P_2F_2(X_2) \vee \ldots \vee P_nF_n(X_n)}$, where

P_i is the logical expression defined the type X_i - the set of values needed for correct execution of $F_{i..}$

WHY DOES SP LOSE ITS EFFICIENCY IN PROCEDURAL PROGRAMMING?

Let us to use the number of functional threads N as the quantity of functional complexity of program system. The amount L of layers in hierarchical module decomposition is the other way to estimate the system's complexity and the value of N might be considered as function F(L). Let's to accept that the lowest layer L_0 of the system has the minimal level of complexity N =1 and on the next upper layer the programs have algorithms include m if statements with functional operator f = $F_{generic}$. In this case the number of the functional threads on the layer L_j ($1 < L_j < L$)can be estimated by the following formula:

$$[1] \qquad \mathbf{N(K, L_j) = \sum_{i=1}^{2^m} K(L_{j-1})_i^m \approx 2^m K^m}$$

The value of K in [1] is the average number of threads in $\mathbf{f_i}$ the layer L_{j-1}. It was accepted too that all possible threads are logically selected or in other words the maximal possible value is used (the minimal number of threads is 2 and correspond to case when all logical expressions in conditional operators are equivalent). For L_j=2 the value of K is 1 as was mentioned above and $N = 2^m$. The problem - extremely increasing of number of threads, will start when program with the number of functional threads K >> 1 should be

embedded as a functional operator $\mathbf{f_i}$ in control construction such as a conditional operator or switch in the algorithms on the next top layer. For example, if K > 100 then N≈ $2^m 100^m$ and the threads defining problem becomes NP-hard as well as the testing task. This process is known as the combinatory explosion and it's important to notice that this problem can't be avoid when the complexity of program system expressed by the number of level L and the number of functional threads in a procedure will increase. As the result programmer's productivity in large and complex projects decreases significantly.

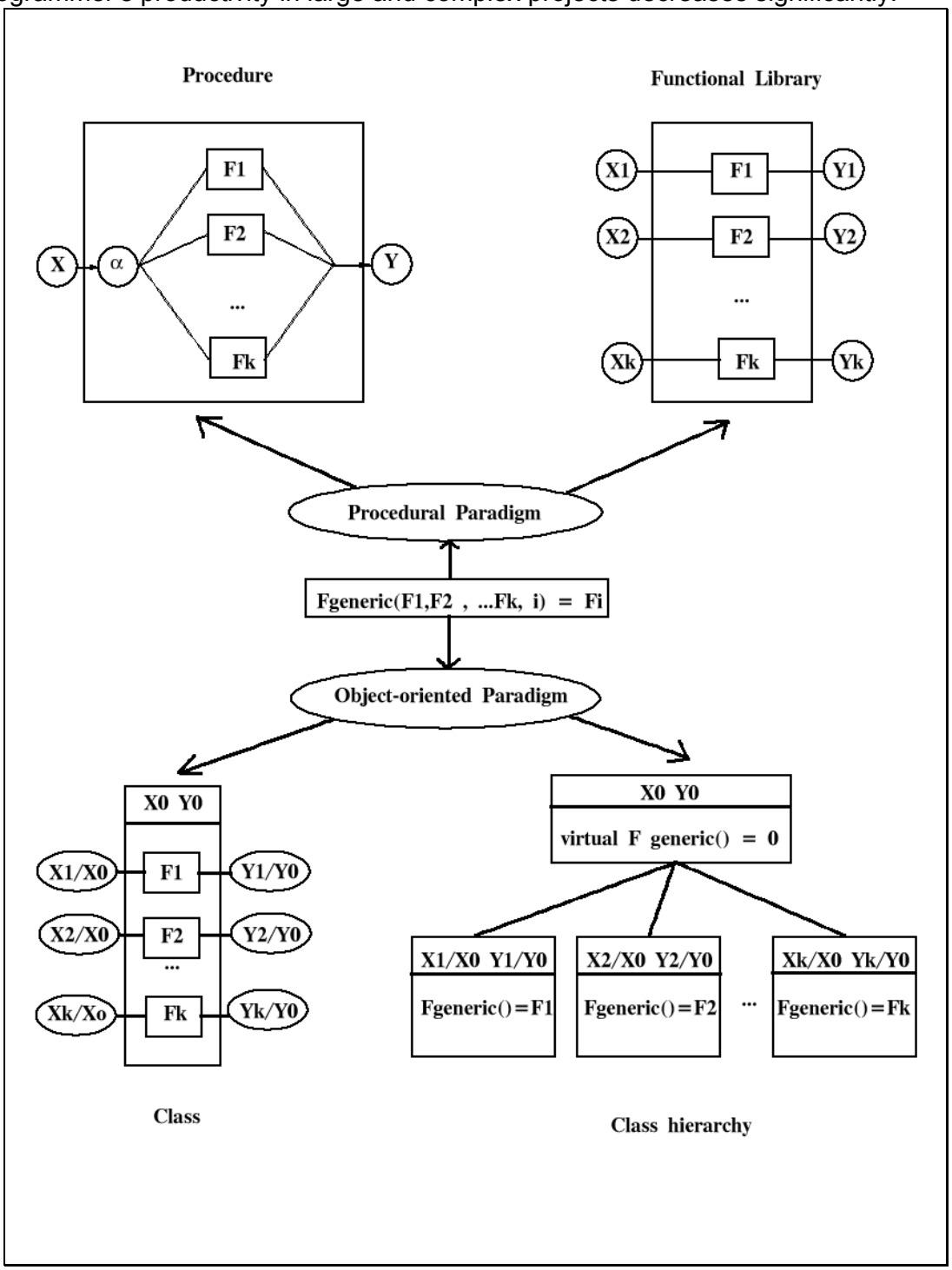

Fig.1 Structured algorithm interpretation in procedural and object-oriented programming

Statistic data about programmer's productivity as the function of software system complexity given by James Martin [5] are shown in table 1.

System size-number of program's row	Productivity- row's number per year
512.000	800
64.000	1300
16.000	2000
2.000	4000
500	8000
<500	15.000

Table1. Programmer's productivity

James Martin used as the value of the system complexity the number of program code rows. These statistic data can be extrapolated with function $f(x) = ax^{-b}$, where x is the system's size, a = 117089.79, b = 0.398. The graphic of this function and statistic data are given on fig.2 – screen shot of MFC Windows program written by the author. The number of program's rows could be obtained very easy for every software system but this coefficient hides the functional complexity – the number of functional threads N. To use formula $N = 2^m K^m$ it's needed to define K=f(system size) for the layer L-1 and m for the upper layer L. To calculate N as the function of program's size let's to accept m=2 and K= size/1000.0 what means that program module with 1000 rows might be defined with minimal functional complexity. In this case $N(size)= 4(size/1000.0)^2$. The graphic of this function is rendered in the same graphic (fig.3) and illustrates that the programmer's productivity reducing can be explained by extremely increasing of number of functional threads. Using the function $f(x)=ax^{-b}$ extrapolated statistic data 5 new values of programmer's productivity for larger program's size were calculated. These values are given in table 2.

System size-number of program's row	Productivity- row's number per year (calculate using ax^{-b})
1.000.000	479
10.000.000	191
100.000.000	76
1000.000.000	30
10.000.000.000	12

Table 2. The forecast for the programmer's productivity

This strong reducing of programmers' productivity in complex and large program system as well as the difficulties to define all threads needed to test system could be considered as the reason for the second crisis in software industry in 1980's. This crisis had done the question was asked by professor Dijkstra :"Whether it's possible at all to develop complex system and what must be done to develop reliable complex systems which can be tested entirely"[2] significantly more actual problem in 80's then in 60's. One thing was obvious – this problem can be resolved using procedural SP because the procedural form of single-interface doesn't already correspond to the functional complexity of program module. This contradiction between old form of program module and new complexity of system can't be resolved in the frame of procedural paradigm. Having analyzed formula 1 one might see that the only way to have the number of functional threads in the range allowing system testing is using $K_L=1$ for functional operators in algorithms at all layers of the system. In procedural paradigm it would be achieved thereby by implementation of $F_{generic}$ as functional library (fig.1) what could be considered as procedural form of class.

FUNCTIONAL LIBRARY AS AN INTERMIDIATE STEP FROM PROCEDURAL SP TO OBJECT-ORIENTED

One of SP recommendation is usage of so-called open data interface what means that global variables should not be used because they connect modules and prevent parallel

work of programmers. Instead procedures should use local variables and take arguments by either value or reference. With system hierarchical complexity increasing the number of embedded structures and union of structure becomes so large that open data interface becomes difficult enough for practical usage and became the source of structural conflict and errors by incorrect setting of arguments values. The main reason for structural conflict in procedural programming is the difficulty to declare data structure for all layers in system's decomposition. For example, if top-down method is used for designing it's necessary to define the structure for the module laid on the upper layer. This structure must include structures and data types for all layers that eventually will be designed. In other words it is needed data interfaces for all modules to be defined at the designing stage. Besides to allow parallel work of programmers it's very important to ensure that data interfaces will not be changed. The necessity of freezing data interfaces is the main reason for potential structural conflict because a programmer can't change interface when the structural error will be discovered. To minimize data interfaces, data structures which are common for more of functions in library had usually been "hidden" in global variables. The functional library with this kind of encapsulation of data could be considered as an intermediate step to class and object in OOP. There were a lot of other questions needed to be resolved and one of them was connected with interaction between modules. How could modules laid on the same layer to call functions of each other? Many new approaches had been developed in OOP started with abstract data types and ended with Frame works (typical example is MFC in Visual C++), Automation, ActiveX controls, COM, DCOM [7]. These module interactions problems can't be explained using $F_{generic}$ but some of OOP concepts can be described using object-oriented interpretation of $F_{generic}$ (fig.1).

OBJECT-ORIENTED INTERPRETATIONS OF $F_{generic}$

The first interpretation of $F_{generic}$ is a class, the second – hierarchy of classes (fig.1). The data X0, Y0 which are common for all functions are encapsulated. Encapsulation allows function's interfaces to be decreased and thereby to overcome the problem of open data interface in procedural SP. It might be said that the most famous new concept in OOP is polymorphism. The concept of polymorphism means that functions can be applied to values of different types. In fact, polymorhism is applied to basic data operations such as +,-,/ etc in the all programming languages. In OOP polymorhism is expanded to be used with programmer's operations – methods or member-functions of classes. There are two types of polymorhism – static and dynamic. Static means that the same name of function will be used for different data types which have been already defined by programmers. For example if all F_i functions in class implementation of $F_{generic}$ (fig.1) act similarly upon different data type (X/X_i) then only one name $F_{generic}$ could be used. Using static polymorphism the system dictionary could be decreased significantly. The idea of dynamic or parametric polymorphism is to define data operations to the unknown data type that will be added to the system in the future. These operations are called pure virtual functions. In COM and DCOM they are known as interfaces. The implementation of as class hierarchy gives the possibility to create classes which inherit the common data and interfaces from base class. Hierarchy of classes is used for developing object-oriented frame works. MFC (Microsoft Foundation Classes) in C++ Visual is typical example [7].

CONCLUSIONS

Of course, OOP can't resolve the decomposition problem of large software system and this problem might be considered as NP-hard because it could be compared with the problem of dividing graph into sub-graphs using such criteria as the minimal connections between sub-graphs [3]. Understanding that every algorithm has functional form could help programmers to resolve two very important practical tasks:

1. To test algorithm entirely trying to define all functional threads and data sets needed for correct executions.
2. To transform procedural implementation of software systems to object-oriented one

using functional model of algorithms.

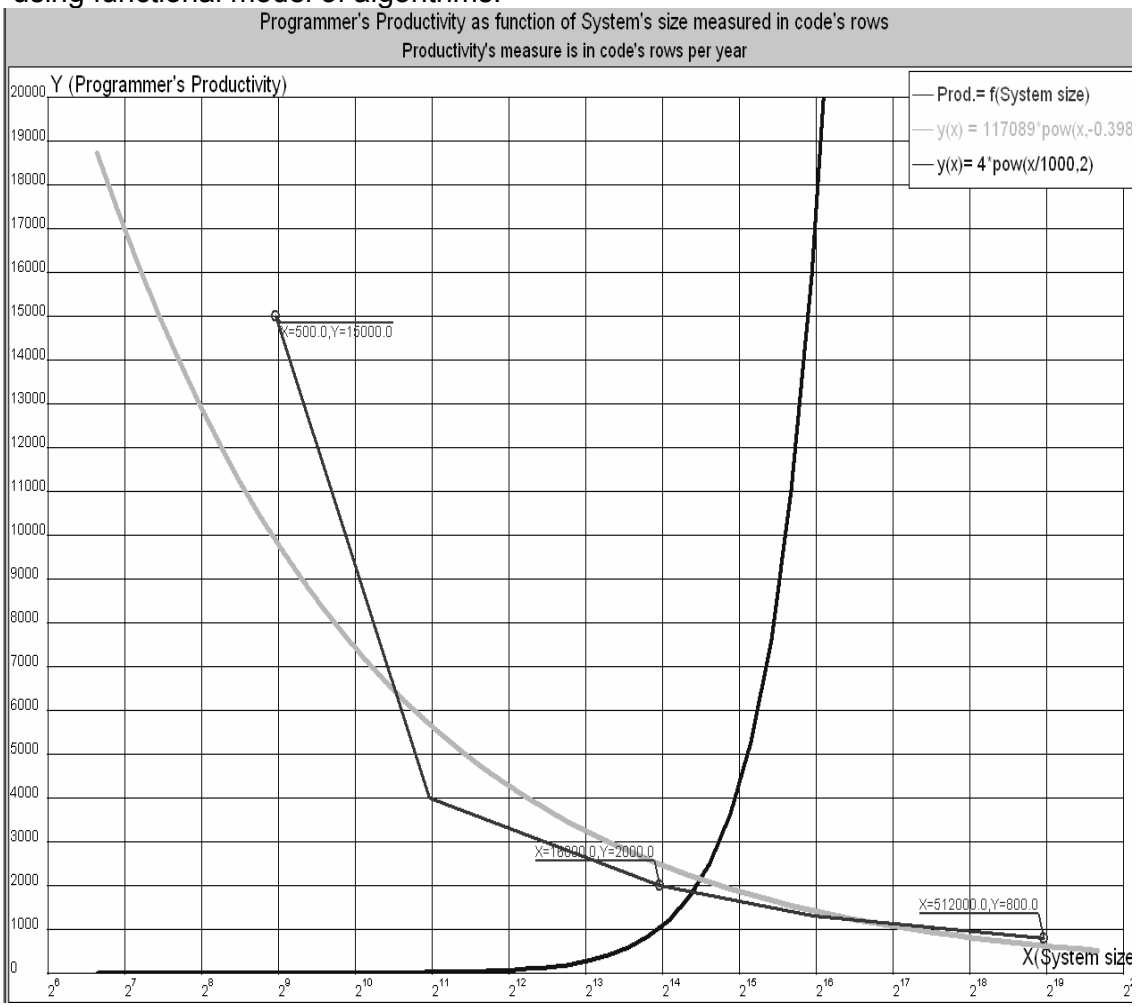

Fig.2 Statistic data about programmers productivity and function extrapolated these data

Is it needed to teach the structured programming nowadays? The answer is yes because OOP is before everything is structural programming and to know how to design and to develop structured algorithms must be studying before programming in C++, Java or PHP.

REFERENCES

[1]Dijkstra Edsger W., Structured Programming, Technological University EINDHOVEN, The Netherlands August 1969.

[2]Dahl O. J., E. W. Dijkstra, and A. R. Hoare. Structured programming. Acad. Pr., 2 edition, 1973.

[3] Garey, Michael, David. Johnson. A Guide to the Theory of NP-Completeness. Bell Laboratories, New Jersey, W. H. Freeman and Company, San Francisco, 1979.

[4] James M., . Forth Generation Languages, Principles. Prentice-Hall, Inc, 1985.

[5] Cutland, N., Computability. An introduction to recursive function theory. Cambridge University Press, Cambridge, 1980.

[6] Istatkova G. Algebra of algorithms in procedural and object-oriented structured programming. Automatica & Informatics, N. 3-4/2001, 56-62

[7] Kruglinski David,Inside Visual ++ 6.0, Microsoft Press, 1999.

ABOUT THE AUTHOR

Engineer, research worker Galina Istatkova , Institute of Computer and Communication Systems, Bulgarian Academy of Science. Phone: (359 2) 76 77 01, GSM – +359 898 89 30 59, E-mail: galina_istatkova@yahoo.com.