SPECIFIC ASPECTS OF FINANCIAL AND ACCOUNTANCY SOFTWARE RELIABILITY

Marian Pompiliu CRISTESCU
PhD, University Professor
"Lucian Blaga" University of Sibiu, Romania

E-mail: mp_cristescu@yahoo.com

Abstract: *The target of the present trend of the software industry is to design and develop more reliable software, even if, in the beginning, this requires larger costs necessary to obtain the level of reliability. It has been found that in the case of software which contain large amounts of components - financial and accounting software are also included here – the actions taken to increase the level of reliability in the operational stage induces a high level of costs. This one is superior to the one that involves obtaining systems of software with an adequate reliability, before releasing them on the market and before using them. Before taking into account the material and financial aspects that involve obtaining the adequate reliability, we must consider the social effects that occur because of the lack of reliability of software. The conclusion is that, if we begin with the idea that a system of accounting software is a fitted and well structured ensemble of different components – from a constructive point of view – which satisfy interconnected needs, the reliability of the entire system depends directly on the reliability of each component.*

Key words: *software reliability; software metrics; object-oriented software; error; fault; failure; financial and accountancy software*

1. Introduction

The main method used in building complex systems is abstractization. A system is built on levels; level B is made out of components from level A. But at the same time, components from level B are used as if they were atoms, independently, to build level C and so on.

An important subject in the theory of reliability is the construction of more reliable software, from components which are more or less reliable. If a system works only when every component is functional, it is impossible to build a complex system because the reliability decreases exponential with the amount of components.

Certain classes of programmes, such as those from air traffic controls and supervision of nuclear power plants, need a high reliability level. In critical programmes, the architects of the systems take into consideration the possibility of failure, which they treat in the software

A system of programmes, from a static point of view, appears as a function f defined through X, with values in Y – of final results. Function $f : X \to Y$ represents in a static way the system of programmes and is:

- a partial function, if for every $x \in X$ there is a value $y \in Y$ so as $y=f(x)$;
- a total function, if for every $x \in X$ there is a value $y \in Y$ so as $y=f(x)$.

In conclusion the total function correspond to a system of programmes that allows the solving of a problem for every initial data, and the partial function corresponds to a system of programmes which supplies us with solutions of certain sets of values.

A system of financial and accounting programmes is identified with a complex process, made out of many subprocesses, based on the rivalling model. This means separating different tasks into performing processes which are parallel different. Figure 1 presents the way such a system of programmes is structured.

A problem which is dealt with by using the calculator is represented through a calculating function, an algorithm. The same function is evaluated by a set of algorithms. There are functions that cannot be evaluated by algorithms.

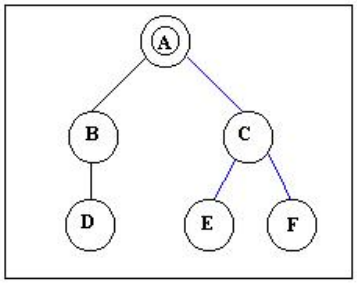

Figure 1. The tree of interconnected components

The structure is dynamical, which means that new performing processes are created or old ones are finished, according to the will of the user. The lines from the figure show the component which is being used and the using component, and the way we look at the tree is from top to bottom.

Between the components there are no implicit connections once these are appealed in the system, but if one wishes, a connection can be made. In this way a total control can be restored over every component of the system. If the example of figure 1 is analysed, we can see that the components A, C, E and F are interconnected; therefore the connection works both ways, no matter if component C has established the connection or component E or F.

Economical procedures and especially those from the financial and accounting field are recognised as having a high level of complexity. The difficulty associated with the solutions of simple problems united is smaller than the one associated to the initial complex problem. The architecture of the system expresses the way the system is entirely organized in components named subsystems. The interaction is produced through the exchange of the performing control. In the case of sequential programmes, the control belongs to only one module. The software architecture also includes information regarding the necessary time needed to perform every module.

The financial and accounting programmes are made out of subsystems; each is made out of smaller subsystems; the lowest level is achieved by modules. A subsystem is a package of connected classes, operations, associations, events and restrictions. These are identified by the services offered; services are groups of functions with the same goal.

A system of financial and accounting programmes offers a multitude of services through its components. The goal is to satisfy the users` needs for a long period of time and at a high quality level. The possibilities that the given functions are correctly executed for some time by the system are done through the help of reliability. The reliability of the system is determined by the reliability of the components, the number of components and the structure of the system.

2. Stimulating the reliability of the financial and accounting systems

A financial and accounting system offers a variety of functions; therefore it contains a big amount of components. Evaluating the reliability of the system is done by analyzing the reliability of it's` components. In this process, the structure scheme must be taken into consideration. The components of the system are represented in figures 2 and 4. In a structural reliability scheme, these are connected in series or parallel.

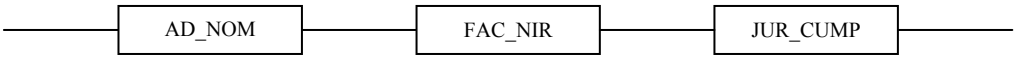

Figure 2. A serial structural scheme from a financial and accounting system

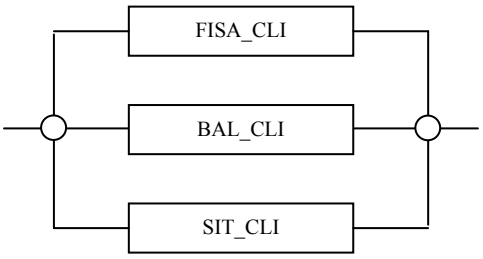

Figure 3. A parallel structural scheme from a financial and accounting system

In order to assure a normal functioning of a performing step, in the series scheme all components have to be working, but in the case of the parallel scheme only one component must be working.

The numerical simulation of the reliability of the financial and accounting system has been achieved through the following algorithm:

- for n in series connected components, each of them having the reliability R, n evenly distributed numbers between 0 and 1 are generated. If all n numbers are smaller or equal to R, the system is functioning properly;
- for n in parallel connected components, each with the reliability R, n evenly distributed numbers between 0 and 1 numbers are generated. If only one of the n numbers is bigger or equal to R, the system is functioning properly.

Estimating the global reliability of the system is made by repeating these numerical simulations for a number of times equal to the number of performing steps allowed. Because in the case of the financial and accounting system this number is high, the problem has been simplified and only 500 simulations have been performed.

a). The numerical simulating programme of performing components connected in series

```
n = [150,300]; % number of simulations
R = 0,8; % reliability of the components
m = 3; % number of series connected components
for j = 1 : length(n)
        k = 0;
        for I = 1 : n(j)
                x = row(1,m);
                if all(x<=R)
                        k = k + 1;
                else
        end
    end
    f(1,j) = k / n(1,j); % reliability
    end
```

b). The numerical simulating programme of performing components connected in parallel

```
n = [150,300]; % number of simulations
R = 0,8; % reliability of the components
m = 3; % number of parallel connected components

for j = 1 : length(n)
        k = 0;
        for I = 1 : n(j)
                x = row(1,m);
                if any(x<=R)
                        k = k + 1;
                else
        end
    end
    f(1,j) = k / n(1,j); % reliability
    end
```

c). The global simulating programme of 500 performed simulations

```
n = 500; R1 = 0,8;R2 = 0,92;
F1 = row(n,1);  F2 = row(n,1);
N = length(F); % number of functions
fprintf(The reliability of the system is %3.2f\n',N/n)
```

The first programme takes figure 2 into consideration and uses components with the reliability *R=0,8*. The second treats the case of figure 3 and also uses components with the reliability *R=0,8*. In order to compare the calculated reliability, a number of 150 and 300 simulations have been conducted. The third programme takes into consideration a series structure made out of two components, the first reliability *R1=0,8,* and the second reliability *R2=0,92*.

After the first programme performed, the reliability obtained was:
[0,5200 0,5000].

After the second programme performed, these values of the reliability were offered:
[0,9920 0,9960].

After the third programme performed, this value was obtained:

The reliability of the system is 0,74.

In practice it was been discovered that for financial and accounting programmes which contain a big number of components, using the series and parallel scheme does not assure a high level of reliability. Therefore, a mixed structure that combines the advantages of both types is used. In figures 4.a and 4.b two specific cases of such mixed structures are presented. These are frequently used for financial and accounting evidences.

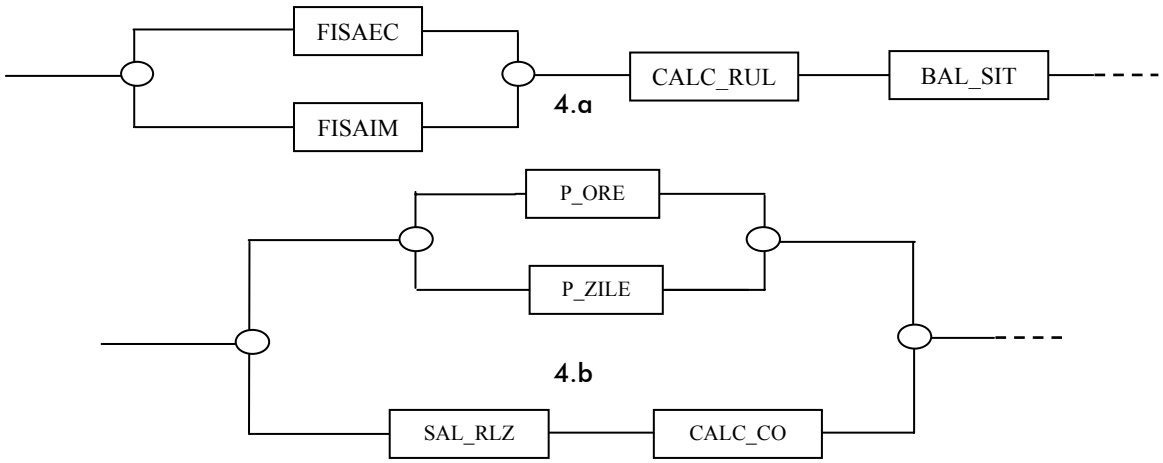

Figure 4.a, b Mixed structural schemes

The following reliability calculations are used in both cases:

- in the first case, from figure 4.a, the reliability is given by the relationship:

$$R_m = \prod_{i=1}^{n} R_i - \prod_{i=1}^{n} R_i (1 - R_i) \qquad (1)$$

- for figure 4.b the reliability is given by the relationship:

$$R_m = 1 - \prod_{i=1}^{n} (1 - R_i) + \prod_{i=1}^{n} R_i (1 - R_i) \qquad (2)$$

Because the complexity level of these schemes is very high, the very difficult necessity of simplification the structure function appears. In specialized literature [SZYP02][1], [PHAM00], [DAVI03] different methods of reducing the structure of the function and calculating the reliability of mixed structural schemes are presented. According to the method presented in [GORO97], the components of a financial and accounting system must be grouped regarding to the way they are situated in the serial or parallel graph and so, we get a primary level of a programming group. This group is formed by components which are connected in series or parallel. A new group on the next hierarchical scale follows and this procedure is continued until a single series or parallel structure of n levels is formed, where levels of n-1 components are displayed. These methods have a low applicability rate due to a set of assumptions on which it relies and too many calculations.

3. Using modern programming techniques to increase the reliability of financial and accounting software

The technique of object oriented modeling is a methodology used to develop financial and accounting software by using a collection of predefined techniques and noting conventions. It follows the entire life cycle which contains: analysing, designing, implementation and testing. These are followed by the stage in which it is used, when the maintenance and improvements on the system are done, to ensure the reliability needs imposed by the client.

For the development of accounting programming objects, two approaches are practiced: quick prototypization and the development of the entire life cycle. In the quick prototypization a small part of the system is initially developed, after this it is improved through gradual improvements of the specification and implementation, until it becomes robust.

The development methodology of software designed for financials and accountings is firstly characterized by the analyzing and projection steps, whereas the implementation and testing steps rely on the first. The analysis process has as a result a formed model which contains three essential aspects of the system: the objects and the relationships that exist between them, the dynamic flow between the orders and the functional transformation of data, using certain restrictions. Therefore the OMT methodology is bases on three models directed towards the object:

- the object oriented model – describes the static structure of data;
- the dynamic model – describes the temporal relationships of orders,
- the functional model – describes the functional relationships between values.

The programming technique frequently used is chosen on criteria such as error and performance tolerance. In [KICM00] it is told that the extension of the traditional library of stopping points is easy to do, so as this one is able to notice more directions from the same process. A multidirectional set library of stopping points, which works at a processing level, must save all directions for a verification point and to restore each of them when it is restarted.

In [TEOD01] it has been demonstrated that this mechanism of stopping points increases the flexibility and efficiency of the error tolerance schemes. Due to these characteristics it is used in the development of financial and accounting systems, in order to increase the efficiency of the tests and to raise the reliability level.

To exemplify the way this mechanism is used, an accounting programming system which, for error tolerance, uses the distributing algorithm of coming and going – present in [KICM00], is taken into consideration.

As a consequence of the existing relationships, the functions of the programming system become interdependent. If one of them fails, the algorithm determines which of the functions is dependent on the one that failed, and these must be performed backwards from the last stopping point. This solution is suboptimal when every function is multidirectional. In practice, it has been observed that only the paths dependent on the failing function must be performed backwards and the others remain unchanged. When establishing stopping points and backward points the following aspects are taken into consideration:

- the minimum frequency of performance for registering the dependencies and other information about the performance of the programme;

- the procedure for establishing selective testing points by using the information and the guiding points so as to develop the restore algorithm;
- the selective backwards algorithm based on guiding points.

To demonstrate how to use the stopping point and backward point technique in order to increase the reliability of financial and accounting software based on object oriented modeling, two arguments are taken into account:

- investigating the way group stopping points, for isolated groups of objects and communication ways of the programme, are established;
- investigating the way in which certain performing ways from a programme can be performed backwards in a selective way and others continue to be performed; during this period the general well being of the programming system is preserved.

Developing error tolerance schemes at a high level involves the usage of selective algorithms. In [ROMA03] it is said that the conventional models, that coordinate the process of restoring, after errors of interacting components are detected, must be implemented at the top of selective algorithms.

By using these techniques, the results obtained due to the growth in error tolerance and, therefore, of the reliability, indicate the fact that using selective schemes at processing levels is better than using techniques based on check points and also using recovery schemes when the number of current functions or error numbers are high.

4. Developing high reliability for object-oriented software

The software developing process is schematized through the next stages: system feasibility studying, problem analysing, designing, codification and system testing.

For a procedural programme system these stages correspond to a "warerfall" pattern. This means that the system is divided into substages and each requirement is previously known so that the tasks are performed one by one.

In the development of bookkeeping software based on this technology the starting point consists of recognizing the requirements of the matter. Therefore, an initial version is designed and then, as the requirements are better defined, the system is completed by adding new components or the existing ones are improved.

Adopting the evolutionary developing design leads to obtaining intermediate forms of the system, called prototypes. These resemble versions of the final form that are improved in time, as the developing process continues. Such an approach allows the client's effective control over the system's final version; the changes that occur in the client's requirements are accepted even if the analysis and design are in an advanced stage. The client's implication in the development process allows setting components and important sequences of execution. This determines the diminishing of the testing effort and implicitly of the costs and reliability increase.

Based on the evolutionary model, the development stages of the programming system are performed with every frequentation and the resulted prototype is evaluated for detecting the errors which are corrected in the next frequentation. In the system feasibility study stage the clients demands are clearly defined and through the client's implication a solution are chosen from the existent ones.

The architecture of the software for bookkeeping is divided in a number of components that consist of one or more objects. These components are collections of objects which collaborate for producing a service set. Each component is described by: functions, internal objects, external objects with which interfaces also interact. It is because of the interface that a component looks like a "black box" that shows only the entrances and emergences.

The testing stage involves the validation of the system results from the previous phases. The organisation of the designing process of the programme systems oriented toward objects involves the existence of different levels of testing. This includes the testing of methods, classes and modules, being based on an established initial plan that is finalised by testing the entire system. Object-oriented technology is used for testing the software and its main effect consists of improving the quality. A new programme system contains reused objects that have already been tested and have an appropriate reliability level. The result is that the testing effort is minimised and the reliability increases. In this case, the testing is aimed toward new components and especially toward the critical ones.

Modularity is another important facility, frequently used for developing programme systems designed for financial bookkeeping and it is based on object modeling. It allows an easier detection of software errors. The repairing process of these systems is also improved by establishing better connections between software items and real objects. As a result of this facility programme systems are divided in autonomous components. This has important effects on the human resources involved in the development and there, on the costs. The structure and organisation procedures of these resources are defined according to the defining manner of the components as well as the integration manner in the whole system.

It is recommended that these components should be developed by interfunctional teams that integrate analysis, designing, codification and testing abilities so that the development of each component is to be accomplished individually. In order to increase the functionality level, the assembly of different components must be done by groups of professionals that are in charge with the testing process of the entire programme system.

Practically it has been uncertain because of the limited resources of the companies which develop programme systems for bookkeeping; some of these recommendations are not followed. Therefore, in most cases the assembly of the components and testing the entire system is made by the same people that have taken part in the analysis, designing and codification phases. Thereby, they sometimes have a subjective vision upon the development process that leads to the decreasing of the ability to detect errors in the initial phases. In this kind of situations, the programme systems are moved to the operational stage, although their reliability level is low. The exploitation costs of these systems are rather high, and the users are not satisfied with the quality of the offered services.

In order to investigate the actual spreading of object-oriented technology among the producers of software destined to keep a financial-accountancy record and to analyse the characteristics of these practices on the software market, along the years, many actions have been undertaken. One of these is represented by the straw poll made by a branch of IBM in 2004. This was based on a questionnaire sent by e-mail and distributed at conferences. The questionnaire was divided into 3 sections: technology, development process and cost.

Based on the results of the straw poll, the weight of the object-oriented software production in the total financial-accountancy software production has been determined. This aspect is shown in figure 5.

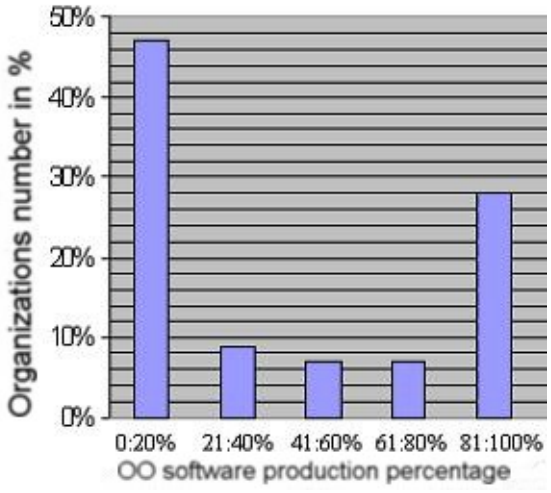

Figure 5. Grouping software producers according to the production of object-oriented software level (Source: http://www.garavelli/poliba/docs.html)

Using the object-oriented technology is in many situations delayed by the high costs required by the preparation. According to the dates, in only 8% of the companies the programmers who have always worked corresponding to this technology represent more than 80%, while in 57% of cases more than two thirds of the personnel has been converted to work on the basis of the principles of object-oriented technology.

Concerning spreading the methods of object orientation in the phases of analysis and designing of the software development process, it has been observed that 35% of the companies do not use any kind of methodology at all. These results were compared to those in section 3 of the questionnaire which refers to the exploitation costs of developed programme systems. It has been established that the costs level for those companies which do not use any kind of methodology is 27% higher compared to those who use object-oriented methodology and 16% compared to those which use the classical object-oriented methodology. 75% of the companies use prototypes during the process of software development. The degree of prototypes use is shown in figure 6.

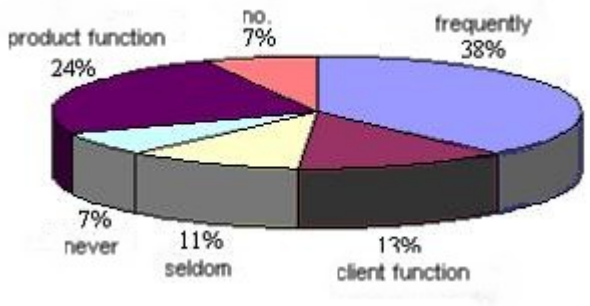

Figure 6. Using prototypes
(Source: http://www.garavelli/poliba/docs.htm)

Analysing the data has shown the fact that using the inter-functioning teams in the process of development is very frequent (68%). For 53% of the companies the size of the team depends on the complexity of the programming system, and for 33%, on its` size. In 69% of the cases the team consists of employees with different abilities, but in 19% of the cases the abilities of the members are homogeneous. The results also show that only 20% of companies use a system of metrics to control the quality, and 64% do not use any kind of metrics.

In figure 6 we can see that the frequency of prototyping is very high for 38% of them, when 24% is dependent on the product, and 13% depends on the client.

The companies questioned were asked for their opinion regarding the improvement of the reliability, as a consequence of four key influential factors which were placed on a scale of 0 to 3. The analysis showed that important factors were considered the development of reusable components (2,38), reusage of the existing components (2,26) or using innovative technological software (2,23). Reusing parts is considered to be the most efficient way of making reliability grow and this is why 66% of the companies produce own software components, designed for future reuse. The reusable components are produced during the development of a specific programming system (50%) or as a result of current activities (23%). Only 16% of companies do not use these reusable components in the process of improvement of the reliability of their software.

Because of this data, the majority of companies that develop software designed to keep the financial and accounting evidence use the technique of object orientation. Using reusable components represents a decisive factor in the process of reducing costs and improving the reliability. To evaluate the reliability of the software of many companies, adequate metrics and models are used.

The cost, from the total of sales that are formed when acquiring these components is smaller than 10%, for most companies (69%), whereas for 19% of companies it is smaller than 20%, growing until 30% for a percentage (12%) of the companies.

5. Specific aspects of the reliability of accounting software

The following factors determine the importance of the study of the reliability of programmes:

- the growth in the complexity of the functioning programmes, as a result of them being included in big software, and of the important functions that these must realize; the consequence is a growth in the cost of the user, in case of errors;
- high expectations regarding the quality of software;
- the complexity of the exploitation needs;
- growing cost of exploitation and maintenance.

One of the characteristics of the annual production of software consists of creating and developing complex systems – from the functional point of view. The programmes are parts of such complex systems; therefore they must match the general conditions of the system. The incorrect function of one of them, may lead to false results. The growing needs related to the functioning quality of programmes and of the systems, find their source, for example, in: a high flexibility, maintainability, portability, integrity, etc. Firstly, these needs must be satisfied in the previous steps, which precede the current exploitation of the product

by the user. When the programming system in the hands of the beneficiary is operational, only its` quality must be confirmed.

The growth in the costs of exploitation and maintenance of software is mostly determined by a lack of the reliability of the programmes it consists of.

In practice it has been discovered that, in the development stage, between the costs of development and the level of reliability of accounting software there is a tight bond; the components that need a high level of reliability have bigger costs in order to achieve this goal.

Testing is a method of improving the reliability of accounting software. A high level of reliability is achieved when the time span in which the testing is done is high and when the test are refined. The testing process involves human and material resource, and an increase in the testing efforts generates a growth in the costs of high level reliability components.

In order to study these connected relationships, three components of the programming system CONTGEST have been analysed. Each component had a specific level of reliability, according to its` operational profile. The time of the tests was record. A specific level for the cost had been attributed to the components, according to the total costs. The characteristics recorded are shown in table 1.

Table 1. The characteristics of the components of the programming system CONTGEST

Component	Time of testing (h)	Costs (%)	Reliability (%)
AD_NOM	51	27	76
ST_NOM	37	15	70
MOD_NOM	82	58	81

By analysing the data presented in table 1, we can observe that between the reliability and the time of testing there is nonlinear dependence. Figure 7 shows the relationship existing between these characteristics. We can see that the time of testing is not the only factor that influences the reliability and this is why other factors must be taken into consideration.

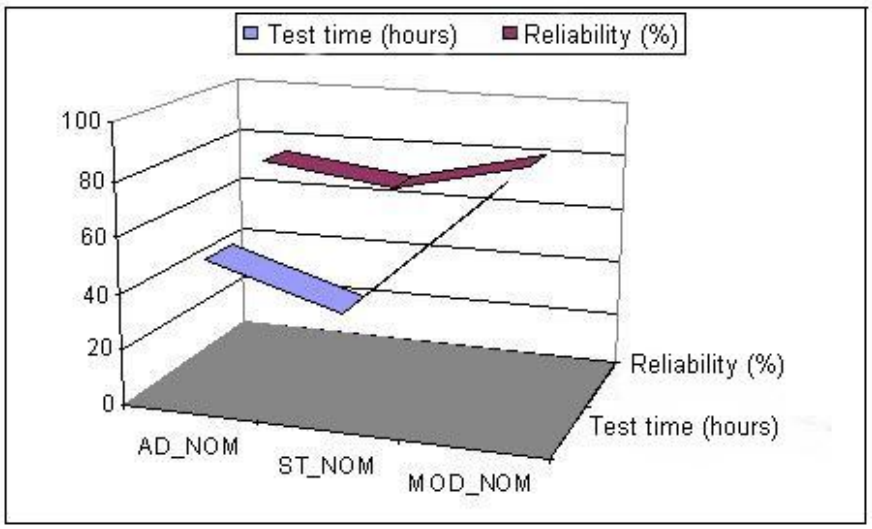

Figure 7. The relationship between the reliability and the time of testing for the components of the programming system CONTGEST

Due to a bigger appealing time of the programme MOD_NOM we can see that it has a high level of reliability, which influences the general reliability of the programming system CONTGEST. To obtain this goal, supplementary tests have been conducted, this leading to a growth in the percentage of the cost for this programme in the total cost of the three components.

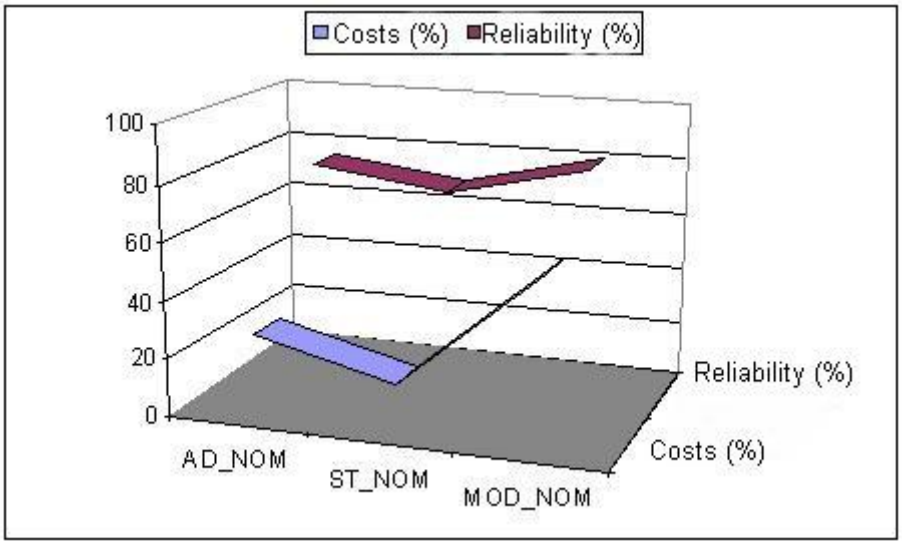

Figure 8. The depending relationship of the reliability and the costs for the components of the programming system CONTGEST

If we take into consideration the complexity of the programming system CONTGEST and its` structure, the analysis should be extended to a global level and we can determine the way characteristics such as testing time and costs influence the reliability of the general system. The information regarding the needs of the users and also the policy of the company which developed the product, regarding the ratio price/quality interfere in this process.

Because the users did not express been given out for usage without maximizing its` reliability. In the exploitation period problems regarding specific needs of the users have appeared, this leading to higher level of maintenance costs, and in two cases, the first version of the product had not been used before the newer version had not appeared.

6. Conclusions

In the study of the reliability of software, an important role is attributed to the existing relationship between the costs of development of a programming system and its` reliability. Obtaining a level of reliability that carries out all the needs imposed by the system, involves big costs in the development stages. These are otherwise smaller that the costs necessary to obtain the reliability needed in the process of exploiting the system – which has low reliability components.

The cost that involve the exploitation and maintenance of a financial and accounting software are directly depending on the level of reliability of the components and the reliability needs imposed by the system. Therefore, when interfering we do not always

achieve noticeable effects. There are cases where if we distinguish an error, others are generated.

If we take into consideration the variety and complexity of accounting situations that the programming system must deal with, the result is that efficiency is given firstly by the way they respond to the needs of the user. When the interferences caused by flaws are rare, the system is said to be more efficient – from every point of view. The frequency of errors and therefore the level of reliability represent signs of loyalty regarding the efficiency of the programming system.

References

1. Chillarege R., Kao W., Condit R. **Defect Type and its Impact on the Growth Curve,** Proceedings International Conference on Software Engineering, May 2004
2. Cristescu M. **Modelarea fiabilitatii sistemelor de programe,** PhD. Thesis, Bucharest, 2003
3. Davis A.M. **Software Requirements: Objects, Functions, and States,** Prentice-Hall, Saddle River, New Jersey, 2003
4. Goron S. **Fiabilitatea softului,** RISOPRINT Publishing House, Cluj-Napoca, 1997
5. Ivan I., Saha P. **Quality characteristics of The Internet Applications,** in DIGITAL ECONOMY - The Proceedings Of The Sixth International Conference On Economic Informatics, Bucharest, May 2003
6. Kim S., Clark J.A. and McDermid J. A. **Class mutation: mutation testing for object-oriented programs,** in Proceedings of the NetObjectDays - Conference on Object-Oriented Software Systems, 2000
7. Pham H. **Software Reliability,** Springer, 2000
8. Schneidewind N.F. **Life Cycle Core Knowledge Requirements for Software Reliability Measurement,** The R & M Engineering Journal, Volume 23 No. 2, June 2003
9. Schneidewind N. F. **Software Quality Control and Prediction Model for Maintenance,** Annals of Software Engineering 9, 2000
10. Simao R., and Belchior A. **Quality Characteristics for Software Components: Hierarchy and Quality Guides,** in Component-Based Software Quality: Methods and Techniques, LNCS 2693, pp. 188-211, 2003
11. Szyperski C. **Component Software Beyond Object-Oriented Programming,** Addison-Wesley and ACM Press, 2002
12. Teodorescu L., Ivan I. **Managementul calității software,** INFOREC Publishing House, Bucharest, 2001

[1] Codifications of references:

[CHIL04]	Chillarege R., Kao W., Condit R. **Defect Type and its Impact on the Growth Curve,** Proceedings International Conference on Software Engineering, May 2004
[CRIS03]	Cristescu M. **Modelarea fiabilitatii sistemelor de programe,** PhD. Thesis, Bucharest, 2003
[DAVI03]	Davis A.M. **Software Requirements: Objects, Functions, and States,** Prentice-Hall, Saddle River, New Jersey, 2003
[GORO97]	Goron S. **Fiabilitatea softului,** RISOPRINT Publishing House, Cluj-Napoca, 1997
[IVAN03]	Ivan I., Saha P. **Quality characteristics of The Internet Applications,** in DIGITAL ECONOMY - The Proceedings Of The Sixth International Conference On Economic Informatics, Bucharest, May 2003
[KICM00]	Kim S., Clark J.A. and McDermid J. A. **Class mutation: mutation testing for object-oriented programs,** in Proceedings of the NetObjectDays - Conference on Object-Oriented Software Systems, 2000
[PHAM00]	Pham H. **Software Reliability,** Springer, 2000
[SCHN03]	Schneidewind N.F. **Life Cycle Core Knowledge Requirements for Software Reliability Measurement,** The R & M Engineering Journal, Volume 23 No. 2, June 2003
[SCHN00]	Schneidewind N. F. **Software Quality Control and Prediction Model for Maintenance,** Annals of Software Engineering 9, 2000

[SIBE03]	Simao R., and Belchior A. **Quality Characteristics for Software Components: Hierarchy and Quality Guides,** in Component-Based Software Quality: Methods and Techniques, LNCS 2693, pp. 188-211, 2003
[SZYP02]	Szyperski C. **Component Software Beyond Object-Oriented Programming,** Addison-Wesley and ACM Press, 2002
[TEOD01]	Teodorescu L., Ivan I. **Managementul calității software,** INFOREC Publishing House, Bucharest, 2001