LIFECYCLE BASED AUDIT PROCESS FOR DISTRIBUTED APPLICATIONS

Mihai Doinea[1]

Abstract

The paper wishes to present the audit process as part of a distributed framework in which a new set of metrics are built. The requirements for an audit process are described and on each of the lifecycle stages, the audit process is emphasized. The existing quality characteristics models from the literature are analyzed based on which a new set of metrics are built for sustaining the overall audit process.

Keywords: audit process, distributed applications, metrics and lifecycle.

Audit process requirements

There is a large list of best practices in distributed software development from which audit procedures must be implemented and passed, for systems to enter into production and give relevant and reliable results.

The audit is a term which stands for listening the facts and figures provided from the activity of an organization. Initially, the term was used for the accounting and financial domain, but gradually spread in every area who needed a safe and impartial point of view on the audited activities. The auditing of the informatics systems is part of the general auditing process.

The audit for informatics systems refers to the following aspects according to [1]:

- the management responsibility;
- the incompatible function segregation;
- the access control;
- the physical security control;
- the disaster effect prevention control.

The literature includes several categories of audit concepts. According to [2] we mention the following ones:

- *Audit of Information Systems* – it aims the assessment of the information systems, the practices and operations regarding them;
- *Audit of Computer Information Systems* – it has the same significance with information system audit in an environment based on computer usage;
- *Computer Auditing* – it has many meanings: computer usage as instrument for auditing, audit in an environment based on computer usage or special investigations connected to financial audit.

[1]Phd Candidate, Amsterdam University, Information Management. The Netherlands (Economic Informatics Department, Academy of Economic Studies, Bucharest, Romania), mihai.doinea@ie.ase.ro

In order to conduct an audit process first of all, the auditors must find out which are the resources that they must evaluate and gather information about all the aspects involved along the activity line. After a well and rigorous documentation in the field examined the next stage is based on making qualitative and quantitative measurements and confront them with the figures provided by the company itself. At the end of the evaluation process they must provide a full documented report about what are the deficiencies of the analyzed field and what are the improvements that can be made to adjust the identified vulnerabilities and risks. The head management must implement the audit report suggestions and make a re-audit process to analyze its efficiency.

The responsibility for prevention and detection of the inconsistency and frauds in an information system depends on the ability of the management staff. The managers has to be sure that the appliance of an adequate internal control system is made without taking high risks and minimizing all that can be minimized with costs as low as possible. The internal control system represents the entire control system established by management staff in order to make the organization’s processes orderly and efficiently. Thus, it assures the conformity of the management methods, the protection of the capital and the completeness and accuracy of the records.

The technical auditor gives direct and detailed information to the management staff regarding all the activity of the auditing process, asking for any necessary additional resources for the audit process to run smoothly and with no interruptions. In his activity, the technical auditor needs the standards and the procedures used to evaluate the system to run the audit process and provide reliable results to the managers. Also, he is involved in the process of elaborating solutions and designing systems aiming a good functionality of the examined system.

The technical auditor has the following roles:

- taking into account all the risks involved by current informatics system implementation;
- carrying out the audit procedures for the targeted system;
- analyzing the obtained results and elaborating solutions to correct undesired behavior.

The following guidelines that presented in ISO 17799 and mentioned also in [2] should be followed in order to have an efficient and useful audit process:

- audit requirements should be agreed with appropriate management;
- the scope of the checks should be agreed and controlled;
- the checks should be limited to read-only access to software and data;
- access other than read-only should only be allowed for isolated copies of system files, which should be erased or given appropriate protection;
- resources for performing the checks should be explicitly identified and made available;
- requirements for special or additional processing should be identified and agreed;
- all access should be monitored and logged to produce a reference trail;
- the use of time stamped reference;

- all procedures, requirements, and responsibilities should be documented;
- the person carrying out the audit should be independent.

Auditors work with the full knowledge of the organization in order to understand the resources that they must check and the complete set of external connections with these assets. The audit process can be classified by its purpose in two different categories:

- internal audit – is conducted by specialists part of the organization having the role of making an internal radiography of the target area with the purpose of knowing the real state of the organization aspects implied;
- external audit – used by companies for getting external and reliable feedback about the current state of the audited domains and gaining trust in front of their possible clients; is made by third parties, independent from the audited organization which certifies through special means the quality of the audited processes.

The audit process for IT&C field is conducted on several levels. Is best that the audit process be implemented at each stage of development for distributed application in order to reduce as much as possible the negative influences given by uncontrolled factors.

A distributed application audit is a systematic, measurable, technical assessment of how organization's policy is used and if the distributed system is acting according to the specifications. Its main objective is analyzing IT&C area of the organization, to test the security levels for the respective information systems. In auditing the IT&C field, auditors must approach the following directions: the organization IT&C resources, the processes enrolled in the IT&C activity, the information security aspects and the computer security aspects.

The audit process follows a certain pattern to stand up to the existing standards and regulations. In auditing a distributed system the following aspects should be carried out: the communication process, the overall information process, the security constraints and the quality of the results provided by the system.

For undertaking an audit process for the information system of an organization, the organization itself must minimize the possible interferences between the audit and the business process. ISO’s standards describe a list of measures that could prevent the interference between this to processes as follows:

- audit requirements supervised by appropriate management;
- the actions taken should be only for check purposes;
- the checks should be limited to read-only access of data;
- back-up copies of vital data should be made before any audit operation;
- all audit operations must be logged for any possible failures;
- the whole process of auditing must be documented in order to provide detailed information about all the aspects involved.

In order for obtaining the best possible results, auditors must be qualified for their activity and also must not be implied in the activities that they are caring out.

Distributed applications lifecycle

Distributed systems are defined in [3] as a piece of software that ensures that a collection of independent computers appears to its users as a single coherent system.

In the realm of social software, examples of internet-based distributed platforms abound: a web-based platform for the shared creation and revision of a free, collaborative multilingual encyclopedia like Wikipedia; an online community platform where people plan and develop a new car along open source principles like OScar; collaborative community spaces for the collective use, modification, and redistribution of software e.g. Linux, Symbian OS, Mozilla, as well as countless distributed platforms found inside organizations that are meant to reduce costs and achieve high level of efficiency by successfully using resources in distributed manner.

Applications must shift to another approach in order to rich the end-user expectations. All the layers of a distributed application, developed stage by stage over the entire period of the software lifecycle, must follow the end-user criteria and meet their expectations. The new generation of distributed applications is built on the needs and aspirations of users to find exactly what they are looking for with no difficulty. Specifications must be developed and the distributed applications with the help of audit must serve totally the end user’s needs.

The new informatics applications are no longer instruments of organizations for which they were initially created, these applications must be orientated towards the end-user satisfaction. Considering the fact that geographical boundaries do not apply in the new society to information streams, the new applications must address a wider segment of users with diverse characteristics and demands [5].

This process can be achieved if, and only if, all the stages of development, in particular, for a distributed application and the lifecycle ones are approached from an end-user perspective.

For this, auditing procedures must be taken for not letting anything that didn’t respect the specifications to pass. Standards and regulations are rigorously defined for that software lifecycle stages to be unanimously accepted and all involved categories who are working along this process, starting from buyers, suppliers, developers, maintainers, operators, managers and ending with technicians to all have a common reference point and a common language.

The [6], which is the ISO standard for software lifecycle processes, in its updated form, that from 2008, *ISO/IEC 12207:2008 Systems and software engineering – Software life cycle processes*, describes processes, activities, tasks and outcomes applied in the process of software acquisition and configuration of software. The primary lifecycle processes together with their corresponding activities are depicted below:

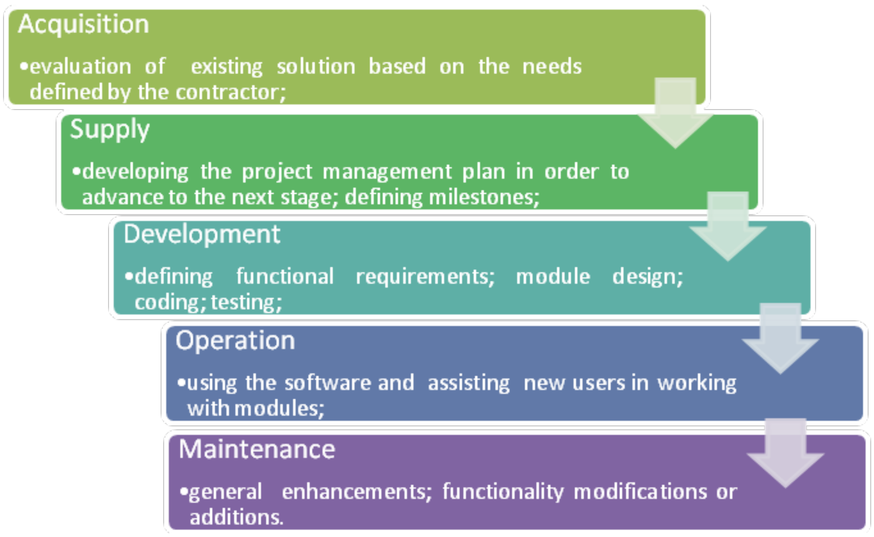

Acquisition
•evaluation of existing solution based on the needs defined by the contractor;
Supply
•developing the project management plan in order to advance to the next stage; defining milestones;
Development
•defining functional requirements; module design; coding; testing;
Operation
•using the software and assisting new users in working with modules;
Maintenance
•general enhancements; functionality modifications or additions.

Fig. 1 – ISO 12207 lifecycle stages

The audit must follow the lifecycle software path in order to have an end product with high expectancies and efficient workflow. An internal audit procedure could be implemented for each one of the aforementioned processes until the operation phase where an external development audit could be taken to guarantee the successful execution of the desired distributed system according to the initially defined specifications.

After the first external audit which will certify or not the final quality of the output, the implementation and operation together with a constant undergoing process of maintenance must be closely monitored and regular external audit procedures must be taken in order to certify the good operation of the product. In this way the end-users are assured that the quality of the final product which they are using is reliable and trust worthy.

The following picture describes an audit proposal procedure to be implemented along the lifecycle stages of a distributed application. Every stage can implement rigorous checkpoints to be followed so that the output of them to verify the figures targeted in the specifications.

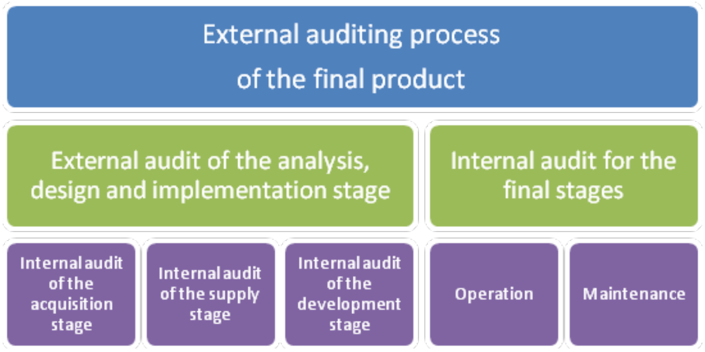

External auditing process of the final product
External audit of the analysis, design and implementation stage
Internal audit for the final stages
Internal audit of the acquisition stage
Internal audit of the supply stage
Internal audit of the development stage
Operation
Maintenance

Fig. 2 – Audit schema based on the software lifecycle stages

The analysis phase determines the requirements and the final purpose of the product, what the system is supposed to do. It's a well-defined picture of the problem what the system aims to solve, having as final output something like a UML Use Case Diagrams. If it's the case that an existing product on the market is satisfying the specified needs, then the system will be bought only after the means by which the software will solve the desired problem are analyzed, determining in which degree can do this without any other added customizations. If the price of customization along with the price of the purchased product are still less than the prices for developing a new one then the purchase is the optimum solution, but still an internal audit must be implemented to reveal the possible flaws of the selected product.

During the development stage the software product is designed, created and tested and will result in a software product ready to be released to the customer with all the expectancies fully functional:

- the design phase aims to develop a solution for the problem wished to be solved. The provided solution will tell how the system is willing to solve the problem and which the most efficient approach to it is;
- in the implementation phase the focus is centered on to coding the solution from the design phase. Here, the code is produced to run in a hardware system and using some specific technologies for specific platforms;
- the testing phase where the produced code is validated according to the requirements and the design class model; the system complete functionality and robustness, in a great part, depends on these tests; this doesn't mean that if a system passes the tests it's right and developed like it supposed to work.

Maintenance and operation processes are both done in the same time. In the operation stage all the flows are revealed because the expected behavior is verified from multiple different views, those of the end-users, which are acting unpredictable and so any miss coding or designing functionality is revealed and the maintenance process must repair it.

In order that the maintenance process to be successful and easy to do, the software shouldn't have used bad technologies, bad design, dirty code, or bad documentation. These all will end up in major obstacle in the maintenance process.

There is a tide bound between the processes found in the software lifecycle and audit as presented also in [6]:

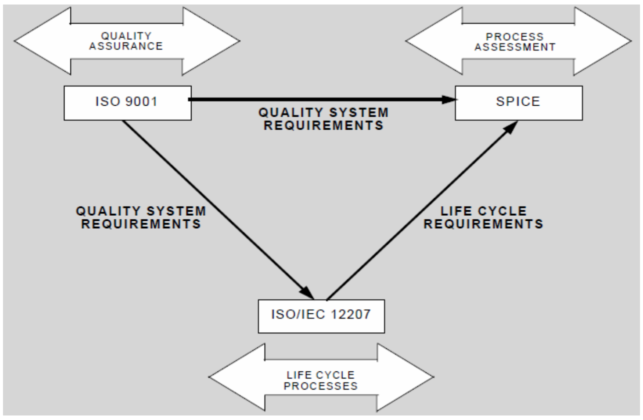


Fig. 3 - Relationships among ISO 9001, SPICE and ISO/IEC 12207 [6]

Informatics applications corresponding to the new society are end-user oriented. They have the target group - the user, as a central element. Distributed applications must be designed for the user, to solve their problems quickly and achieve the highest degree of satisfaction. The characteristics of end-user oriented distributed applications are:

- permanent access to resources;
- large number of users facilitated by the distributed architecture;
- high degree of user satisfaction given by the end-user developing orientation;
- easiness in accessing the resources of the distributed application by a wide category of beneficiaries;
- no need of resource consumption on training procedure given by the engageability characteristic [7] of this distributed systems.

The higher the degree these characteristics are met, the better the user oriented applications' quality is. The reorientation of applications to meet the users' needs is required [8]. For the design of the user oriented applications the target group is analyzed. The applications' quality characteristics depend on the target group's members. The users' satisfaction level is a good indicator of the application's final quality [9]. User oriented applications are realized to solve their problems.

Quality characteristics for distributed applications

Different ways for classifying the quality characteristics have been observed in literature [10][11] and a comparison between them should emphasize the strong points of each one, with considerations about which are the best to use in different situations and from which perspective should they be analyzed: (i) users, (ii) administrators or (iii) organizations.

In order for distributed systems to behave according to the requirements and needs of people and contemporary organizations, stringent quality characteristics and sub-characteristics need to be met. Quality characteristics are defined by [12] as being a set of attributes of a software product by which its quality is described and can be evaluated.

The way in which these systems are designed must reflect the quality characteristics needed for usage and need to be carefully managed throughout the entire system life cycle. The McCall Quality Characteristics Model is based on the life cycle of a product and the characteristics are grouped by three major categories: operations, revisions, transitions, figure 4.

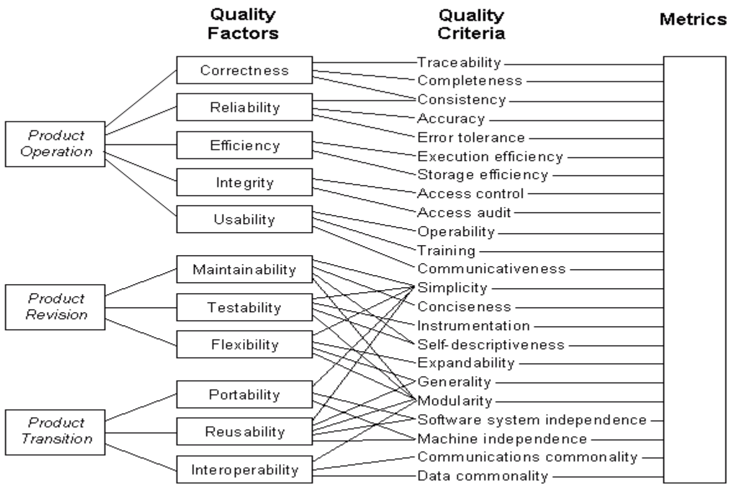


Fig. 4 - McCall Quality Characteristics Model [13]

The Boehm Quality Characteristics Model is more or less the same as the previous but he reorganized the nature of some characteristics, figure 5.

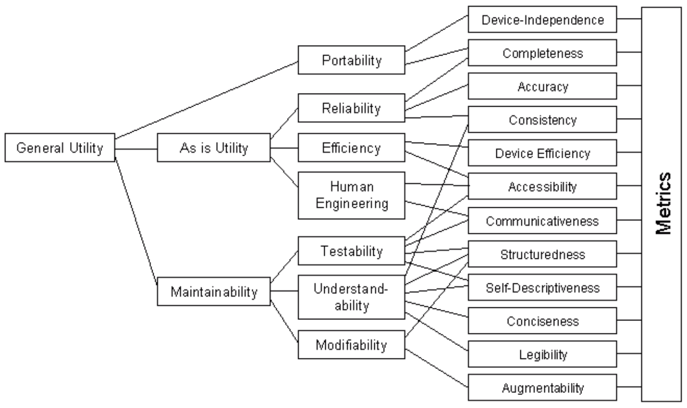


Fig. 5 - Boehm Quality Characteristics Model [13]

During all stages of the lifecycle—that is from starting to develop a distributed system using modeling languages, doing a back stage analysis upon the characteristics that are required in order to achieve the purposes for which the system is being built, through to the final stage of the lifecycle in which the system is removed from use or replaced—quality characteristics inevitably influence all the components with which the system interacts. By understanding better the underlying processes that constitute the distributed system’s Graphical System Interface, GUI and identifying the aspects that are relevant in providing useful and reliable results an analysis upon the quality characteristics at different levels for distributed systems must be made.
ISO had introduced in 1991 the following classification, figure 6 and identified six major characteristics and for each one, a set of sub characteristics based on which internal and external metrics could be developed.

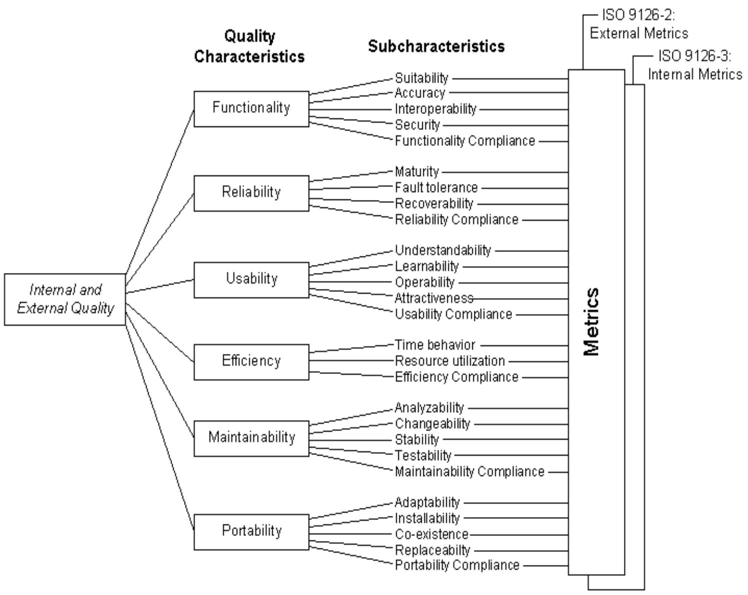


Fig. 6 - ISO 9216 Quality Characteristics Model [14]

The quality characteristics and their sub characteristics should be treated from several different perspectives to get a good understanding of how these can influence the overall performance of the system. The purpose of distributed systems can be unfolded as follows:

- increase efficiency by means of better interactions between participants;
- share resources to achieve a higher level of collaboration;
- dividing tasks for obtaining reliable results more effectively and efficiently.

Depending on the perspective we adopt, different characteristics can be identified for a distributed system. These perspectives can be categorized according to the following criteria:

- target group criteria – the users for which a distributed system is designed and who have their own opinions about how such a system must react when dealing with their requests;
- technological aspect criteria – from a technological perspective distributed systems must enclose all the characteristics of a distributed system much more;
- social or economic area of activating – depending on the type of jobs distributed systems are performing they must meet different characteristics, some more important than others; e.g., in the military field, the distributed systems of defence [15] has a stronger need for security than informal distributed systems such as Wikipedia;
- the nature of activities it serves: informing, negotiating, collaborating or cooperating; based on the nature of the activity that the system needs to support, certain characteristics are required.

In what follows, the four criteria will be discussed in more detail. Regarding the first criteria we can identify some important characteristics that a distributed system must have, defined also in [16], in order to satisfy the designated target group for which it is built; these are important characteristics viewed from a user perspective:

- accessibility – is a general term used to describe the degree to which a distributed system is accessible by as many people as possible, with as less of assistance or none, preferable;
- availability – is the characteristic which reflects the time in which a system is fully operational and users can access its processes without interference;
- usability – describes the extent to which a distributed system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context without any kind of impediments;
- reliability – refers to the capacity of distributed systems to perform tasks given a certain stressful setting without any unpredictable stops;
- efficiency – determines the relationship between the level of performance of a distributed system and the amount of resources used for generating results.

The second topic contains characteristics that must be met during the design stage but also after the distributed system is put into production. These kinds of characteristics are defined in [12] as quality software characteristics without which parts of a distributed system cannot serve its purposes: functionality, portability, maintainability, reusability.

The third aspect will require that specific characteristics are more rigorously treated than others depending on the domain in which distributed systems are integrated:

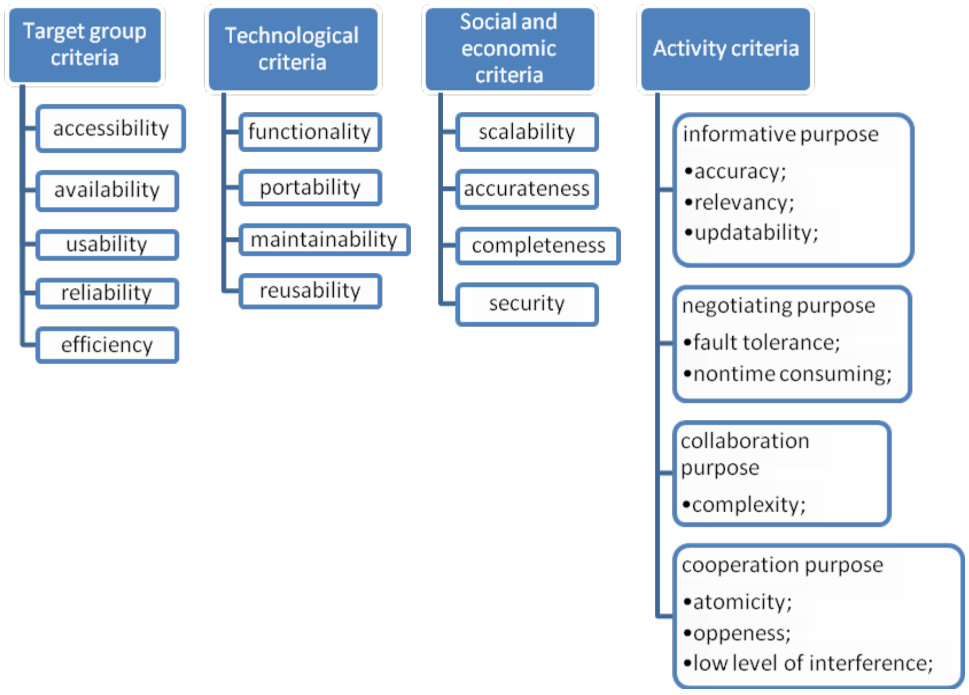
- scalability – geographical distributed systems rely heavily on two elements, one is the algorithms that are needed to compute all the data and the other one is a database in which attributes of the geographical elements are stored. From this point of view scalability—the power and easiness with which data or new elements can be added and the system’s ability to handle these growing amounts of data—requires careful management;
- security – distributed systems of defence have a greater need for security characteristics like confidentiality, integrity and authenticity due to the nature of the tasks for which these systems are implemented;
- accurateness – distributed medical systems must work with efficient algorithms and high precision otherwise situations in which doctors are performing remote procedures on patients could lead to dramatic results;
- completeness – informing distributed systems need to have huge databases filled with different types of data and they must cover as much as possible in relation to the specific subject domains for which these are employed.

For the fourth category—the purpose of a distributed system— we can identify several characteristics that are worth mentioning. Every category stresses certain of the aforementioned aspects while ignoring others, as follows:

- informative purpose – these types of distributed systems aim to gather and store information under a common aegis, helping people share their knowledge, hence, these systems must offer among other characteristics, the intrinsic characteristics of information: (i)*accuracy*, (ii)*relevancy* and (iii)*updated*; e.g. Wikipedia;

- negotiating purposes – such distributed systems play a vital role in the negotiation process and therefore (i)the *fault tolerance* and (ii)*non time consuming* characteristics must persist; e.g. BidRivals negotiation system;
- the collaboration aspect is presented here as a group that is working towards a single goal but at an individual level different tasks are assigned in order to accomplish the common goal; such an approach must rely on a complex (i)*document* and (ii)*process management* system; e.g. Office SharePoint server;
- the cooperation captures the aspects of a distributed system used by a group of people who are trying to achieve the same goal, therefore having a common objective plus, using and sharing the same resources and working simultaneously and collectively like brainstorming activities; the characteristics which must emphasis in such systems are (i)*atomicity* and a (ii)*boundless framework* together with a (iii)*low level of interference.*

A new model can be achieved based on the four points of view, defined as the big four set of characteristics that wish to cover all the drawbacks and advantages of the aforementioned models. The model tries to regroup all what exists into a new approach that will help developers, managers, technicians and other users aim exactly what are the important characteristics that need to be closely monitored based on the aforementioned



criteria.

Fig. 7 – The big four quality characteristics criteria

Based on the lifecycle stages, a hierarchy of the quality characteristics is made to reveal the order in which they must be taken into consideration for getting the most out of the final product. For each step of a product lifecycle we can rank the characteristics based on the moment in time where they should be discussed and implemented and also where the effects of their implementation are visible.

The implementation of the quality characteristics are taken into consideration at the first three processes of the software lifecycle in the following order:

- acquisition and supply – functionality, accessibility, reliability, relevancy;
- development – security, availability, efficiency, usability, accurateness, completeness, fault tolerance, resource consuming, analyzability, maintainability, atomicity, complexity, portability, reusability, usability.

The effects are visible at the following stages:

- operation – accessibility, availability, reliability, efficiency, security, usability, functionality, accurateness, completeness, relevancy, fault tolerance, resource consumers;
- revision – analyzability, changeability, stability, testability, updateability, complexity, atomicity;
- transition – portability, reusability and interoperability, scalability, low level of interference.

Is important that each one of the quality characteristic should be treated in the right time and on the right place because between all of them exists interconnections that will affect later the final result.

Metrics for the audit process

For doing a correct and relevant assessment upon the implications of the lifecycle processes for a distributed application upon the quality characteristics and sub characteristics, their quantitative aspects must be synthesized by means of some metrics that can measure the dynamics and the behaviour of each and every input factor in order that the audit process to be undertaken.

The following set of metrics are constructed based on the new quality characteristic model described in the previous chapters with the scope of getting significant output from final product when will be audited.

Based on the previous model, metrics can be defined like usability, portability, functionality, maintainability, reliability and efficiency helping auditors to get relevant intel about the behaviour of the distributed system according to the specifications.

Usability describes the extent to which a distributed system can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context without any kind of impediments, is the characteristic of being intuitive, [17] and can be measured by the following formula:

$$UM = \frac{\sum_{i=1}^{n}(1 - p_i)}{m}, \qquad (1)$$

where:

UM – usability metric;

n – the number of user’s successful operations with or without assistance;

m – the number of user's total operations;

– the degree in which a user was assisted for completing the *i* operation.

The metric will take values between . If a user completed all operations successfully, then n will be equal to m and further more if it didn't had need of any kind of assistance, , then *UM = 1*, which means that the system has maximum usability.

The following results from table 1, describe the behavior of a number of successful operations *n=15* from a total number of *m=20* operations, recorded for a group of users with regards to the usability of the distributed application.

Table 1 – Usability data set

No.	**The degree of assistance ( )**	**No.**	**The degree of assistance ( )**	**No.**	**The degree of assistance ( )**
1	0.52	**6**	0.75	**11**	0.85
2	0.71	**7**	0.87	**12**	0.52
3	0.62	**8**	0.23	**13**	0.83
4	0.34	**9**	0.96	**14**	0.87
5	0.36	**10**	0.78	**15**	0.95

The utility indicator based on the data from table 1 has the following value:

$$UM = \frac{\sum_{i=1}^{n}(1-p_i)}{m} = 0.242$$

The low degree of usability is given by:

- the fact that not all operations were completed successful, just *75%* of them;
- the successful operations needed a high level of assistance in order to be completed.

Another characteristic that can be also measured is portability which defines the degree of changing the environment in which a distributed system is functioning without being forced to spend extra costs for these modifications. The metric used could be determined by the following formula:

, (2)

where:

PM – portability metric;

– the costs supported for the porting process;

– the costs supported for the developing process of a distributed system.

The domain for this metric is defined by the following values: .

In the best case scenario, if a distributed system must be moved in other working environments without extra costs then PM = 1, otherwise the metric will behave as follows:

(3)

The scenario in which is the one in which the costs for changing the environment are greater than the design costs and it represents an unpractical solution, in this case, a new system should be developed.

If the , then the PM metric has the following values:

$$PM = 1 - \frac{C_{int}}{C_{dev}} = 0.912$$

The value of this metric should be close to 1, in order to have a high degree of portability.

Functionality is defined as the ability of a distributed system to interact properly among its own components and the external ones. Functionality can be calculated for distributed systems that already have passed the implementation phase, using the following formula:

,
(4)

where:

FM – functionality metric;

– number of bytes affected by error *k*;

n – total number of errors;

– number of bytes transferred between *i* and *j* components;

m – total number of components.

The domain for the functionality metric is . When no errors are detected functionality metric equals 1, otherwise, if errors occur in the transfer between the internal or external components affecting all information processed, *FM=0,* because each error is counted for every transfer failed.

Table 2 contains the number of bytes affected by a each one of the *n=8* errors.

Table 2 – Functionality data set

Error no.	**BN_k (KB)**	**Error no.**	**BN_k (KB)**
1	10	**5**	23
2	14	**6**	30
3	320	**7**	15
4	65	**8**	45

The matrix MCT – component transferred matrix is a symmetric matrix, where the element $a_{ij} = a_{ji}$, with .

$$MCT = \begin{pmatrix} 23 & 12 & 120 & 56 \\ 12 & 52 & 98 & 43 \\ 120 & 98 & 256 & 27 \\ 56 & 43 & 27 & 120 \end{pmatrix}$$

The functionality metric for the data extrapolated above has the value:

0.269,

with:

KB and *KB*.

Maintainability is the characteristic which give users, if present, easiness in correcting bugs, meet new requirements or improve future maintenance. Metrics have been developed for measuring the maintenance index like:

- lines of code measures;
- McCabe measures – cyclomatic complexity [18];
- Halstead complexity.

Maintainability characteristic is defined by several system properties and for each one, metrics could be built:

- changeability;
- testability;
- analyzability.

The following metric aims to reveal the maintainability power of a distributed system from the perspective of adding new features, changing existing ones to comply with the new requirements or even eliminating some, aggregated also with the level of complexity defined by the graph associated with the distributed system. Changeability metric is defined below:

, (5)

where:

– initial number of source line;

– number of lines removed from the system;

– number of lines added to the system;

– number of lines modified from the system.

Another metric used for the maintainability index is the one which describes the number of paths from the initial point of the distributed system's oriented graph to every functionality described as a node. The distributed system's graph is represented by its adjacency matrix AM defined below:

(6)

Based on this matrix GCM, the graph complexity metric is defined as:

(7)

The maintainability index MM, will be computed based on (5) and (7) as follows:

(8)

The two components found in the MM index influence in the following way:

- CM – if no changes are made in the system at the maintenance stage then CM=1, otherwise CM will decrease to zero accordingly with the degree of changes made;
- GCM – the minimum value of the GCM is 1, representing just a single functionality, otherwise if increase the number of functionalities and the connections between them, the complexity will increase unrestrictive;
- representing the weight of the changeability metric resulting in

Other factors could also influence maintainability but the power of code changeability and system complexity level are ones of great importance for a distributed system.

Let SL_0 = 500, SL_R = 50, SL_A = 100 and SL_M = 150, than

$$CM = \frac{SL_0 - SL_R + SL_A}{SL_0 + SL_R + SL_A + SL_M} = 0.688.$$

If the application has *n*=5 functionalities implemented and the adjacency matrix of the interactions between each functionality is defined below:

$$AM = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{pmatrix},$$

$$GCM = \sum_{i=1}^{n} \sum_{j=i}^{n} am_{ij} = 9$$

than:

$$MM = \left(1 - \frac{1}{GCM}\right) * CM = 0.611$$

Reliability refers to the capacity of distributed systems to perform tasks given a certain stressful setting without any unpredictable stops. The metric referring reliability is described below:

(9)

where:

RM – reliability metric;

DS_{0i} – data segment before unpredicted errors occur (bytes) in functionality *i*;

DS_{1i} – data segment after re-entering in normal functionality (bytes) in functionality *i*;

T_i – time needed for recovery in functionality *i*;

– data segment average calculated for the *i* functionality of the system;

m – total number of functionalities for the distributed system.

The domain for the reliability metric is . The reliability metric represents an average, of every single reliability functionality and the more it gets higher the less effective is the system.

The following data were recorded for a distributed application which has a number of *m=7* functionalities:

Table 3 – Reliability data set

Funct. No.	**DS_{0i} (Kb)**	**DS_{1i} (Kb)**	**T_i (ms)**	**(Kb)**	**Funct. No.**	**DS_{0i} (Kb)**	**DS_{1i} (Kb)**	**T_i (ms)**	**(Kb)**
1	50	40	40	70	**5**	40	40	5	40
2	30	30	21	10	**6**	25	20	3	20
3	100	13	14	20	**7**	30	20	10	80
4	340	280	45	50	**8**	100	10	79	90

$$RM = \frac{\sum_{i=1}^{m} \frac{DS_{0i} - DS_{1i}}{T_i * \overline{DS}_i}}{m} = 2.19,$$

meaning a good value based on the percentage of data recovered, *pdr*, which is:

.

Efficiency is defined as the ability of a distributed system to achieve its objectives without waste of resources, such as memory, space and processor utilization, network bandwidth, time, etc.

If we know the following values, we can define the Halstead complexity, HC [19]:

- n_1 – the number of distinct operators;
- n_2 – the number of distinct operands;
- N_1 – the total number of operators;
- N_2 – the total number of operands.
- Program length: $N = N_1 + N_;$

- Program vocabulary: n = $n_1 + n_2$;
- Volume:

- Difficulty:
- Efficiency: HC = D*V.

The efficiency metric, *EM* is described as an aggregated index of the efficiency of each functionality. Number of operations per time weighted with the complexity of an operation given by the Halstead complexity describes the efficiency for one functionality of a distributed system.

(10)

where:

nop_j – the number of operations done for the *j* functionality;
HC_j – Halstead complexity for the functionality *j*;
T_j – necessary time for doing operations nop_j;
m – number of functionalities.

The efficiency metric, being defined as an average of each functionality efficiency found in $(0; +\infty)$, has also its values in $EM \in (0; +\infty)$.

When the time factor $T \to \infty$ then efficiency is defined as $\lim_{T \to \infty} EM = 0$, meaning the lowest level of efficiency, otherwise the value of the indicator is improving.

Table 4 contains data for calculating the efficiency metric based on the time spent for executing a number of operations:

Table 4 – Efficiency data set

Funct. No.	***nop$_j$***	***HC$_j$***	***T$_j$ (ms)***	**Funct. No.**	***nop$_j$***	***HC$_j$***	***T$_j$(ms)***
1	40	150	40	**5**	45	34	65
2	50	100	30	**6**	10	143	45
3	100	40	10	**7**	80	25	30
4	20	300	50	**8**	50	10	15

$$EM = \frac{\sum_{j=1}^{m} \frac{nop_j * HC_j}{T_j}}{m} = 123.997$$

EM value is greater than 0, meaning a good efficiency for the total number of functionalities.

The metrics used for auditing a distributed system must also be validate by statistical data gathered from the application usage and also by verifying the main properties of a metric, which are: representative; sensitive; general; stable; non-compensatory; non-catastrophic.

After a final review of these metrics the audit process can be carried out based on the values provided from measuring the quality characteristics of a distributed system.

Conclusions

The audit is a standard procedure that should be applied to a distributed system in order to certify whether or not the system is functioning like it was intended to and described in the specifications. For such procedure to take place and also to provide reliable and trustful results, a set of regulations and a strict set of procedures should be carried out. When auditing distributed applications, there are many things that must be taken into account due to the high complexity of such systems. For this a lifecycle approach is preferred because all the audit process is split it and mapped on each one of the software lifecycle stage. If for each one of the stages the audit is undertaken, analyzing the symmetry between what it does and what it supposed to do, then the risks of missing important aspects is diminishing.

Bibliography

[1] Ivan, I., Nosca, G., Capisizu, S. – *Auditul sistemelor informatice*, ASE Printing House, Bucharest, 2005

[2] Popa, M., Doinea, M. – *Audit Characteristics for Information System Security*, Economic Informatics Journal, INFOREC Press, Vol. 11, No. 4, 2007, pp. 103 – 106, ISSN 1453-1305

[3] Tanenbaum, A.S., Steen, V.M. – *Distributed Systems: Principles and Paradigms, second edition,* Pearson Prentice Hall, 2007, 704 pg. ISBN 013-2392-275

[4] Ivan, I., Ciurea, C., Visoiu, A., Doinea, M. – *Quality Demands Of Data In Collaborative Systems*, Supplement International Workshop in Collaborative Systems and Information Society, Annals of the Tiberiu Popoviciu Seminar, Mediamira Science Publisher, vol. 6, pg. 212 – 221, Cluj-Napoca, 2008, ISSN 1584-4536

[5] Rosca, I. G., Ghilic-Micu, B., Stoica, M. – *Informatica. Societatea informatională. E-Serviciile,* Economica Press, Bucharest, 2006, 488 pg, ISBN 9789737092663

[6] Singh, R. – *International Standard ISO/IEC 12207 Software Life Cycle Processes*, Federal Aviation Administration, Washington, DC, USA, 17 pg.

[7] Doinea, M., Osch, V.W. – *Collaborative systems: Defining and Measuring Quality Characteristics*, Journal of Applied Collaborative Systems, Vol. 2, No. 1, 2010, ISSN 2066-7450

[8] Jiang, L., Wu, R., Li, Y., Wang, H. – *Citizen-oriented community e-government service platform*, International Conference on Services Systems and Services Management, Proceedings of ICSSSM, Vol. 2, 13-15 June, 2005, 1434–1437 pp.

[9] Ivan, I., Ciurea, C. – *Entry data validation in citizen oriented applications*, The 4th International Conference on Applied Statistics, November 20-22, 2008, NIS Publishing House, Bucharest, Romania

[10] Gheorghe, N. – *Analiza comparata a modelelor calitatii software*, Informatica Economica Journal, Vol. 28, No. 4, 2003, 44-49 pp., INFOREC Publisher, ISSN 1453-1305

[11] Perez, M., Rojas, T. – *Construction of a Systematic Quality Model for Evaluating a Software Product*, Software Quality Journal, Vol. 11, No. 3, 2003, 219-242 pp., ISSN 0963-9314

[12] Wikipedia ISO/IEC 9126-1991 International Standard, *Information technology – Software product evaluation - Quality characteristics and guidelines for their use*, 2010, [Online] http://en.wikipedia.org/wiki/ISO_9126

[13] Cote, M.A., Suryn, W., Georgiadou, E. – *Software Quality Model Requirements for Software Quality Engineering*, Software Quality Management & INSPIRE Conference, 2006, pp. 31-50, ISSN 1363-0679

[14] Behkamal, B., Kahani, M., Akbari, M. K. – *Customizing ISO9126 quality model for evaluation of B2B applications*, Information and Software Technology, Vol. 51, No. 3, 2009, pp. 599-609, ISSN 0950-5849

[15] Ivan, I., Ciurea, C. – *Using Very Large Volume Data Sets for Collaborative Systems Study*, Informatica Economică Journal, Vol. 13, No. 1, 2009, INFOREC Publisher, ISSN 1453-1305

[16] Ivan, I., Ciurea, C. – *Quality Characteristics of Collaborative Systems*, Proceedings of The Second International Conferences on Advances in Computer-Human Interactions, ACHI 2009, February 1-7, 2009, Cancun, Mexico, ISBN 978-1-4244-3351-3

[17] Ivan, I., Doinea, M. – *Social Networks Security*, The Journal of Applied Collaborative Systems, Vol. 1, No. 2, 2009, pp. 101 – 110, ISSN 2066-7450

[18] Ivan, I., Doinea, M., Săcuiu, L. – *Evaluating security of non-homogenous distributed applications*, 4th International Conference on Applied Statistics, Bucharest, November 2008, ISBN 1018-046x

[19] Pedrycz, W., Succi, G. – *Genetic granular classifiers in modeling software quality*, Journal of Systems and Software, Vol. 76, No. 3, June 2005, 277-285 pp.

Acknowledgments

This work was supported by CNCSIS –UEFISCSU, project number PNII – IDEI 1838/2008 and by „Doctoral Programme and PhD Students in the education research and innovation triangle" project which is co funded by European Social Fund through The Sectorial Operational Program for Human Resources Development 2007-2013, coordinated by The Bucharest Academy of Economic Studies (project no. 7832, "Doctoral Programme and PhD Students in the education research and innovation triangle, DOCECI").