THE MANAGEMENT OF SOFTWARE PRODUCTS THROUGH ECONOMIC INFORMATION SYSTEMS

Lecturer PhD Nelu BURCEA
Athenaeum University, Bucharest, Romania
neluburcea@yahoo.com

Lecturer PhD Dănuţ-Octavian SIMION
Athenaeum University, Bucharest, Romania
danut_so@yahoo.com

Abstract
The paper presents the activity of managing the software products through economic information systems that ensure the storage of data, manipulation of different facts and also the interrogation of large quantities of data that is used in various reports for the decisions of managers. The internal characteristics of the product, process or resource are those that are measured by examining the product, process or resource, separate from their behavior. Internal features include: maintainability, flexibility, portability, reusability, program readability, testability, understanding, ease of building. The external characteristics of the product, process or resource are those that are measured only with reference to how the product, process or resource reacts with the environment. External features of software quality directly affect the value of the product to the user. Achieving a product quality model involves identifying the tangible internal tangible properties of the product, measurable and / or evaluable, which have the greatest effect on external quality attributes. The benefits of choosing a good software product is reflected in a better productivity and a good management of business flows that ensure a stabile growth in relations between producers, supliers and customers that may collaborate with companies.

Keywords: *Software products, business environment, methods of evaluations, quality attributes, information systems.*

JEL Classification: *C23, C26, C38, C55, C81, C87*

1. Introduction

The quality of a product is sometimes defined as "the totality of its characteristics by which it meets a number of defined or imposed needs". The quality of a software product is due to its ability to be effectively and comfortably used by a set of users for a set of purposes under specified conditions. The quality characteristics of a software product are properties of the product to which users are sensitive. For example: ease of use, reliability, response time, etc. There are different models to classify the quality attributes (attributes) of a software product. Models often include measures to determine the degree to which the product meets each quality attribute. Each model may have a different attribute set at the highest level of the classification, too, the selection and attribute definitions may differ at all levels.

The quality required for a software product must be defined in the software requirements definition document (SRD). Also, the definitions of quality attributes, measurement methods, and attribute acceptance criteria must be specified [1], [4].

The quality of program products can be appreciated both by the quality characteristics specific to the current execution of the programs, as well as by those that ensure the maintenance of the execution programs as the initial conditions of the problems change. The quality of a software product is at the end of the development process only if the internal properties that determine the level of the quality characteristics are built during development. It is considered as the fundamental axiom of software quality that: the tangible internal or internal properties or characteristics of the product determine the quality of its external characteristics.

2. The characteristics of software products and the economic value of those

Standardization of software product terminology has led to ISO 9126 (InformationTechnology-Software Product Quality, Part 1: Quality Model, 1998). The standard contains definitions in particular for the final product. Six quality features are defined, divided into 21 sub-features.

a) Functionality: Achieving the basic purpose for which the product was made

- Opportunity: the presence of a set of appropriate functions for specified tasks
- Accuracy: Delivering correct or agreed results or effects
- Interoperability: the ability of the product to interact with specified systems
- Security: the ability to prevent unauthorized, accidental or deliberate access to programs or data

- Compliance: adherence to standards, conventions, laws and protocols

b) Reliability: the product's ability to maintain its performance level under defined conditions for a defined period of time.

- Maturity: attribute based on the frequency of failures due to software mistakes
- Fault tolerance: the ability to maintain a specified level of performance in cases of software failures or unexpected inputs
- Fallback recovery: the capacity and effort required to restore the performance level, recover affected data after possible falls
- Compliance

c) Usability: the effort required to use it by a defined set of users

- The ease of understanding: the effort required by a user to recognize the logical concept and its applicability
- Learning ease: the effort required by a user to learn the application, operation, inputs and outputs
- Operability: ease of operation and control by users
- Power of attraction: the ability of the product to be attractive to users
- Compliance

d) Efficiency: the relationship between the product's performance level and the amount of resources used under defined conditions

- Execution time: response speed, processing times, output rate at function execution
- Use of resources: the amount of resources used and the duration of use for performing its functions
- Compliance

e) Maintenance ease: the effort required to make the changes, including corrections, improvements or adaptations of the product to changes in the operating environment, requirements and functional changes

- The ease of analysis: the effort required to diagnose defects, causes of falls, to identify parts that need to be modified
- Change ease: the effort required to remove defects or change
- Stability: the risk of unexpected effects from changes
- Test ease: the effort required to validate the modified product
- Compliance

f) Portability: The ability of the product to be transferred from one organization or software platform to another

- Adaptability: ability to adapt to different specified environments
- Installation ease: the effort required to install the product in a specified environment

- Co-existence: the ability to coexist with other independent products in the same environment
- Opportunity and effort to use the product instead of another product in a particular environment
- Compliance

Special types of systems and quality requirements

There are many particular quality requirements that fall or fall within ISO 9126. Certain special classes of applications may have other quality attributes to consider.

Examples:

• Systems whose fall can have extremely severe consequences:

- The degree of trust of the system as a whole (hardware, software people) is the main purpose, in addition to the achievement of basic functions.

A high degree of trust includes attributes such as: fault tolerance, operational safety, security, usability.

• Smart and knowledge-based systems:

- Property "at any time" (guarantees the best answer that can be obtained in a given time if a response is requested within that time frame)
- Explaining ability (explains the thought process of providing an answer).

• Human Interface and Interaction Systems

- Easy to adapt to user features and interests, Smart help, etc.

• Information systems

- Easy to query
- Accuracy in providing answers (relevant information only)

Software quality features that affect the software engineering process

- Code style
- Reusability of the code
- Code modularity and module independence

Relationships between external and internal characteristics of quality

Because there is no precise relationship between the two groups of features, the quality models decompose into external, user-perceived quality and internal quality characteristics that depend on the developer.

The quality model of a software product is structured into three parts:

- definition and decomposition of external attributes of quality - consumer orientation;

- defining and classifying the internal quality characteristics - targeting the developer;
- making detailed links between external sub-attributes and internal sub-characteristics.

The internal characteristics of the product, process or resource are those that are measured by examining the product, process or resource, separate from their behavior. Internal features include: maintainability, flexibility, portability, reusability, program readability, testability, understanding, ease of building.

The external characteristics of the product, process or resource are those that are measured only with reference to how the product, process or resource reacts with the environment. External features of software quality directly affect the value of the product to the user. Achieving a product quality model involves identifying the tangible internal tangible properties of the product, measurable and / or evaluable, which have the greatest effect on external quality attributes [2], [6].

Correctness includes those properties on which the proper functioning of the product depends software. These properties are so important that they are classified separately. Correctness properties are internal, associated with individual components, or contextual, associated with how components are used in the context. Internal properties measure the extent to which a component has been developed in accordance with its intended use or how well it has been composed. Contextual properties are determined by how components are composed.

Quality models for software products

The complex character with multiple meanings of the quality concept necessarily implies a clear and operational definition of quality. This is done by defining a quality model, built by decomposing the concept of quality to the primary characteristics. As a result of the research in the field of software quality, several software quality models were proposed.

The Mc Call model groups the quality factors into three categories:

- exploitation / use with quality factors, fairness, integrity, usability, reliability;
- product revision with quality factors maintainability, flexibility, testability;
- transition produced with quality factors reusability, portability, interoperability.

The model has been developed to improve product quality and separates quality characteristics for the developer, user and reuser.

The Boehm model is one of the first software quality models where quality characteristics are determined by internal attributes, and metrics for quantification that are described.

The quality model according to ISO / IEC 9126 proposes the use of a set of six quality features: functionality, reliability, usability, performance, maintainability, portability. For each feature, a set of sub-features is detailed, and the last level represented by metrics is not standardized. In this standard, the task of defining appropriate metrics for each quality feature lies with the software developer.

The emergence of new concepts, techniques and methods of software development also determines the design of new quality models. The Dromey / Griffith quality concept involves separating software quality characteristics into behavioral features and usage features. The concept distinguishes between product components, quality attributes and quality carrier characteristics [3], [5].

From the comparative analysis of the aforementioned models we find that there are static models that do not describe how the metrics are projected from the current values to the next values of the important points of the development process. It is important for the model to link the software metrics to the expected quality at the time of delivery of the software. The models also give no guidance on how to use metrics and attributes to identify and classify risks. Although there are more visions of quality according to the participants' position in the software development and use process, the practice has demonstrated that the project manager has a very strong influence on the quality aspect in the development process.

The vision of quality project managers is pragmatic and relatively simple:

High quality software is the one that works pretty well to fulfill the function for which it was designed and is available when it is needed to perform this function. Thus, the project manager is interested in a pragmatic model of quality and in collecting a set of metrics to ensure the successful development of a specific operational system.

Based on these considerations, the Software Assurance Technology Center, SATC, has developed a project-oriented software model for the project manager.

The set of objectives selected for the SATC quality model includes both process-oriented and traditional product-oriented indicators. The SATC model also has objectives whose metrics are based on data collected from process and product rather than expert assessments.

A customer-oriented quality model focused on the degree the customer appreciates a product is the Kano Model. Using the definitions in ISO 9000, the Kano Model splits the quality factors into three categories:

- evidence - necessarily present to any product for sale, but without giving any credit to the product;
- asked - is what the consumer asks for. I am in favor of the manufacturer and the more there is, the better;
- surprise - the term is used in a positive sense;

These factors are not demanded and not expected by buyers, but their presence increases the competitiveness of the product. For the three types of quality factors there are specific discovery mechanisms and techniques:

Surprise factors that depend on market research;

Factors demanded by customers or the company's marketing department;

The underlying factors, referred to in the ISO 9000 standard. Traditional software enhancement methods do not ensure the presence of surprise factors. For this purpose, an appropriate method is needed for the software.

3. The advanced software evaluation methods

The software can be evaluated either directly or indirectly. By direct assessment of the software engineering process, it is understood that costs and associated efforts are determined. It involves calculating the number of written lines of code (LOCs), determining the execution speed, the size of the memory, and the number of defects reported over a certain time interval.

Indirect product evaluation is in fact an analysis of functionality, quality, complexity, efficiency, reliability, maintenance and many other features.

The cost and effort required to develop software, calculating the number of lines of code (LOC) and other direct estimates are relatively easy to estimate initially. However, quality and functionality or efficiency and maintenance are much more difficult to assess and can only be measured indirectly. Product evaluation methods can be described as follows:

- Productive evaluation focuses on the final results of the software engineering process;
- Qualitative assessment provides an indication of how close the software is to the customer's implicit and explicit requirements;
- The technical evaluation highlights the characteristics of the software (eg logical complexity, degree of modularisation) rather than the process by which it has been developed;

- Dimensional evaluation is used to "collect" direct assessments of the results and quality of the software engineering process;
- Functional evaluation provides an indirect evaluation;
- Human resource assessment provides information on how developers develop a software product as well as perceiving the effectiveness of development tools and models.

The evaluation methods most commonly used by software makers are:

Method of dimensional evaluation

The dimensional evaluation of the software is a direct estimation of the software as well as the process by which it is developed. If a project manager maintains simple records, a table with data ordered by the size criterion can be created. For each project, the usual dimensional data is:

- the effort estimates the need for human resources and is measured in programmers-per-month or programmers-per-year;
- Kilo Lines of Code (KLOC) - thousands of lines of code;
- value is the monetary expression of the effort;
- documentation pages;
- the number of errors reported by users over a period of time.
- the number of programmers who worked on software development.

From the primary data contained in such a table, a productivity and dimensional dimensional assessment for each project can be made:

Productivity = KLOC / Programmers per month
Quality = Number of Errors / KLOC

In addition, other interesting parameters can be calculated:

Cost = Value / KLOC
Documentation = Documentation pages / KLOC

The use of dimensional parameters (KLOC, effort, etc.) is controversial and they are not universally accepted as the best method of evaluating the software development process. The controversy revolves around the use of the LOC code lines as the main size. Supporters of the LOC variables state that this is an artifact of all software development projects and can be easily calculated that many estimation models use LOC or KLOC as the main input and that there is already a huge literature (plus associated data) dedicated to LOC. On the other hand, opponents claim that the LOC variable is program-dependent, that LOC can penalize well-designed but short programs, that it can not be easily associated with non-

procedural languages, and that its use in estimation requires a level of detail that can be difficult (Eg the project manager has to estimate the number of code lines that need to be produced long before the analysis and the project plan have been completed) [3], [6].

Functional assessment / evaluation of for software products

The parameters that functionally characterize the software represent an indirect evaluation of the software and of the process by which it is developed. By avoiding LOC calculation, functional parameters focus on the "functionality" or "utility" of the program. This type of assessment was proposed for a productivity measurement approach, called the functional score method. Functional Score (SF) is obtained using an empirical relationship based on calculable estimates of the product information domain as well as evaluations of the complexity of the application.

The values of the information domain are defined in the following way:

- Number of user entries: Each user input that provides the application with distinct data oriented to it is taken into account. Inputs will have to be distinguished by queries, which are calculated separately.
- Number of user outputs: Each output to the user, which provides application-oriented information, is taken into account. In this context, the term "output" refers to reports, screens, error messages, etc. Individual report data is not calculated separately.
- Number of user queries: A query is defined as an on-line entry that results in an immediate response of the application as an on-line output. Each distinct query is taken into account.
- Number of files: Each "master" logical file, such as a logical collection of data that can be part of a large database or individual file, is taken into account.

Number of external interfaces: All machine readable interfaces (data files on tape or hard disk) that are used to transmit information to another system are taken into account.

Once the above data has been collected, a complexity index is associated with each calculation. Organizations using the functional score method develop criteria to determine whether a particular entry is simple, medium or complex. Of course, determining complexity is a relatively subjective process. To calculate the functional score, the following relationship is used:

*SF = Total of calculation * 0.65 + 0.01 * SUM (Fi);*

Where the total-of-calculation is the sum of the partial results obtained by weighing the values of the information domain. The constant values in the above equation as well as the influence factors that are applied to the calculation of the information domain are determined empirically.

Complexity adjustment values - Fi, i = 1 ... 14 are determined by evaluating the influence of 14 factors:

- Does it require back-up and recovery?
- Are data communications facilities required?
- Are distributed processing functions required?
- Is the criterion of critical performance?
- Will the system run in an intensely operating environment?
- Does the data entry system require online?
- Does the data entry system need to be on-line, that the data input process takes place on screens or through multiple operations?
- Are the files updated online?
- Are the inputs, outputs and complex queries?
- Is the internal process complex?
- Is the code designed to be reused?
- Is the conversion and installation of the program included in the design?
- Is the system designed for multiple installations in different organizations?
- Is the application designed to facilitate the user's change and ease of use?
- Each factor is rated with a score of 0 to 5, meaning:

0 - Does not Influence;
1 - Incidentally;
2 - Moderate;
3 - Environment;
4 - Significant;
5 - Essential;

Once the functional score has been calculated, it is used in a LOC-like manner as a measure of productivity, quality, and other attributes that define the program:

Productivity = SF / Programmers per month
Quality = Defective Number / SF
Cost = Value / SF
Documentation = Documentation Pages / SF

Rating based on the functional score was originally designed to be used in business information systems. However, the later proposed extension, called the characteristic score - SC, may allow this method to be

applied to programs in the field of engineering systems. The characteristic score is appropriate for describing applications where the complexity of algorithms is high. Real-time, process-control and object-oriented applications tend to have a large algorithmic complexity and are therefore suited to evaluation by the characteristic score method [2], [5]. To calculate this score, the values of the information domain are again counted and weighted. Unlike the functional score calculation, the characteristic score takes into account yet another information area (algorithms), and the weighting values are fixed. The final characteristic score is obtained from the equation:

*SC = Total of calculation * 0.65 + 0.01 * SUM (Fi);*

The characteristic score considers a new dimension of the software, namely algorithms. Reversing a matrix, decoding a bit string, or treating an interrupt are all examples of algorithms. The characteristic and functional scores mean the same thing, namely the functionality or utility provided by the software. The evaluations result in the same value of the SF in the case of conventional engineering calculation or information management applications. For more real-time real-time systems, the score is 20-35% higher than the one calculated using the functional score exclusively. Using the functional score - or characteristic - is controversial.

The fact that SF - SC is independent of programming languages, making it ideal for applications written in conventional and non - procedural languages. They also claim that it is based on data that is supposed to be known much earlier in the project evolution, making SF - SC much more attractive as an estimate. Opponents of the idea say that the method requires little prestidigitation and that the calculation is made in part on rather subjective rather than objective data; So that information can be difficult to tighten after the events have occurred and that SF - SC does not have direct physical significance - it is just a simple number.

Techniques of Decomposition

There is a natural approach to solving problems: if the problem to be solved is too complicated, we tend to divide it into a series of sub-problems until we reach a level where sub-problems can be resolved. We then solve each of the sub-problems in the hope that solutions can be combined to form a global solution. Estimating the software project is a form of problem solving and in most cases the problem to be solved (eg developing an effort and cost estimate for a software project) is too complex to be considered as

a whole. Thus, we decompose the problem, redefine it as a collection of sub-problems with less complexity and therefore more resolvable.

The code lines and the functional score are initial data starting from which productivity can be calculated. LOC and SF data are used in two ways during the software project estimate:

- as an estimation variable that is used to dimension each element of the software;
- Basic measurements collected from old projects and used in conjunction with estimation variables to develop effort and cost estimates.

LOC and FP estimates are distinct estimation techniques. However, both have a number of common features.

The project manager begins by presenting a synthetic description of the final function of the software and, starting from this statement, tries to break down the project of the future IT product into small sub-functions that can be estimated individually. The estimation variable LOC or SC is then calculated for each sub-function. Basic productivity measures (e.g. LOC / programmers-per-month or SC / programmers-per-month) are then applied to the most appropriate estimation variable and the effort or cost of the sub-function is derived. Estimates of the sub-function are combined to provide a global estimate for the whole project. The estimation techniques LOC and SC differ to the level of detail required for decomposition.

When LOC is used as an estimation variable, functional decomposition is absolutely essential and is often driven to considerable levels of detail. Because of the data required to estimate the functional score or more macroscopic, the decomposition level when SC is used as an estimation variable is considerably less detailed. It should also be taken into account that LOC is estimated directly while SC is determined indirectly by estimating the number of inputs, outputs, data files, interrogations and external interfaces as well as the 14 complexity adjustment values described above [1], [3].

Independently of the estimation variable that is being used, the project manager typically provides a range of values for each function decomposed into the subfunctions. Using historical data, the project manager estimates a LOC or SC value for each function, in the most optimistic, most likely, and most pessimistic case. An implicit indication of the degree of uncertainty is provided when a range of values is specified.

An expected value for LOC and SC is then calculated. The expected value for the estimation variable E can be calculated by averaging the LOC or SF estimates in optimistic (a), possibly (m), and pessimistic (b) cases.

The estimate $E = (a + 4m + b) / 6$ gives the most credibility the most probable estimate (m) and follows a probability beta distribution.

Suppose there is a very low probability that current LOC or SF results are outside the range defined by the estimated values in the optimistic or pessimistic case.

Using standard statistical techniques, we can calculate estimates. It should be noted that a deviation based on uncertain (estimated) data has to be used judiciously. Once the expected E value for the estimation variable has been determined, LOC and SC are used.

4. Conclusions

The evaluation of software products remains an important activity for a company that wants to ensure a rapid growth and profit, because the benefits of good system information ensure a better record of data, the updates of different values and the interrogation through queries that are reflected in different reports. The software can be evaluated either directly or indirectly. By direct assessment of the software engineering process, it is understood that costs and associated efforts are determined. It involves calculating the number of written lines of code, determining the execution speed, the size of the memory, and the number of defects reported over a certain time interval. Indirect product evaluation is in fact an analysis of functionality, quality, complexity, efficiency, reliability, maintenance and many other features. The cost and effort required to develop software, calculating the number of lines of code and other direct estimates are relatively easy to estimate initially [2], [4]. However, quality and functionality or efficiency and maintenance are much more difficult to assess and can only be measured indirectly. A good evaluation refers to a wide range of aspects such as costs, benefits, complexity, reliability, portability and other economic aspects that may improved the quality of information systems that give a competitve advantage in the area of businesses.

References

1. A Rae, P Robert, HL Hausen, “Software Evaluation for Certification”, dl.acm.org, 2015;
2. J Guthrie, R Petty, K Yongvanich, “Using content analysis as a research method to inquire into intellectual capital reporting”, Journal of Intellectual Capital, 2015;
3. M Azuma, “Software products evaluation system: quality models, metrics and processes - International Standards and Japanese practice”, Elsevier, Information and Software Technology, 2016;

4. KE Emam, W Melo, JN Drouin, "The theory and practice of software process improvement and capability determination", dl.acm.org, 2015;
5. M Paulk, "Capability maturity model for software", Wiley Online Library, Encyclopedia of Software Engineering, 2016;
6. Weiss S. M., Kulikowski C. A., "A practical Guide to Designing Expert System", Chapman and Hall Ltd., London, 2015;
7. URI: http://www.expertsystem.com
8. URI: http://www.sciencedirect.com
9. URI: http://ecomputernotes.com
10. URI: http://ieeexplore.ieee.org