Software Quality Evaluation and Assurance: Analysis, Tendencies, Problems

Tetyana Hovorushchenko

System Programming Department, Khmelnitskiy National University, Institutska Str., 12, Khmelnitskiy, 29016, UKRAINE,

e-mail: tat_yana@ukr.net

Abstract – The authors have analyzed the impact of the software quality for work of complex hardware-software systems, the software quality standards, the models, methods and tools of software quality evaluation. The analysis provides to discover an important tendency - the need of software quality evaluation and prediction at the early stages of the life cycle. The authors have proved the subjective dependence and artificial adaptable of used concepts and the lack of mathematical models, theory and methodology in the area of software quality evaluation and assurance.

Key words – software, software quality, software quality evaluation, software quality assurance.

I. Introduction

Today production of software is the largest sector of the world economy, which employs about three million professionals (programmers, software developers, etc).

"Volume of economic costs due to faulty and defective software in the U.S. reaches billions of dollars per year, representing about 1% of the national GDP" - Report of the USA National Institute for Standards and Technologies [1].

Until the development of quality software is not the norm, and general technologies, by which developers can build reliable and safe software with corresponding cost and within a specified time, don't exist. Sources of modern software faults are extremely diverse, and it only complicates the problem and increases its size and cost.

During the last years the software industry has reached the level of development, in which quality requirements are compulsory point of the contract of software developing, because software quality is its most important feature from the user perspective.

Software Quality Assurance is the problem, whose solution requires the comprehensive research in the following areas:

1) development of tools of software quality analysis and evaluation at different stages of its lifecycle;

2) definition and management of the parameters that affect the software quality at all stages of its lifecycle.

II. Problems of Software Bugs Identification at the Different Lifecycle Stages

The crisis in the field of software quality assurance was evident even 50 years ago - developed product lacked the necessary functionality, its productivity was poor, low quality software did not satisfied the customers.

The several methods and tools are available, the best specialists are involved in the development of technologies and standards to ensure the software systems quality, but software quality is still dependent on the knowledge and experience of developers.

Currently, significant and inalienable feature of software systems is their complexity. The growing complexity of software functions inevitably leads to an increase of their volume and development complexity.

The typical density of bugs of different software are represented in Table 1 [2].

TABLE 1

TYPICAL DENSITY OF DIFFERENT SOFTWARE BUGS

Software Size	Typical Density of Bugs
Less than 2K	0-25 bugs per 1000 lines of code
2K-16K	0-40 bugs per 1000 lines
16K-64K	0,5-50 bugs per 1000 lines
64K-512K	2-70 bugs per 1000 lines
More than 512K	4-100 bugs per 1000 lines

From Table 1 it follows that the current software up to millions of lines of code in principle can not be infallible. The need software quality is ensured with the fact that some unknown number of software bugs and defects always remains, and their negative effects should be blocked or reduced to an acceptable level.

According to approximate estimates the cost of software development is about 275 billion dollars, but only 72% software projects reach the implementation stage and only 26% software projects completed successfully [3]. Software projects often fail because of inadequate requirements formulation, design failure or ineffective planning, incorrect understanding or inadequate analysis of the specification and the project, because of bugs at the early software lifecycle stages.

The analysis of built-in software bugs and their consequences are represented in Table 2.

TABLE 2

ANALYSIS OF BUILT-IN SOFTWARE BUGS AND CONSEQUENCES

Event	Reason	Consequences
Requirements Formulation Stage		
In 1971 "Cosmos-419" did not start to Mars	Specification infidelity	Loss of device
In 1971 the station "Mars 2" could not undock from the ship	Specification infidelity	The task was not completed
Chinook helicopter crash in 1994	Specification infidelity	29 people were killed
"Death" sessions of radiation therapy with Therac-25	Specification incomplete	6 patients received a lethal dose
Explosion of rocket Ariane 5 in 1996	Mismatch of requirements	9.5 billion $

Design stage		
"Death" sessions of radiation therapy with Therac-25	Errors in the project; incorrect risk assessment	6 patients received a lethal dose
Processor Intel Pentium Error in 1994	Deviations from specifications	Loss of company were 475 million $
Accident of station Mars Climate Orbiter and the disappearance due to the Mars Polar Lander in 1999	The error in the project through the use of different measurement units	327.6 million dollars – apparatus, 91.7 million dollars - launch
Emergency drop of rocket "Rokot" in 2005	Logic error in the control system algorithm	Loss of European research satellite CryoSat
Crash of airbus A330 in October 2008	Error in the system data processing algorithm	110 passengers and 9 crew members were injured
Fall rocket "Zenit-3SL" in 2013	Wrong project of control system	Loss of satellite "Intelsat-27"

The analysis of application software bugs and their consequences are represented in Table 3.

TABLE 3

ANALYSIS OF APPLICATION SOFTWARE BUGS AND CONSEQUENCES

Event	Reason	Consequences
Requirements Formulation Stage		
The failure of New York Bank system	Lack of memory due to incorrect requirements	Loss of 32 billion dollars
In 1990 AT & T held the 9-hour accident	Problems with boundary conditions	75 million unrealized calls, loss of 60 million dollars
In 1998 AT & T held the 26-hour accident	Problems with hidden boundary conditions	Disability of services associated with data transmission
Error Y2K - Error of the second digits saving in the year of the date (1999)	Infidelity or incomplete specifications	Loss - 500 billion dollars
Design stage		
In 1983, at the station "Serpukhov-15" detection system false worked	Recognition system is incorrectly designed	The world was on the brink of nuclear war
In 1991, the Patriot missile defense system was not intercepted Iraqi missile	Rounding error - incorrect calculation of approaching missiles location	28 American soldiers were killed and about 100 people were injured
Error Y2K - Error of the second digits saving in the year of the date (1999)	Invalid project	Losses of 500 billion dollars
"Lethal" therapy at the National Cancer Institute in Panama City in 2001	Incorrect calculation of radiation doses in the software of Multidata Systems International company	28 patients were suffered excessive exposure, several patients died
The accident at the uranium processing plant in Western Australia in 2001	The logical error in the algorithm	Releases of radioactive materials

Analysis of Tables 2 and 3 shows that many bugs of built-in and application software, that caused the crash and incidents, were made at the early lifecycle stages.

Table 4 shows the percentage of bugs, that occur at the stages of requirements formulation, design and implementation of various software [2]. Distribution of the bugs, implied at the various lifecycle stages are represented in Table 4 [2].

TABLE 4

DISTRIBUTION OF BUGS, IMPLIED AT THE VARIOUS STAGES

Lifecycle Stage	**Software Volume**				
	2K	**8K**	**32K**	**128K**	**512K**
Require-ments	Up to 10%	Up to 15%	Up to 20%	Up to 22%	Up to 23%
Design	Up to 15%	Up to 19%	Up to 25%	Up to 28%	Up to 32%
Coding	Up to 75%	Up to 66%	Up to 55%	Up to 50%	Up to 45%

Table 4 shows the requirements formulation bugs and design bugs are 25-55% of all bugs, and the greater the volume of software, the more bugs introduced at the early stages.

The cost of the design bug correcting in 2-4 times higher than the cost of the coding bug correcting. Correcting of bug before the construction (coding) is in 10-100 times less than its removal at the end of the project, during testing or after release. The dependence of the bugs correcting cost from the lifecycle stage are represented on Fig.1 [4].

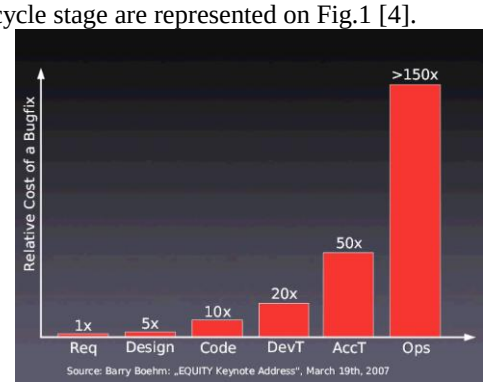

Fig.1 The Dependence of the Bugs Correcting Cost from the Lifecycle Stage

The longer a bug remains in the chain of software development, the stronger it gets into other parts of the software, the greater it caused the harm to the next stages. The policy of early bugs correction may be several times lower financial and time costs for the software developing, that is a strong argument in favor of early detection and removal of bugs. "Increased attention to early quality control can significantly reduce the bugs rate, but does not increase the total cost of development" - NASA Software Design Laboratory [2].

All studies in the field of software quality assessment at the early lifecycle stages are chaotic and non-systematic. Of course, there are some basic research (works of Boehm, Dijkstra, Meyer), but *there is no complete, tested and approved theory and methodology for the development of quality software, and methodology for assessment and prediction of software quality at the early lifecycle stages*.

Consequently, the industry of software quality assessment requires major changes, otherwise the world will continue to expect man-made disasters, caused by software errors.

III. The Analysis of Software Quality Standards

Software organizations must be guided by the standards on the processes on development and on the processes of quality evaluation and assurance. Standards provide the communication between developers and users and should contain the previous experience of the developers. But the long-term development of standards, their untimely updating and improving lead to inhibition of technologies, gap with practical needs and prevent the introduction of innovations.

The main criteria for the examination of standards in the software development and quality evaluation [5]: 1) arbitrariness of standard interpretation for conformity assessment; 2) relevance given to product requirements analysis; 3) process conceived as an asset of organized and reusable practices (it survives across projects); 4) relevance given to management practices; 5) means to keep pace with evolving technology; 6) relevance given to system theory and system engineering culture; 7) relevance given to safety culture; 8) relevance given to human factor; 9) separating abstraction levels in standard; 10) introduction of integrity (SIL) or criticality levels; 11) definition of purpose of the norm and stakeholders; 12) properties of components and systems vs. properties of functions (safety, SIL, reliability); 13) support given to independent assessment of certification; 14) relevance given to system operational phase, including human-system related processes.

In [5] for each of the criteria using a 5-point scale (0 to 4 points) standards [6-8] has been evaluated and founded that the standard [6] has 25 points, the standard [7] - 32 points, standard [8] - 10 points. This analysis showed that the standard [6], published in 2008, worse by the main criteria than the standard [7], published in 2003. Not always the updated standards take previous experience of developers and most innovative technologies.

Some current standards don't fully meet the modern software requirements and all user needs. Now more standards were created, they different standardize and regulate the same processes (usually routine mass processes). As a result the incomplete coverage of the standardization objects, incompatible of regulations of various organizations, lobbying of interests of individual software developers, adaptation of standards to developers needs arise.

The way out of this situation is using of the documentation of one or two developers, particularly for software quality evaluation and assurance standards ISO and IEEE will be considered.

Software quality is the software characteristic, that reflects the degree of software compliance to requirements, the suitability of software to meet specific needs. *Software quality characteristic* is the set of software properties, by which the quality is described and assessed [9]. *Software quality index* is the software quality characteristic, which is to be precise description and measurement and has a quantitative value [9].

Standard [10] combines the characteristics from various software quality standards. Software quality (Q) is the function of the six basic quality characteristics: functionality (F), reliability (R), usability (U), efficiency (E), maintainability (M), portability (P): $Q = f(F, R, U, E, M, P)$.

As today is measured the quality characteristics? Software quality indexes are measured - the metric is chosen, the scale of assessment is calibrated depending on possible degrees of compliance of index and restrictions. Set of "measurable" indexes is a criterion for assessing of charateristic.

BUT! Selection of metrics to software quality indexes evaluation is done subjectively, because thousands of metrics were created, but the only standards for their selection and using are not available. *Interpretation of the metrics values is also subjective*, because standardized «etalon» values of metrics are not available. *Evaluation scale calibration is also subjective*, because it depends on possible degrees of compliance of index and restrictions, and degrees of compliance are not standardized and are determined by the software organization.

Therefore, *evaluation of software quality* (as a function of the main characteristics) *is subjective*, because the software organization chooses its favorable metrics, interprets the choosing metrics values as maximum, calibrates the assessment scale of each characteristic on the basis of its own interpretation of the metrics values. As a result software company obtains maximum values of each characteristic, and accordingly the maximum value of software quality. In fact, *there is a formal satisfaction of software quality*.

In addition, *comprehensive methodologies*, that provide the evaluation of not only the impact of each characteristic on the software quality (this issue is devoted to the series of works), but also *will provide the evaluation of characteristics interference, are not available*.

IV.Software Quality Evaluation

In the research of software quality models (SquaRE [10] and CMM [11]) the following problems were identified: 1) immaturity of quality measurement technology; 2) lack of details and the possibility of different interpretations of standards and quality models based on auditor perceptions; 3) inaccurate assessment of processes quality by SQuaRE-model, involved in software designing and implementing; 4) lack mechanisms, that contribute to the improvement of existing processes, in SQuaRE model; 5) CMM model is the property of Software Engineering Institute (SEI) and are not generally available, so further development of model is no involvement of programmer community; 6) CMM model is focused on the using at the large software companies.

Methods of Software Quality Management [12]: 1) Static techniques; 2) People-intensive techniques; 3) Analytical techniques; 4) Formal techniques; 5) Dynamic techniques.

Research of software quality assessment and management methods showed, that *evaluation of software quality doesn't provide the fundamental methodology*. All developed methods are disparate, are used at the discretion of developers, because clear criteria for the method using in each concrete case are not available.

Common drawbacks of software quality assessment tools: 1) subjective dependent of selection of metrics that the tool of quality evaluation automation will form; 2) subjective interpretation of metrics values, because the etalon metrics values are not available; 3) focus of automation tools on source code testing and metrology, instead of software metrics calculations at the design stage, when finished source code is not available, but informative, functional and behavioral models of requirements analysis are available; 4) all existing automation tools are aimed for maximum quality assessment, but not for improving or assurance quality.

Existing models, methods and means of software quality evaluation are inefficient at the design stage and don't meet modern requirements for software.

Acknowledgments

Analysis of the impact of quality on the complex hardware and software systems work have found important tendency, that must be considered when software quality evaluating and ensuring: *project quality evaluation and prediction of developed software quality at the design stage will provide the early bugs detection, that will provide software quality increasing and the cost of its development reducing.*

Actuality of the software quality improving problems *necessitates the development of fundamental theory and methodology of software quality evaluation and assurance (guarantee).*

Research of software quality evaluation models and methods revealed the following specific problems:

1) lag and ineffective of quality evaluation methods and tools caused by neglect and dissatisfaction with their current requirements for software;

2) some restrictions when using of methods for processes of development and software quality evaluation due to lack of methodology and criteria for methods selection;

3) subjective dependence and imitation adaptability of used concepts in the field of software quality evaluation and assurance;

4) lack of software quality evaluation and assurance models, which will consider the impact of various factors in quality management;

5) *lack of theory and methodology in the field of software quality evaluation and assurance, which will ensure the same quality software with corresponding cost and within a specified time with the use of the same development technologies and the same standards.*

Further efforts of authors will be directed to the solving of the aforementioned problems.

References

[1] The Economic Impacts of Inadequate Infrastructure for Software Testing Research. - Triangle Institute, NIST Planning Report № 02-3, RTI Health, Social, and Economics Research

[2] S.McConnell. Code Complete - Microsoft Press, 2013 - 896 p.

[3] V.Mishchenko, O.Pomorova, T.Hovorushchenko. CASE-assessment of Critical Software Systems. Volume 1. Quality / Kharkiv: The National Aerospace University "KhAI", 2012. - 201 p. [in Russian]

[4] Cost of a bug within a software lifecycle // http://www.testically.org/2012/02/09/cost-of-a-bug-within-a-software-lifecycle/

[5] M.Fusani. Examining software engineering requirements in safety-related standards // Radioelectronic and computer systems – Kharkiv: NAU "KhAI", 2009 – № 7, pp.268-274

[6] ISO/IEC 12207 FDAM Information technology – Software life cycle processes / ISO/IEC, 2008

[7] ISO/IEC 15504-2 Information technology -- Process assessment - Part 2: Performing an assessment / ISO/IEC, 2003

[8] ISO/IEC 9126-1 Software engineering- Product quality- Part 1: Quality model / ISO/IEC, 2001

[9] IEEE Std. 1061-1998 IEEE Computer Society: Standard for Software Quality Metrics Methodology, 1998 - 20 p.

[10] ISO/IEC 25010:2011. Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models / ISO/IEC, 2011

[11] Chrissis M. CMM: Guidelines for Process Integration and Product Improvement / M. Chrissis, M. Konrad, S. Shrum. - Addison-Wesley Professional, 2006

[12] Guide to the Software Engineering Body of Knowledge (SWEBOK) - A project of the IEEE Computer Society Professional Practices Committee, 2004