Problems of Evaluating the Software Quality for Systems Important for the Nuclear Power Plant Safety

E.Ph. Jharko
V.A. Trapeznikov Institute of Control Sciences of the Russian Academy of Sciences
zharko@ipu.ru

Abstract—During the last decade, automated process control systems (APCS) have approached a qualitatively new level of the development; and a consequence of this progress is considerable complicating the APCS software and its life cycle. In 2000, at the V.A. Trapeznikov Institute of Control Sciences of the Russian Academy of Sciences (ICS RAS), software complexes for perspective nuclear power plant (NPP) APCS has been elaborated. The required life cycle of the total APCS is not less than 30 years, what leads to a necessity of developing and implementing comprehensive procedures on maintaining and modification of the software complexes. In the paper, an experience gathered in the ICS RAS on maintaining NPP APCS software, assuring quality of the updated software code and forecasting of costs of its modification and maintenance is presented.

Keywords—test and documentation, monitoring of product quality and control performance, control system design, software quality, automated process control, safety, nuclear power plant.

1. INTRODUCTION

Automated process control systems (APCS) have reached a qualitatively new level of their development concerned with growth of the level of automation of industrial plants and, as a consequence, with the growth of the number of diagnostic and controlling signals processed by the system per time unit. From another hand side, practically linear growth of the computer system capacity, that may be used within the APCS, has enabled one to implement considerably more complicated algorithms of control and data analysis by use of complex soft/hardware tools. However, the qualitative jump appeared in the make-up of the solved tasks, made one to revise the relationship of the components of their life cycle.

These changes are explicitly traced by an example of developing software for the nuclear power plant (NPP) APCS with the required lifetime of the total NPP APCS being not less than 30 years. This considerably exceeds the average lifetime and shelf life of the hardware, and enforces to pay more attention to thorough elaborating the stage of modification and maintaining the software developed.

In advanced NPP APCS, software (SW) is applied everywhere, starting with controllers and completing with common plant-level systems intended to organize performance of multi-unit NPP as a whole. NPP APCS of the Russian manufacturing, that are built both in Russia and ones supplied abroad, are not an exclusion.

Software quality assurance is a persistent process within all the life-cycle of the software, that covers:
• Methods and tools of analysis, design, and coding;
• Technical reports implementing at each step of software development;
• Procedures of multi-level testing;
• Monitoring software documentation and changes introduced into it;
• Procedures of assurance of correspondence to standards in the field of software development, correspondence to which is defined in the assignment on development of given software;
• Algorithms of measurements and forming reports.

The software quality may be defined as correspondence to explicitly set functional and operational requirements, correspondence to explicitly indicated standards of the development, and correspondence to implicit characteristics that are expected from professionally developed software. Such a definition of the quality underlines three important circumstances:
• Requirements to the software is the basis with regard to which the software quality is determined;
• Indicated standards define a set of design criteria, that define the style of software development;
• There exists variety of implicit requirements which are not frequently mentioned (for instance, maintainability, modification ability). If some software corresponds to the explicit requirements to its development, but is not in position to meet the implicit requirements, then the software is doubtful.
These circumstances are most brightly traced with regard to software of high reliability systems that involve subsystems of NPP APCS, since, besides complete correctness, the software possesses other characteristics being of interest to a consumer of this software, such as absence of errors under execution, data integrity, temporal characteristics, accuracy, correctness of types, completeness, functional reliability, safety, maintainability, intelligibility, modification ability, and others.

2. CLASSIFICATION OF SYSTEMS IMPORTANT FOR THE NPP SAFETY

In accordance to the international classification [1], one emphasizes systems important for the NPP safety with regard to functions implemented by these systems:

**Category A** involves functions that play the main role in achieving or maintaining of the NPP safety in order to prevent growth of accidents up to inadmissible subsequences;

**Category B** involves functions that play some supplementary role with regard to the functions of the category A in achieving or maintaining the NPP safety, especially those functions that are required for performance after achieving a controlled state in order to prevent growth of design envisioned events up to inadmissible subsequences or to moderate subsequences of the design envisioned events.

**Category C** involves functions that play an auxiliary or indirect role in achieving or maintaining the NPP safety.

Basic principles of development of control systems important to the NPP safety have found their reflection in international standards [2, 3]. There is no unique classification of NPP systems. Table 1 presents comparison of classes of NPP safety systems, presented in various regulatory documents. In dependence on the safety class, software, developed for these systems, is imposed by constraints concerned with applicability of operation systems, programming languages, details in documentation, etc.

### Table 1. Comparison of NPP safety classes presented in different documents

<table>
<thead>
<tr>
<th>Standard or regulatory document</th>
<th>Safety classes (the degree of importance is increased from the left to the right)</th>
</tr>
</thead>
<tbody>
<tr>
<td>PNAEG-01-011</td>
<td>Class 4</td>
</tr>
</tbody>
</table>

The NPP APCS make-up involves systems of the 2\textsuperscript{nd}, 3\textsuperscript{rd}, and 4\textsuperscript{th} safety classes in accordance to [4], or, in accordance to the international classification [1] (see Table 1), systems of classes A, B, C. Thus, under development of software for NPP APCS subsystems, one should follow standards [2] (for systems of class A), [3] (for systems of classes B, C).

3. DEFINITION OF THE SOFTWARE QUALITY

Definition of the software quality helps to:
- Evaluate software products;
- Evaluate principles of software organization;
- Improve processes of software creation.

One differentiates the following stages of evaluating the software quality:
- defining characteristics of the software product quality;
- developing indexes to determining quality characteristics;
- recording the values and comparing with preceding values;
- introducing changes into the software to improve its quality.

Figure 1 represents the process of “measurement” of a software product. It is impossible to measure some characteristics directly, but these may be measured via the quality indexes. Figure 2 represent the dependence between the characteristics and quality indexes.

![Fig. 1. The process “measurement” of a software product.](image-url)
Fig. 2. The interconnection between the indexes and characteristics of the quality.

Conditions to be met under measuring the software characteristics via the quality indexes are as follows:

- The quality indexes are to be measured accurately;
- A dependence between indexes and characteristics of the quality is to exist;
- The dependence is to be expressed as a formula or model;
- The quality indexes are of two kinds: management indexes and prediction indexes.

The control indexes are used by the staff to manage the process of software development. These indexes provide information on the process quality. These are not representative for software, i.e., any processing manufacture may be managed and monitored via such indexes. Management indexes involve, for instance, the effort, consumed calendar time or activity factor for partial tasks or activities, percentage of checked operands, etc.

The prediction indexes determine product characteristics that predict the product quality. The product characteristics are predicted if indexes corresponding to them are determined. One differentiates two types of the prediction indexes: dynamic and static indexes.

The dynamic indexes collect measurements made within the process of a software code execution for effectiveness and reliability. These are closely related with the software quality and are comfortable to be determined. The effectiveness may be calculated on the basis of measuring the time of execution, while the reliability may be calculated on the basis of the number of system faults and types of the faults.

The static indexes collect measurements made in the process of presentation of the system for evaluating the complexity, clarity, and maintainability. These indexes have indirect relation to the software quality, which assumes existence of the dependence between the characteristics and indexes of the quality.

Let us consider experience gathered during more than ten years of modification and maintenance of complex highly reliable software created at the V.A. Trapeznikov Institute of Control Sciences for a top unit-level system of APCS of built and modified nuclear power plants [5, 6].

Let us formulate basic notions used further in the paper. Modification is alteration in already coordinated documentations (a program code).

Maintenance is a modification of a software product concerned with correcting errors, improving functionality and capacity of the software product, or with changing the environment in which the program code run is implemented. The maintenance and, correspondingly, modification may be of several types:

- Correction (type A) is the works concerned with a necessity of correcting errors in a previously elaborated software;
- Adaptation (type B) is the works concerned with changing environment conditions in which the program code runs;
- Improvement (type C) is the works concerned with improving the existing functionality and adding a new one of the software.

4. THE SOFTWARE MAINTENANCE

NPP APCS software is a complex SW with the volume of the code exceeding 800 MB, involving more than 100 software components. The procedure of its modification and maintenance is to assure preserving the high quality of the provided functionality and compatibility with the application software after implementing the modification.

For developing and maintaining the NPP APCS subsystems software, the “standard” model of the software life cycle [7] have been used.

The stage of the life cycle concerned with the maintenance of the software, from a procedural point of view, is very close to developing properly a new software, being, in a certain sense, a recursive procedure within the scopes of the life cycle and may be separated onto the following phases:

1) Problem identification;
2) Problem analysis (technical assignment);
3) Technical design;
4) Developing;
5) Testing and quality assessment;
6) Delivering (the modified software).

Phases 3, 4 and 6 are, in entity, identical to the corresponding phases of developing new software. To test and assess quality of the modified software, there are used methods applied to assess the quality of a newly developed SW [6, 8]. Further, consider in details the job make-up within implementing phases 1, 2 and 5.

3.1. Problem identification

Actions implemented at the phase of the problem identification of the stage of maintenance are similar to those ones implemented for new software. Within both the cases the basic initial data are existence of a problem which is intended to be removed during the work implementation. However, the
maintenance stage is usually characterized by a better specific description of the problem as a result that both the software developer and user possess an experience of work in the given subject area.

The identification phase should be preceded by an activity concerned with accounting non-conformities in the software performance (the notion of non-conformity should be understood in a wide sense and include all the types of maintenance, A to C). This activity is one of the key ones to reach the high quality of the software product [9]. Remarks on the software performance are appeared as a card of accounting remarks. In the document with indication of remarks appeared, characteristic indicators of faults are involved: problem isolation, author of the program code containing the error, time period required to insert corrections into the program code, what has caused arising of the given problem, what has caused its detection.

As a result of implementing the identification phase, issuing a request on modification containing a list of remarks that are proposed to be removed under the software modification is to be. Issuing the request on the modification has also a purpose to accumulate and group the non-conformities revealed in accordance to the features to simplify implementing the next stage of the software maintenance. As a classification system, the so called “Scheme of classification in accordance to the attribute” [10] is applied. Following to this scheme, the data over the cards of accounting remarks were classified in accordance to the following attributes:

- Category (data management, defining initial data, system (kernel of the operation system));
- Type elucidating the category (for instance, definition of the data type);
- Code availability (the non-conformity has been caused by availability of an odd code, absence of the code, coding error).

Introducing the classification presented has enabled one to specify the cause and situation of arising the non-conformity more exactly.

### 3.2. Problem analysis

The phase of the problem analysis is implemented on the basis of the request to the modification and evaluate the possibility and necessity of removing each of the remark listed in the request.

Based on the analysis of the request on modification, a technical assignment is issued, containing the list of requirements which are to be met within the software modification process.

The most difficulty for developers during implementation of the phase of analysis is concerned not only with the technical realization, but also with the labor expenditures of the modification implementation. Besides direct introducing changes concerned with the realized (new) functionality, one should take into account the expenditures on checking influence of the changes introduced on the adjacent fields of the modified software and already implemented in the original (non modified software) functionality. Simultaneously, under the expenditure evaluation (including the testing expenditures), one should consider the following factors:

- **The degrees of modularity (independence of components) of the software subject to modification.** As a module, a system unit is understood, which implement some independent function in the software make-up. The more apart over the module groups the functionality subject to modification is distributed, the less expenditure are required to test the adjacent modules.

- **The size of the code in each of the modules.** These may be essentially differed from each other, and, consequently, the modification complexity depends on that how much “complex” modules are involved into the modification process. If the modification is applied to a code possessing lower complexity, the easier is implementing analysis of the modified module, the less are expenditures on testing the realized functionality.

To evaluate the software complexity, a number of metrics have been developed [11, 12], but their applying, basically, reflects the subjective choice of the expert rather than advantages or disadvantages of given metrics. To evaluate the module (a group of modules) complexity, one may use a metrics based on calculating the sum of the code line (CL). However, as it has been shown in paper of [13], the program code size in accordance to the metrics CL is not a stable characteristic reflecting the program code complexity. We have used also two other metrics: the cyclomatic complexity (CC) [14] and the metrics of the expended efforts (EE) [15]. To calculate the latter metrics, one does not require a profound structural analysis of the code, and the metrics enables one to evaluate the number of errors in the code and the expenditures on its modification. So, this metrics has been selected as the basic one, although for compact modules, it is more acceptable the complexity evaluate using the metrics based on the module cyclomatic complexity.

### 3.3. Testing and quality assessment

Quality assessment, as a constituent part of testing and verification of the software, is implemented during all the life cycle of the software. In accordance to standards [1-3], one of the most important metrics of the software quality, affecting the maintenance and modification, is the clarity of the source code. Figure 3 represents an example of analysis of a functional module in the part of the “source code clarity”. The metrics “Clarity” is characterized by: Total comments, Comments in Headers, Comments in Declarations, Total Comments/Exe. Lines, Average Length of Basic Blocks, Expansion Factor and etc.
Fig. 3. Analysis of a functional module in accordance to the quality metrics “Clarity”.

For the software quality, the testability and maintainability of software (functional) modules are of the same degree of importance. Figures 4 and 5 represent an example of analysis of functional modules in the part of the “source code testability” и “source code maintainability” correspondingly. The metrics “Testability” is characterized by: Knots, Cyclomatic Complexity, Executable reformatted Lines, Number of Basic Blocks, Total Operands, Unreachable Lines, Unreachable Branches, Number of Loops and etc. The metrics “Maintainability” is characterized by: Essential Knots, Essential Cyclomatic Complexity, Cyclomatic Complexity, Vocabulary, Unreachable Lines, Unreachable Branches, Number of Procedures and etc.

Creating qualitative complex software is not possible without wide-ranging testing. From one hand side, any software application, due to its implementation of some known algorithm of performance, may by completely tested under limitation of the input data set. However, this statement is valid for only trivial or some special cases of the software. Sources provided for software testing in majority of cases are insufficient to provide the complete test coverage of the program code. So, a task arises to classify parts of the software code in order to elicit parts containing errors with a maximal probability.

4. CONCLUSIONS

The software quality may be considered as “sufficiently good”, when potentially positive results of creation and use of the developed software are suitable and exceed potentially negative opinions of customers. Such an approach tests, from the point of view of the traditional notion of the software quality, various variants of implementation. Under the approach, strict, but not checked, requirements are substituted by optimal ones. This approach is focused on indentifying problems and improving possibilities of the decision making. Thus, the project of development of software for systems important for the safety is to be, basically, problem-oriented rather than software quality goal-seeking. Also, one may say that the software quality, in accordance to the notion “sufficiently good”, is the optimal set of solutions for this series of problems. Such a way of interpretation is to coordinate the considered problems, elaborate compromise
variants, opposing them to corresponding life-cycle process [7].

The required software quality is hard to be achieved. The process of obtaining the required software quality concerns with the process of development, methods and management of the process. The software quality is achieved due to applying the methodology of development and using verification and validation methods during the life-cycle of software of systems important for the safety. Figure 6 represents the place of software verification and validation within the context of quality assurance and hierarchy of standards.

Fig. 6. The place of software verification and validation in the software quality assurance of systems important for the NPP safety.

REFERENCES