SOFTWARE RELIABILITY. WHAT IS IT?

Dmitry A. Maevsky, Igor A. Ushakov, Ludmila N. Shapa
•
Odessa National Polytechnic University, Odessa, Ukraine
e-mail: Dmitry.A.Maevsky@gmail.com

ABSTRACT

This article is about how the confusion of terms can give rise to confusion in the approaches and methods, and bring to a deadlock the whole research area. It doesn't mean that we would like to teach someone or to present "the ultimate truth". It is rather another attempt to declare that "software reliability» does not exist in the form it is treated in most of "traditional" sources.

1 INTRODUCTION

More than 40 years have passes since the term "software reliability" firmly entrenched in the everyday life of modern science. In the early seventies of the last century, programming started to convert "from an art to a science." The growth of software (SW) complexity and its extensive penetration into all spheres of human life have shown that its errors can lead to serious financial losses, human casualties, technological and environmental disasters. In this regard, there arose a need of prediction of software failure and modeling of their consequences that led to the rapid development of a new field of science. In [1, 2, 3] the models of software reliability designed to analyze and forecast the process of defect detection in software products were first proposed.

Over the last years a large number of reliability models have been created. Valuation models allow assessing the approximate number of defects in them on the basis of special metrics before testing or operation of the program. Predictive models based on the study of the process of identifying defects allow to foresee of the development of this process over time, and to obtain estimated reliability of the program. The most comprehensive and systematic presentation of the modern theory of software reliability with a description of the existing models can be found in [4]. However, as the experience shows, none of these models can claim to be a universal one, each model "serves" its class of software systems, boundaries between the classes remain very shaky.

At present, the reliability of software is considered as a special case of the general reliability theory. Practically all reputable textbooks on reliability certainly contain the relevant section with regularly repeated attempts to use the same mathematical tools as for the analogous process in technical systems for the description of defect detection in software. Nevertheless, in the authors' opinion triumph on this way has not been reached and is not expected to occur. If you pay attention to the essence of scientific publications on the theory of software reliability, we can see that in the last twenty years, no fundamentally new ideas in this area have been proposed. In [5] the attempts of mechanical introduction into the existing variable models, designed to take into account the secondary defects, were made. However, according to the authors [5], it is practically impossible to introduce these variables into some models and for some of them such complicated mathematical expressions are obtained, that their practical application becomes difficult. It seems to us that all of the above-stated just give evidence to the above mentioned dead-end, the theory of software reliability has come to.

To find a way out of the dead end, it is necessary to go back to the roots - basic principles, which laid the foundation of a new theory forty years ago. We should find out whether the principles themselves turned out to be false, or incorrect premises were made out of the right principles. It's impossible to break the deadlock without such seditious attacks on fundamentals and recognized experts, only by hanging another patch on an improperly working model.

At the dawn of the development of software reliability theory the phenomenon of obtaining “wrong result” from the program, by analogy with the theory of complex systems reliability, became known as "failure". The first step towards the truth has been made. It was followed by the second one - the ability of a software system to operate without failures was called "reliability." A theory of reliability of complex systems by that time was, first - sufficiently well developed, and second - tested through practice. Therefore, the implementation of methods and mathematical tools of the existing reliability theory to a new physical phenomenon was absolutely natural.

Talking of criticism of modern software reliability, we are not the first here. The inherent differences between software and technical systems were once and again referred to by many authors [6, 7, 8, 9]. Getting back to the roots we should find out whether the use of the term "reliability" to the software was competent. Now, let’s talk about terms.

2 WHAT IS “RELIABILITY”?

In theory of technical systems reliability the term "reliability" is defined as a property of the system to perform reliably the functions it is designed to. Consequently, the inherent attributes of reliability are:

1. The time in which the system performs the specified function.
2. Randomly occurring system element failures, leading to a definite inefficiency of functioning or even to a complete cessation of functioning of the system.

Technical systems are subdivided into recoverable and unrecoverable. The first include, for example, vehicles, air traffic control systems, etc., and the second - a system designed to perform a single mission (for example, from household light bulbs to satellites).

The random nature of failures in the system leads to such natural probabilistic characteristics (parameters) as the average run time, the probability of failure-free operation for a specified time, and the availability factor. With the appearance of the theory of software reliability the direct transfer of the concept of complex systems reliability on the software systems was, of course, the easiest way. For example, the standard [10] defines software reliability as the probability of failure-free operation for a specified period of time in an appropriate environment. Similar definition is in monograph «Handbook of Software Reliability Engineering» [4]. Herewith, software failure itself is treated as an event where the given result is different from that which would be expected from an ideal program. Most often the fact of software failure is actually determined in a forensic way, with the help of a man - an operator. If there were no operator control, the development of operated that way software process could become unpredictable.

Further, the failure in the existing theory of software reliability is thought of as a random event. However, its randomness of this event is placed in question by many authors [6, 7, 8, 9]. In particular, in [6] it is noted: «… Errors caused by software have no stochastic nature: they will repeat as soon as some conditions will be repeated. Errors of software, in a sense, are not “objective”, they depend on type of operations, type of inputs and, at last, on type of users. … Errors caused by software do not depend on time in a usually understandable way: if you don’t use software it cannot fail! At a pinch, in this sense software can be compared with a spare unit which can be used but nobody knows the timing of this usage. At last independence of errors. There is no such concept as a “sample” for software: there is a phenomenon of cloning. “Replacement” of “failed” software has no sense! You will change one Mollie for another Mollie with the same genes, with the same illness, with the same properties».

Six years later, in [7] the author draws the attention again to the impossibility of mechanical transfer of the reliability theory fundamental principles of technical systems to the software. The chapter on the software reliability ends like this: «… attempts to put “hardware reliability shoes” on “software legs” are absolutely wrong and, moreover, will lead only to a logical dead end”. Thus we can come to a conclusion that for the six years between the publications of the said articles, the theory of software reliability has not undergone major changes, and the term “reliability” as it is understood in the technical systems cannot be used for software.

Its reason lies in the fact that stochastic apparatus of reliability theory is practically fully used for the description of processes in software. But having questioned the random nature of software failures, we automatically put into question all modern probability theory of reliability as well! Is it good or bad? We consider it to be very good. In the modern theory of software reliability there are more questions than answers. Therefore, the rejection of willful false assumptions may allow the creation of a new, more reasonable and “workable” theory of software reliability..

3 WHAT IS “SOWTWARE”?

Let’s study the notion of “software” or “program”.

The international standard ISO/IEC 2382-1 in the section “01.01.08 . Software” indicates that the software is “Any part of Programs, Procedures, rules and associated documentation of an Information processing system”. Thus, the reliability of the software – means “the reliability … of programs, procedures, rules and documentation”. Already here, in this only line there is so much confusion that such document should be treated either as the Scriptures (taking all without hesitation and doubt), or we should try all the same to critically evaluate the resulting. After all, it turns out that when talking about the reliability of software, we talk about the “failure-free” procedures, rules, and even the documentation! And what, if I may ask, is “failure-free documentation” or “MTBF”? So where is, I beg your pardon, a coincidence here? And where is the “time of use”? Let's see, if the term “reliability” can be used only for the part of this definition for the program? The same standard in the section “01.05.01. Program; Computer program” defines program as “Syntactic unit that conforms to the rules of a particular Programming language and that is composed of Declarations and Statements or Instructions needed to solve a certain function, task, or problem”. A close examination of this definition raises a number of questions again (Fig. 1).

Suppose that some programmer wrote his program in “C” in strict accordance with its rules. This correspondence is confirmed by the fact that during the compilation of the program no warning messages were even issued. After the compiler operation, this program turned out to be converted into an object code, but by the rules of a completely different language, which corresponds to the assembly language of the used processor. Formally, according to the definition, we are dealing with another program that conforms to a completely different language. However, on the other hand, from the point of view of the main functions of this software – controlling the computer hardware - the program has not changed. Just the form of its presentation has changed, nothing more, and the information content of the program has remained the same. Obviously, we assume herewith that the compiler translates high-level language operators into object code correctly. But then it turns out that the program is not a “syntactic unit that conforms to a particular programming language rules”. From the point of view of the main control functions of the program, it is nothing more than information that controls the computer's hardware. And then the syntactic rules of the language are nothing but the rules of coding this information. The encoding rules are subject to change - information in such a case should remain unchanged. This information defines a sequence of computer operations to transform input (initial) data to output data- that is the result of its operation. Therefore, we can state that the program is encoded programming information in the form of instructions of a specific programming language about the way of converting the input data set into the output.

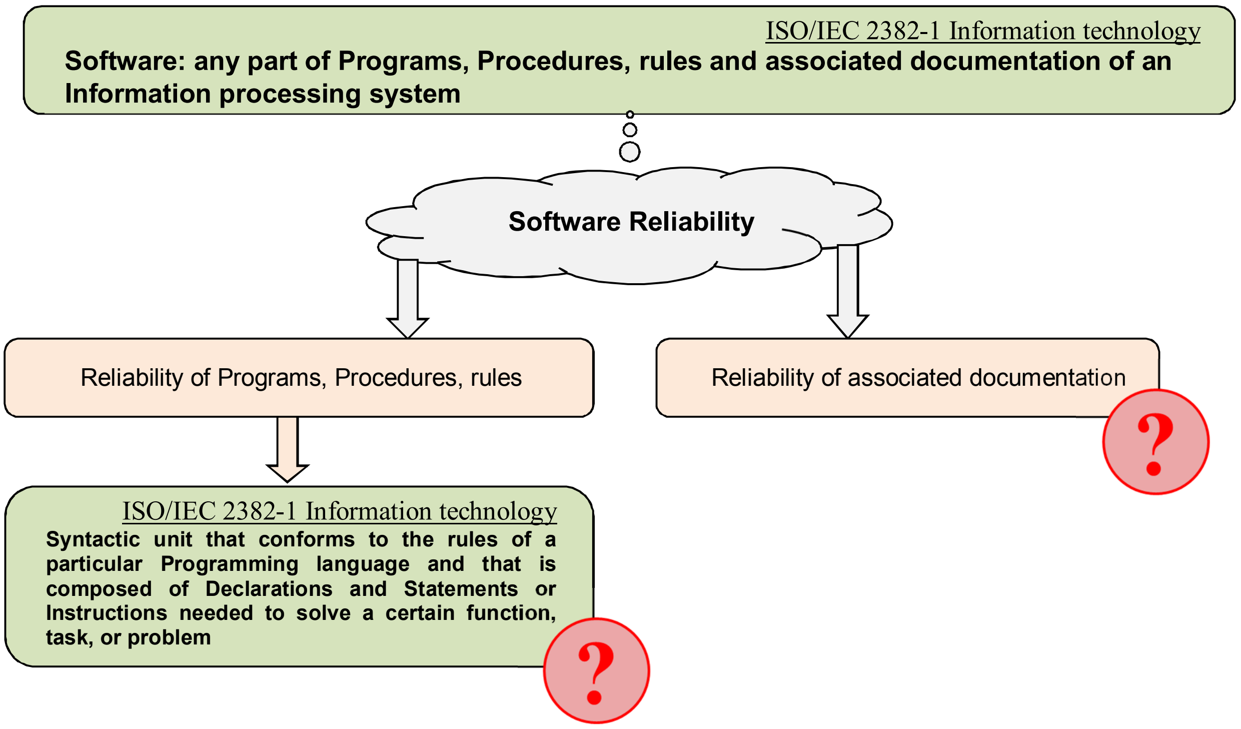
ISO/IEC 2382-1 Information technology
Software: any part of Programs, Procedures, rules and associated documentation of an Information processing system
Software Reliability
Reliability of Programs, Procedures, rules
Reliability of associated documentation
?
ISO/IEC 2382-1 Information technology
Syntactic unit that conforms to the rules of a particular Programming language and that is composed of Declarations and Statements or Instructions needed to solve a certain function, task, or problem
?

Fig. 1 Structure of the definition "Software"

Thus, the program is information. This information should be presented in a certain way. Such presentation comprehends a particular programming language with its own rules. Anything can act as a storage medium - from a sheet of paper with hand-written source code to the hard drive with binary code. The essence of the program, as information, herewith remains absolutely unchanged. Thus, speaking of the reliability of the software, we have to comprehend the reliability of information. However, the term "reliability" is unacceptable for information.

4 FAILURES OF SOFTWARE – DOES IT EXIST?

In 1995 Nozer D. Singpurvalla (George Washington University) published an article [8], the name of which posed the question: "The failure rate of software: does it exist?" In the article it was stated that the notion of failure rate with respect to the software has no mathematical sense. Thus the author writes: "… it does not make mathematical sense to talk about the failure rate of a piece of software. Furthermore, even when it does make sense, the failure rate is personal, and does not exist outside of the software engineer's mind". This is due to the fact that the occurrence or non-occurrence of software failures depends not only on the software itself, but also on the person using it. Indeed, let's imagine a case where the used software performs merely two functions of any kind. (Fig. 2).

The program code of the first function is perfect, and the second code is written very negligently. Then the first user who only uses the first function of the program, will state with full right that it always works correctly. However, the second user, who needs only the second function for work will constantly deal with incorrect results and assume that the program is unworkable. And most interestingly, both of them will be right. It turns out that the failure rate is characterized not so much by the software itself as by its members. The result obtained is remarkable, especially when you consider that all models of software reliability in some way try to calculate specifically the failure rate and use it to obtain the other reliability factors [4].

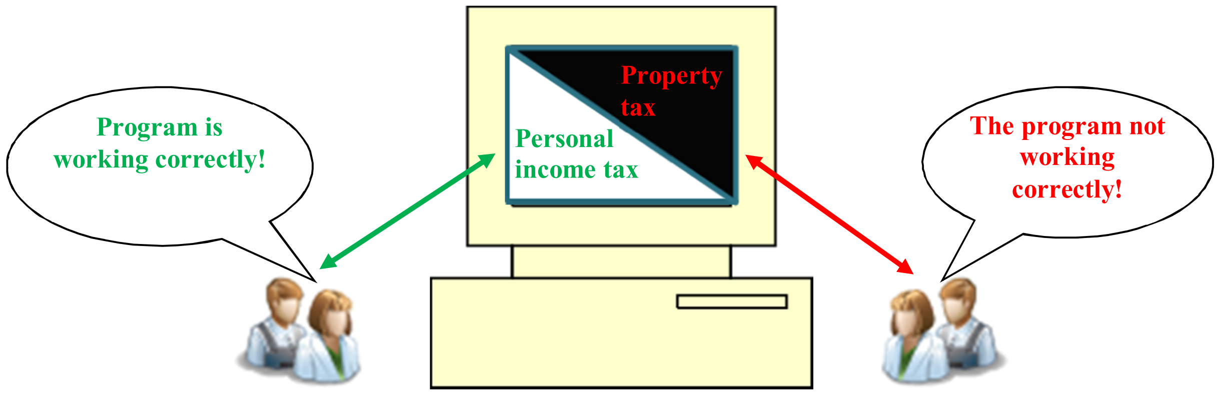
Program is working correctly!
Property tax
Personal income tax
The program not working correctly!

Fig. 2. So how does the program work?

All the above mentioned allows us to go further and ask the question – do any software failures exist at all? The traditional reliability theory defines failure as the event of violation of operational condition of an object. With the failure of the software it's not all that simple. In fact, the program may fail, performing some task that lies "outside" its tested area. Or the software developer can even make a mistake in the very logic of the program. After identifying the program defect (in coding or solution logic), the program gets finalized and becomes able to handle the problem unsolvable before (assuming that during updating the new "secondary" defects were not introduced).

5 CONCLUSION

The authors are well aware that, after fourty years of use in many thousands of scientific publications and hundreds of textbooks, the use of a new term may lead to embarrassment, as if it is something parallel, and the "software reliability" in the old sense, continues to live and thrive. However, it seemed to us that it is necessary to dot once again all the "I"s .

6 REFERENCES

1. Wolverton R.W., Shick G.J. Assessment of Software Reliability, TRWSS-73-04, September 1972. - C. 11 - 18.
2. Jelinski, Z, Moranda, P. Software reliability research. In: Freiberger W,, ed. Statistical Computer Performance Evaluation. New York: Academic; 1972, 465–484.
3. G. J. Myers. Software reliability: principles and practices. John Willey & Sons, New York, 1976
4. Lyu M.R. Handbook of Software Reliability Engineering, McGraw-Hill Company, 1996. – 805 p.
5. Odarushchenko, O.N. Consideration of secondary defects in software reliability models / O.N. Odarushchenko, A.A. Rudenko, W.S.Kharchenko // Mathematical Machines and Systems. - 2010. - № 1. - p. 205-217
6. Ushakov, I.A. Reliability: past, present, future [Electronic resource] / I. A. Usakov // Reliability: Theory & Applications. - 2006. - Vol.1, № 1(01). - p. 10 - 16. Access mode: http://www.gnedenko- forum.org/Journal/2006/012006/ art_1_01(1)_2006.pdf
7. Ushakov, I.A. Reliability theory: history & current state in bibliographies [Electronic resource] / I. A. Usakov // Reliability: Theory & Applications. - 2012. - Vol.7, № 1(24). - p. 10 - 16. Access mode: http://www.gnedenko-forum.org/Journal/2012/012012/RTA_1_2012-01.pdf

8. Singpurwalla, N.D. (1995). The failure rate of software: does it exist? IEEE Transactions on Reliability. Vol.44, No. 3.
9. Smagin, V.A. Fundamentals of the theory of software reliability. Textbook / V.A. Smagin // St-Petersburg: Military Space Academy named after A.F Mozhaisky- 2009.
10. ANSI/IEEE, "Standard Glossary of Software Engineering Terminology", STD-729-1991, ANSI /IEEE, 1991.