The development and reliability analysis environment of fault-tolerance multiversion software

I V Kovalev¹,²,³, D I Kovalev², V S Chefonov¹, N A Testoedov⁴, A A Koltyshev⁴ and A G Krivogornitsyn⁴

¹Siberian Federal University, 79, Svobodny pr., Krasnoyarsk, 660041, Russian Federation
²Reshetnev Siberian State University of Science and Technology, 31, Krasnoyarsky Rabochy Av., Krasnoyarsk, 660037, Russian Federation
³Krasnoyarsk State Agrarian University, 90, Mira pr., Krasnoyarsk, 660049, Russian Federation
⁴JSC Academician M.F. Reshetnev «ISS» Zheleznogorsk, Krasnoyarsk region, Russia

E-mail: chefonov99@mail.ru

Abstract. The article considers the software toolkit «SA-Analysis» which is intended for designing and the analysis of fault tolerant software reliability. Fault tolerance is reached due to application multiversion software architecture. The description of software system toolkit is given. Application of similar system allows to raise efficiency of software engineering process and to create highly reliable software products.

1. Introduction
The development of many technical systems is carried out using special software products. They allow not only to design the future system, but also to evaluate its reliability [1]. Typically, such software products are used to develop complex technical systems.

Currently, many technical systems include software, which serves as a control system or is engaged in information processing. Since the software is directly related to technical systems, the reliability and fault tolerance of the software is a necessary condition for ensuring the reliability and safety of technical systems [2].

Thus, the problem of designing highly reliable software is relevant, especially when it comes to the development of complex information management systems [3].

The widespread use of computer technology has given rise to the concept of «software architecture», which, despite its prevalence, is perceived, as a rule, intuitively and is used most often in the process of comparing computers. In a narrow sense, software architecture, as well as any other information processing tools in general, is understood as a combination of their properties and characteristics designed to satisfy user requests. In a wide sense, software architecture is a generalized model of an information system, sufficient to understand the principles of its functioning [4].

Architecture development is the process of breaking up a large system into smaller parts. There are many names for these parts: programs, components, subsystems, and levels of abstraction. The
architecture development process is a necessary step in the software or software complexes (SC) design process. SC is a set of solutions to many different, but related problems.

Architecture reliability includes reliability of individual elements [5]. Failure of an individual element causes this and possibly other elements to fail, but not the system as a whole. A failure in the system or its functions can result in the period of system downtime. The downtime of a system is defined as the length of time over which the system or part of it cannot perform functions. The period of system downtime leads to a sharp decrease in performance, so reducing system downtime is one of the most important factors in the development of distributed software architectures.

The basic concepts of the reliability theory of software complexes are based on the concepts of the reliability theory, originally developed in relation to hardware complexes. But there are significant differences in the principles of ensuring the reliability of software and other technical systems.

Within the framework of a certain software project, the complexity of creation and the possibility of achieving the specified reliability parameters are determined, on the one hand, by the volume of allocated resources, and on the other by the technological means used in a software design [6].

2. Software architecture design

Low failure rates for hardware components can be achieved by duplicating the most critical architecture components. In some cases, when a component fails, its functions can be performed by the duplicating component. Software reliability cannot be increased by this method, because duplication of software components will also lead to duplication of its errors. There are methods that use diversity in specification, development, implementation, and testing [7].

Experiments show that applying redundancy at the earliest stages of development, as far as possible, reduces the probability of errors. Reliable software can be created by carefully designing the architecture and identifying errors in the components that have the largest effect on system reliability. These components are defined as the most commonly used or architecturally related to many other components, thus affecting their reliability.

Modular organization involves the division of software into functionally complete parts (modules), the unification of relations between them and the establishment a hierarchy of components interaction, which is determined by the sequence of their call. The call is made by transferring control from the calling module to the called one, which returns control to the calling module at the end of execution. The call to a group of programs is carried out by transferring control to the manager of the group at the lower level of the hierarchy included in this SC.

The main methods of the modern practical development of software systems are based on functional decomposition using modular-hierarchical principles [8]. Moreover, at each hierarchical level, the complexity of the components and their relationship is limited. As a result, the overall complexity of the system grows much more slowly than with unstructured design. These principles led to the creation of structural programming as a standard way of building modular programs.

The definition of a modular program is based on restrictions on the size of programs and on the concept of their independence, i.e., a modular program should consist of modules of finite sizes that have an entry point and an exit point. The advantages of modularity are to simplify the design and modification of the program, facilitate testing and debugging of programs, the ability to create a library of standard modules, etc. The disadvantages are the increase in program execution time and the amount of memory, the complexity of inter-module interaction. In general, they are payed off by reducing the development period and simplifying the maintenance of programs.

Structural programming is the main way to implement modular programs. The goals of structural programming are to increase the readability and clarity of programs, increase the productivity of programmers and simplify the development process. In general, the structural programming concept is a set of principles of writing programs in accordance with a list of certain rules. According to the «structural theorem», there are three basic constructions needed to construct any program:

1) a simple computational sequence, meaning that two actions must be performed one after another;
2) alternative or branching, in which a choice is made between two possible ways based on checking some condition;

3) a loop or iteration that re-executes a sequence of actions.

The end-to-end structural control used in structural programming is necessary for detecting and correcting errors in the early stages of design, while the cost of correcting errors is minimal, and the consequences of their presence are negligible. Such control is ensured by regular discussion of decisions made by all interacting developers. In addition, the basic structures, the unification of component interaction rules and the known sequence of their development in the order of subordination create prerequisites for automating the process of structural control.

The modularity of the construction leads to the use of a hierarchical structure of program modules interaction. The hierarchical scheme, reflecting the functions of the modules, at the same time shows the structure of the relationships between them. Hierarchical structures of the system are characterized, on the one hand, by vertical control, when the upper-level modules can intervene and coordinate the work of the lower-level modules. On the other hand, the actions of the top-level modules depend on the information received as a result of the functioning of the lower hierarchical levels. Thus, the control actions mainly go from top to bottom, and information about the corresponding decisions and variables go from bottom to top.

Software architecture is formed from a number of components connected by various means of dependence and communication. An architectural component can be defined differently depending on the architectural approach and the details of the architecture description. The most critical components of software architecture are components that are frequently accessed, or components that are architecturally connected (through dependencies and relationships) with many other components, thus affecting their reliability. A process is a component that occupies a slot in the processor process table. A process can also be a component that is managed as a process through the use of operating system primitives. A module consists of several files that are logically linked and form part of the functional code. The size of the module can vary over a wide range and depends on the application software created.

Depending on the properties of the architecture, software components, as well as their relationship and dependencies can be defined in different ways. The following architecture properties are considered:

- static and dynamic dependency of components;
- component interfaces;
- relations and management;
- system load.

A software architecture with a static dependency of components is formed as a tree of systems, subsystems, modules, and functions. With this approach, a failure in a higher-level node can lead to inaccessibility of lower-level nodes. This architecture with a static dependency of components does not include the relationship between dynamic processes, thus, the model does not provide information regarding the use of software components during user request processing, administration, and maintenance.

When components are dynamically dependent, the software architecture is formed as a graph of processes that use subsystems, modules, and functions. Processes can be executed repeatedly using the same subsystems, modules, and functions. A process can use several different functions during its execution.

Component interfaces include a description of the interaction between different components, as well as interfaces within software components placed in different hardware components according to hardware configurations. Component interfaces depend on the distribution of the software and on the relationship imposed by the hardware architecture.

Relations and management include communication (in the sense of messages) that exists between processes during user request processing as well as in maintenance and administration. Hardware
architecture also affects both relations and management, because different software components are placed in hardware components that are interconnected.

The system load depends on the number and type of user requests, as well as on the administration and maintenance procedures. The system load is reflected in the number and type of processes and messages and the use of software components. Component dependencies allow a fault to propagate from the component in which it occurs to other components. This propagation can cause failures in the chain (or tree) of components. Failure detection depends on the tests performed or on the number and type of user requests. An error can occur in any component. This error may be caused by another component failure, or it may be a failure that occurred in this component. An error can be traced through a chain (or tree) of component dependencies to eliminate all failures that are associated with this error [5].

The reliability of the software depends on the level corresponding to the various components and their dependencies. The duration of the system failure and its impact on system reliability varies depending on where the failure occurred. Failure can occur at different levels of architecture, in a module, process, component interface, or in a communication and control mechanism.

The number of architectural levels in the software architecture model depends on the particular (applied) system design. Each architectural level contains a component graph of each type (module graph and process graph). Processes use modules during their execution time. A process can use several different modules, and a module can be used by several different processes. The process graph represents the dynamic dependency of components, and the module graph represents the static dependency of components. As shown in [9], this architecture can be extended to an arbitrary number of levels with component graphs at different levels. In the architecture model, the process graph is at the top level, and the graphs of other components, such as functions, primitives, data, structures, and messages, are at separate lower levels.

3. Multiversion software architectures

Structural redundancy methods are used to increase the reliability of software [10, 11]. These methods are aimed at creating fault-tolerant software architectures and consist in applying an excessive number of versions (multiversions) of the software module and organizing the process of functioning of these versions in a special way. Versions are developed independently (by different programmers, possibly using different programming languages and different algorithms), but at the same time remain functionally equivalent. Using an excessive number of versions eliminates the effect of single version failures on the software as a whole.

The basic version of the SA-Analysis software system [12] includes two methods for designing fault tolerant software architectures: the recovery block method (RB - architecture) [13] and N-version programming (NVP - architecture) [3]. Their structural diagrams are presented in figure 1.
The recovery blocks method is one of the first multiversion approaches to software fault tolerance. In the RB method, each software module contains an acceptance test that verifies its operation. The module has a sub-programme, a version that repeats the calculations if the test finds an incorrect execution. But the re-calculations are performed by another (alternative) version of the module, which is built on another interpretation of the requirements imposed on the module. Thus, the module has several versions (recovery blocks) that perform the same functions, but implement different algorithms [1]. The acceptance test checks the result of version execution not for correctness, but for validity. The test makes a decision about the validity of the result produced by the version based on the boundaries with the correct result, the type of output, and the runtime of the version. If an execution error is detected, the acceptance test returns the module to a working state, and then the input data is transferred to the backup version. Since the acceptance test can accept an incorrect result as correct, the software package «SA-Analysis» provides the possibility of distinguishing errors.

N-versioned programming is an independent creation of \( N \geq 2 \) functionally equivalent versions in accordance with the same original specifications [6]. These versions run in parallel on separate computers. The results of their work are compared using voting algorithms [3]. The SA-Analysis software package implements an absolute majority voting algorithm. With this algorithm, the result that is received from more than half of the versions is considered to be correct.

Thus, multiversion software architectures can significantly increase the reliability of the software even despite the failures of individual software components.

4. The development and reliability analysis environment «SA-Analysis»

The developed software complex «SA-Analysis» is an environment for designing highly reliable fault-tolerant software, which may include multiversion software architectures. The software package allows not only to design the future of the software, but also to assess its reliability. The SA-Analysis program is developed using the visual programming environment Borland C++ Builder.

The development and reliability analysis environment has a user-friendly interface. The user interactively works with the environment, choosing the number of modules, the type of multiversion architecture, the number of multiversions and their characteristics. The program implements a multifunctional graphical editor based on OpenGL, which allows the user to design the architecture of future software tools of any complexity. The algorithms implemented in the program make it possible to calculate the reliability of software built on the conveyor principle.

Figure 2 shows the SA-Analysis software package. Work with the program begins with the development of the structure of future software in a graphical editor. The structure is formed from the proposed elements: a monolithic module (without reserve), RB and NVP architecture, and two auxiliary elements to indicate the beginning and end of the software structure. The user forms the software structure in the form of a chain of elements, choosing the necessary elements. The number of versions in RB and NVP architectures and the number of modules in the software are unlimited and depend only on the needs of the user.
Figure 2. Software complex «SA-Analysis».

The use of a graphical editor increases the efficiency of the software design process. The editor allows the user to manipulate elements (add, delete and move), as well as build a chain of elements in the sequence in which it is necessary. Graphic objects change accordingly the number of versions in RB and NVP architectures. It provides a visual representation of the scale of the software being developed.

After developing the software structure, the user sets the parameters of each module and each version for multiversion architectures and the operating time of the software.

At any time, the user can save the data to a separate file. In this case, all graphic objects and their parameters will be saved. The user can download data from the file for further work with the software structure and analysis of its reliability.

The software package operates in one of two modes, depending on the type of initial information about the reliability of the software components, which can be set either as the failure rate and recovery rate, or as the probability of failure. If the software parameters are specified through the probability of failures, the result of the program will be the probability of failure-free operation of each module and the software as a whole. If the failure rates and recovery of the software components are known, then the program will give the result in the form of probabilities graphs of failure-free software operation and each of its components.

Additionally, the program implements the ability to form models of their functioning for multiversion architectures. Models are implemented as a state graph and a system of differential equations. This information is useful when the reliability of the software is estimated on the basis of information about the failure and recovery rates of software component versions. Models can be used to evaluate other software reliability characteristics.

The SA-Analysis software package consists of three parts: the shell which implements a graphical editor and procedures for calculating reliability and two modules of RB and NPV. RB and NPV modules are designed to calculate the reliability of the respective multiversion architectures and build
their graphical and mathematical models. In addition, the RB and NPV modules are self-contained and independent, so the user has the opportunity to work with them directly.

5. Conclusion
The application of the SA-Analysis program for the development and analysis of software architectural reliability will help developers to create high-quality software products that required the level of reliability, as well as to identify weaknesses in the software structure. Using the SA-Analysis program allows software developers to save time and money on the designing and testing software components. The presence of two multiversion software architectures makes it possible to design fault-tolerant software and choose the architecture that meets the required operating conditions and functional tasks solved by the redundant component.

The basic version of the SA-Analysis software package was examined and registered in the Industry Fund of algorithms and programs of the Russian Federation (FAP No. 10675) [12], which makes it accessible to a wide range of specialists in modeling complex software architectures of information management systems.

Acknowledgments
The reported study was funded by Russian Foundation for Basic Research, Government of Krasnoyarsk Territory, Krasnoyarsk Regional Fund of Science to the research project: «Multiversion method for improving information reliability of environmental monitoring of thermal power plants» (No 18-48-240007).

References
[1] Sommerville I and Sawyer P 1997 Requirements Engineering: A Good Practice Guide Chichester (John Wiley and Sons)
[2] Parnas D L et al 1990 Evaluation of safety-critical software Comm. ACM 33(6) 636-87
[3] Laprie J-C and Arlat J 1995 Architectural issues in software fault tolerance Software Fault Tolerance ed M R Lyu (Chichester: John Wiley and Sons) pp 47-127
[4] Kovalev I V 2014 Analysis of problems in the research area of software reliability: a lot of stages and architectural aspect Vestnik of SibGAU 3(55) 78-92
[5] Randell B 1975 System structure for software fault tolerance IEEE Trans. on Software Engineering SE-1(2) 220-52
[6] Kovalev I V et al 2011 Diversified management work on the development of N-variant software systems Basic Research 8-1 124-7
[7] Offen R J and Jeffrey R 1997 Establishing software measurement programs IEEE Software 14(2) 45-99
[8] Antamoshkin A N, Kovalev I V and Carev R Ju 2011 Mathematical and software fault-tolerant control systems and information processing (Krasnoyarsk State Agrarian University)
[9] Kovalev I V, Carev R Ju and Zav’jalova O I 2010 Analysis of architectural software reliability information management systems Instruments 11 24–6
[10] Avizienis A 1985 The N-version Approach to Fault-Tolerant Software IEEE Trans. on Software Engineering SE-11(12) 1491-92
[11] Avizienis A 1995 A methodology of N-version programming Software Fault Tolerance (M R Lyu ed) (Chichester: John Wiley and Sons) p. 23-69
[12] Novoj A V and Kovalev I V 2008 System design and analysis of architectural reliability of software (Software system «SA-Analysis ver.1.0.») Moscow: VNTIC 50200600852.
[13] Randell B and Xu J 1995 The evolution of the recovery block concept Software Fault Tolerance (M R Lyu ed) (Chichester: John Wiley and Sons) pp 1-23