The concept of architectural reliability of software for ensuring the functioning of request-free measuring stations

Igor Kovalev[1,2,3,4*], *Dmitry* Kovalev[2,5], *Roman* Kovalev[6], *Valeria* Podoplelova[2,7], *Vasiliy* Losev[4], *Anna* Voroshilova[1], *Dmitry* Borovinsky[8], and *Mavlyuda* Gadoeva[9]

[1]Siberian Federal University, Krasnoyarsk, Russia
[2]Krasnoyarsk State Agrarian University, Krasnoyarsk, Russia
[3] Navoi State University of Mining and Technology, Navoi, Uzbekistan
[4]Reshetnev Siberian State University of Science and Technology, Krasnoyarsk, Russia
[5]National Research University "Tashkent Institute of Irrigation and Agricultural Mechanization Engineers", Tashkent, Uzbekistan
[6]JSC "Academician M F Reshetnev Information satellite systems", Zheleznogorsk, Russia
[7]Sochi State University, Sochi, Russia
[8]FSBEE HE Siberian Fire and Rescue Academy EMERCOM of Russia, Zheleznogorsk, Russia
[9]Bukhara State University, Bukhara, Uzbekistan

Abstract. The article proposes the concept of architectural reliability of software included in the hardware and software complex of request-free measuring stations. This concept is based on previously completed activities to shape the appearance of a complex of hardware and software for the functioning of measuring stations. It is noted that in order to solve problems related to increasing the architectural reliability of software, the security functions and information and logical interaction of the components of the measuring station are essential. Since software architecture is determined by the concept of architectural design, the work presents the stages of designing a complex software system. The description provided includes the basic operations of architectural design, with decomposition necessary to structure and organize system specifications. The final stage lists the main steps that should be taken to enhance the strategic influence of the software architecture on the functionality and reliability of the set of technical means for ensuring the operation of request-free measuring stations.

1 Introduction

The development of the concept of architectural reliability of software included in the hardware and software complex of request-free measuring stations is an urgent task. This follows from the fact that currently in modern satellite navigation the principle of request-free rangefinder measurements between navigation satellites and the consumer is actively used [1-3]. The implementation of this principle is that information about satellite coordinates

* Corresponding author: kovalev.fsu@mail.ru

is transmitted to the consumer as part of the navigation signal. At the same time, ranges to navigation satellites are measured simultaneously (synchronously) in automatic mode. To ensure the correct implementation of the considered principle of query-free rangefinder measurements, a special method for measuring ranges is required. This method is based on calculating time delays. In particular, the time delay of the received signal from the satellite is calculated, which is compared with the delay of the signal generated by the consumer equipment [4, 5].

Figure 1 shows the circuit for determining consumer location (x, y, z coordinates) using four navigation satellites (NS) for ranging. Colored thick lines show circles with satellites located in the center. The radii of the circles correspond to the true ranges, i.e. the true distances between the satellites and the consumer. The colored thin lines are circles with radii corresponding to the measured ranges, which differ from the true ranges and are therefore called pseudo-ranges.

The true range differs from the pseudodistance by an amount equal to the product of the speed of light by the clock departure b, i.e., the amount of offset of the consumer's clock with respect to system time. Figure 1 shows the case when the consumer's clock drift is greater than zero, i.e. the consumer's clock is ahead of the system time, so the measured pseudodistance is less than the true distance.

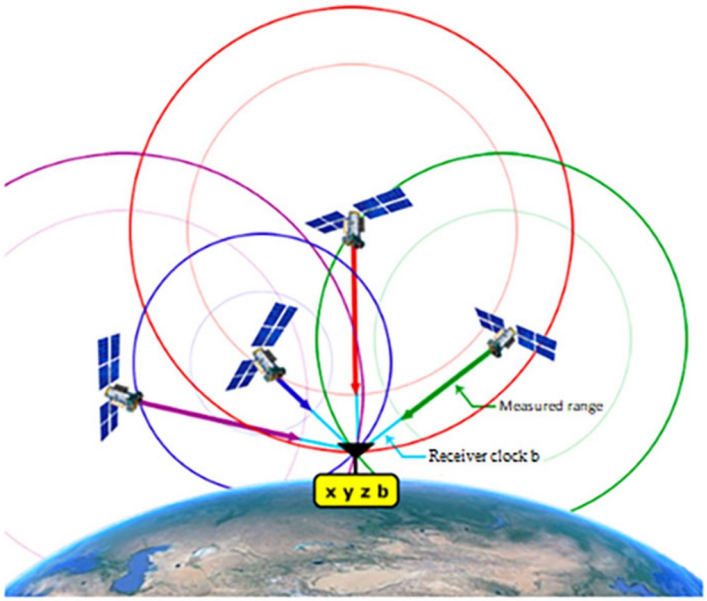

Fig. 1. Circuit for determining consumer location (x, y, z coordinates) using four NS for ranging (source: https://glonass-iac.ru/guide/navfaq.php).

The basic option can only include three NS. However, in this case, measurements must be made with high accuracy. In fact, the clock readings of the satellites and the consumer must match. Then it is enough to take measurements of up to three NS to determine the position of the consumer in space [6].

However, it should be noted that such a coincidence of satellite and consumer clock readings is not always achieved. Often these values do not match. That is, in real conditions of satellite navigation, the readings of the clocks that are part of the consumer navigation equipment (CNE) may differ from the readings of the NS on-board clock. To improve the method and obtain a correct solution to the navigation problem, one more parameter is added.

It is necessary to add one more parameter to the previously unknown parameters (three consumer coordinates). This additional parameter uses the offset between the consumer clock and the system (onboard) time NS. Thus, to correctly solve a navigation problem, in the general case, it is necessary that at least four NS are available to the consumer. This distinguishes this approach from the basic option discussed above.

2 Materials and methods

In order to ensure the functioning of demand-free measuring stations, the following activities are carried out according to the results of theoretical and experimental studies conducted at the stage of technical design [7, 8] to form the appearance of a set of technical means and software for the functioning of measuring stations:

- development of the structural scheme of a set of technical means to ensure the functioning of the measuring station - a set of technical means to ensure the functioning of the measuring station (STM-MS);
- determination of the purpose and characteristics of STM-MS components;
- determination of variants of STM-MS constructive execution;
- analysis of measuring equipment that can be used for construction and testing of STM-MS.

The fulfillment of the listed activities allows to determine the main approaches to the construction of STM-MS in accordance with the requirements of the technical task at the stage of technical design of the system. These requirements include:

- analysis of the product design for manufacturability taking into account feedback from organizations and enterprises that are manufacturers of industrial production;
- development of necessary schematic diagrams, connection diagrams, etc.;
- assessment of the possibility of storage, transportation, as well as installation of the product at the place of its operation;
- finalization of applications for the development and manufacture of new products and materials used in the product under development.

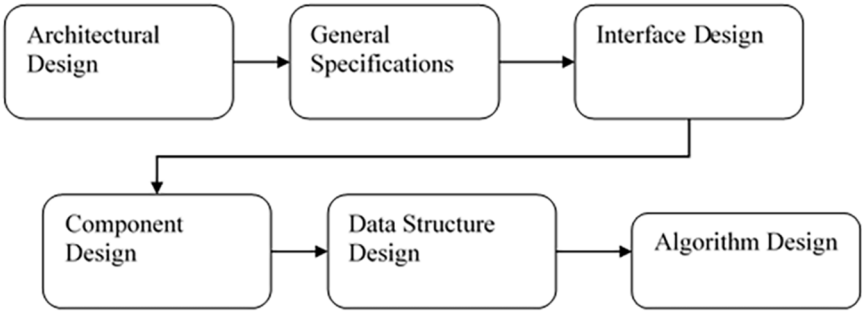

Fig. 2. Stages of software architectural design.

To solve the problems related to the increase of architectural reliability of the software, the functions of security and information-logical interaction of the components of the measuring station are essential, in particular:

- information protection;
- safety of operating personnel and equipment;
- possibility of information-logical interaction with internal and external components of STM-MS;

- transfer of information on functional control of the station elements, temperature-humidity and power supply parameters, as well as information on unauthorized intrusion, on the threat of fire.

Software architecture, as a rule, is determined by the concept of architectural design [9]. Figure 2 shows the stages of designing a complex software system.

In this paper, architectural design is understood as the definition of subsystems, control structure and subsystems interaction, which is a connecting link between the design process and the requirements development stage. The purpose of architectural design is to describe the software architecture. At the same time, decomposition is necessary for structuring and organizing system specifications [10].

A system whose operations (methods) do not depend on services provided by other subsystems is considered as a subsystem. A subsystem consists of modules. A module is a system component that provides one or more services for other modules.

3 Results and discussion

3.1 STM-MS architectural design

STM-MS architectural design consists of the following steps:

- system structuring - formation of a set of relatively independent systems;
- control modeling - management of relationships between parts of the system;
- modular decomposition - subsystems are divided into modules.

The result of architectural design is a document showing the architecture of the software system [11].

The following architectural models are considered:

- static structural model of the program system - subsystems are developed independently;
- dynamic process model - organization of processes during the system operation;
- interface model - services provided by each subsystem through a common interface are defined;
- relationship model - the relationships between the parts of the system are considered.

Depending on the requirements, different attributes of the model are selected [12]:

1. Performance. A minimum number of subsystems with minimum interaction between them are responsible for all critical situations. It is better to use large-modular system components.
2. Protection. This is ensured by a multi-layer structure in which critical system elements are protected at internal levels.
3. Security. The number of subsystems that affect security should be minimized as much as possible.
4. Reliability. This is achieved by including redundant components that can be replaced and updated without interrupting system operation.
5. Maintainability. This is accomplished by using small structural components that can be easily modified. Programs that create data are separated from the components that process and use that data.

3.2 Software Architectural Reliability

Issues related to software architecture are increasingly attracting the attention of researchers and developers. Software architecture is understood by many authors [13, 14] as a multifaceted approach to ensure that software meets its intended purpose.

It is obvious that no subject area is already without software. With the help of software, enterprises can not only achieve their goals, but also define as clearly as possible what exactly these goals are. Careful attention to software development enables enterprises to achieve strategically significant goals that can affect their position in the marketplace. This section suggests strategies that are significant in implementing modern approaches to analyzing and designing software architecture.

Modern software architecture differs significantly from that used during the nascent period of the industry. The developer no longer draws simple block diagrams, defining individual functional modules, or manually writes each application. Modern architecture offers a detailed and accurate model of the system. The techniques of software architecture formation assume a detailed analysis of the system before its implementation. Techniques such as Attribute Driven Design (ADD) [15] ensure that software implemented based on a pre-formed architecture will accurately fulfill its intended purpose.

There are quite a few examples of using architecture for strategic purposes. One of the best known is the Common Object Request Broker Architecture (CORBA) [16, 17]. This architecture serves to link legacy systems, i.e., to integrate systems written in different languages and to support communication between computers with different hardware architectures.

CORBA makes it possible to give new life to legacy systems and at the same time quickly integrate new applications into them, giving enterprises a strategic advantage over competitors modifying legacy systems. A more recent example is the freely available integrated development environment Eclipse [18, 19].

Let us focus on current concepts in the field of software architecture, analyzing how software architecture can influence the formation of management strategy in a system, and consider a number of possible concrete steps that allow architects to contribute to the definition of new strategies and models for the development of architectural approach in STM-MS software development.

3.2.1 General description of program architecture

The architecture of a software system defines its structure, or more precisely, several structures, each of which includes elements and the relationships between them. Elements can be computational objects bound by control flow or business objects bound by semantic constraints.

In general, the architecture design process consists of a systematic decomposition of top-level elements into sets of smaller elements.

For example, using the ADD approach, the architect chooses a particular decomposition in an effort to improve certain properties of the final product. Note, however, that each decomposition tends to degrade some properties. MVC (Model-View-Controller) based decomposition extends the possibilities of modifying the system, in particular to add new model views more efficiently. But this decomposition will lead to performance degradation, since any changes to the Model will have to make a notification to the View element when any changes to the Model occur. From the architect's point of view, this is an acceptable compromise, since the system does not operate in real (computer) time, but in expectation of human perception of changes.

The process of creating an architecture involves designing a system with specific properties and functionality, with each such property having a given priority. Starting with an overall structure that supports all required functionality, the architect methodically decomposes the functionality and distributes it among the components.

The decomposition process continues until the components are properly detailed. In other words, an optimal ratio between the characteristics of the given features must be achieved. Working groups can then proceed with the design and implementation of their subset of architecture elements.

To professionals familiar with the MVC design pattern, the visual representation of the model may seem quite informative, but it lacks essential details not understood by users or designers. Several architecture description languages [20] have been developed to support detailed architecture specifications, but they have never been widely adopted. Many of the concepts presented in these languages are implemented in the latest version of UML (Unified Modeling Language) [21].

A UML diagram allows you to present the interfaces of software modules in detail. To achieve this, the diagram contains additional information about the interfaces. This is the request and receive information for each module. In this case, the architect can create different diagrams to show the component structure, detailing the interactions between components and their placement in the software architecture.

Then the architecture description can have several “views”. These different views are used both to display different types of information and to display different software structures. It is important that the dynamic interactions between elements of a static architecture are easily described and visualized using a UML sequence diagram.

4 Conclusion

The above description includes the basic operations of architecture design, but does not reflect a typical architecture development process. In most cases, an architect starts with a predefined, reference architecture. This may be an actual industry standard (e.g., J2EE) or a prescription (e.g., the Command, Control, Communications, Computer, Intelligence, Surveillance and Reconnaissance architecture platform provided for special systems in the United States). Reference architectures provide a high-level decomposition that establishes the basic characteristics of the structure's properties, but leaves the architect free to perform low-level decompositions that more accurately define the quality of the software product's properties.

Having chosen a reference architecture, it is possible to use the components developed with its help as a basis for design. An important factor in choosing an architecture is the commercial community of manufacturers of standard components that is formed around it. The larger and more diverse it is, the more likely you are to acquire components that allow you to significantly accelerate product development. Besides, there are templates, references, examples and other assets for this architecture.

This article concludes by listing the key steps that should be taken to enhance the strategic impact of the software architecture on STM-MS functionality and reliability.

1. A deployed, state-of-the-art, proven STM-MS software architecture should be developed. Keep in mind that this is the first but most important step in the project. It is recommended to use UML or similar notations to describe the architecture so that these descriptions are accurate and unambiguous.
2. Coordination of separate STM-MS software architectures used in the same organization. Techniques are used to form software product series for solutions that are closely related to each other.

3. Ensuring close coordination of processes related to the development of a set of technical means by organizing them on the basis of architecture. ADD or similar architecture design techniques should be used to support consistency between the software architecture and the hardware suite.
4. Trade-offs should be made to support the realization of specified system properties and, ultimately, the performance goals of the entire hardware-software complex. Architectural representations should be created that show the correspondence between goals and architectural decisions and the evolution of the architecture.
5. Continuous study of new technological and software solutions, analyzing the activities of industry associations formed around a particular technology (e.g., Object Management Group). It is necessary to track software architectures that are defined by groups specializing in a particular subject area, such as space navigation or telecommunications, etc.

By consistently following these steps, it is possible to ensure that the software architecture helps to define goals and achieve them with high reliability and readiness of the resulting IT applications for STM-MS.

References

1. H.Kh. Saad, M.A. Loban, Reports of the Belarusian State University of Informatics and Radioelectronics **20(8)**, 51-58 (2022)
2. P.A. Kudriasheva, A.S. Davydenko, Computer Science, Telecommunications and Management **13(2)**, 14-23 (2020)
3. I. Kartsan, Modern Innovations, Systems and Technologies **1(2)**, 64-71 (2021). https://doi.org/10.47813/2782-2818-2021-1-2-64-71
4. I.V. Kovalev, et al., Transportation Research Procedia **68**, 796-801 (2023). https://doi.org/10.1016/j.trpro.2023.02.111.
5. V.N. Tyapkin et al., Journal of the Siberian Federal University. Mathematics and Physics **12(6),** 772-779 (2019)
6. R. Kovalev, et al., Spacecraft and technology **2(20),** 72-75 (2017)
7. I.V. Kovalev et al., IOP Conf. Ser.: Earth Environ. Sci. **1112,** 012156 (2022). https://doi.org/10.1088/1755-1315/1112/1/012156
8. I. Kovalev et al., E3S Web of Conferences **417**, 06008 (2023). https://doi.org/10.1051/e3sconf/202341706008
9. J. Manishaben, SSRN Electronic Journal **6(11)**, 2452-2454 (2019). https://doi.org/10.2139/ssrn.3772387
10. A.A. Mitsyuk, N.A. Jamgaryan, Proceedings of the Institute of System Programming RAS 33(3), 7-26 (2021)
11. M. Richards, N. Ford, *Fundamentals of Software Architecture: An Engineering Approach* (O'Reilly Media, 2020), p. 432
12. I.N. Kartsan, Modern Innovations, Systems and Technologies **3(4)**, 0322-0331 (2023). https://doi.org/10.47813/2782-2818-2023-3-4-0322-0331
13. R. N. Taylor, A. van der Hoek, *Software Design and Architecture: the once and future focus of software engineering* Proceedings of Future of Software Engineering (FOSE '07), Minneapolis, MN, USA, 226-243 (2007). https://doi.org/10.1109/FOSE.2007.21
14. O. Sievi-Korte, I. Richardson, S. Beecham, Journal of Systems and Software **158**, 110400 (2019). https://doi.org/10.1016/j.jss.2019.110400

15. H. Reza, Journal of Software Engineering and Applications **10**, 483-499 (2017)
16. A. Gupta, S. Kar, IETE Technical review **19**, 31-45 (2002). https://doi.org/10.1080/02564602.2002.11417009
17. B. Kenneth, CORBA: The Common Object Request Broker Architecture. In: Reliable Distributed Systems (Springer, New York, NY., 2005). https://doi.org/10.1007/0-387-27601-7_6
18. J. desRivieres, J. Wiegand, IBM Systems Journal **43(2)**, 371-383 (2004). https://doi.org/10.1147/sj.432.0371
19. B. Nuryyev, S. Nadi, N.U. Bhuiyan, L. Banderali, *Challenges of Implementing Software Variability in Eclipse OMR: An Interview Study* Proceedings of IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Madrid, ES, 31-40 (2021). https://doi.org/10.1109/ICSE-SEIP52600.2021.00012
20. W. Eoin, H. Rich, Architecture Description Languages in Practice Session Report. Proceedings of 5th Working IEEE/IFIP Conference on Software Architecture, WICSA, 243-246 (2005). https://doi.org/10.1109/WICSA.2005.15
21. F. Pecoraro, D. Luzi, International Journal of Environmental Research and Public Health **19(20)**, 13456 (2022). https://doi.org/10.3390/ijerph192013456