UDC 621.3.019.3

P.D. CESPEDES GARCIA[*]

N-VERSION PROGRAMMING AS AN OPPORTUNITY TO EXCLUDE ERRORS IN SOFTWARE

[*]Institute of Mathematical Machines and Systems National Academy of Sciences of Ukraine, Kyiv, Ukraine

Анотація. *Одним із методів підвищення рівня гарантоздатності комп'ютерних систем є метод багатоверсійного програмування. Ця концепція була запропонована Лімінгом Ченом і Альгірдасом Авіженісом у вигляді основної гіпотези про те, що «незалежні зусилля при розробці програмного забезпечення значно зменшать імовірність ідентичних збоїв, що виникають у двох або більше версіях програми». Метою багатоверсійного програмування є підвищення надійності роботи програмного забезпечення за рахунок локалізації помилок проектування. Ефективність багатоверсійної системи залежить від різноманітності варіацій на етапах утворення груп виконавців, використання різних алгоритмів, засобів проектування та випробувань. Багатоверсійне програмне забезпечення визначається, як незалежно розроблені дві або більше функціонально еквівалентні програми, написані з однієї початкової специфікації. Різні версії програм мають усе необхідне для їх одночасного виконання і при цьому взаємодіють одна з одною на етапах порівняння результатів. Дослідження припускають, що незалежна розробка програмного забезпечення і подальше впровадження методів диверсності приведуть до створення версій, які містять суттєво різні помилки, що не приводять до відмов на етапах порівняння. Таким чином, якщо більшість версій видасть однакові значення, цей загальний результат, імовірно, буде вірним. Також слід звернути увагу на проектування багатоверсійної системи як системи з декількома видами версійної надлишковості на всіх етапах її життєвого циклу. Це дозволить диверсифікувати систему не тільки на етапі розробки, але й на попередніх етапах проектування системи. Незважаючи на те, що були проведені дослідження, які ставлять під сумнів актуальність багатоверсійного проектування, ряд дослідницьких груп підтвердили значно вищу коректність обчислень, яких здатні досягти багатоверсійні системи у порівнянні зі звичайними системами.*

Ключові слова: *гарантоздатність, N-версійне програмування, багатоверсійні системи, надлишковість, диверсність, відмовостійка система, розробка програмного забезпечення.*

Аннотация. *Одним из методов повышения уровня гарантоспособности компьютерных систем является метод многоверсионного программирования. Эта концепция была предложена Лимингом Ченом и Альгирдасом Авиженисом в виде основной гипотезы о том, что «независимые усилия при разработке программного обеспечения значительно уменьшат вероятность идентичных сбоев, возникающих в двух или более версиях программы». Целью многоверсионного программирования является повышение надежности работы программного обеспечения за счет локализации ошибок проектирования. Эффективность многоверсионной системы зависит от разнообразия вариаций на этапах образования групп исполнителей, использования разных алгоритмов, средств проектирования и испытаний. Многоверсионное программное обеспечение определяется как независимо разработанные две или более функционально эквивалентные программы, написанные из одной исходной спецификации. Разные версии программ имеют всё необходимое для их одновременного выполнения и при этом взаимодействуют друг с другом на этапах сравнения результатов. Исследования предполагают, что независимая разработка программного обеспечения и дальнейшее внедрение методов диверсности приведут к созданию версий, которые содержат существенно разные ошибки, не приводящие к отказам на этапах сравнения. Таким образом, если большинство версий выдадут одинаковые значения, этот общий результат, вероятно, будет верным. Также следует обратить внимание на проектирование многоверсионной системы как системы с несколькими видами версионной избыточности на всех её этапах жизненного цикла. Это позволит диверсифицировать систему не только на этапе разработки, но и на предварительных этапах проектирования системы. Несмотря на то, что были проведены исследования, ставящие под сомнение актуальность многоверсионного проектирования, ряд исследовательских групп подтвердили значи-*

тельно более высокую корректность вычислений, которых способны достичь многоверсионные системы в сравнении с обычными системами.
Ключевые слова: *гарантоспособность, N-версионное программирование, многоверсионные системы, избыточность, диверсность, отказоустойчивая система, разработка программного обеспечения.*

Abstract. *One of the methods for increasing computer system dependability level is the N-version programming method. Liming Chen and Algirdas Avizienis proposed this concept with the main hypothesis that «independent efforts in software development will significantly reduce the likelihood of identical failures that occurs in two or more versions of the program». The main goal of N-version programming is to increase the reliability of the software by bounding design errors. The effectiveness of a multiversion system depends on the variety of variations at the executive teams formation stage, different algorithms implementation, design and testing tools. Multiversion software is defined as independently developed two or more functionally equivalent programs written from the same specification source. Different program versions have everything they need for simultaneous execution, and at the same time, they interact with each other at the results comparison stages. Researches suggest that independent software development and further implementation of diversity methods will lead to the creation of versions that contain significantly different errors and they will not lead to failures at the comparison stages. Thus, if most versions produce the same values, this overall result is likely will be correct. Moreover, a multiversion system design must be considered, as a system with several types of versioned redundancy at all stages of its life cycle. This will allow diversifying the system not only at the development stage, but at the preliminary stages of system design as well. Despite the conducted studies that discredited the relevance of multiversion design, a number of research groups have confirmed the significantly higher calculations accuracy, which multiversion systems are capable to achieve, in comparison with conventional systems.*
Keywords: *dependability, N-version programming, multiversion systems, redundancy, diversity, fault-tolerant system, software development.*

1. Introduction

Complex systems with its constant software failures causes lots of trouble, any failure in security systems can lead to unpleasant consequences, which is why modern systems are subject of high reliability requirements. The most common source of computer system failures are software defects. They had been formed during the development phase, no one detected them during testing and verification, and finally they appeared while operating a set of an input data or due to the physical or informational environment features [1]. The testing stage cannot always eliminate every software error, more accurate testing of a critical application system is expensive. This forces to search for more affordable methods to improve the quality of a complex system [2].

2. The emergence of N-version programming approach

Duplication technique (hardware redundancy) is often used in critical systems, this approach protects against hardware accidental failures. In addition to redundancy, there is a multiversion (*N*-version) programming (NVP) technique, it is based on the idea of using redundancy in software. Multiversion programming partially transfers hardware duplication approach to software development. The idea is to execute simultaneously several independently developed but functionally equivalent software versions by different development teams. A kind of software redundancy that increases the system chances to provide correct results by compensating each other's errors. Thus, the risk of simultaneous and single-type failures in a multiversion system decreases with an increase in the number of implemented versions. This *N*-version method provides the necessary level of critical systems safety and reliability when designing a multiversion system, this diversity can be implemented in the following stages [3]:

1. Training, experience, and location of developers.
2. Algorithms and data structures.

3. Programming languages.
4. Software development methods.
5. Development tools and environments.
6. Testing tools and methods.

The research on *N*-version efficiency received much efforts, Liming Chen and Algirdas Avizienis introduced the concept itself in 1977 with the main hypothesis that independent software development efforts would significantly reduce the chances of identical failures using two or more program versions. At the early stages, attention was focused mainly on modeling a system that implements the principle of the *N*-version programming approach. There were discussions about simultaneous failures possibilities in different versions, later this topic was researched in pair with *N*-version programming software failures. For example in [2], we see that constraints of the software reliability stemmed from multiversion method problems. The authors also paid attention to the pricing issue and to the multiversion system optimal structure as well.

The research has led to significant breakthroughs in the field of *N*-version programming, A. Avizienis, P. Popov, L. Strigini have made a significant contribution to the NVP methodologies development, they describe processes and tools for multiversion software implementation, testing, practical use and maintenance. These and other studies were focused on diversity and reducing the odds of simultaneous and similar failures, which led to meaningful results in developing multiversion applications. Further, developments in the modeling field and quantified influence of diversity on the reliability of multiversion systems were carried out [3].

John Knight and Nancy Leveson demonstrated [4] that independent version development is not enough for the system to become completely crash independent. Their research was important for the NVP, since they encouraged researchers to study the question of how to effectively implement diversity and quantify its effect more carefully. Many researchers, who were involved or interested in *N*-version programming, misunderstood their criticism and perceived it as a definitive statement against the NVP, even though it was not true. D. Knight and N. Leveson warned that independence of failures, though mathematically convenient in theoretical work, could not be assumed in reality. However, according to A. Avizienis, independent failure is merely an ideological objective, and is not a basis for, or even an assumption of *N*-version programming [3].

In [5], there is a suggestion to divide redundancy into external and internal at the early multiversion system designing stages, and to carry out the project formation, respectively, in two stages: theoretical design and a software development. Thorough discussion about taxonomy and versioned redundancy classification is occurred in [1], diversity principles and their terminological aspects are clarified, and different types of versioned redundancy being analyzed, taking into account the capabilities of modern computer technologies and experience in developing multiversion systems. Also worth mentioning the performer's personal characteristics diversity, different developers even if they work separately, often make the same mistakes. The most likely reason is a random personal characteristics coincidence between the participants in a multiversion project where the same techniques and tools for development are used (e.g. similarities in implementations of the most common techniques in solving various tasks during the learning process) [6].

3. The concept of N-version programming

Traditional software has only one version. Input data always being processed in the same way, any failures lead to an error in the calculations results. To avoid this problem, when writing and testing code, software developers have to eliminate as many errors as possible. Since, it is almost impossible to eliminate all errors during a complex system development, the concept of a fault-tolerant software was introduced in pair with redundancy. With its introduction, users assumed a significant reliability level increase along with the software systems performance boost. *N*-version programming is one of the most common methods for developing fault-tolerant application systems. It is defined as an independently developed set of $N \geq 2$ structurally different, but functionally equivalent programs (versions) that corresponds to the same initial specification. Each software version process source data separately and simultaneously, the voting/matching mechanism is an inter-stage point for output data from each version to interact with each other. This mechanism will not allow any failure to affect the entire system result.

When we talk about «independently developed programs», it means that there are *N*-number of developers groups who do not interact with each other, they use different development tools, programming languages and algorithms for the task implementation. A visual representation of multiversion software is shown in Fig. 1.

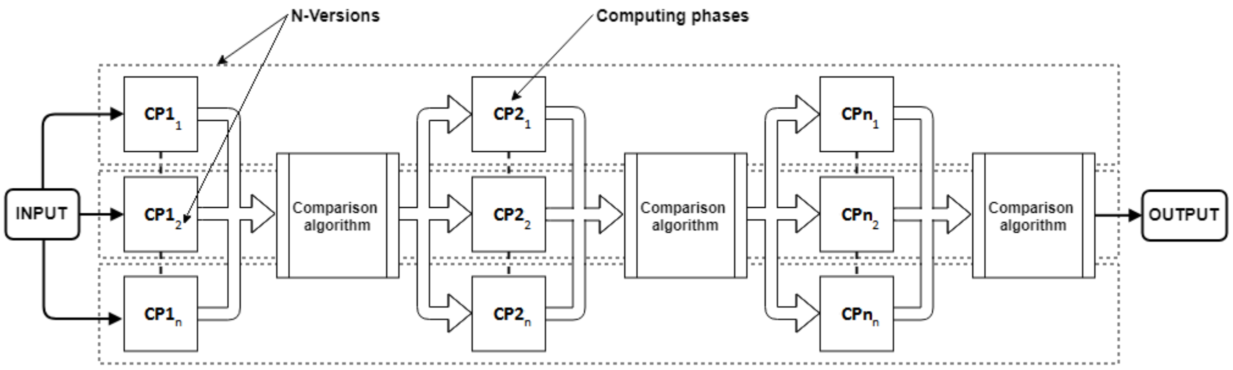

Figure 1 – An example of a *N*-version programming system

The main goal of the technical, or the so-called initial specification, is to establish the functional requirements completely, while leaving the widest possible choice of implementations for designers and developers. It also specifies special functions that are necessary for each version execution in accordance with the fault-tolerance requirements. The initial specification should define [7]:

1. Functions that must be implemented by multiversion software.
2. Data formats for special mechanisms: comparison vectors, comparison status indicators and synchronization mechanisms.
3. Cross-check points for comparison vector generation.
4. Comparison algorithm.
5. Response to the possible outcomes.

At various system operation stages, these special mechanisms allow to control the entire system using *N*-version execution environment (NVX) that acts as an additional software and/or hardware [8]. With comparison algorithm implementation, it is necessary to establish explicitly the acceptable range of numerical results. «Comparison algorithm» expression is used as a general term to the comparison principle. Specifically, «matching» algorithm refers to systems with two implemented versions, while «voting» algorithm refers to systems with more than two versions.

Some NVX implementations are developed in such way, that they make the final decision, taking into account each individual result of each *N*-version. Decision-making algorithms implementation can vary from simple, where the most common results considered as valid, to algorithms that are more complex. Among other duties, the NVX must: provide the necessary interversion communication; manage synchronization control; perform error-masking for each stage at the inter-stage comparison points to ensure version functioning; execute error-correction functions and, in general, manage system efficiency [3].

4. Versioned redundancy types classification

Form of redundancy in a multiversion project determines the system development direction. Options for engineering implementation are called external multiversion. Internal multiversion is responsible for technical and software solutions.

Formation of a multiversion project with an external multiversion diversity starts from consideration of all factors that would be taken into the project account. Next, based on the initial specification, a hierarchy is formed from a set of variants, which are preliminarily divided into categories. From a variety of options consider reasonable combinations, and at the final stage, generate system versions. At the initial planning stage, project managers make decisions about: hardware diversity options, operating system, software, recommended programming languages, specifications, development tools, and testing options. After forming the general development concept, the project developers establish functional specifications. They start to develop architecture, data structures, algorithms, afterwards they write code and test it. Also, they are engaged in providing quality control, designing supporting documentation, and ultimately, they maintain software support [9].

Details about delivering redundancy to a multiversion system at all its life cycle stages considered in [1]. It is proposed that system level consists of six types of versioned redundancy (VR):

1. Various teams of performers achieve Subjective VR. The work [6], on the diversity of a programmers personal characteristics, considers the reasons why the development of a project by several groups is the basis for a successful results in terms of personality differences.

2. VR of design is based on applying various concepts of technologies and architectures in the development of hardware and software and the system as a whole.

3. VR of software suggests applying different software versions. Their diversity caused by various types of VR's implementation, it applies to the software development stages: algorithms and logic implementation, software architecture, programming languages, operating systems and databases.

4. Functional VR is achieved by implementing different functionality to achieve an identical result.

5. Signal VR is provided by using various data sources that determines the system functioning algorithm.

6. VR of equipment is achieved with the help of various components, printed circuit boards, computer bus organization, and different equipment manufacturers.

Implementation of multiversion system at the program stage is achieved based on diversity of life cycle models levels. It offers a choice between a model with the minimum required and maximum set of processes to ensure quality. By choosing the programming languages, development tools and the developers themselves, you can achieve a diversity of resources and tools. Project solutions diversity is achieved by taking into account the possible choices of architecture and platforms, records, formats and ways of representing data [1].

Options for classifying multiversioning at a conceptual level are considered in accordance with approaches to system design and calculation systems that are used for representing and processing information in multiversion system channels [1].

5. Conclusion

Multiversion system development combines the principles of redundancy with the condition of identical functionality. The main goal of this approach is to increase the system dependability level. Performer groups diversity at all project life cycle stages implements a multiversioning principle, starting with the requirements development and finishing with commissioning. With the same specification, independent development teams create several project implementations using different hardware, methods, tools, programming languages, etc. To achieve correct output results in the multiversion system, there is either comparison or voting algorithm at the final calculation stage, systems with inter-stage testing and error masking are also possible.

There are disputes between NVP method researchers regarding the errors independence when developing different versions for the system. It is assumed that independence in the system development will lead to statistically independent errors, the experiment [4] suggests opposite. The multiversion system weak point is the comparison algorithm, system calculations accuracy and output data fidelity depends on the algorithm that is used in the *N*-version execution environment. It could be either «voting» or «matching» algorithm, depending on quantity of implemented versions in the system, it determines which of the results among each version is correct. The results of several independent versions are more likely to provide «reliable» data than the results of a single version, but there are some debates whether the effect of the multiversion method implementation is worth the time, efforts and costs [3].

The positive effect from designing multiversion software can be achieved when comparing output data using voting algorithm, therefore the number of versions should be at least 3, keep in mind that it will also significantly increase development costs. It's noted that when the development of complex multiversion software is completed, it has to be upgraded, since it is morally and technologically obsolete. Multiversion software is efficient in long-lived especially critical systems – astronautics, armament, etc., where software has been tested and developed for years and, subsequently «lashed» into the ROM.

REFERENCES

1. Сиора А.А., Скляр В.В., Харченко В.С. (n,m)-версионные системы: таксономия, модели и технологии. *Вісник Харківського національного університету. Математичне моделювання. Інформаційні технології. Автоматизовані системи управління*. 2008. № 833. С. 231–246.
2. Min X., Chengjie X., Szu-Hui Ng. A study of N-version programming and its impact on software availability. *International Journal of Systems Science*. 2014. Vol 45, N 10. P. 2145–2157.
3. Liburd D.S. An N-version Electronic Voting System. *Massachusetts Institute of Technology*. Dept. of Electrical Engineering and Computer Science. 2004. P. 15–31.
4. Knight J.C., Leveson N.G. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE Transactions on Software Engineering*. 1986. Vol. 12, N 1. P. 96–109.
5. Дужий В.И., Шостак А.В., Дужий И.В. Оценка показателя версионной избыточности при проектировании программного обеспечения. *Радіоелектронні і комп'ютерні системи*. 2009. № 6. С. 159–165.
6. Варламова Н.В., Мищенко В.О. Разнообразие личностных характеристик программистов как основа успешности программной диверсности. *Вісник Харківського національного університету імені В.Н. Каразіна. Математичне моделювання. Інформаційні технології. Автоматизовані системи управління*. 2016. № 30. С. 27–35.
7. Chen L., Avizienis A.A. N-Version Programming: A Fault-Tolerance Approach to Reliability of Software Operation. *Fault-Tolerant Computing. Highlights from Twenty-Five Years*. 1995. P. 113–119.
8. Avizienis A.A. The Methodology of N-version Programming / ed. M. Lyu. Software Fault Tolerance: John Wiley & Sons, 1995. P. 35.
9. Дужий В.И., Дужий И.В., Шостак А.В. Разработка многоверсионной иерархии решений при проектировании программного обеспечения. *Радіоелектронні і комп'ютерні системи*. 2007. № 8. С. 173–176.

Стаття надійшла до редакції 12.02.2019