The decision algorithm selection for ensuring the quality assurance of the components of the multi-version execution environment

Igor Kovalev, Vasiliy Losev, Mikhail Saramud, Mariam Petrosyan

Abstract: the article suggests a selection technique for optimal voting algorithm for the decision block of the multi-version execution environment. It allows you to choose an algorithm that guarantees the quality characteristics of the developed component. Different decision-making algorithms have their own strengths and weaknesses: some are more resistant to related faults, but they do not work adequately with a large percentage of "inaccuracies," while others, on the contrary, are resistant to both "inaccuracies" and relatively unreliable versions, but they are mistaken for each related error, etc. Therefore it is necessary to check all of the algorithms in an environment simulating the characteristics of the system under development. Thus, we get the characteristics of the quality of the algorithm in conditions exactly the same, that it will work in our system. The existent algorithms of a decision making in multi-version execution environments are described in the article. In addition, the own modifications of existing voting algorithms and t / (n-1) algorithm are suggest by authors. The software implementation of the simulation environment that implements simulations of versions with specified characteristics and proposed modified algorithms is considered. Given the characteristics of the system, the environment makes it possible to obtain the quality characteristics of all implemented decision algorithms. If, by results of modeling, there exists an unambiguously superior algorithm, the system specifies it explicitly, if there is no such algorithm, then, the developer is provided with numerical and graphical simulation results for self-selection. The results of the simulation are considered; moreover, the dependence of the reliability indicators of the system on its input parameters is shown. The comparative analysis of various decision algorithms is made basing on simulation results.

Key words: NVP; NVS; fault tolerance; quality assurance; life cycle; reliability; voting algorithms; simulation modeling; NVX.

I. Introduction

In modern time, the task of creating fault tolerance control systems is actual, both for dangerous and complex production processes, and for autonomous unmanned objects. While optimizing the manufactures the management of the complicated continuous processes is transferred to modern industrial controllers and computer systems. The field of unmanned autonomous objects is also actively developing, beginning with the “copter”, multi-rotor systems which find more applications in everyday tasks from photo-video shooting from the air to the delivery of goods from online stores, ending with unmanned cars, the commercial models of which has already begun to appear on public roads [1].

The authors are with the Reshetnev Siberian State University of Science and Technology, Russian, Krasnoyarsk, Russia

For all these tasks there is a need for a reliable software as a whole as a control system and for specific processes that are particularly critical to reliability. The most effective method of increasing the reliability of a software to date is the approach with the introduction of software redundancy to the multi-version programming [2].

However, in the software systems it is impossible to duplicate versions as it is done in a hardware. Thus, we will duplicate all errors, both algorithmic and coding errors, and we will not achieve an increase in the reliability of the redundant system [3]. The composition of the multi-version system is to consist of functionally equivalent but algorithmically different versions, ideally all versions must be done by different developers, in different programming languages, in different development environments, using different libraries, if necessary[4].

This approach permits to minimize the most "dangerous" type of faults, multi-version or related faults [5], these are incorrect but coincident outputs of the versions. This type of faults is the most dangerous because it is the most difficult to detect, since in a situation where, for example, 3 out of 5 versions gave different erroneous outputs, it is possible to determine the correct output for the system by eliminating three faults. In the case where the erroneous outputs are equal, it is extremely difficult to decide correctly.

To guarantee the quality of a complex system as a whole, we need to ensure the quality of its constituent components. In the case of fault-tolerant software based on software redundancy, we need to guarantee not so much the quality of versions as the decision block that will choose the correct output from the version of the response collection, and it is the quality of its work that is most critical for the fault-tolerance of the system that being developed. Therefore, the main task of system developers is to guarantee the quality of the decision block. At the same time, the quality assurance of the versions will for the most part be the task of third-party developers, since the creation of a version for ensuring diversification is transferred to various developers.

Considering the stage of the life cycle of the design of fault-tolerant on-board software, we need to determine the algorithms underlying the decision block. At the current time, there are many algorithms used to determine the correct exit from the set of response versions. Most often - these are different voting algorithms, we will consider the most

common: voting by an absolute majority, voting by an agreed majority, fuzzy voting by an agreed majority, a median vote. Also we will consider the t / (n-1) decision-making algorithm and suggest modifications to existing voting algorithms and t / (n-1) decision algorithms.

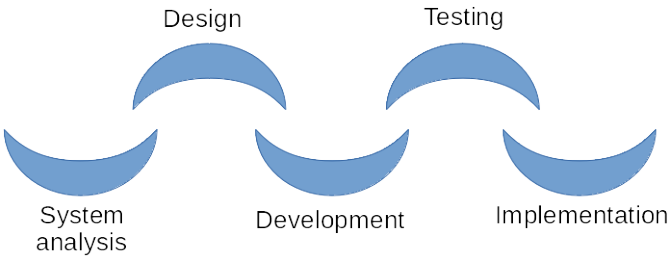

Figure 1. Stages of the life cycle

If you develop a system without evaluating the quality of the system in the early stages of the software life cycle (Figure 1), then assessing the quality at the testing stage of the resulting system, you can find that its quality indicators do not reach the required values, because the algorithms embedded in it do not allow it with any software implementation. This is a very dangerous situation, because it leads to the abandonment of the already developed product and return to the stage of system analysis to replace the algorithms with those that will allow the system to reach the required quality indicators. As can be seen in figure 1, this will actually lead to a repetition of all the work done. Therefore, it is most reasonable to evaluate the characteristics of the system from the very earliest stages of the life cycle, so that solutions that knowingly fulfill the qualitative requirements to the system are always transferred to the next stage. Thus, we will get rid of the risk of abandoning the developed system because of the error in choosing the algorithm in the early stages of the life cycle. This will not only ensure the quality of the system being developed, but also reduce the risks of material and time losses during its implementation.

We offer a quality assurance methodology for guaranteeing the quality of the component of the fault-tolerant software - the decision-making unit in the multi-version execution environment at the design stage by choosing a algorithm that is known to be optimal under known system characteristics. Different decision-making algorithms have their own strengths and weaknesses: some are more resistant to related faults, but they do not work adequately with a large percentage of "inaccuracies," while others, on the contrary, are resistant to both "inaccuracies" and relatively unreliable versions, but they are mistaken for each interverted error [6], etc. Therefore it is necessary to check all of the algorithms in an environment simulating the characteristics of the system under development. Thus, we get the performance characteristics of the algorithm exactly in the conditions in which it will work in our system.

In the case of development for embedded systems and controllers, all software devices, from the operating system to the application software, is actually one executable file, and if an incorrect algorithm choice is detected at the operational stage, replacing even a small software component is a problem, since it leads to the need to recompile the entire project and replace the firmware on the device, which is not always possible, either because of the hardware features of the device, or its inaccessibility, in the case when it o has already been put into operation.

For making a decision what output from multiple versions should be recognized as a true one and sent to the output are applied various algorithms [7], the most common voting algorithms are:

1. The voting algorithm by an absolute majority. It is necessary that one variant is voted by an absolute majority of versions, i.e. (N + 1) / 2, where N - number of versions. For example with five versions it is necessary that one variant is voted at least by three versions, otherwise it is considered that the correct answer cannot be selected.

2. The voting algorithm by an agreed majority. It is necessary that in one version more versions are voted. It is not necessary that the number of votes is more than half of the number of versions. For making a decision, it is enough that more versions are voted for a certain variant than for the others. In the event that the same number of versions voted for in several variants, any one of them is chosen, since it is considered that they are equally "correct".

3. The fuzzy voting algorithm by an agreed majority. This algorithm is similar to the past algorithm by the mechanism of voting. However, the elements from the theory of fuzzy sets are added here, each version can vote for several close answers, but with different degrees of membership to a number from 0 to 1. A number from 0 to 1, which determines the "closeness" to a given value, where 1 is equal to the value, 0 is farther from the compared value than the tolerance E, and in the interval is not equal to the value, but is no more than the tolerance E from it. As a result of this voting, the versions receive a different and no longer an integer number of votes, the version with the largest number of votes is recognized as correct or correct (in case of a coincidence of the number of votes).

4. Median voting. In this version, it assumes that the outputs of all versions are erroneous and as the output is taken the average value of all outputs. This approach is often used in cases where it is impossible to compare directly the outputs of versions. For example, when the outputs are the direction of motion, vector, etc. There are weighted modifications of the median voting, when the contribution of each version in response is different. There are various implementations of the median vote, in our case, all answers are sorted and the middle answer is taken.

II. MODIFICATIONS OF EXISTING ALGORITHMS

As noted earlier, the most dangerous faults are related. For increasing the resistance to these faults, we propose modifications of the basic voting algorithms by an agreed majority and fuzzy voting by an agreed majority. The modifications consist of introducing a dynamic evaluation of the reliability of each version or its "weight", which also has an element of forgetting. The weight is considered as the total of the results of voting, divided by their number. Technically, we implement it in the following way: for each software

version, the system creates a boolean stack of a given length, to which 0 is added. If the voting block decides that, the version gave the wrong answer and 1 if it is a true one. In case of a fuzzy vote, if the value of the version's output belongs to the winning class> 0, that is, all versions that added weight to the class that won the voting will be marked as true, regardless of the indicator of belonging.

Since the stack length is fixed within the simulation (it is set on the form), the new data will replace the oldest ones, that is, the stack works according to the FIFO (First In, First Out) principle. This allows us to enter the forgetting element to the depth of the stack. If, for example, the depth of the stack is 100, then the results of the version work will be older than 100 votes and will not be taken into account. The element of forgetting is necessary to ensure the operative response of the system to changes in the behavior of versions. In the case where versions can significantly change their reliability when changing the input data stream, it is necessary to promptly change their rating for the most correct weighted voting [8].

The rating is determined by the summation of all the elements of the stack, for example, if a stack of 1000 elements has 993 values of "1" and 7 values of "0", the weight of this version will be 0.993.

With software implementation, one restriction is made: the weight of the version cannot be equal to one, which can happen in practice, sufficiently reliable versions give the correct answer 100, 1000, 10000, etc. times and without restraint, versions would get the whole stack of units (or TRUE), which would give them a rating of 1. Such situations should not be allowed, since if this version is not answered correctly, it will receive a weight of 1, while the correct answer with the remaining N-1 versions by weight will only approach 1 and lose the vote. Besides, analytically, the reliability rating of the version equal to 1 does not make sense, because if we have an absolutely reliable software module, the sense of the system is lost. Thus, there is a limitation in calculating the rating of the version with each vote.

III. T/(N-1) ALGORITHM

In addition, the decision making algorithm in multi-version systems proposed by Jie Xu from the University of Newcastle upon Tyne, based on t / (n-1) diagnosability, is of interest [9]. For simplicity, we call it the t / (n-1) decision making algorithm. The essence of the algorithm is not in the voting of all outputs of versions, but only in comparison of some of them, sufficient for making a decision. Let us consider the example of a system with the number of versions N = 5 and the maximum number of faults t = 2, that is, let us consider the 2 / (5-1) version. If the number of faults does not exceed ***t***, the algorithm guarantees that the correct version of the N output versions is selected. However, if the number of incorrect version outputs is exceeded, the system does not necessarily choose the wrong one, but the correct output is determined with certain probability, but this is no longer guaranteed [10]. Let us consider the algorithm in more detail on the given example. The outputs of four versions are compared in pairs - 1 with 2, 2 with 3, 3 with 4, we get three results of comparisons ω12, ω23, ω34, equal to 0, if the outputs are the same and 1 if different. On the basis of only these three comparison results the algorithm makes the decision to switch the output between outputs 1, 4 and 5 versions, that is, versions 2 and 3 are used only for comparison. The values of their outputs are never used as an output of the system. More clearly, the scheme of work can be studied in Figure 1.

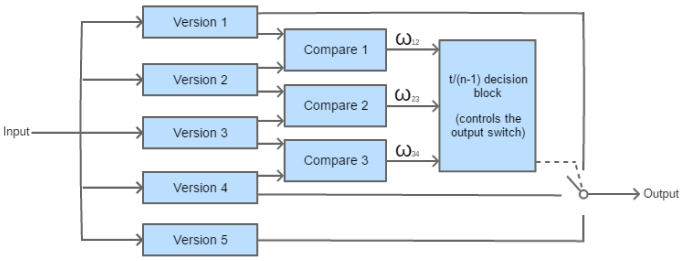

Fig. 2. The architecture of the t / (n-1) algorithm for n = 5 and t = 2.

The figure 1 shows the decision making scheme in the t / (n-1) algorithm is relatively simple. In the case of five multi-version s, only the results of three pair comparisons of the outputs of the four versions are necessary for making a decision (correct control of the output switch). The output value of the fifth version for making a decision is not used. With the logic for controlling the output switch based on the comparison results for n = 5, you can see Table 1.

TABLE 1. POSSIBLE CHOICES BASED ON COMPARATOR OUTPUTS FOR N = 5;

ω_{12}	ω_{23}	ω_{34}	**Supposedly correct versions**
0	0	0	1, 2, 3, 4
0	0	1	1, 2, 3
0	1	0	5
0	1	1	1, 2
1	0	0	2, 3, 4
1	0	1	5
1	1	0	3, 4
1	1	1	5

Having studied Figure 1 and Table 1, one can conclude that with relatively reliable versions in most cases the comparators will return (0; 0; 0) and an output will supply with the value of the execution of the first version. We can also conclude that there is no need to execute the fifth version every time, but only in case of the corresponding values of the results of the comparisons, when it is necessary to submit exactly the result of the fifth version ((0; 1; 0), (1; 0; 1) , (1, 1, 1)). This fact reduces the average load required by the multi-version software execution environment (in most cases, 4 out of 5 versions will be calculated). The decision-making algorithm itself is also significantly less resource-intensive than voting. Especially with its weighted modifications, where every version runs all versions, creates classes and calculates weights for each of them. For t / (n-1) while n = 5, only three simple comparison operations with binary output are needed. Next, an unambiguous, a priori defined output choice is made for one of the eight possible combinations of comparator output values (Table 1).

IV. The Software Implementation of the Simulation Environment

The program implements version simulations that work according to the parameters specified on the form to the number of versions from 3 to 9. Probability of correct operation with three consecutive data streams for each version, the length of the data sets (respectively, the total number of iterations is equal to three lengths), the probability of occurrence of an related fault, the probability of inaccuracy and the tolerance of E. With each vote, the function returns N responses, with probabilities corresponding to each version and current set of input data. The change of reliability working version during the simulation for three sets of input data was introduced in order to investigate the response of the system to a sudden change in the reliability of the versions. For example, if version number 3 had reliability 0.97 in the first set, in the second, reliability dropped to 0.58, and in the third again rose to 0.95.

There are four decision making algorithms implemented in the environment. They are weighed voting by an agreed majority with forgetting, its fuzzy version, t / (n-1) algorithm and its fuzzy modification with modified comparators. Let us consider this process in more detail. When voted by an agreed majority, the voting block receives the responses of the version simulations. If the output value does not coincide with the value obtained before, a new class is created. If the value coincides with the existing class, then the weight of this class will be recalculated as the

$$Ptotal = Pclass + (1- Pclass) * Pversion. \quad (1)$$

After all versions gave the answer, there is a comparison of the weights of the resulting classes, the class with the highest weight wins, as it is clear - this is not always the class for which the largest number of versions voted. Value has the weight of each version, the reliability of each individual version is taken into account. After determining the correct output, the versions that voted for it get a "1" weight on the stack, and the versions that voted differently get "0".

There is no restriction on the response time of versions in this model, since simulations work by one algorithm and there is no sense in comparing their resource intensity. However, when considering real versions, known hardware limitations and requirements to the system reaction time, it makes sense to introduce such a restriction in order to take into account the likelihood that resource-intensive versions may not be able to answer at the time of voting and will not be taken into account. These data can be very important in the case of building systems that operate in real time[11].

In case of a fuzzy voting by an agreed majority after the creation and evaluation of all classes based on the outputs of the versions, the program produces one more pass. In the process of which, the versions whose output value did not coincide with the value of the class, but differ from it no more than the tolerance E, and hence the membership of the class is> 0 also adds weight to the class and weights are recalculated one more time:

$$Ptotal = Pclass + (1- P\text{-}class) * Pversion * Kmembership \quad (2)$$

where K membership = 1- (|X-class-Hversions| / E), X-values of the class and version giving a non-matching answer, but falling within the tolerance E.

In the system under consideration, version simulations give 3 types of errors: a random error simulating a malfunction in the module, an related error and an inaccuracy - a response close to the correct one, a distanced from it no more than tolerance, but not equal to it. This type of error simulates "inaccuracy" - rounding errors when there is a shortage of digits, inaccurate digitization of analog sensor outputs, etc., that is the situation, when the version worked out algorithmically correct, but gave an inaccurate response due to rounding errors, digitization, shortage of capacity, a large difference in the order of magnitude in floating-point operations. The error occurs with the probability given for each version and each data set, in the case of an error, the following checks occur: if this is not the first error in the current poll, then an related error is generated with a given probability - that is, a value that matches the value of the past error is returned, this error simulates a related error-an admitted algorithmic miscalculation, the same in several versions, which will give the same error with the same input. Then, with a given probability, an "inaccuracy" is generated - the output returns, it is not equal to the correct one, but the distant from it for no more than a predetermined deviation E. If the previous probabilities do not work, then a random error is returned, simulating a malfunction in the current version of the module. From the results of the program it is clear that in the absence of "inaccuracies" the difference in the work of clear and fuzzy options for voting methods is absent, a random error is always too "far" from the correct answer to change the weight of classes. From this it can be concluded, that if there are no possible places for the occurrence of "inaccuracies" in the system - the digitization of analog signals, the lack of capacity for mathematical operations, etc., then using a more resource-intensive fuzzy voting algorithm will not give advantages, however, if such "inaccuracies" are possible, then a fuzzy algorithm will increase the reliability of the system. The only drawback will be the same evaluation for the versions that gave the ideally correct output, and the versions algorithmically correct, but having "inaccuracies".

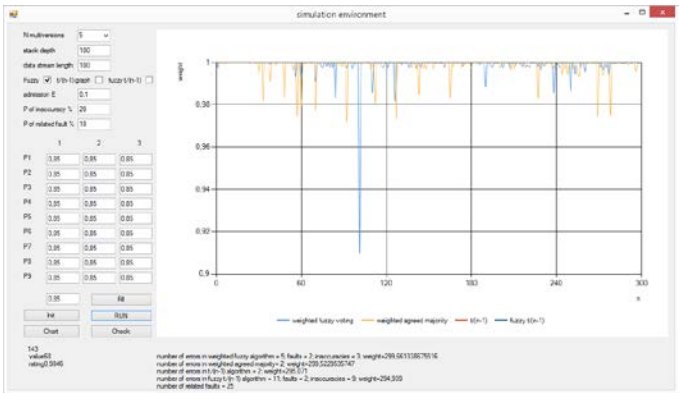

Fig. 3. Interface of the simulation environment (graphics t/(n-1) are disabled)

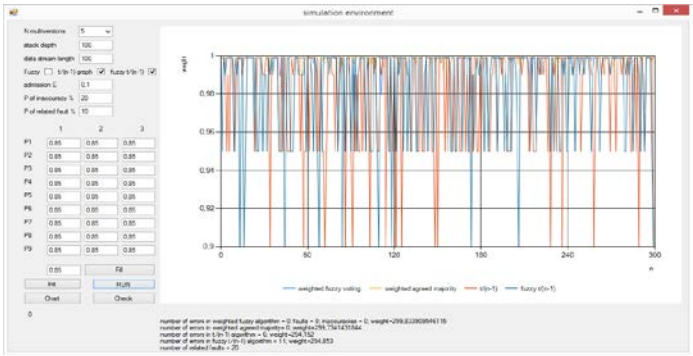

Fig. 4. Interface of the simulation environment (all graphs are displayed)

For convenience, the form displays information about the last recorded error at the output of the voting block - the iteration number, the output value, and the weight of this class. The system allows you to plot the weights of each version and the winning classes, along the axes of the iteration number and weight, it is possible to change the scale for clarity, for example, when examining the weights of the winning classes, their values do not fall below 0.9 and on a scale from zero to one represent of itself a practically flat line in the region of unity, for a better perception in this mode, the scale changes to a range from 0.9 to 1 along the weight axis.

In order to be able to compare by the sums of weights t / (n-1) algorithm with weighted voting algorithms, we introduced weight estimates for the t / (n-1) algorithm based on the output of the comparators, since these are not real weights, but only an estimate that takes only one of 4 preset variants (0.999, 0.99, 0.95, 0.9), the graphs are not informative and greatly complicate the perception of graphs of scales of voting algorithms. Therefore, the graphs of t / (n-1) algorithms are made with the capability to turn-off on the form, for more convenient study of the voting algorithms, however, disabling the graphs does not disable counting the sum of weights (or their estimates) during simulation. You can compare the appearance of the graphs in Figures 2 and 3.

Based on the simulation results, the sum of errors for all iterations for each algorithm, the number of "inaccuracies" admitted for fuzzy algorithms, and the sum of the resulting related errors are displayed. If based on the simulation results exists algorithm that is uniquely superior to other algorithms, the system displays a message about the selected optimal algorithm (Figure 5).

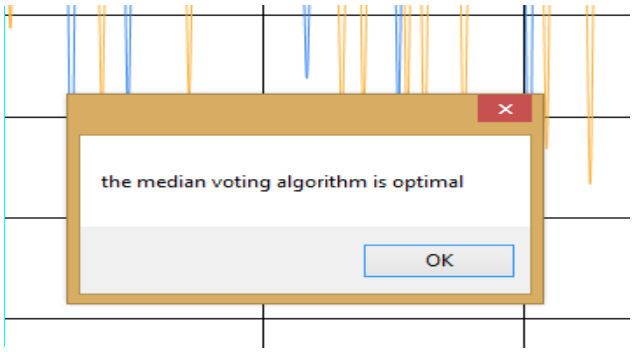

Figure 5. System message about the selected optimal algorithm.

V. Simulation Results

Let us consider the results of a program with different input parameters.

First, let's study the graphs of the version weights presented in Figures 6-9. There are graphs for a stack of 100 deep and three consecutive flows of data for 100 votes for each. In Figure 6, we can observe the operation of the system for all reliable versions (version reliability in all data streams from 0.9 to 0.99), as you can see - the versions weights also change in the minimum limits, without falling below the values of 0.9.

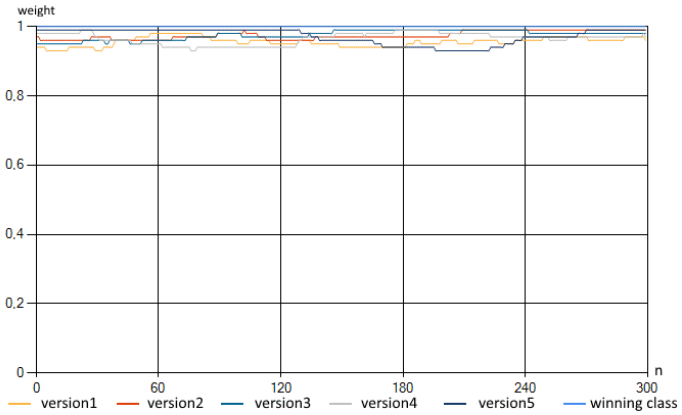

Figure 6. Results of the simulation model implementation of the voting algorithm by an agreed majority (high reliability of all versions 0.9-0.99).

In Figure 7, we can observe the operation of the system with the average reliability of versions (reliability of versions in all data streams from 0.7 to 0.93), as can be seen - the versions weights change already in large limits, but do not drop much below 0.7.

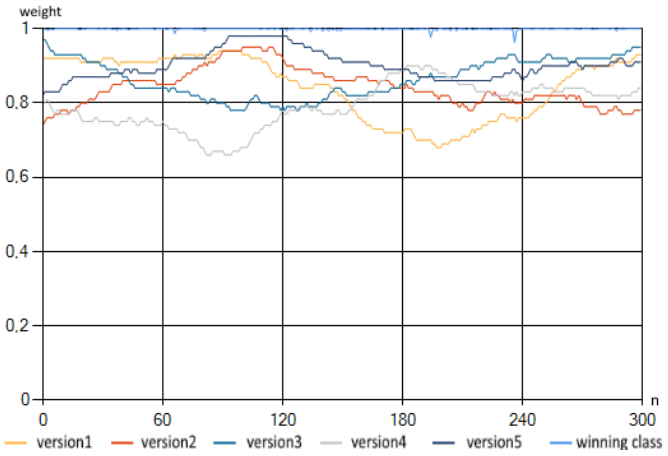

Figure 7. Results of the simulation model implementation of the voting algorithm by an agreed majority (average reliability of all versions 0.7-0.93).

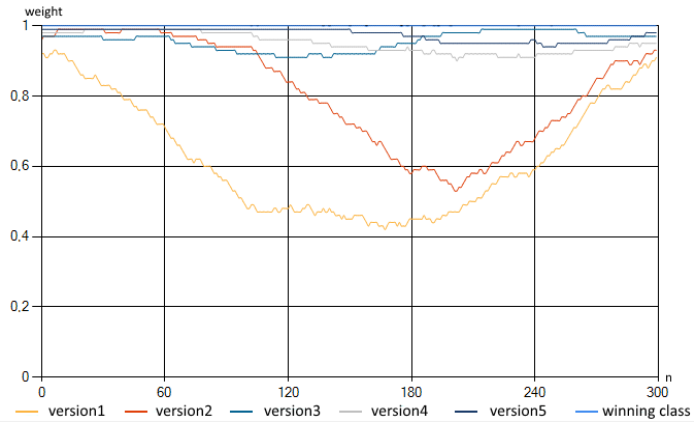

Figure 8. Results of the simulation model implementation of the voting algorithm by the agreed majority (the first version of the first and second data sets failed in the first version of the second data set with a 50% probability).

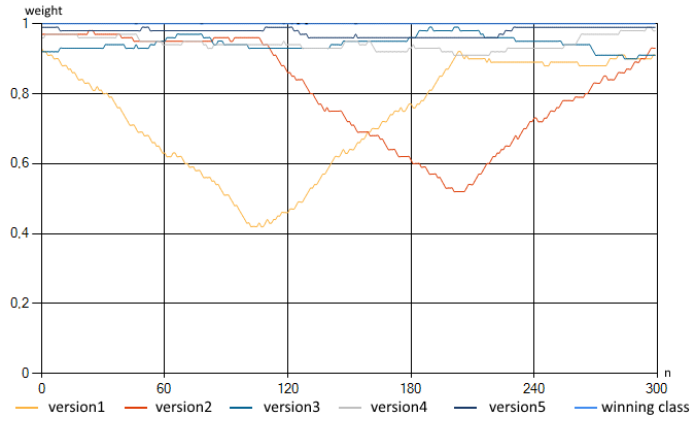

Fig. 9. The results of the simulation model of the implementation of the voting algorithm by an agreed majority (the first version of the first data set has failed and in the second version of the second data set with a 50% probability).

The graphs (Figures 8-9) shows the system's response to the behavior change of versions, when a reliable version starts to give errors or vice versa, the low-weight version ceases to be wrong (at the beginning of the experiment, all weights of the versions are relatively high). In Figure 8, the first version make mistakes with a 50% probability on the first and second data stream, the third version stop making mistakes. In Figure 9, the first version is mistaken with a 50% probability on the first data stream, ceases to make mistakes on the second and third streams, and the second version starts to make errors on the second data stream and stops at the third one. It can be concluded that the system reacts fairly quickly to changes in the behavior of versions and, through the number of votes equal to the depth of the version stack, receive estimated weights that are fairly accurate in their probability of a correct answer.

TABLE 2. THE RESULTS OF THE SYSTEM EXECUTION AT DIFFERENT NUMBERS OF MULTI-VERSION S FOR UNRELIABLE VERSIONS WITH P=0.7.

N		**3**	**4**	**5**	**6**	**7**	**8**	**9**
Clear	***Number of errors***	49	22	16	7	5	3	0
	Sum of weights	269,43	286,53	291,89	296,05	298,16	298,67	299,21
Unclear	***Number of errors***	88	45	28	13	11	4	2
	Sum of weights	279,45	289,04	295,54	297,70	298,70	299,46	299,70
Related error			22	42	48	84	89	123

Table 2 summarizes the simulation results for a different number of multi-version s - from 3 to 9, with the probability of all versions in all data sets equal to P = 0.7, the depth of stack 100, the tolerance E = 0.1, the probability of inaccuracy P = 20%, the probability of related error P = 10 %. Such a low reliability index of all versions is taken as 0.7 for the sake of clarity of the simulation results, because with quite reliable versions (P> 0.95) even with the number of multi-version s N = 3, both voting algorithms give no errors for 300 votes [12]. From the data we see how the reliability of the system as a whole is increased, despite the use of extremely unreliable software modules, despite the fact that each module emits an error in 30% of cases, it can be observed that in a system with 9 versions all 300 answers are correct, the algorithm of clear voting by the agreed majority always chooses the right answer from the proposed ones. The fuzzy algorithm shows more errors, because sometimes a class with a value close to the correct one (no further than E from the correct one), but not equal to it, wins in voting. The system makes such a response erroneous, but the versions that add weight to the winning class still get in stack 1, increasing its own weights, therefore, despite the greater number of errors, the sum of weights in the fuzzy algorithm is higher. In the model, the t / (n-1) algorithm is implemented only for N = 5, therefore, only voting algorithms are presented in this table, because they are able to work with any number of versions N≥3.

Let us compare the reliability indexes of the t/(n-1) algorithm in comparison with the weighted modifications of the voting algorithm by the agreed majority that we proposed, its clear and not clear version. To do this, we simulate in the simulated imitation environment with the following model parameters: the number of iterations = 300, the same reliability of all five versions in all 300 runs, the stack depth = 100 (for weighted algorithms), the tolerance E = 0.1 (for fuzzy modification), the probability of inaccuracy P = 20%, probability of related error P = 10%. Let's change the version reliability from the relatively reliable value 0.95 to 0.65 in steps of 0.05 and get the number of errors for 300 iterations for each algorithm. The simulation results are presented in table 3.

TABLE 3. THE RESULTS OF MODELING WITH DIFFERENT VERSION RELIABILITY.

Given version reliability.		**0,65**	**0,7**	**0,75**	**0,8**	**0,85**	**0,9**	**0,95**
Clear voting by an agreed majority	***Number of errors***	22	12	9	1	1	0	0
	Sum of weights	287,99	294,05	296,36	298,33	299,75	299,96	299,99
Unclear voting by an agreed majority	***Number of errors***	24	14	7	3	1	0	0
	Of these, failures	14	10	3	2	1	0	0
	inaccuracies	10	4	4	1	0	0	0
	Sum of weights	293,75	295,86	298,43	299,43	299,74	299,97	299,99
t/(n-1) algorithm	***Number of errors***	39	24	17	10	3	1	0
	Sum of weights	285,07	287,06	290,65	291,74	293,06	295,8	297,87
Fuzzy modification of t/(n-1) algorithm	***Number of errors***	44	35	34	21	16	4	3
	Of these, failures	19	14	14	2	2	0	0
	Inaccuracies	25	21	20	19	14	4	3
	Sum of weights	290,44	291,85	291,68	296,61	294,68	296,98	298,39
Median voting	***Number of errors***	73	45	29	16	8	2	0
Related errors		134	94	66	41	26	10	1

It can be seen from the simulation results that with relatively reliable versions all algorithms provide error-free operation for 300 iterations, although each of the 5 versions gives an average of 15 erroneous outputs during this time (with a reliability of 0.95, the simulation version will give an average of 5 errors per 100 iterations). The exception is a fuzzy t / (n-1) algorithm, the simulation results show that the fuzzy modification is very unstable to the occurrence of inaccuracies. It has not missed a single failure, but when the inaccuracies are returned by the first or fourth version, fuzzy comparators return a match with the neighboring versions that gave perfectly correct answer and the system sends the output with an inaccurate result. With decreasing version reliability, algorithms start to make mistakes, but in different numbers. The algorithm of coordinated voting, more precisely, its weighted modification with forgetting turned out to be the most reliable. The greater number of mistakes in the fuzzy algorithm is due to cases where the answer of "inaccuracy" is chosen; the answer is remote from the correct one by no more than the E tolerance, but not equal to it. Such responses are also counted by the system as erroneous. t/(n-1) algorithm begins to yield significantly in reliability to the weighted voting algorithm by an agreed majority with relatively unreliable versions, since more often happen situations in which more than t = 2 versions give the wrong answer, and in such cases the correct operation of the algorithm is not guaranteed. The results also show that the fuzzy version of the t/(n-1) algorithm gives more errors in general, but most of the errors are past inaccuracies, the number of failures is even less than the base algorithm has.

From the presented above, we can conclude that the use of t/(n-1) algorithm is possible with relatively reliable versions, when there will not be situations of simultaneous failure of more than t versions. Its application will be justified in situations where it is necessary to reduce the computational load on the system, especially in cases where the comparators are simple to implement, but to create multiple classes each time and calculate their weights is too labor-intensive (depends on the architecture of the system). However, in cases of using relatively unreliable versions, or high probability of related error (when several versions will give the same failure simultaneously), its application is not desirable, since in such situations it shows less reliability in comparison with multi-version voting algorithms. Concerning the fuzzy modification, its application is justified in systems with the probability of inaccuracies, the passage of which is not critical for the system, since the algorithm often allows them to pass, but it cuts off failures well enough.

Let us explore the behavior of the system with different probabilities of the occurrence of inaccuracies, the results are presented in table 4.

TABLE 4. THE RESULTS OF THE SIMULATION FOR DIFFERENT PROBABILITY OF INACCURACY AND P = 0.8 FOR ALL VERSIONS.

Probability of inaccuracy		**5%**	**10%**	**20%**	**50%**	**100%**
Clear voting by an agreed majority	***Number of errors***	2	2	3	2	3
	Sum of weights	298.55	298.87	298.84	298.46	298.34
Unclear voting by an agreed majority	***Number of errors***	2	3	7	8	5
	Of these, failures	1	3	2	3	0
	Inaccuracies	1	0	5	5	5
	Sum of weights	299.09	299.09	299.52	299.74	299.98
t/(n-1) algorithm	***Number of errors***	8	10	13	11	11
	Sum of weights	291.72	292.16	291.81	291.83	292.71
Fuzzy modification of t/(n-1) algorithm	***Number of errors***	10	12	25	30	69
	Of these, failures	5	5	8	2	0
	inaccuracies	5	7	17	28	69

	Sum of weights	292.91	293.27	292.94	295.95	299.65
Median voting	***Number of errors***	14	14	15	17	18
Related fault		48	41	44	43	50

From the results, it can be seen that when the probability of inaccuracy increases, fuzzy modifications of decision-making algorithms allow the passage of inaccuracies, but remain resistant to failures [13], and in order to choose the most suitable algorithm it is necessary to understand how critical the system is for passing exactly the inaccuracies. For example - for many course maintenance systems with permanent correction of inaccuracy with a slight deviation from the correct value will not have a negative effect on the system work, in contrast to failures - when the system significantly changes direction, which can lead to catastrophic consequences. In the case of systems that are unstable to inaccuracies, it is better to use classical variants of algorithms, since they better screen out the inaccuracies, considering them incorrect, regardless of their proximity to the correct answer, unlike fuzzy variations.

Let us explore the behavior of the system with different probability of occurrence of an related error, which is of the greatest interest, since it will show the stability of all the studied algorithms to the most dangerous type of errors, the results are presented in table 5.

TABLE 5. THE RESULTS OF THE SIMULATION FOR DIFFERENT PROBABILITY OF OCCURRENCE OF AN RELATED ERROR AND P = 0.8 FOR ALL VERSIONS.

The probability of occurrence of an related error.		**5%**	**10%**	**20%**	**50%**	**100%**	**100% (P=0,95)**
Clear voting by an agreed majority	***Number of errors***	2	3	4	5	13	0
Unclear voting by an agreed majority	***Number of errors***	2	3	6	13	15	0
	Of these, failures	1	2	2	10	11	0
	Inaccuracies	1	1	4	3	4	0
t/(n-1) algorithm	***Number of errors***	4	7	10	16	21	1
Fuzzy modification of t/(n-1) algorithm	***Number of errors***	10	17	19	23	22	7
	Of these, failures	3	4	6	12	13	2
	inaccuracies	7	13	13	11	9	5
Median voting	***Number of errors***	14	15	19	16	17	0
Related errors		26	41	66	204	392	31

The simulation results show a much greater stability of the voting algorithms to related errors, compared with the t/(n-1) algorithm. An additional column is also added to the table to show that with quite reliable versions in the whole (P = 0.95 for all) even 100% of related errors do not affect the reliability of the system with the proposed modified voting algorithms, which can not be said for t/(n-1) algorithm that tolerates failures even with reliable versions. It is interesting to note that the related errors do not affect the operation of the median voting algorithm, because it does not compare version responses for changing the weights of classes, and the collection of answers is simply sorted by the value of the values. For a median vote, there is no difference, the matching errors give out versions or not. Let's consider an example - if 5 versions gave answers (3, 3, 6, 19, 19) and answers "3" and "19" are related errors, it will still choose the answer "6", as average. This same property of its work makes it resistant to emissions, unlike the implementation of voting algorithms with averaging of outputs, if one or several versions give an answer differing by several orders in any direction, this will not affect the operation of the algorithm.

The results obtained with the help of the developed simulation imitation environment show the effectiveness of the proposed modifications of the voting algorithms, and also prove the possibility of creating a reliable system from unreliable software modules [14]. As for the t/(n-1) algorithm, which is of interest as an alternative to voting algorithms, its use is justified only in systems with significant limitations on computational resources and the use of sufficiently reliable versions, since the algorithm shows a worse resistance to unreliable versions and different types errors.

CONCLUSION

the developed system ensures the quality of the component, since it allows to obtain its qualitative characteristics in the conditions in which it will work in a real system. this result is very important for software development, especially for complex fault-tolerant systems, since it allows to get an estimate of the quality characteristics of a component at an early stage of development. this allows you to select deliberately appropriate decision algorithms. the choice of the decision algorithm with guaranteed quality characteristics that meets the requirements excludes the case of re-development of the system due to the fact that only during the testing phase it turns out that the chosen algorithm is unable to meet the established requirements. this not only guarantees the quality of the software product being developed, but also makes the implementation time more predictable.

ACKNOWLEDGMENT

This work was supported by Ministry of Education and Science of Russian Federation within limits of state contract № 2.2867.2017/4.6.

REFERENCES

[1] Engel E.A., Kovalev I.V., Engel N.E., Brezitskaya V.V., Prohorovich G.A., «Intelligent control system of autonomous objects», IOP Conf. Series: Materials Science and Engineering 173 (2017) 012024. doi:10.1088/1757-899X/173/1/012024.

[2] Eckhardt, D.E., and Lee, L.D., "A Theoretical Basis for the Analysis of Multi-version Software Subject to Coincident Errors," IEEE Transactions on Software Engineering, vol. SE-11, no. 12, 1985, pp. 1511-1517.

[3] Laprie, J.-C., Arlat, J., Beounes, C., and Kanoun, K., "Definition and Analysis of Hardware- and Software-Fault-Tolerant Architectures," IEEE Computer, vol. 23, no. 7, July 1990, pp. 39-51. Reprinted in Fault-Tolerant Software Systems: Techniques and Applications, Hoang Pham (ed.), IEEE Computer Society Press, 1992, pp. 5-17.

[4] L. Chen, A. Avizienis, "N-Version Programming: A Fault-Tolerant Approach to Reliability of Software Operation", Proc. Int. Symp. Fault-Tolerant Computing FTCS-8, pp. 3-9, 1978.

[5] Knight, J.C., and Leveson, N.G., "An Experimental Evaluation of the Assumption of Independence in Multi-version Programming," IEEE Transactions on Software Engineering, vol. SE-12, no. 1, 1986, pp. 96-109

[6] Kovalev, I., Losev, V., Saramud, M.Email Author, Petrosyan, M., « Model implementation of the simulation environment of voting algorithms, as a dynamic system for increasing the reliability of the control complex of autonomous unmanned objects», MATEC Web of Conferences Volume 132, 31 October 2017, № 04011.

[7] G. Latif-Shabgahi; S. Bennett Adaptive majority voter: a novel voting algorithm for real-time fault-tolerant control systems Proceedings 25th EUROMICRO Conference. Informatics: Theory and Practice for the New Millennium, 1999, pp.113 - 120 vol.2

[8] Kovalev I.V., Zelenkov P.V., Losev V.V., Kovalev D.I., Ivleva N.V., Saramud M.V. Multi-version environment creation for control algorithm implementation by autonomous unpiloted objects [Electronic resource] // IOP Conf. Series: Materials Science and Engineering 173 (2017) 012025.

[9] J. Xu, “The t(n-1)-diagnosability and its applications to fault tolerance”, Digest of Papers. Fault-Tolerant Computing: The Twenty-First International Symposium, 1991, pp. 496 – 503.

[10] Jie Xu; B. Randell, “Software fault tolerance: t/(n-1)-variant programming”, Software fault tolerance: t/(n-1)-variant programming

IEEE Transactions on Reliability, 1997, Volume: 46, Issue: 1, pp. 60 – 68.

[11] Lee, I., Tang, D., lyer, R.K., and Hsueh, M.C., "Measurement-Based Evaluation of Operating System Fault Tolerance," IEEE Transactions on Reliabil ity, vol. 42, no. 2, June 1993, pp. 238-249.

[12] Brilliant, S.S., Knight, J.C., and Leveson, N.G., "Analysis of Faults in an N-Version Software Experiment," IEEE Transactions on Software Engineering, vol.16,no.2, 1990, pp.238-247.

[13] McAllister, D.F., Sun, C.E., and Vouk, M.A., "Reliability of Voting in Fault –Tolerant Software Systems for Small Output Spaces," IEEE Transactions on Reliability, vol. 39, no. 5,1990, pp. 524-534.

[14] Avizienis, A., "The N-Version Approach to Fault -Tolerant Software," IEEE Transactions on Software Engineering, vol. SE-11, no. 12, December 1985, pp.1491-1501.