UDC 004.048 **doi:** https://doi.org/10.20998/2522-9052.2025.2.08

Svitlana Krepych[1], Iryna Spivak[1], Serhii Spivak[2], Roman Krepych[3]

[1] West Ukrainian National University, Ternopil, Ukraine
[2] Ternopil Ivan Puluj National Technical University, Ternopil, Ukraine
[3] Kamianets-Podilskyi State Institute, Kamianets-Podilskyi, Ukraine

THE METHOD OF ASSESSING THE RELIABILITY OF SOFTWARE SYSTEMS BASED ON A GRAPHIC MODEL OF THE DEPENDENCE OF METHODS OF THE SYSTEM UNDER TEST

Abstract. Today, software has become an integral part of many areas of our daily life — from automation and optimization of production processes to the creation of individual comfort. Programs make our lives easier, solving tasks in seconds that used to take hours or even weeks, as well as giving us convenience and comfort that people of previous generations could not even dream of. In order to meet the growing demand for new IT software, the market around the world and in the country in particular is also growing and changing rapidly. According to the IT Ukraine Association, compared to 2017, the number of employed specialists in the labor market of Ukraine increased by approximately two times, and the volume of export of IT services - by two and a half. Despite the fact that due to the full-scale invasion, the pace of development has slowed down in 2022-2024, it is clear that the industry has not reached its peak, which means that it will continue to develop. In addition to the obvious changes related to the expansion of the market, there are also internal changes in the processes of the industry due to the desire to increase the speed of program development, as well as to reduce the final price of the software product. It is common knowledge that high quality software is an integral part of a successful product. However, even at a fairly low pace of development, developers often make mistakes that lead to serious problems, affecting security, reliability, and user satisfaction. So what can be said about the development in a short time? That is why ensuring the high quality of the software product is one of the main tasks that must be solved at the development stage. **The object of the research** is the process of assessing the reliability of software systems. The **subject of the research** is a method of assessing the reliability of software systems based on a graphic model of the dependence of the methods of the system under test. **Conclusion**: on the basis of the method of evaluating the reliability of software systems based on the graphical model of the dependence of the methods of the system under test, software was developed in the Java and Kotlin programming languages for evaluating the reliability index of software systems of any architectural complexity.

Keywords: software quality; software reliability; testing; modeling of complex systems; graphical system model.

Introduction

Today, software has become an integral part of many areas of our daily life — from automation and optimization of production processes to the creation of individual comfort. Programs make our lives easier, solving tasks in seconds that used to take hours or even weeks, as well as giving us convenience and comfort that people of previous generations could not even dream of.

In order to meet the growing demand for new IT software, the market around the world and in the country in particular is also growing and changing rapidly. According to the IT Ukraine Association, compared to 2017, the number of employed specialists in the labor market of Ukraine increased by approximately two times, and the volume of export of IT services - by two and a half [1, 2]. Despite the fact that due to the full-scale invasion, the pace of development in 2022-2024 has slowed down, it is obvious that the industry has not reached its peak, which means that it will continue to develop [3].

In addition to the obvious changes associated with the expansion of the market, there are also internal changes in the processes of the industry due to the desire to increase the speed of software development, as well as to reduce the final price of the software product [4].

It is common knowledge that high quality software is an integral part of a successful product [5, 6]. However, even at a fairly low pace of development, developers quite often make mistakes that lead to serious problems, affecting security, reliability and user satisfaction. So what can be said about the development in a short time? That is why ensuring the high quality of the software product is one of the main tasks that must be solved at the development stage.

Many concepts and approaches to software reliability assessment have been adapted from older but successful hardware systems reliability assessment methods [7]. However, due to significant fundamental differences in the nature of hardware and software and their failure processes, such approaches can (and usually do) not work very well for software [8, 9]. Since the processes of natural destruction are statistically independent, in the context of hardware, the use of redundancy, that is, hot or cold redundancy, allows creating systems with very high reliability indicators. Another source of failure is design errors. Mostly they arise due to the human factor in the process of development or maintenance. The probability of a design defect, as a rule, depends only on the method of use and does not depend on time. Unlike hardware, software, at least in theory, can be completely error-free [10, 11].

Let's take a closer look at the existing evaluation models and start with models based on the black box. A large number of models of this type have been created, but the basis of all of them is the study of two subjects:

- the number of failures for a certain time (calendar time or time in computing units, which takes into account the load on the software);
- time intervals between failures.

The distribution of the number of failures over time plays an important role in the classification of reliability models. According to this principle, they are divided into Poisson and binomial [12]. Binomial-type models assume that there is an initial specific number of errors in the program, and that there is a one-to-one correspondence between errors and failures. After each failure, its cause is eliminated, so that this failure is no longer repeated, and the number of errors decreases (no new errors are introduced).

This assumption causes each bug in the software to occur only once and is independent of the others. At the same time, each failure occurs randomly according to the failure intensity, which is the same throughout the debugging time [13, 14].

Piason-type models are based on the fact that the initial number of faults is a Poisson random variable with a certain mean [15].

As an alternative to these models, models that assume debugging imperfections have been proposed, assuming that [15, 16]:

● failure detection coefficient is not a constant value;

● during each failure, the error that caused the failure is corrected, but new errors may occur.

Architecture-Based Reliability Models (White Box). Large software systems almost always consist of smaller blocks that are responsible for separate parts of the functionality [17]. The main advantage of architectural reliability prediction models is that it is possible to predict system reliability already at the early stages of software design. System-wide failure data is not required, as is the case with black-box models, so potential quality problems can be identified before the system or prototype is fully completed and where a black-box approach can be applied [18, 19].

Statement for the task

As mentioned above, in order to reduce the cost of development and shorten its terms, it is important to detect errors as early as possible, because with iterative development, the complexity and, as a result, the cost of correcting an failure increases over time, and the probability of correcting it correctly (without introducing new failures) decreases [20]. On the other hand, almost all large software products are too complex to contain any bugs at all. In view of the above, the task of development is not to guarantee the absence of errors, but to maintain a balance between the conditional reliability of the final software system and costs at all stages of development.

The purpose of the research is to create software that would allow conducting research on the reliability of software systems at various stages of testing and taking into account their complexity and structure.

To achieve this goal, the following tasks must be solved:

● Create an algorithm for determining dependencies between program methods.

● Create algorithms for analysis and processing of data coming from the evaluated system during the testing process. This includes determining the probabilities of calling methods, handling exceptions, and other aspects.

● Implement an algorithm that allows you to numerically assess the reliability of the program based on the received data. In addition to estimating the probability of failure-free operation, the algorithm can provide for the determination of other reliability metrics.

● Create a user interface that will allow you to set the necessary input parameters, as well as display the results of the analysis in an easy-to-understand form.

● Test the program on real examples to check the correctness of the developed software, as well as its efficiency and accuracy.

Main part

A block diagram demonstrating the essence of the proposed approach is shown in Fig. 1. The decision to develop the main part of the system as a plug-in to the integrated development environment IntelliJ IDEA from the company JetBrains is due to the fact that the IDE has an open and documented API that provides access to a large the number of functionalities related to the analysis of software code and interaction with it. Thanks to this, there is no need to implement algorithms for parsing java files, in order to find methods based on formalized features, as well as various ways of interacting with the code during its execution.

To isolate the call tree, we will use the IntelliJ Idea API, which allows you to get a link to all nested methods in the body of the parent in a few lines of code. Fig. 2 shows the program code of the processMethod() method, which is responsible for building the method call tree.

This code recursively bypasses all nested methods in the body of the parent, preserving the structure and sequence of calls. For the correct construction of the graph, the sequence of calls is very important, since the transfer of stack control occurs synchronously, and therefore the first submethod in the body of the parent, after starting execution, will own the stack until all instructions in its body, as well as all its submethods, are executed. Only after that, stack control will be transferred to the second submethod (if available).

To increase the efficiency of calculations, the algorithm skips getters and setters of objects when creating a tree of method calls.

Getters and setters are special methods that are responsible for getting and setting the values of object fields. These methods, usually with few exceptions, do not contain business logic and are simple, which essentially means that they cannot cause the application to failure. The isGetter() and isSetter() methods are responsible for checking whether a method belongs to the category of getters or setters.

As mentioned above, the program does not have its own graphical interface and uses the context menu of the studio for interaction. Fig. 3 shows the context menu that opens when you right-click on a method in the studio editor.

Each menu item has validation and is displayed and activated only under certain conditions, for example, for the "Add method to assessment" item to be active, firstly, the click must take place on the method, and secondly, this method must not be in the list of previously added ones.

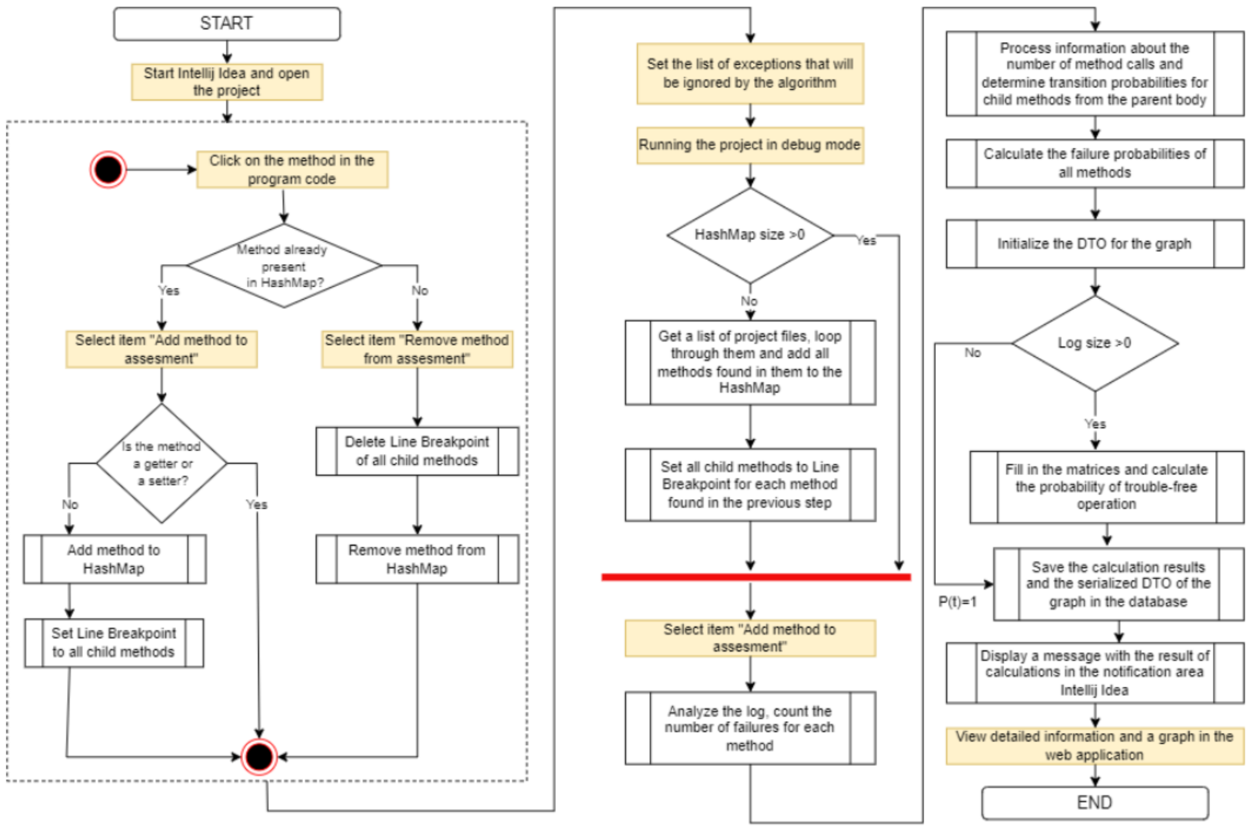
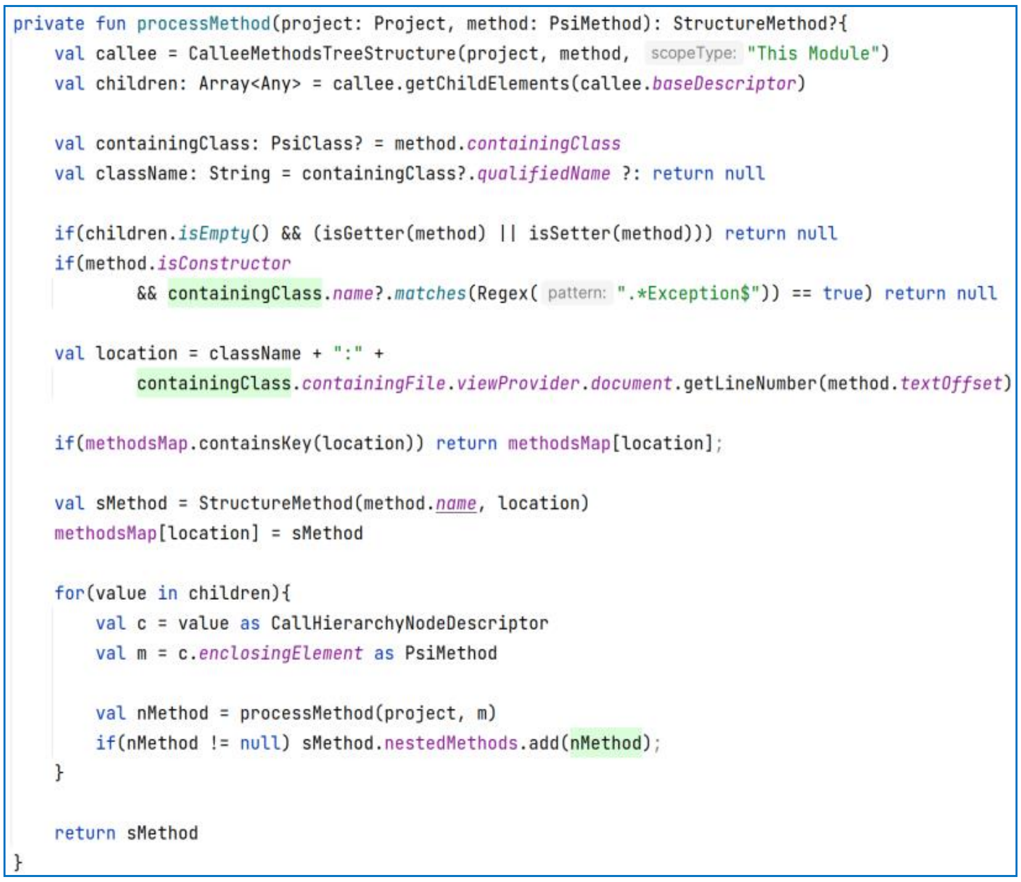

START
Start Intellij Idea and open the project
Click on the method in the program code
Method already present in HashMap?
Yes
No
Select item "Add method to assesment"
Select item "Remove method from assesment"
Is the method a getter or a setter?
No
Yes
Add method to HashMap
Set Line Breakpoint to all child methods
Delete Line Breakpoint of all child methods
Remove method from HashMap
Set the list of exceptions that will be ignored by the algorithm
Running the project in debug mode
HashMap size >0
Yes
No
Get a list of project files, loop through them and add all methods found in them to the HashMap
Set all child methods to Line Breakpoint for each method found in the previous step
Select item "Add method to assesment"
Analyze the log, count the number of failures for each method
Process information about the number of method calls and determine transition probabilities for child methods from the parent body
Calculate the failure probabilities of all methods
Initialize the DTO for the graph
Log size >0
No
Yes
Fill in the matrices and calculate the probability of trouble-free operation
P(t)=1
Save the calculation results and the serialized DTO of the graph in the database
Display a message with the result of calculations in the notification area Intellij Idea
View detailed information and a graph in the web application
END

Fig. 1. Block diagram of the proposed approach

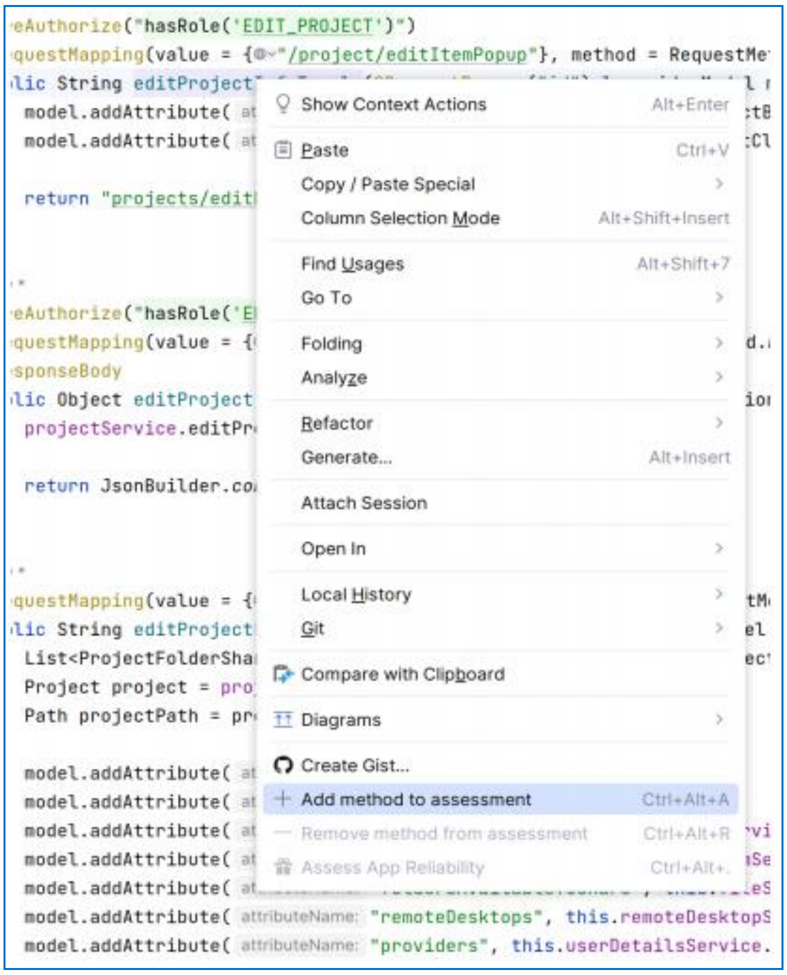


```
private fun processMethod(project: Project, method: PsiMethod): StructureMethod?{
    val callee = CalleeMethodsTreeStructure(project, method,  scopeType: "This Module")
    val children: Array<Any> = callee.getChildElements(callee.baseDescriptor)

    val containingClass: PsiClass? = method.containingClass
    val className: String = containingClass?.qualifiedName ?: return null

    if(children.isEmpty() && (isGetter(method) || isSetter(method))) return null
    if(method.isConstructor
            && containingClass.name?.matches(Regex( pattern: ".*Exception$")) == true) return null

    val location = className + ":" +
            containingClass.containingFile.viewProvider.document.getLineNumber(method.textOffset)

    if(methodsMap.containsKey(location)) return methodsMap[location];

    val sMethod = StructureMethod(method.name, location)
    methodsMap[location] = sMethod

    for(value in children){
        val c = value as CallHierarchyNodeDescriptor
        val m = c.enclosingElement as PsiMethod

        val nMethod = processMethod(project, m)
        if(nMethod != null) sMethod.nestedMethods.add(nMethod);
    }

    return sMethod
}
```

Fig. 2. Program code of the method processMethod()

The function "Remove method from assessment" works according to the reverse principle, the only difference is that when removing a method from the tree, the presence of dependencies of other branches on this method is checked, and if there are any, then the method is removed only from that branch, according to the method which was clicked with the mouse. To take into account the complexity of the software system in the process of reliability assessment, a probabilistic graph model of methods is used, which allows you to present the program in the form of a directed graph. The vertices are the methods (functions) of the program, and the connections between these methods are the edges. Figure 4 shows a graphical representation of a program containing 4 methods. This directed graph contains 6 vertices, four of which are responsible for methods, and two more are fictitious, representing entry and exit points from the code.

The values P_1, P_2, P_3 and P_4 are the probabilities of failure-free operation of the corresponding methods, and $p(i, j)$ are the probabilities of calling the method j from the body of the method i. The probability of failure-free operation of each method is calculated as the ratio of the number of its calls that resulted in an error to the total number of method calls.



```
eAuthorize("hasRole('EDIT_PROJECT')")
questMapping(value = {⊕∨"/project/editItemPopup"}, method = RequestMe
lic String editProject
  model.addAttribute( at                                              tE
  model.addAttribute( at                                              Cl

  return "projects/edit

*
eAuthorize("hasRole('E
questMapping(value = {                                                d.
sponseBody
lic Object editProject                                                io
  projectService.editPr

  return JsonBuilder.co

*
questMapping(value = {                                                tM
lic String editProject                                                el
  List<ProjectFolderSha                                               ec
  Project project = pro
  Path projectPath = pr

  model.addAttribute( at
  model.addAttribute( at
  model.addAttribute( at                                              vi
  model.addAttribute( at                                              Se
  model.addAttribute( at                                              eS
  model.addAttribute( attributeName: "remoteDesktops", this.remoteDesktopS
  model.addAttribute( attributeName: "providers", this.userDetailsService.
```

Fig. 3. Program code of the method processMethod()

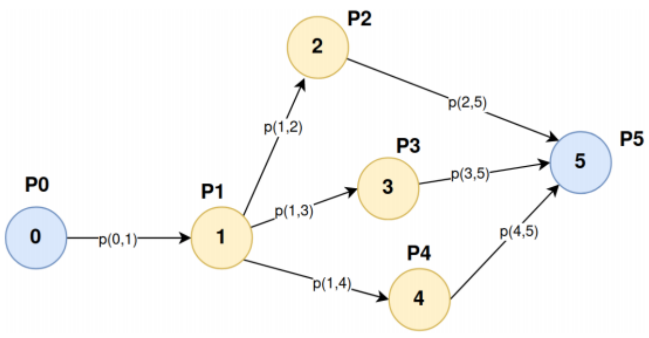


Fig. 4. An example of a graphical program model

Transition probabilities are calculated in proportion to the number of calls of nested methods from the body of the i-th method, so that the sum of all transition probabilities from the body of the i-th method equals:

$$\sum_{j=j_{\min}}^{j_{\max}} p(i, j) = 1 . \qquad (1)$$

The next step is to construct a matrix of transition probabilities P with dimension m, where m is the number of vertices of the graph. In this matrix, the value of the element $P_{i,j}$ is the value of the probability of the corresponding transition between the corresponding methods, i.e. $p(i, j)$. If there are no connections between the corresponding methods, zero is written in place of the corresponding element. The next step is to calculate the matrix G, for which the j-th row of the matrix P is multiplied by the probability of failure-free operation of the j-th method (for fictitious vertices, this parameter is equal to one).

Finally, it is necessary to calculate the matrix T, which in the case of an acyclic graph is calculated according:

$$T = I + G + G^2 + G^3 + ... + G^m , \qquad (2)$$

where I is the unit square matrix of dimension m.

If the graph of the studied program is not acyclic, that is, it contains paths leading from the vertex to itself, then the matrix T is calculated:

$$T = (I - G)^{-1} . \qquad (3)$$

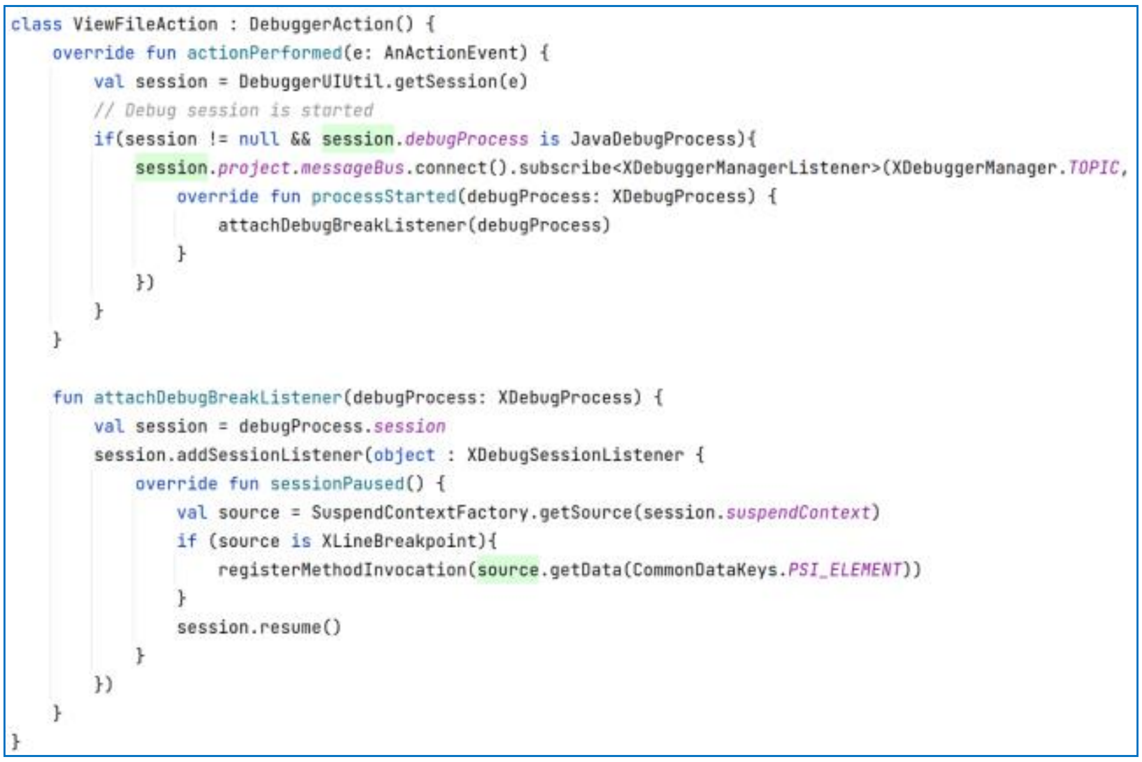
The probability of failure-free operation of the studied program will be the value of the element $T_{o,m-1}$ of the matrix T.

The next important element of the algorithm is the construction of connections between methods in the chain, determination of transition probabilities, as well as the probability of error-free operation of each method to build a graph. To solve this problem, it is necessary to count the number of executions of each method, as well as the number of failures that occurred in each of them [21]. The debug mode in Intellij Idea will do a great job with this, in which the studio can record method entries and all exception generation in the log. When adding a method to a HashMap, a "Breakpoint" is automatically set on it. All breakpoints set in the code can be viewed by going to the Intellij Idea settings [22].

Fig. 5 shows the code that is responsible for fixing the number of method calls. This code is called every time the session is stopped. It checks whether this stop was caused by a row-level Breakpoint, commits the call (if all conditions are met) and resumes the session. Although stopping the session also slows down the execution of the main program, it ensures that all calls will be caught, because attempts to catch calls in asynchronous mode can cause problems because the plugin code does not keep up with the main code. To increase the speed of the program, a variable of type unsigned long is used to save the number of method calls, which allows you to store values up to $2^{64} – 1$.

This information, together with data on the number of calls and a tree of methods, allows you to build a graphical model of this software system, as well as fill in the matrices necessary to calculate its reliability. It is important to understand that the method tree constructed above does not contain enough information to construct a graph [23]. This is due to the fact that this hierarchy of methods in no way takes into account the conditional statements and various checks that may be present in the program code. The very task of estimating transition probabilities is extremely non-trivial and cannot be solved algorithmically. Within the framework of the considered model, transition probabilities are calculated statistically as the ratio of the number of calls of the child method to the number of calls of the parent method. Table 1 shows an example illustrating the influence of conditional operators on the structure of the graph [24, 25].

To obtain a reliability indicator, you need to use the last item of the context menu (Assess App Reliability), as shown in Fig. 6. This method is active only when the studio was started in debug mode, the HashMap contains at least one method, and the call count table contains at least one method called by the debug input.

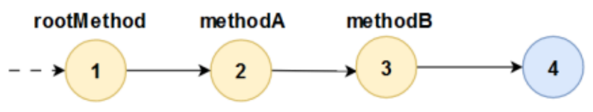
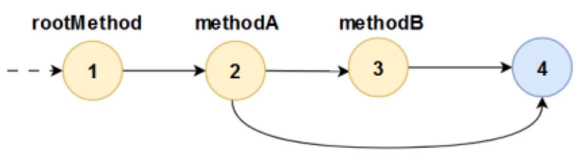


```
class ViewFileAction : DebuggerAction() {
    override fun actionPerformed(e: AnActionEvent) {
        val session = DebuggerUIUtil.getSession(e)
        // Debug session is started
        if(session != null && session.debugProcess is JavaDebugProcess){
            session.project.messageBus.connect().subscribe<XDebuggerManagerListener>(XDebuggerManager.TOPIC,
                override fun processStarted(debugProcess: XDebugProcess) {
                    attachDebugBreakListener(debugProcess)
                }
            })
        }
    }

    fun attachDebugBreakListener(debugProcess: XDebugProcess) {
        val session = debugProcess.session
        session.addSessionListener(object : XDebugSessionListener {
            override fun sessionPaused() {
                val source = SuspendContextFactory.getSource(session.suspendContext)
                if (source is XLineBreakpoint){
                    registerMethodInvocation(source.getData(CommonDataKeys.PSI_ELEMENT))
                }
                session.resume()
            }
        })
    }
}
```

Fig. 5. Program code for registration of method calls

Table 1 – **The influence of conditional operators on the structure of graphs**

void rootMethod(){
methodA();
methodB();
}
	void rootMethod(){
	methodA();
	if (something) methodB();
	}

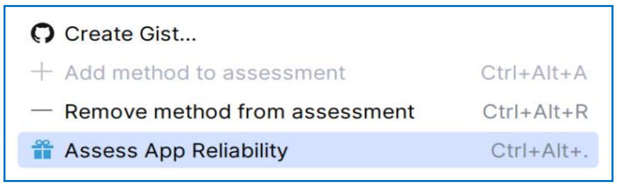

Create Gist...
Add method to assessment Ctrl+Alt+A
Remove method from assessment Ctrl+Alt+R
Assess App Reliability Ctrl+Alt+.

Fig. 6. The Assess App Reliability function in the context menu

After activating this item, the program analyzes the log entries, determines all the necessary values for building the matrices and calculates the probability of the program's error-free operation (P_f, Table 1). First, the probability of error-free operation is calculated as one minus the number of errors divided by the number of calls. Since the number of calls is counted only for methods on which Breakpoint is set, errors in all other methods will be missed in the counting process.

Table 2 – **Log analysis results**

Method	Number of calls	Number of failures	P_f
calculateSomething()	2147483647	23798952	0.9889
generateDivident()	2147483647	102261126	0.9524
genRandomValue()	613566756	18592932	0.9697
getConstantValue()	1431655765	15070061	0.9895
generateDivisor()	2011559528	0	1

The next step is to calculate the probability of calling the method from the parent's body. For this, for each method, the body of which method was called is checked and the number of calls "with context" is counted. Context is important because the same method can be used in different parts of the program. At this step, transition probabilities are calculated, that is, values for each of the edges of the graph.

Next, the program proceeds to actual calculations of the probability of failure-free operation. After all operations and calculations are completed, the final data will be automatically written to the database, and an informational message with the result will be displayed in the notification area, as shown in Fig. 7.

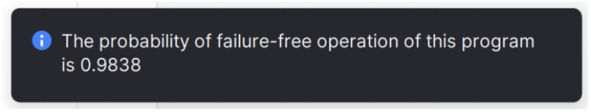

The probability of failure-free operation of this program is 0.9838

Fig.7. Information message with the result of the reliability assessment

For a visual representation of the implementation of the proposed approach for evaluating the reliability of a software system based on a graph model of method dependence, we will illustrate the reliability evaluation of a simple program, the program code of which is shown in Fig. 8. For the convenience of visual demonstration, all methods of the software system are located in one class, but the proposed approach will work even if all methods belong to different classes. To emulate the presence of failures, conditional operators are added to the body of methods that generate exceptions of the NumberFormatException type for certain values of the iterative variable. Binding to the iterative variable was done purely from the point of view of repeatability of results for ease of testing [26]. After the calculations are completed, a browser window containing the constructed graph of the program with probability indicators opens automatically.

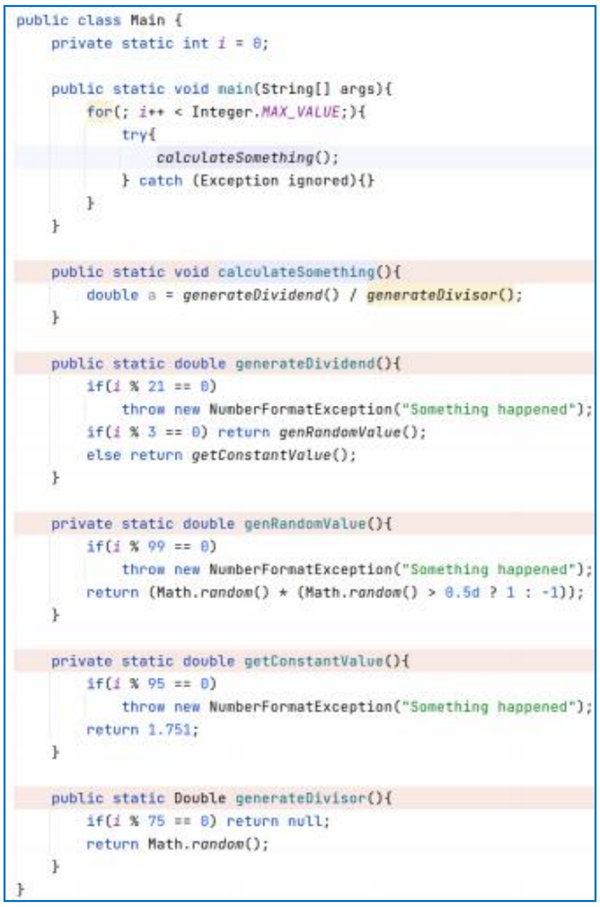

```java
public class Main {
    private static int i = 0;

    public static void main(String[] args){
        for(; i++ < Integer.MAX_VALUE;){
            try{
                calculateSomething();
            } catch (Exception ignored){}
        }
    }

    public static void calculateSomething(){
        double a = generateDividend() / generateDivisor();
    }

    public static double generateDividend(){
        if(i % 21 == 0)
            throw new NumberFormatException("Something happened");
        if(i % 3 == 0) return genRandomValue();
        else return getConstantValue();
    }

    private static double genRandomValue(){
        if(i % 99 == 0)
            throw new NumberFormatException("Something happened");
        return (Math.random() * (Math.random() > 0.5d ? 1 : -1));
    }

    private static double getConstantValue(){
        if(i % 95 == 0)
            throw new NumberFormatException("Something happened");
        return 1.751;
    }

    public static Double generateDivisor(){
        if(i % 75 == 0) return null;
        return Math.random();
    }
}
```

Fig. 8. Program source code containing 5 methods

A screenshot of the graph is shown in Fig. 9. It is worth noting that despite its simplicity of operation and free for commercial use, the dracula.js library has a not very optimized algorithm for calculating Bezier curves for constructing the edges of the graph, so quite often the graph does not look very obvious and requires manual manipulation with nodes In addition, this library has certain performance problems - it takes 0.5 seconds to build a graph containing 100 nodes, 5.5 seconds to build a graph with 500 nodes, but a graph for a program containing 1000 methods will be built in 40 seconds.

Since the graph contains seven vertices, the matrices P, G and T will also have dimension 7 × 7. Fig, 10 shows the values of all matrices [27].

As it was mentioned above, the probability of failure-free operation of the system is equal to $T_{o,m-1}$, since in our case $m = 7$, then $T_{0,m-1} = T_{0,6} = 0.93$.

We will apply the proposed method to determine the reliability of a more complex program. The class diagram of the test program is shown in Fig. 11.

After the completion of the program execution and completion of calculations, we receive information about the number of method calls and errors that occurred. The resulting data are shown in Table 3.

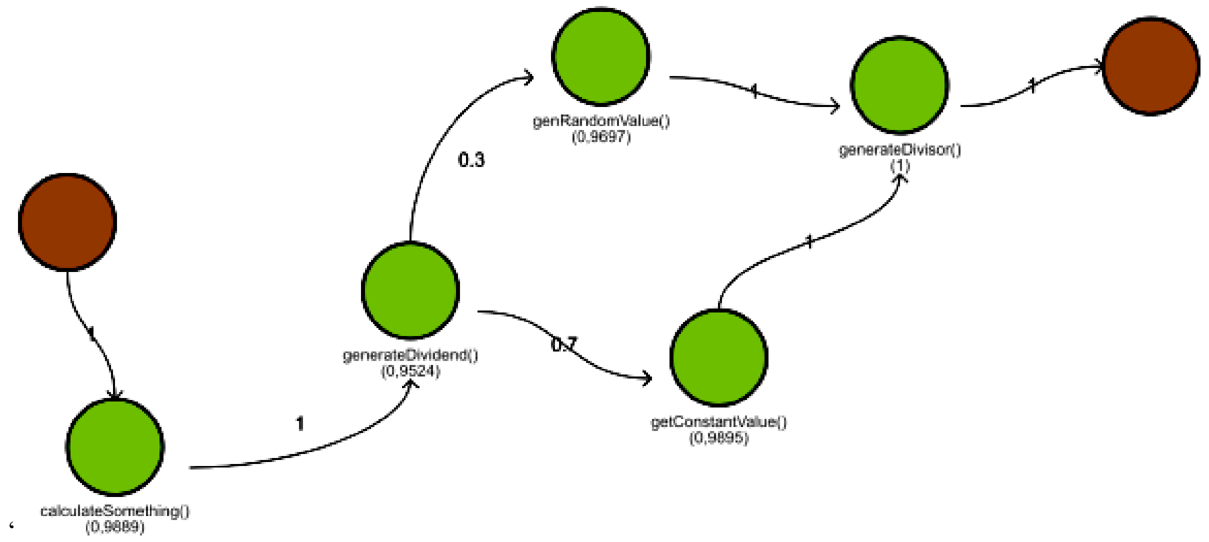


Fig. 9. Graph of the program under study

$$P=\begin{pmatrix}0&1&0&0&0&0&0\\0&0&1&0&0&0&0\\0&0&0&0.3&0.7&0&0\\0&0&0&0&0&1&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&1\\0&0&0&0&0&0&0\end{pmatrix};\ G=\begin{pmatrix}0&1&0&0&0&0&0\\0&0&0.99&0&0&0&0\\0&0&0&0.29&0.67&0&0\\0&0&0&0&0&0.97&0\\0&0&0&0&0&0.99&0\\0&0&0&0&0&0&1\\0&0&0&0&0&0&0\end{pmatrix};\ T=\begin{pmatrix}1&1&0.99&0.28&0.66&0.93&0.93\\0&1&0.99&0.28&0.66&0.93&0.93\\0&0&1&0.29&0.67&0.94&0.94\\0&0&0&1&0&0.97&0.97\\0&0&0&0&1&0.99&0.99\\0&0&0&0&0&1&1\\0&0&0&0&0&0&1\end{pmatrix}.$$

Fig. 10. The values of the matrices P, G and T

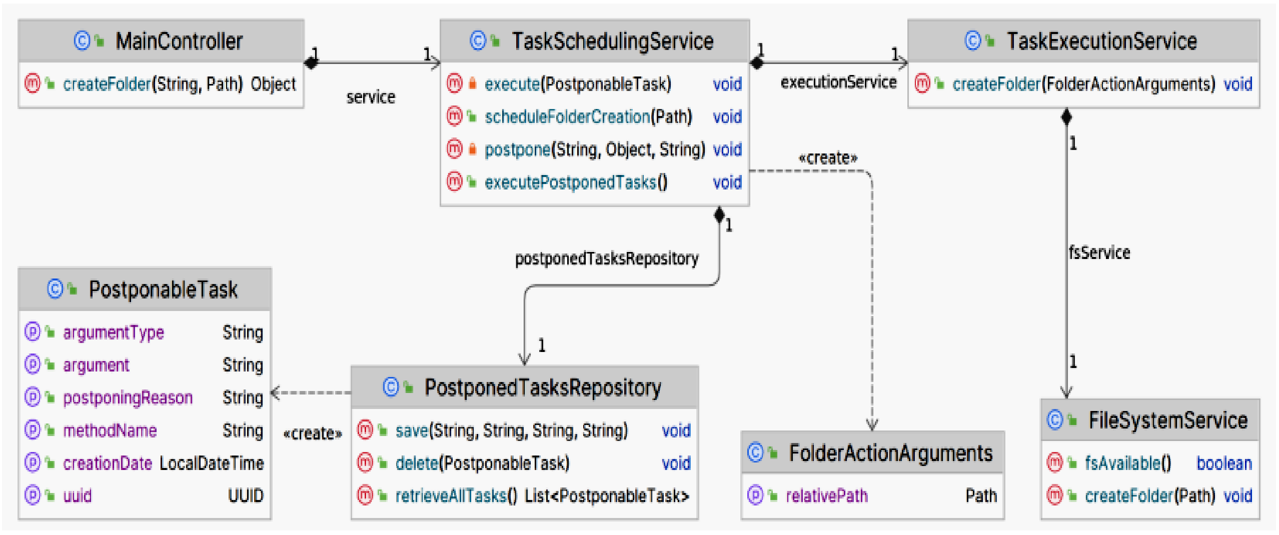


Fig. 11. Class diagram of the program under study

Table 3 – **Log analysis results**

Method	Number of calls	Number of failures	The probability of failure-free work
MainController.createFolder()	1000	0	1
scheduleFolderCreation()	1000	0	1
TaskExecutionService.createFolder()	3765	0	1
fsAvailable()	3765	35	0.991
postpone()	333	5	0.985
save()	328	2	0.993
FileSystemService.createFolder()	1024	75	0.93
delete()	326	1	0.997
executePostponedTasks()	114	0	1
retriveAllTasks()	114	2	0.982
execute()	2768	13	0.995

Fig. 12 shows the values of all matrices P, G and T for this program.

The probability of failure-free work of the system is equal to $T_{o,m-1}$, since in our case $m = 13$, then

$$T_{0,m-1} = T_{0,12} = 0.97 .$$

The graph of this program, containing 13 vertices and 17 edges, is shown in Fig. 13 [27].

$$P = \begin{pmatrix} 0 & 0.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.06 & 0 & 0.29 & 0 & 0 & 0 & 0 & 0.63 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0.06 & 0 & 0.3 & 0 & 0 & 0 & 0 & 0.64 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}; G = \begin{pmatrix} 0 & 0.9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.06 & 0 & 0.29 & 0 & 0 & 0 & 0 & 0.62 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.99 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.99 \\ 0 & 0 & 0 & 0 & 0 & 0.06 & 0 & 0.28 & 0 & 0 & 0 & 0 & 0.6 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.98 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix};$$

$$T = \begin{pmatrix} 1 & 0.9 & 0.9 & 1 & 1 & 0.1 & 0.09 & 0.29 & 0.08 & 0.1 & 0.1 & 0.1 & 0.97 \\ 0 & 1 & 1 & 1 & 1 & 0.1 & 0.09 & 0.29 & 0.08 & 0 & 0 & 0 & 0.97 \\ 0 & 0 & 1 & 1 & 1 & 0.1 & 0.09 & 0.29 & 0.08 & 0 & 0 & 0 & 0.97 \\ 0 & 0 & 0 & 1 & 1 & 0.1 & 0.09 & 0.29 & 0.08 & 0 & 0 & 0 & 0.97 \\ 0 & 0 & 0 & 0 & 1 & 0.1 & 0.09 & 0.29 & 0.08 & 0 & 0 & 0 & 0.97 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0.99 & 0 & 0 & 0 & 0 & 0 & 0.98 \\ .0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0.99 \\ 0 & 0 & 0 & 0 & 0 & 0.06 & 0.05 & 1 & 0.28 & 0 & 0 & 0 & 0.93 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0.98 & 0.98 & 0.09 & 0.09 & 0.28 & 0.08 & 1 & 1 & 0.98 & 0.95 \\ 0 & 0 & 0 & 0.98 & 0.98 & 0.09 & 0.09 & 0.28 & 0.08 & 0 & 1 & 0.98 & 0.95 \\ 0 & 0 & 0 & 1 & 1 & 0.09 & 0.09 & 0.29 & 0.08 & 0 & 0 & 1 & 0.96 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

Fig. 12. The values of the matrices P, G and T

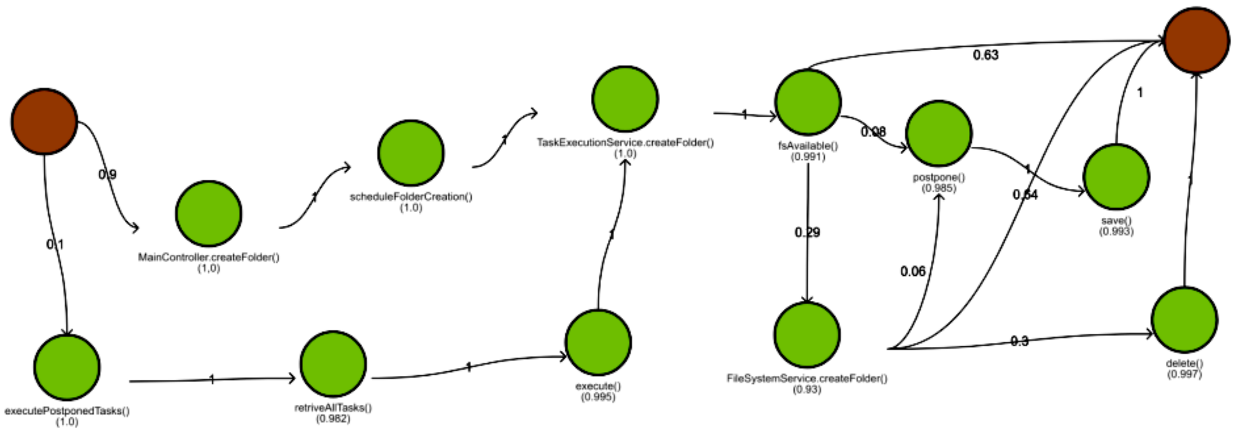


Fig. 13. Graph of the program under study

Conclusions

The paper provides a solution to the current problem of quality analysis of complex software systems. During the analysis of the available solutions, various models for assessing the reliability of software systems were investigated.

Unfortunately, a large number of them were primarily created to assess the reliability of hardware systems, so they are based on assumptions and simplifications that are not very common to software systems, so they are not always able to give an acceptable result. A method for assessing the reliability of software systems based on a graph model of the dependence of the methods of the system under test is proposed. An experimental study of the software system was conducted for the possibility of its application to determine the reliability of real programs written in the Java programming language. The obtained results show that the complexity of the program does not significantly affect the time of execution of calculations. Considering the obtained results, we can see that the proposed approach shows good performance indicators, which means that it can calculate reliability indicators for much larger software systems in an acceptable amount of time.

References

1. Yakovyna, V. and Symets, I. (2021), "Reliability assessment of CubeSat nanosatellites flight software by high-order Markov chains", *Procedia Computer Science*, vol. 192, pp. 447–456, doi: https://doi.org/10.1016/j.procs.2021.08.046
2. Skanda, V.C., Srinivasa Prasad, S, Dheemanth, G.R. and Kumar, N.S. (2019), "Assessment of quality of program based on static analysis", *IEEE 10th International Conference on Technology for Education(T4E)*, pp. 276–277, doi: https://doi.org/10.1109/T4E.2019.00072
3. Lu, S., Li, H. and Jiang, Z. (2020), "Comparative study of open source software reliability assessment tools", *IEEE International Conference on Artificial Intelligence and Information Systems (ICAIIS)*, China, pp. 49–55, doi: https://doi.org/10.1109/ICAIIS49377.2020.9194946
4. Yakovyna, V., Seniv, M., Symets, I. and Sambir, N. (2020), "Algorithms and software suite for reliability assessment of complex technical systems", *Radio Electronics, Computer Science, Control 4*, pp. 163–177, doi: https://doi.org/10.15588/1607-3274-2020-4-16
5. San, K.K., Washizaki, H., Fukazawa, Y., Honda, K., Taga, M. and Matsuzaki, A. (2021), "Deep cross-project software reliability growth model using project similarity-based clustering", *Mathematics*, vol. 9, no. 22, article number 2945, doi: https://doi.org/10.3390/math9222945
6. Krepych, S., Stakhiv, P. and Spivak, I., (2013), "Analysis of the tolerance area parameters REC based on technological area scattering", *12th International Conference "The Experience of Designing and Application of CAD Systems in Microelectronics"*, Polyana Svalyava, Ukraine, pp.179–180. available at: https://ieeexplore.ieee.org/document/6543231
7. Wu, C.-Y. and Huang, C.-Y. (*2021), "A study of incorporation of deep learning into software reliability modeling and assessment", IEEE Transactions on Reliability, vol. 70, no. 4, pp. 1621–1640, doi: https://doi.org/10.1109/TR.2021.3105531*
8. *Jagtap, M., Katragadda, P. and Satelkar, P. (2022), "Software reliability: development of software defect prediction models using advanced techniques", Annual Reliability and Maintainability Symposium (RAMS), pp. 1–7, doi: https://doi.org/10.1109/RAMS51457.2022.9893986*
9. *Nafreen, M., Luperon, M., Fiondella, L., Nagaraju, V., Shi, Y. and Wandji, T. (2020), "Connecting software reliability growth Models to software defect tracking", IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), pp.138–147, doi: https://doi.org/10.1109/ISSRE5003.2020.00022*
10. *Yakovyna, V. and Symets, I. (2021), "A method of high-order Markov chain representation through an equivalent first-order chain for software reliability assessment", Computer systems and information technologies, vol. 3, pp. 66–73, doi: https://doi.org/10.31891/CSIT-2021-5-9*
11. *Jain, R. and Sharma, A. (2019), "Assessing software reliability using genetic algorithms", The Journal of Engineering Research, [TJER], vol. 16(1), pp. 11–17, doi: https://doi.org/10.24200/tjer.vol16iss1*
12. *Micro, R., Chren, S. and Rossi, B. (2022), "Applicability of soft-ware reliability growth models to open source software", 48th Euromicro Conference on Software Engineering and Acvanced Applications*, (SEAA), pp. 255–262, doi: https://doi.org/10.1109/SEAA56994.2022.00047
13. Lu, S., Li, H. and Jiang, Z. (2020), "Comparative Study of Open Source Software Reliability Assessment Tools", *2020 IEEE International Conference on Artificial Intelligence and Information Systems*, (ICAIIS), Dalian, China, pp. 49–55, doi: https://doi.org/10.1109/ICAIIS49377.2020.9194946
14. Kim, T., Ryu, D. and Baik, J. (2024), "Enhancing software reliability growth modeling: a comprehensive analysis of historical datasets and optimal model selections", *IEEE 24th International Conference on Software Quality, Reliability and Security*, QRS, pp. 147–158, doi: https://doi.org/10.1109/QRS62785.2024.00024
15. Chen, Y., Yan, X. and Khan, A.A. (2019), "A Novel Reliability Assessment Method Based on the Effects of Components", *IEEE 19th International Conference on Software Quality, Reliability and Security (QRS)*, Sofia, Bulgaria, pp. 69–76, doi: https://doi.org/10.1109/QRS.2019.00022
16. Nafreen, M., Luperon, M., Fiondella, L., Nagaraju, V., Shi, Y. and Wandji, T. (2020), "Connecting Software Reliability Growth Models to Software Defect Tracking", *IEEE 31st International Symposium on Software Reliability Engineering*, ISSRE, Coimbra, Portugal, pp. 138–147, doi: https://doi.org/10.1109/ISSRE5003.2020.00022
17. Saini G.L., Panwar, D. and Singh, V. (2021), "Software reliability prediction of open source software using soft computing technique", *Recent Advances in Computer Science and Communications (Formerly: Recent Patents on Computer Science)*, vol. 14, no. 2, pp. 612–621, doi: https://doi.org/10.2174/2213275912666190307165332
18. Teuber, S. and Weigl, A. (2021), "Quantifying Software Reliability via Model-Counting",in: Abate, A., Marin, A. (eds) Quantitative Evaluation of Systems. QEST 2021. Lecture Notes in Computer Science, vol 12846. Springer, Cham. doi: https://doi.org/10.1007/978-3-030-85172-9_4.
19. Nafreen, M., Bhattacharya, S. and Fiondella, L. (2020), "Architecture-based Software Reliability Incorporating Fault Tolerant Machine Learning", *Annual Reliability and Maintainability Symposium (RAMS)*, Palm Springs, CA, USA, pp. 1–6, doi: https://doi.org/10.1109/RAMS48030.2020.9153718.
20. Kuang, P., Zhao, Q.-M. and Xie, Z.-H. (2015), "Algorithms for solving unconctrained optimization problems", *12th International Computer Conference on Wavelet Active Media Technology and Information Processing*, pp. 379–382, doi: https://doi.org/10.1109/ICCWAMTIP.2015.7494013
21. Bayurskii, A. and Krepych, S. (2018), "Intelligent Syatem Analyzing Quality of Land Plots", CEUR Workshop Proceedings 2300, pp.166-169, available at: https://ceur-ws.org/Vol-2300/Paper40.pdf
22. (2024), Breakpoints|intellij idea documentation, available at: https://www.jetbrains.com/help/idea/using-breakpoints.html
23. Kuchuk, N., Kashkevich, S., Radchenko, V., Andrusenko, Y. and Kuchuk, H. (2024), "Applying edge computing in the execution IoT operative transactions", *Advanced Information Systems*, vol. 8, no. 4, pp. 49–59, doi: https://doi.org/10.20998/2522-9052.2024.4.07

24. Mezhuev, P., Gerasimov, A., Privalov, P. and Butkevich, V. (2021), "A dynamic algorithm for source code static analysis", *Ivannikov Memorial Workshop (IVMEM)*, pp.57-60, doi: https://doi.org/10.1109/IVMEM53963.2021.00016
25. Zhang, Y., Sun, Y., Si, G., Dong, B. and Chen, W. (2022), "An overview of source code static analysis method based on knowledge graph", *IEEE 5th Advanced Information Management, Communicates, Electronic and Automation Control Conference (IMCEC)*, vol. 5, pp. 1772–1775, doi: https://doi.org/10.1109/IMCEC55388.2022.10019850
26. Spivak, I., Krepych, S., Litvynchuk, M. and Spivak, S. (2021), "Validation and data processing in json format", *IEEE EUROCON 2021 19th International Conference on Smart Technologies*, pp. 326–330, doi: https://doi.org/10.1109/EUROCON52738.2021.9535582
27. Krutko, V., Spivak, I. and Krepych, S. (2023), "An approach to assessing the reliability of software systems based on a graph model of method dependence", *6th Worksop for Young Scientists in Computer Science & Software Engineering*, pp. 37–47, available at: https://ceur-ws.org/Vol-3662/paper11.pdf

Received (Надійшла) 30.10.2024
Accepted for publication (Прийнята до друку) 26.02.2025

ВІДОМОСТІ ПРО АВТОРІВ/ ABOUT THE AUTHORS

Крепич Світлана Ярославівна – кандидат технічних наук, доцент, доцент кафедри комп'ютерних наук, Західноукраїнський національний університет, Тернопіль, Україна;
Svitlana Krepych – Candidate of Technical Sciences, Associate Professor, Associate Professor of Computer Science Department, Western Ukrainian National University, Ternopil, Ukraine;
e-mail: msya220189@gmail.com ORCID Author ID: https://orcid.org/0000-0001-7700-8367;
Scopus ID: https://www.scopus.com/authid/detail.uri?authorId=55225606100.

Співак Ірина Ярославівна - кандидат технічних наук, доцент, доцент кафедри комп'ютерних наук, Західноукраїнський національний університет, Тернопіль, Україна;
Iryna Spivak – Candidate of Technical Sciences, Associate Professor, Associate Professor of Computer Science Department, Western Ukrainian National University, Ternopil, Ukraine;
e-mail: spivak.iruna@gmail.com ORCID Author ID: https://orcid.org/0000-0003-4831-0780;
Scopus ID: https://www.scopus.com/authid/detail.uri?authorId=55226024100.

Співак Сергій Михайлович – доктор економічних наук, професор, завідувач кафедри бухгалтерського обліку та аудиту, Тернопільський національний технічний університет імені Івана Пулюя, Тернопіль, Україна;
Serhii Spivak – Doctor of Economic Sciences, Professor, Head of the Accounting and Audit Department, Ternopil Ivan Puluj National Technical University, Ternopil, Ukraine;
e-mail: spivak_s@tntu.edu.ua ORCID Author ID: https://orcid.org/0000-0002-7160-2151;
Scopus ID: https://www.scopus.com/authid/detail.uri?authorId=57210559132.

Крепич Роман Володимирович – викладач, Кам'янець-Подільський державний інститут, Кам'янець-Подільський, Україна;
Roman Krepych– lector, Kamianiets-Podilskyi State Institute, Kamianets-Podilskyi, Ukraine;
e-mail: jagmstar@gmail.com ORCID Author ID: https://orcid.org/0000-0003-4831-0780;
Scopus ID: https://www.scopus.com/authid/detail.uri?authorId=27368089600.

Метод оцінювання надійності програмних систем на основі графової моделі залежності методів системи, що тестується

С. Я. Крепич, І. Я. Співак, С. М. Співак, Р. В. Крепич

Анотація. На сьогодні програмне забезпечення перетворилося на невід'ємну складову багатьох сфер нашого повсякденного життя — від автоматизації і оптимізації процесів на виробництві до створення комфорту окремої людини. Програми роблять наші життя простішими, в секунди вирішуючи задачі, на які раніше йшли години чи навіть тижні, а також даруючи нам зручність та комфорт, про які люди минулих поколінь не могли навіть і мріяти. З метою задовольнити зростаючий попит на нове програмне забезпечення ІТ ринок в усьому світі і в У країні зокрема також стрімко росте і змінюється. За даними IT Ukraine Association, порівняно з 2017 роком, кількість працевлаштованих фахівців на ринку праці України зросла приблизно в два рази, а обсяги експорту ІТ послуг - в два з половиною. Не дивлячись на те, що через повномасштабне вторгнення темпи розвитку в 2022-2024 роках сповільнились, очевидним є, що галузь не досягла свого піку, а значить продовжить розвиватися. Крім очевидних змін, пов'язаних з розширенням ринку, є й внутрішні зміни в процесах індустрії зумовлені прагненням підвищити швидкість розробки програм, а також знизити кінцеву ціну програмного продукту .Загальновідомо, що висока якість програмного забезпечення є невід'ємною частиною успішного продукту. Проте, навіть при доволі невисокому темпі розробки, розробники доволі часто припускаються помилок, які призводять до серйозних проблем, впливаючи на безпеку, надійність та задоволеність користувачів. То що вже казати про розробку у стислі терміни? Саме тому, забезпечення високої як ості програмного продукту є одним з основних завдань, яке має вирішуватися на етапі розробки. **Об'єктом дослідження** виступають процеси оцінювання надійності програмних систем. **Предметом дослідження** є метод оцінювання надійності програмних систем на основі графової моделі залежності методів системи, що тестується. **Висновок**: на основі методу оцінювання надійності програмних систем на основі графової моделі залежності методів системи, що тестується, розроблено програмне забезпечення на мові програмування Java та Kotlin для оцінювання показника надійності програмних систем будь-якої архітектурної складності.

Ключові слова: якість програмного забезпечення; надійність програмного забезпечення; тестування, моделювання складних систем; графова модель системи.