УДК 004.052.3
DOI: 10.31673/2409-7292.2023.030606

Чмут О. В., Калініченко О. Г., Бодашевський Є. М.

ТЕХНОЛОГІЯ СТВОРЕННЯ ВІДМОВОСТІЙКОГО БАГАТОМОДУЛЬНОГО ПРОГРАМНОГО КОМПЛЕКСУ НА ОСНОВІ ПРОЦЕДУРИ ВЗАЄМНИХ ВНУТРІШНІХ ПЕРЕВІРОК

У статті розглянуто проблему створення відмовостійкого багатомодульного програмного комплексу на основі процедури взаємних внутрішніх перевірок. Ця проблема є особливо гострою у випадках використання комп'ютерних систем критичного призначення, де вимоги до безпеки програмного забезпечення є надзвичайно високими. Головна ідея статті полягає в тому, що багатомодульний програмний комплекс, як система, що базується на обчислювальних ресурсах, як на етапі створення так і на етапах експлуатації може самостійно здійснювати діагностування між окремими модулями, забезпечуючи тим самим, його працездатність. Метою статті є розробка процедури взаємних внутрішніх перевірок для її імплементації в технологію створення відмовостійкого багатомодульного програмного комплексу. Модель діагностування базується на припущеннях, що зв'язки між модулями можна подати повнозв'язним неорієнтованим графом і кожна пара модулів може обмінюватись повідомленнями. Кожен модуль за допомогою засобів самодіагностування визначає свій стан «справний» чи «несправний») і передає його іншому модулю. Після накопичення результатів перевірок приймається колективне рішення щодо справності модулів. Розроблено оригінальний алгоритм реалізації технології діагностування. Правильність розробленої концепції забезпечення стійкості до відмови програмних комплексів діагностуванням перевірено математичним моделюванням. Одержано графіки залежностей достовірності діагностування від кількості модулів в програмі та кількості допустимих відмов. Оцінено довірчі інтервали достовірності, які показують, що зі збільшенням кількості відмов довірчий інтервал розширюється, а при зменшенні – звужується. Зроблено висновок про те, що розроблений алгоритм вимагає меншої надмірності системи і дозволяє отримати результат всього два раунди обміну повідомленнями між модулями програми. При цьому він забезпечує діагностування багатомодульного програмного комплексу при відмові майже половини його модулів.

Ключові слова: програмний комплекс, програмний модуль, відмовостійкість, самодіагностування, достовірність.

Вступ

Створення багатомодульних програмних продуктів завжди пов'язано з низкою проблем, які обумовлюються численними помилками коду та нераціональним розподілом ресурсів системи між модулями. Швидкий розвиток цифрових технологій, збільшення кількості підключених до Інтернету пристроїв, зміни в нормативно-правовому середовищі, фінансові кризи, відмови в роботі важливого обладнання, кібератаки та зростання кіберзлочинності – лише деякі з причин, які стимулюють суспільство до постійного удосконалення своїх інформаційних систем та засобів захисту програмного забезпечення [1].

Постановка проблеми

Проблематика стає особливо гострою у випадках використання комп'ютерних систем критичного призначення, де вимоги до безпеки програмного забезпечення є надзвичайно високими. Згідно з керівними документами [2–4], існують різні методи забезпечення безпеки програмного забезпечення, такі як обґрунтоване розроблення та ефективне виконання політики безпеки, оперативна реакція на події, що стосуються безпеки ПЗ, безпечне програмування і інші аспекти. При створенні безпечного програмного забезпечення важливо обрати належну методологію для його реалізації [5], що сприятиме підвищенню ефективності та якості продукту. Варто враховувати, що процес розробки програмного забезпечення складається з кількох етапів у рамках обраної методології [6,7], які можна розділити на три основні групи: послідовні, циклічні та гнучкі. Проте, досить мало уваги приділяється автоматизованій методології створення надійного програмного забезпечення, що дозволило б автоматично контролювати його працездатність на всіх етапах життєвого циклу.

Дослідження публікацій за темою

Публікація [8] є першою в серії, що пропонує розуміння центральних тем стабільності програмного забезпечення. Автори досліджують унікальні характеристики програмного

забезпечення, які відрізняють його від інших галузей інженерії. Також вивчаються артефакти аналізу, дизайну, розробки та інші фактори, які мають тенденцію створювати стабільні чи нестабільні програмні продукти. У статті [9] автори представляють метод аналізу ризиків у розробці програмного забезпечення. У контексті розробки програмного забезпечення це будь-яка потенційна ситуація чи подія, яка може негативно вплинути на проект програмного забезпечення в майбутньому та може виникнути в результаті певних поточних дій. Ризик обговорюється в контексті вимог проекту та релізів. Вимоги (або модель вимог) представляють запити замовника, які повинні бути перетворені в функції та реалізовані в програмному продукті. Релізи стосуються фази життєвого циклу розробки програмного забезпечення.

У статті [10] описано, як застосувати теорію управління в інженерії програмного забезпечення. Оскільки програмний процес можна визначити кількісно, ви можете використовувати загальні принципи теорії управління та методи в програмному забезпеченні для способів автоматичного налаштування, координації та оптимального керування, за допомогою яких програмний процес знаходиться в діапазоні стабільної рівноваги. В публікації [11] автори зосереджуються на методах розробки програмного забезпечення для реалізації архітектурної стабільності, оскільки архітектура є одним із артефактів програмного забезпечення, який має глибокий вплив протягом життєвого циклу програмного забезпечення.

Робота [12] розглядає способи вимірювання якості архітектури програмного забезпечення. Вимірювання цих якостей має важливе значення для цієї підгалузі розробки програмного забезпечення. Ця робота досліджує стабільність і зрозумілість архітектури програмного забезпечення, кілька показників, які на них впливають, і огляд літератури щодо цих якостей. В публікації [13] автори роблять акцент на стабільності та зрозумілості архітектури програми. Стабільна архітектура матиме менше залежностей між компонентами, вимагатиме менше змін для впровадження нових функцій або виправлення помилок і вноситиме зміни швидше. Зрозумілість означає ступінь, до якого архітектура програмного забезпечення може бути легко зрозуміла розробникам та іншим зацікавленим сторонам. Автори вводять показники, які використовуються для вимірювання зрозумілості та стабільності архітектури програмного забезпечення, включаючи кількість компонентів та їхні зв'язки, рівень використаної абстракції та ступінь узгодженості між різними частинами архітектури.

Разом з тим, всі розглянуті публікації є надто далекими для формування архітектури автоматизованого комплексу перевірок програмного забезпечення.

Метою цієї роботи є розробка процедури взаємних внутрішніх перевірок для її імплементації в технологію створення відмовостійкого багатомодульного програмного комплексу.

Обґрунтування діагностичної моделі процедури взаємних внутрішніх перевірок

Розглянемо програмний комплекс [14], що складається з $N \geq 2t+2$ модулів, у якому виникає не більше t відмов. Приймемо такі припущення:

1. Зв'язки між модулями можна подати повнозв'язним неорієнтованим графом, тобто кожна пара модулів може обмінюватись повідомленнями.

2. Програмний комплекс є синхронним, тобто в ньому існує механізм, що дозволяє всім модулям програми одночасно перейти до діагностування.

3. Модулі обмінюються повідомленнями з множини $Z = \{a, \bar{a}, \varnothing\}$, де a – повідомлення, семантика якого залежить від конкретного випадку; $\bar{a}$ – протилежне a; $\varnothing$ – порожнє повідомлення, тобто, протягом відведеного інтервалу часу каналом зв'язку не надійшло жодного повідомлення.

4. Кожен модуль за допомогою засобів самодіагностування визначає стан $S(i)$. Стан може приймати два значення «справний» та «несправний».

© Чмут, О. В., Калініченко, О. Г., & Бодашевський, Є. М. (2023). Технологія створення відмовостійкого багатомодульного програмного комплексу на основі процедури взаємних внутрішніх перевірок. Сучасний захист інформації, 4(56), 52–61. https://doi.org/10.31673/2409-7292.2023.030606.

5. При передачі повідомлень з різною семантикою процесор витрачає на кожне повідомлення час *DELAY*, незалежно від того, у скільки модулів це повідомлення надсилається.

6. Модуль *i* витрачає на отримання повідомлення від модуля *j* час *DELAY_MES*(*i*,*j*).

7. Виконується така нерівність: *DELAY*(*i*) > (*N*-1) *DELAY_MES*(*i*,*j*).

Нехай модуль *k* (*k* = 1 ... *N*) є «старшим». Кожному з інших модулів («підлеглим») модуль *k* повинен передати повідомлення відповідно до припущенням 3. Після цього «підлеглі» обмінюються повідомленнями про отриману від «старшого» інформацію, при цьому кожен формує вихідний набір, за яким визначаються «нелояльні підлеглі» і потім решта досягає угоди з приводу «наказу» «старшого» – «прийняти» або «відкинути». Нелояльність модуля полягає в довільному спотворенні інформації, що передається. При цьому в різних повідомленнях нелояльний підлеглий передає суперечливі повідомлення іншим.

Припустимо a=0, $\bar{a}$ =1 і, кожному модулю у програмі присвоєно індивідуальний номер n=1...N, який відомий всім модулям програми. Вихідний набір, що формується в n-му модулі, має вигляд наступної матриці

$$A_n = \begin{bmatrix} a_{11}^n & a_{12}^n & a_{13}^n & \cdots & a_{1L}^n \\ a_{21}^n & a_{22}^n & a_{23}^n & \cdots & a_{2L}^n \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{n1}^n & a_{n2}^n & a_{n3}^n & \cdots & a_{nL}^n \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{L1}^n & a_{L2}^n & a_{L3}^n & \cdots & a_{LL}^n \end{bmatrix}, \quad (1)$$

де L=2t+1; у i-му рядку містяться значення повідомлень всіх j-х модулів, прийняті в першому раунді i-й і передані нею в другому раунді в n-му модулю; в j-му стовпці містяться значення повідомлень одного і того ж j-го модуля, передані ним всім i-м модулям; на головній діагоналі розташовані елементи a_{ij}^n, що є значеннями повідомлень j-х модулів, передані ними самим собі у першому раунді.

В алгоритмі використовується функція *majority*, яку можна визначити за наступними міркуваннями. Нехай заданий вектор e={e_i} (i=1,2,...,n) і розряди групуються за ознакою рівності значень групи $M_1, M_2, \ldots, M_m$, які не перетинаються. Очевидно $m \le n$. Нехай $|M_1|>|M_2|>\ldots>|M_m|$, де через $|M|$ позначено потужність множини M, тобто кількість елементів множини. Якщо розряди {e_i}, що входять до M_1, мають значення v, то функція *majority* визначається рівністю: $maj\{e_i\} = v$.

Алгоритм діагностування багатомодульного програмного комплексу на основі взаємного інформаційного узгодження.

Розглянемо програмний комплекс, що складається з N модулів із номерами 1...N. Діагностування здійснюється шляхом обміну повідомленнями при припущеннях про синхронність роботи програмного комплексу та про можливість одержувачем повідомлення визначити його відправника. Алгоритм А1 складається з двох етапів: етапу пересилок повідомлень (кроки 1–3) та етапу аналізу отриманих повідомлень (кроки 4–7). Етап аналізу виконується кожним модулем автономно з урахуванням повідомлень, отриманих ним на етапі пересилок.

Алгоритм А1 реалізується наступним чином.

Крок 1. k-й модуль надсилає іншим n-им модулям ($n \ne k$) повідомлення Z_k з множини Z={ $a, \bar{a}, \varnothing$ }. Для зручності вважатимемо номер k рівним 2t+2.

Крок 2. *n*-і модулі обмінюються повідомленнями, отриманими від *k*-го модуля на кроці 1. При цьому модулі, у яких має місце відмова, можуть передавати суперечливі повідомлення іншим модулям.

Кожен *n*-й модуль (n=1...N–1; $n \neq k$) формує з отриманих повідомлень вектор $STR(n)$, що містить $2t$+1 елемент, для розміщення повідомлень, отриманих від усіх модулів, включаючи себе на даному кроці $STR(n)=(Z_1,Z_2,...,Z_{N-1})$.

Крок 3. *n*-і модулі обмінюються векторами $STR(n)$ (n=1...N–1), у тому числі кожен формує вихідний набір як матриці A_n, у якій вектори $STR(n)$ (n=1... N–1) розташовуються як рядки у порядку зростання номерів модулів-відправників, які можна завжди однозначно визначити згідно з припущенням 1.

Крок 4. *n*-й модуль, застосовуючи функцію *majority* до стовпців матриці A_n, отримує по одному значенню цієї функції для кожного стовпця, з яких формує вектор PRS_n, елементи якого дорівнюють $PRS_n(j)=majority\{a_{ij}^n \mid i=1...N\}$.

Крок 5. *n*-й модуль визначає елементи a_{ij}^{n*}, i,j= 1...N–1 власної матриці A_n, які знаходяться на перетині стовпця j з рядком i, якщо $PRS_n(j) \neq a_{ij}^{n*}$.

Загальна кількість позначених елементів через L_{max}. Якщо L_{max} = 0, перейти до кроку 7.

Крок 6. *n*-й модуль визначає підозрювану область у вигляді логічного виразу

$$\sum \Pi = \bigwedge_{l=1}^{L_{\max}} (i_l \vee j_l), \qquad (2)$$

де i,j – номери модулів з елементів матриці a_{ij}^{n*}, що відповідають номеру рядка i та стовпця j елемента матриці A_n, що відрізняється від $PRS_n(j)$.

Далі вираз (2) приводиться до нормальної диз’юнктивної форми, для чого розкриваються дужки і виконуються перетворення, тобто вираз із кон'юнкції приводиться до виду диз'юнкції кон'юнкцій. При цьому для врахування обмеження не більше ніж у t несправних модулів виключаються з розгляду терми з більш ніж t елементами. Кожен з терм, що залишився, визначає допустиме поєднання збоїв за умови, що їх в програмі є не більше ніж t. Якщо у виразі залишається більше одного терму, то результат діагностування неоднозначний і $DIAG$=1, інакше $DIAG$=0.

Крок 7. *n*-і модулі формують матриці B_n отримані з A_n шляхом викреслення рядків і стовпців, відповідних модулів, номери яких присутні в термах, отриманих після перетворень виразу $\sum \Pi$ на кроці 6. За отриманими B_n *n*-і модулі визначають стан (справність) *k*-го модуля за такими правилами:

1) якщо матриця B_n містить ідентичні елементи, то *k*-й модуль справний ($DIAG$=0);

2) якщо матриця B_n містить неідентичні елементи, то *k*-й модуль несправний ($DIAG$=0);

3) якщо частина матриць B_n містить неідентичні елементи, а частина – ідентичні, то справність *k*-го модуля не визначена і $DIAG$=1.

Крок 8. Кінець алгоритму.

Якщо після алгоритму $DIAG$=1, то діагностування не виконано успішно, що є сигналом у верхній рівень ієрархії про неспроможність Алгоритму А1. Якщо $DIAG$=0, діагностування виконано успішно.

Алгоритм проілюструємо на прикладі. Розглянемо програмний комплекс, що складається з N=8 модулів при числі тих, що відмовили рівним t=3 (N=2t+2). Несправними модулями є «старший» та «підлеглі» з номерами 4 та 7.

Приклад застосування алгоритму. Досліджувана програма представлена у вигляді графа (рис. 1), на якому «нелояльні» модулі позначені штрихуванням. Для простоти на графі

показані лише зв'язки з «нелояльними» модулями і значення, які передають нелояльні модулі. Лояльні модулі передають без змін ті значення, які вони набули.

Таким чином, існує 4 рішення. Виходячи з припущення, що в програмі існує не більше t=3 відмов, терми, які дають більшу кількість модулів, що відмовили, виключимо з $\sum\Pi$ (крок 6 алгоритму). Єдиним правомірним рішенням для A_1 буде матриця B_1, яка отримана з A_1 після викреслення рядків та стовпців з номерами 4 та 7 (модулі з номерами 4 та 7 вважаються несправними).

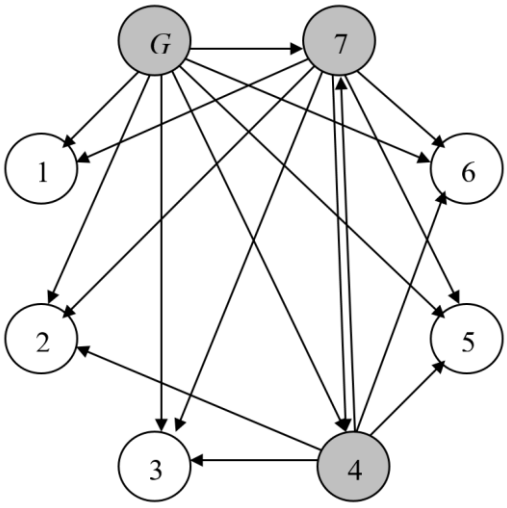

Рис. 1. Граф досліджуваної системи

$$A_1 = \begin{vmatrix} I & 0 & 0 & 0 & I & I & I \\ I & 0 & 0 & 0 & I & I & 0^* \\ I & 0 & 0 & I^* & I & I & 0^* \\ I & 0 & 0 & 0 & I & I & I \\ I & 0 & 0 & I^* & I & I & I \\ I & 0 & 0 & I^* & I & I & 0^* \\ I & 0 & 0 & 0 & I & I & I \end{vmatrix}. \qquad B_1 = \begin{vmatrix} I & 0 & 0 & I & I \\ I & 0 & 0 & I & I \\ I & 0 & 0 & I & I \\ I & 0 & 0 & I & I \\ I & 0 & 0 & I & I \end{vmatrix}.$$

$$PRS_1 = |\,I \;\; 0 \;\; 0 \;\; \boxed{0} \;\; I \;\; I \;\; \boxed{I}\,|.$$

Для визначення справності «старшого» (модуль з номером 8, що передав вихідне повідомлення) відповідно до кроку 7 алгоритму проаналізуємо матрицю B_1. З аналізу видно, що модулі, що залишилися, після викреслення несправних, а саме 1, 2, 3, 5, 6 отримали від «старшого» різні повідомлення (непомічені розряди вектора). Таким чином, модуль з номером 1 визначає, що «старший» поводився нелояльно. Алгоритм А1 завершується із *DIAG*=0, тобто. діагностування виконано успішно та визначено модуль із відмовами – 4, 7, 8.

Оцінка достовірності процедури взаємних внутрішніх перевірок

Постановка експерименту. Достовірність результатів теоретичних досліджень будемо досліджувати математичним моделюванням процедури діагностування програмного комплексу, що складаються з N = 7, 8, 9 модулів. Мета експериментальних досліджень полягає в наступному: 1) аналіз правильності алгоритму А1; 2) визначення ефективності алгоритму А1; 3) оцінка показників діагностування: достовірності та часу діагностування.

© Чмут, О. В., Калініченко, О. Г., & Бодашевський, Є. М. (2023). Технологія створення відмовостійкого багатомодульного програмного комплексу на основі процедури взаємних внутрішніх перевірок. Сучасний захист інформації, 4(56), 52–61. https://doi.org/10.31673/2409-7292.2023.030606.

У процесі моделювання за допомогою генератора випадкових чисел ставився стан програмного комплексу для наступних параметрів програми: t=1...N–2/2, де t – кількість модулів, що відмовили, N – кількість всіх модулів програми. Після цього імітувалася робота алгоритму А1. Внаслідок виконання алгоритму визначається технічний стан програми. Якщо в результаті виконання алгоритму А1 всі справні модулі узгоджено і однозначно визначають несправні, то діагностування програмного комплексу з відмовами завершується успішно і *DIAG*=0, інакше *DIAG*=1. При *DIAG*=0 отриманий стан програмного комплексу порівнюється із заданим. За результатами порівняння заданого та отриманого розподілу відмов у програмі визначалася правильність результатів діагностування.

За результатами моделювання одержано залежності достовірності діагностування D (рис. 2.а) та часу діагностування $t_д$ (рис. 2.б) від кількості відмов t для різних N. Загалом математичне моделювання підтвердило правильність теоретично розробленої концепції забезпечення стійкості до відмови програмних комплексів діагностуванням на основі взаємного інформаційного узгодження. Зокрема, з графіків залежностей $D=f(t)$ (рис. 2.а) видно, що кількість відмов t не перевищує допустиме (з прийнятої моделі програми $t \le \left\lfloor \frac{N-2}{2} \right\rfloor$), а достовірність діагностування досить висока D>0.95.

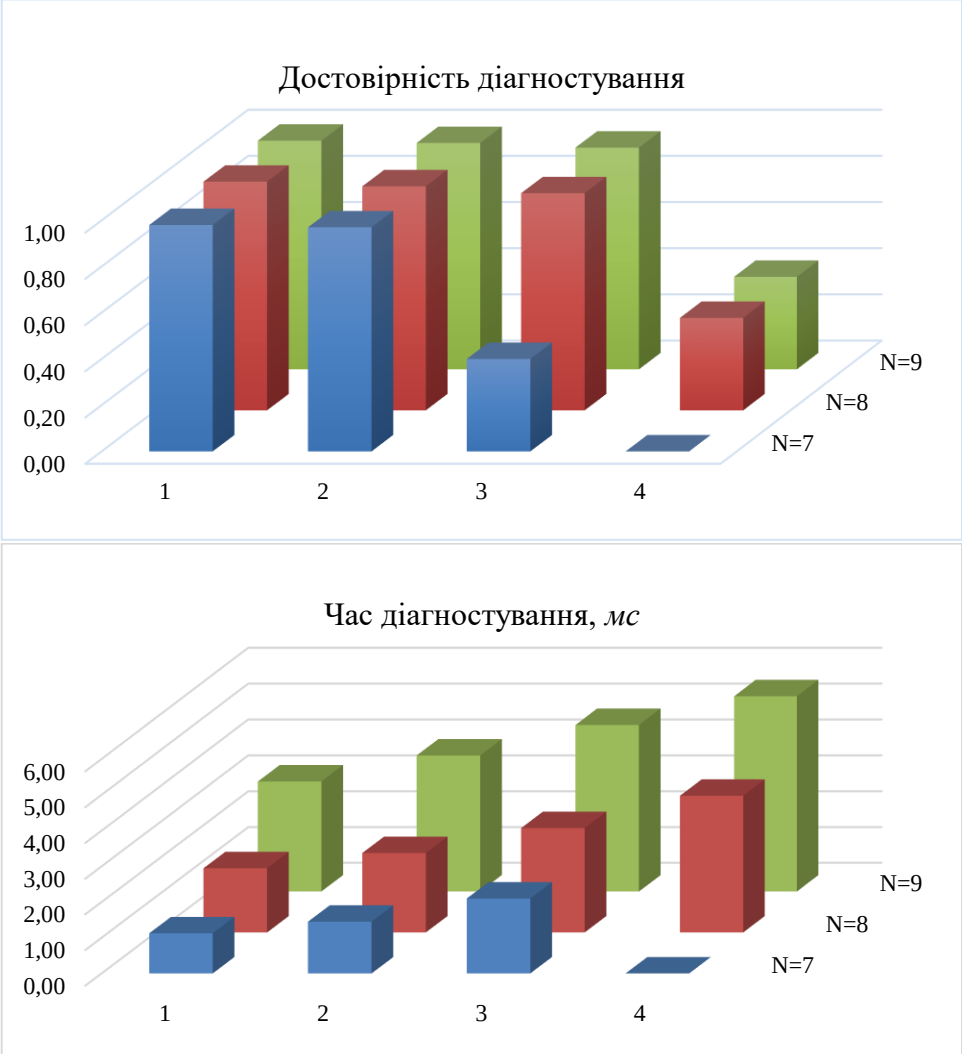

Рис. 2. Результати розрахунку достовірності діагностування D, часу діагностування $t_д$ в залежності від кількості модулів, що відмовили, і кількості модулів N в програмі

З графіків $t_д=f(t)$ (рис. 2.б) за різних N видно, що час діагностування лінійно змінюється від кількості відмов. Для реалізації даного алгоритму необхідно, щоб кожен модуль мав $(N-1)\times(N-1)$ чарунок пам'яті для зберігання булевих змінних вихідного набору, та $N\times\left\lfloor\frac{N-2}{2}\right\rfloor$ чарунок для зберігання проміжних значень.

Перевірка результатів дослідження методом статистичного моделювання

Достовірність діагностування програмного комплексу визначає міру довіри до отриманого результату діагностування його стану порівняно з дійсним станом об'єкта. Достовірність у роботі визначалася як відношення кількості дослідів, у яких правильно визначено стан програми до загальної кількості випробувань:

$$D^* = \frac{K_{пр}}{K_{заг}} \tag{3}$$

Так як достовірність є випадковою величиною, що змінюється за певним законом розподілу, то така точкова оцінка може виявитися дуже далекою від дійсного значення визначається параметра. І тут доцільно визначати інтервальну довірчу оцінку достовірності. При невідомій дисперсії σ^2 випадкової величини доводиться використовувати оцінку дисперсії s^2. При цьому величина меж довірчого інтервалу

$$t_{qk} = \frac{\sqrt{n}\cdot(\overline{D}-\hat{D})}{s} \tag{4}$$

підпорядковуватиметься закону розподілу Стьюдента з $k=n-1$ ступенями свободи. Тут n – обсяг вибірки (кількість дослідів); $\overline{D}$ – середнє арифметичне значення достовірності D, що визначається:

$$\overline{D} = \frac{1}{n}\sum_{i=1}^{n} D_i^*; \tag{5}$$

де $\hat{D}$ – оцінка достовірності; s^2 – оцінка дисперсії, що визначається:

$$s^2 = \frac{1}{n-1}\cdot\sum_{i=1}^{n}(\overline{D}-D_i^*)^2 \tag{6}$$

Позначимо через t_{qk} межі довірчого інтервалу для розподілу Стьюдента з k ступенями свободи за довірчої ймовірності $(1-q)$. Оцінка достовірності $\hat{D}$ з довірчою ймовірністю $(1-q)$ буде в інтервалі:

$$\overline{D} - t_{qk}\frac{s}{\sqrt{n}} < \hat{D} < \overline{D} + t_{qk}\frac{s}{\sqrt{n}}, \tag{7}$$

де t_{qk} – визначається з таблиць; $\overline{D}$ – визначається з виразу (5); s – визначається з виразу (6); n – обсяг випадкової вибірки (кількість прогонів моделі).

Наступним етапом знаходження довірчих оцінок є визначення оптимального обсягу вибірки n. Очевидно, що чим більше n, тим менший довірчий інтервал і знайдена оцінка буде

© Чмут, О. В., Калініченко, О. Г., & Бодашевський, Є. М. (2023). Технологія створення відмовостійкого багатомодульного програмного комплексу на основі процедури взаємних внутрішніх перевірок. Сучасний захист інформації, 4(56), 52–61. https://doi.org/10.31673/2409-7292.2023.030606.

ближчою до справжньої величини. Раніше було доведено, що з n>30 довірчий інтервал мало змінюється. Проте, зі зростанням n пропорційно збільшується обсяг обчислень шуканої випадкової величини. В цьому випадку при виборі n виходитимемо з умови забезпечення мінімуму похибки вимірювань. Задамося похибкою визначення достовірності діагностування 1%. У цьому випадку слід 100 разів виконати алгоритм діагностування та підрахувати кількість дослідів, у яких об'єкт правильно продіагностовано. За формулою (3) необхідно визначити статистичну достовірність D*. Описану процедуру слід виконати n разів та отримати n реалізацій D*.

Дослідження показали, що найдоцільніше прийняти n=6...9, оскільки при цьому величина знайденого довірчого інтервалу змінюється від 3 до 7%. При збільшенні n до 30 і більше інтервал не звужується менш ніж на 2%, а при зменшенні n до 3...4 – довірчий інтервал може досягати 10...20% і більше.

На рис. 3 представлена вибірка обсягом n=35 статистично певної достовірності діагностування програмного комплексу, що складається з 8 модулів при двох відмовах. За цією вибіркою визначені довірчі інтервали достовірності з довірчою ймовірністю 0,95 для перших n=6 реалізацій випадкової D* і всієї вибірки з n=35 реалізацій. У першому випадку величина довірчого інтервалу становить близько 4%, у другому – 1,5...2%. Виходячи з цього, можна дійти невтішного висновку, що з погляду мінімуму обчислень і максимуму точності отриманої довірчої оцінки, слід прийняти n=6.

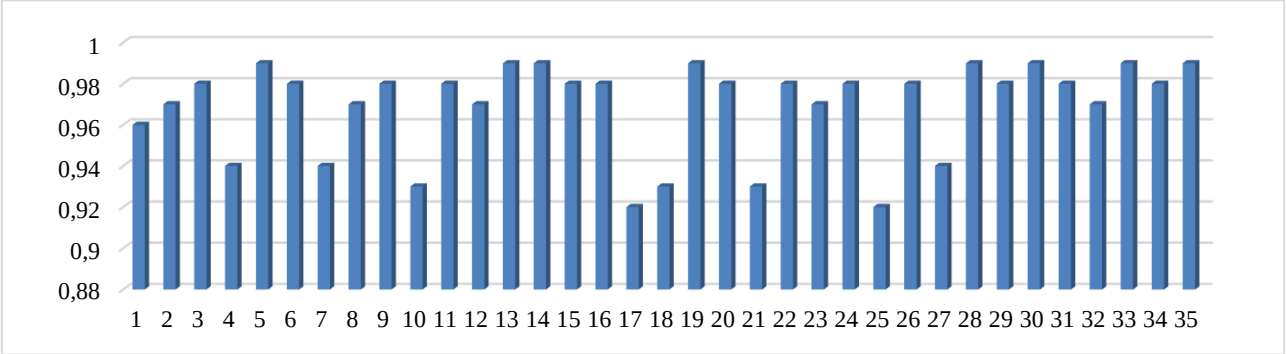

Рис. 3. Розрахунок інтервальних оцінок діагностування з довірчою ймовірністю P = 0,95. Середнє значення D = 0,9695. Нижня границя $D_н$ = 0,961. Верхня межа $D_в$ = 0,977

На рис. 4–6 представлено по шість реалізацій D* кожної відмовної ситуації (t=1...4) для програм з N = 7, 8, 9 модулів відповідно. Для наведених реалізацій визначено: середнє значення $\overline{D}$, нижня $D_н$ і верхня $D_в$ межі довірчого 95% інтервалу (табл. 1).

Таблиця 1

Середнє значення $\overline{D}$, нижня $D_н$ і верхня $D_в$ межі довірчого 95% інтервалу

N	D	Кількість модулів, які відмовили (t)			
		1	2	3	4
7	$\overline{D}$	0,983	0,967	0,367	–
	$D_н$	0,967	0,949	0,345	–
	$D_в$	0,997	0,984	0,388	–
8	$\overline{D}$	0,988	0,970	0,942	0,407
	$D_н$	0,975	0,952	0,927	0,363
	$D_в$	1,00	0,988	0,956	0,450
9	$\overline{D}$	0,993	0,980	0,957	0,412
	$D_н$	0,985	0,966	0,942	0,377
	$D_в$	1,00	0,994	0,972	0,446

Аналіз знайдених довірчих інтервалів показує, що зі збільшенням кількості відмов *t* довірчий інтервал розширюється, а при зменшенні *t* – звужується. Це пояснюється великим розкидом результатів діагностування при великих значеннях *t*.

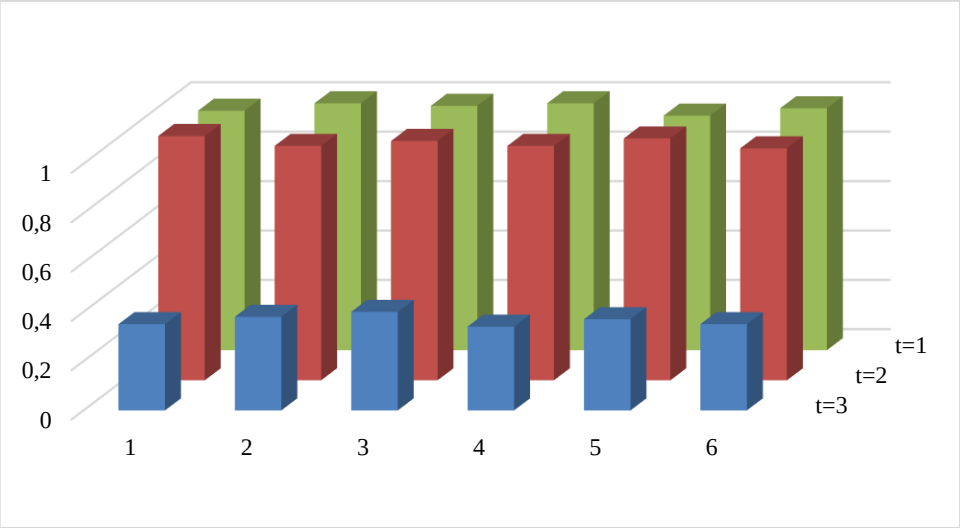

Рис. 4. Реалізації *D** та 95% довірчі інтервали для *N*=7 модулів

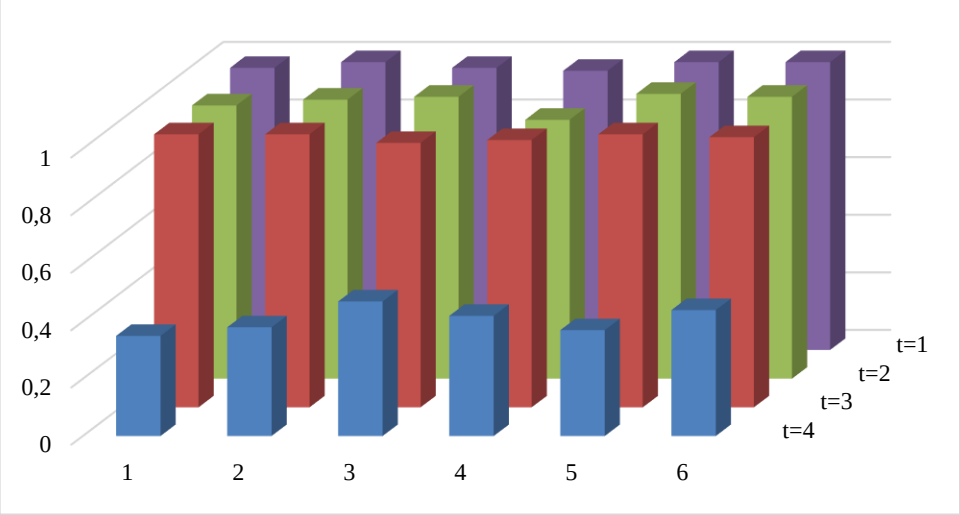

Рис. 5. Реалізації *D** та 95% довірчі інтервали для *N*=8 модулів

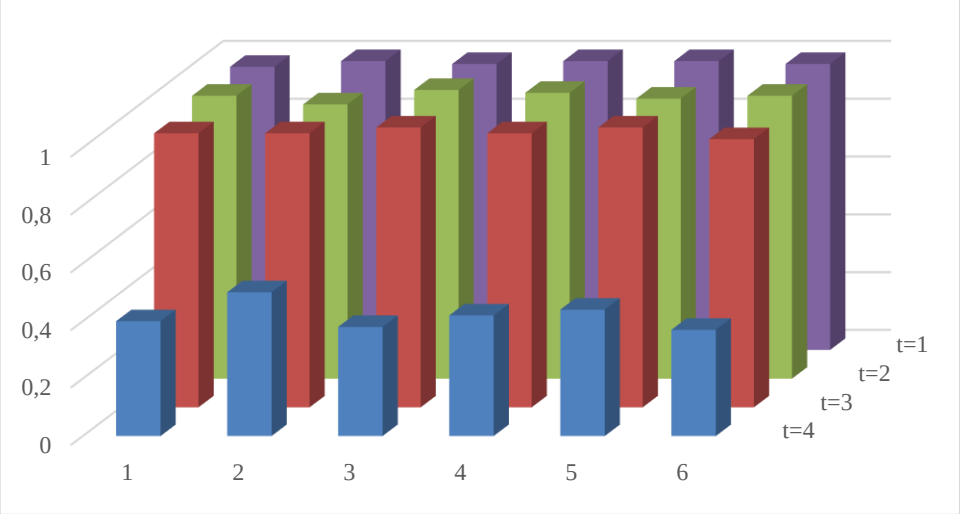

Рис. 6. Реалізації *D** та 95% довірчі інтервали для *N*=9 модулів

© Чмут, О. В., Калініченко, О. Г., & Бодашевський, Є. М. (2023). Технологія створення відмовостійкого багатомодульного програмного комплексу на основі процедури взаємних внутрішніх перевірок. Сучасний захист інформації, 4(56), 52–61. https://doi.org/10.31673/2409-7292.2023.030606.

Висновки

1. Перспективні технології створення відмовостійких багатомодульних програмних комплексів мають базуватися на методології створення надійного програмного забезпечення та автоматичному контролі його працездатності на всіх етапах життєвого циклу без участі людини.

2. Розроблений алгоритм діагностування відмов на основі взаємних внутрішніх перевірок дозволяє здійснити діагностування з різною якістю за різний час, що узгоджується концепцією автоматичного забезпечення відмовостійкості програмних комплексів. Переваги розробленого алгоритму на основі взаємної інформаційної узгодженості полягають у тому, що він вимагає меншої надмірності системи і всього два раунди обміну повідомленнями між модулями програми. При цьому він забезпечує діагностування багатомодульного програмного комплексу при відмові майже половини його модулів.

3. Запропонована методика діагностування забезпечує діагностування збоїв, які можуть виникнути в програмі багатомодульного програмного комплексу за будь-якої форми його прояву. У процесі виконання за цією методикою процедури діагностування накопичується інформація про форми прояву збоїв, достатня для того, щоб визначити її приналежність до однієї з трьох груп, що забезпечує елементи самонавчання технології.

4. Основні результати досліджень можуть бути використані компаніями-розробниками програмного забезпечення при створенні засобів забезпечення відмовостійкості та систем діагностування складних багатомодульних програмних комплексів.

Перелік посилань

1. Коваленко О.В. Моделі та методи розроблення безпечного програмного забезпечення комп’ютерних систем. Дис. на зд. наук. ст. д. т. н. 05.13.05 – комп'ютерні системи та компоненти». Черкаси, ЧДТУ, 2020. 304 с.
2. Закон України «Про захист інформації в інформаційнотелекомунікаційних системах» [Електронний ресурс]. – Режим доступу до ресурсу: http://zakon4.rada.gov.ua/laws/show/2594-15
3. ISO/IEC/IEEE 15939:2017 Systems and software engineering. Measurement process. 2017. 39 p.
4. ISO/IEC TS 33030:2017 Information technology. Process assessment. An exemplar documented assessment process. 2017. 33 p.
5. Gavrylenko, S. Software security overview /Anoushirvan Rashidinia, S. Gavrilenko, M. Pochebut, O. Sytnikova// Системи управління навігації та зв’язку. – ПНУ, 2019. – Вип. 2 (54) с.55-58.
6. Krishnan, M. Soumya Software Development Risk Aspects and Success Frequency on Spiral and Agile Model / M. Soumya Krishnan // International Journal of Innovative Research in Computer and Communication Engineering (An ISO 3297: 2007 Certified Organization). – 2015. –Vol.3. – №1. – pp.301-310
7. Putu, Adi Guna Permana Scrum Method Implementation in a Software Development Project Management / Putu Adi Guna Permana // Int. Journal of Adv. Comp. Science and Appl. – 2015. – Vol. 6. № 9. – P. 199-205.
8. Fayad, Mohamed & Altman, Adam. (2001). An Introduction to Software Stability. Comm. ACM. 44. 95-98.
9. Jureczko, Marian. (2012). Automatic Control of the Software Development Process with Regard to Risk Analysis. 10.5772/45736.
10. Hu, W., Yang, X., Zhu, M., Zhang, Y. (2011). The Application of Automatic Control Theory in Software Engineering. In: Ma, M. (eds) Communication Systems and Information Technology. Lecture Notes in Electrical Engineering, vol 100. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-21762-3_126
11. Salama, Maria & Bahsoon, Rami & Lago, Patricia. (2019). Stability in Software Engineering: Survey of the State-of-the-Art and Research Directions. IEEE Transactions on Software Engineering. PP. 1-1. 10.1109/TSE.2019.2925616.
12. Alenezi, M., “Software Architecture Quality Measurement Stability and Understandability” International Journal of Advanced Computer Science and Applications(IJACSA), 7(7), 2016. http://dx.doi.org/10.14569/IJACSA.2016.070775
13. Himayat, Saif and Ahmad, Dr. Jameel, Software Understandability using Software Metrics: An Exhaustive Review (May 7, 2023). Available at SSRN: https://ssrn.com/abstract=4447189 or http://dx.doi.org/10.2139/ssrn.4447189
14. Чмут О.В. Методика діагностування автоматизованої системи управління повітряним рухом на основі взаємної інформаційної узгодженості. Дис. на зд. наук. ст. к. т. н. за спец. 05.13.06 «інформаційні технології». К., НАУ, 2010. 174 с.

Надійшла: 24.10.2023 Рецензент: д.т.н., професор Вишнівський В.В.