DOI 10.18372/2310-5461.58.17654
УДК 519.718.2

С. С. Штаненко, канд. техн. наук, доцент
Військовий інститут телекомунікацій
та інформатизації імені Героїв Крут
orcid.org/0000-0001-9776-4653
e-mail: sh_sergei@ukr.net;

Ю. Я. Самохвалов, д-р техн. наук, професор
Київський національний університет
імені Тараса Шевченка
orcid.org/0000-0001-5123-1288
e-mail: yu1953@ukr.net;

С. В. Толюпа, д-р техн. наук, професор
Київський національний університет
імені Тараса Шевченка
orcid.org/0000-0002-1919-9174
e-mail: tolupa@ukr.net

ПІДХІД ДО ВИЯВЛЕННЯ ПОМИЛОК ТА ВІДНОВЛЕННЯ ПРАВИЛЬНОГО ФУНКЦІОНУВАННЯ ПРОГРАМНИХ ЗАСОБІВ СУЧАСНИХ СИСТЕМ УПРАВЛІННЯ, РЕАЛІЗОВАНИХ ЗА ПРИНЦИПОМ «СИСТЕМА НА КРИСТАЛІ»

Вступ

Сучасні системи управління різного призначення (автоматизовані системи управління – АСУ, автоматизовані системи управління технологічними процесами – АСУ ТП, автоматизовані системи управління військами – АСУВ тощо) в якості основних елементів містять засоби обчислювальної техніки – процесори, блоки пам'яті, програмне забезпечення, різного роду перетворювачі, датчики, вимірювальні та виконавчі прилади тощо. Зазначимо, що останнім часом в якості основи проєктування сучасних систем управління застосовують обчислювальні засоби, реалізовані на сучасній елементній базі – універсальних і спеціалізованих мікропроцесорних системах, що являють собою систему на одній платі (кристалі) або вбудовану систему.

Аналізуючи останні публікації в літературі та описи різних технічних виробів, які називають «системами на кристалі», можна сформулювати таке визначення, що система на кристалі, однокристальна система (System-on-Chip – SoC) – це електронна схема, яка виконує функції цілого пристрою (наприклад, обчислювальної системи) та розміщена на одній інтегральній схемі, типова структура якої представлена на рис. 1.

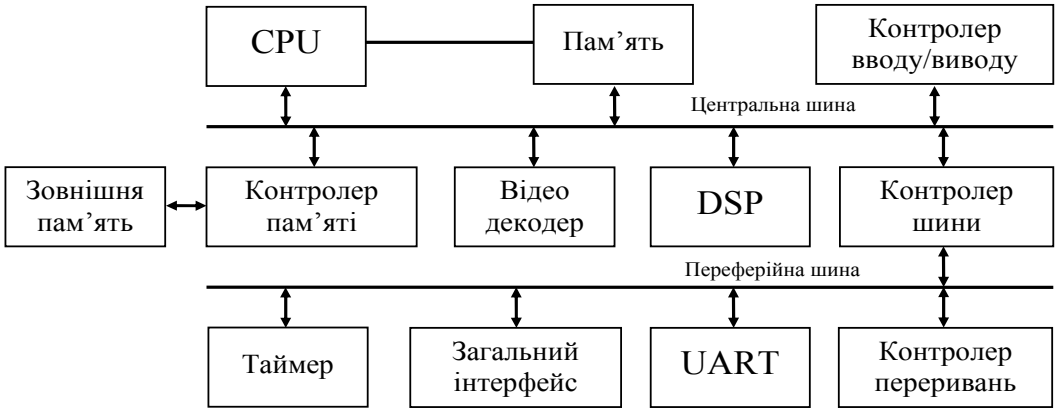

Рис. 1. Структурна схема системи на кристалі

У свою чергу, вбудована система (Embedded system) є спеціалізованою мікропроцесорною системою управління, контролю та моніторингу, концепція розробки якої полягає в тому, що така система функціонує, будучи вбудованою безпосередньо в пристрій, яким вона управляє [1].

Слід зазначити, що перші вбудовані системи розроблялися як спеціалізовані цифрові пристрої

на базі інтегральних схем малого та середнього ступеня інтеграції. Однак, з появою мікроконтролерної та мікропроцесорної техніки, а пізніше й інтегральних схем із програмованою структурою, поняття вбудованої системи сильно трансформувалося. Так, якщо мікропроцесорна вбудована система являла собою спеціалізовану систему, технологічно виготовлену на одній друкованій платі, яка мала у своєму складі центральний процесор, окремі інтегральні схеми контролерів, периферійного обладнання, цифрових пристроїв, то сучасна вбудована система може бути єдиною інтегральною схемою, що включає в свій склад усі вищезазначені компоненти [2]. При цьому в англомовній літературі термін System-on-Chip дуже точно передає тенденцію розвитку сучасних вбудованих систем [3].

Так, на сьогодні в якості елементної бази проєктування Embedded system та System-on-Chip широко застосовуються:

– спеціалізовані ВІС/НВІС (ASIC – Application-Specific Integrated Circuit), що мають індивідуальний характер функціонування, а також використовують традиційні маршрути проєктування інтегральних схем;

– програмовані логічні інтегральні схеми (ПЛІС), що являють собою програмовану вентильну матрицю FPGA (Field-Programmable Gate Array) та жорстку процесорну систему HPS (Hard Processor System), з'єднані між собою інтерфейсом обміну даними. При цьому характер функціонування обчислювальної системи на ПЛІС, що проєктується, задається за допомогою мов опису апаратури або шляхом використання бібліотечних IP-блоків (Intellectual Property), які можуть бути двох типів: Soft IP – описані на рівні реєстрових передач RTL (register transfer level) та Hard IP – описані на топологічному рівні, реалізовані в САПР.

Слід зазначити, що сучасні обчислювальні системи характеризуються великою різноманітністю функціональних можливостей, ступенем складності та специфікою технологій, що застосовуються для їх розробки та виготовлення. Крім цього, кожна зі складових обчислювальної системи (апаратна або програмна частина) має свої особливості та характерні властивості, які зумовлюють існування досить специфічних різновидів несправностей її стану в цілому. Таким чином, перераховані вище специфічні особливості сучасних обчислювальних систем викликали значні зміни, як у процесі їх проєктування, так і розробки методів та засобів контролю, тестового та функціонального діагностування, з метою виявлення та локалізації несправностей, пов'язаних із хіміко-фізичними процесами, а також із навмисними чи ненавмисними несприятливими впливами.

Аналіз останніх досліджень і публікацій

На сьогодні питанням проєктування, функціонування, технічної діагностики, надійності обчислювальних систем присвячено велику кількість наукових праць.

Так, у статті [4] на підставі аналізу вітчизняних та зарубіжних публікацій представлені особливості проєктування «систем на кристалі», основою яких є складні функціональні блоки (СФ-блоки) або бібліотечні IP-блоки (Intellectual Property), а також вбудовані процесорні ядра.

У статті [5] порушено питання підвищення надійності програмного забезпечення шляхом застосування відомих математичних моделей. Розглянуто існуючі способи підвищення надійності програмного забезпечення, що використовують часову та програмну надмірність. Зроблено висновок про відсутність універсального вирішення проблеми надійності програмного забезпечення за наявності множини часткових рішень.

Робота [6] спрямована на організацію тестування та налагодження вбудованих обчислювальних систем зі складною гетерогенною структурою. Запропоновано способи документування тестових впливів вбудованих систем на різних фазах їх створення, а також способи організації процесів тестування, верифікації, валідації та налагодження в комплексних проєктах.

У роботі [7] представлені наукові та практичні результати досліджень у галузі тестового діагностування обчислювальних систем. Наведено основні характеристики оригінальних рішень у галузі стендового обладнання для тестування цифрових модулів, контролепридатного синтезу обчислювальних систем, методів компактного тестування, теорії сигнатурного аналізу та методів самотестування обчислювальних машин і систем.

У статті [8] розглянуто класичні методи діагностування ПЕОМ, розкрито їхні переваги та недоліки. При цьому основну увагу приділено вбудованим засобам діагностування, а саме процедурі Power On Self Test – самоперевірці при включенні, що виконується програмним чином у BIOS/UEFI материнської плати.

Постановка проблеми

Проведений аналіз показує, що на сьогодні не в повному обсязі опрацьовано питання розпізнавання стану програмних засобів спеціалізованих обчислювальних систем, реалізованих за принципом «система на кристалі», способи забезпечення та підвищення надійності програм як невід'ємної складової сучасних апаратно-програмних систем управління, а також питання відновлення правильного функціонування програмних засобів систем, які розглядаються. З

огляду на зазначене, **метою статті** є аналіз існуючих математичних моделей прогнозування надійності програм, способів забезпечення та підвищення їхньої надійності, а також відновлення правильного функціонування програмних засобів сучасних систем управління внаслідок реалізації методів, що ґрунтуються на ідеях теорії штучного інтелекту.

Прогнозування надійності програмних засобів сучасних обчислювальних систем

Термін надійність програмних засобів (ПЗ) виник за аналогією з поняттям надійність технічних виробів. Однак аналогія обмежується формою прояву програмних та технічних відмов, виражених у втраті досліджуваним об'єктом працездатності. Незважаючи на зовнішню схожість проявів відмов апаратури та дефектів ПЗ, механізми їхнього формування мають різну природу. Для програмного забезпечення фундаментальною є ідея проєктних дефектів, а теорія надійності техніки враховує в основному відмови через знос, старіння або інші хіміко-фізичні процеси, механічні причини, що змінюють характеристики об'єкта в часі. Якщо в апаратурі в момент прояву відмови відбуваються деякі зміни (поломка, коротке замикання, зникнення контактів тощо), то програма залишається такою самою, як і була до моменту виявлення помилки (дефекту). Отже, програми не мають фізичного зносу чи старіння, тобто. не змінюють своїх властивостей у часі.

Проте, один із підходів до терміну надійності ПЗ, заснований на теорії надійності технічних виробів, знаходить свій відбиток у тлумаченні цього терміна [9]. Найбільш показовим у цьому відношенні є визначення надійності як властивості системи зберігати протягом необхідного інтервалу часу здатність правильно виконувати задані специфікацією правила переробки інформації в реальних умовах експлуатації.

Інший підхід до терміну надійність ПЗ відображає прагнення до розширювального тлумачення надійності ПЗ і значною мірою ототожнює поняття надійності та якості [10]. Відповідно до цього підходу програма має властивість надійності, якщо можна припускати, що вона «задовільно виконуватиме необхідні функції» або «задовольнятиме очікування користувача».

Третій підхід пов'язує властивості надійності ПЗ з наявністю у програмі помилок, що залишилися після завершення її налагодження, тестування та випробувань. У цьому випадку під надійністю програм розуміють одиничний показник, що характеризує властивість ПЗ виявляти в процесі експлуатації помилки, що залишилися в ній [11]. Таким чином, даний підхід, на відміну від попереднього, звужує характеристику надійності до поняття коректності ПЗ, оскільки відповідно до зазначеного визначення коректна програма вважається абсолютно надійною.

Відомий також підхід [12], згідно з яким під надійністю ПЗ розуміється комплексна властивість, що складається, як і у разі технічних об'єктів, з набору характеристик: коректність, стійкість, відновлюваність та виправність. При цьому коректність і стійкість ПЗ пов'язані з виконанням прогонів програми, тобто процесами, що вимірюються, як правило, інтервалами до сотень секунд. Відновлюваність і виправність – пов'язані з процесами, тривалість яких становить хвилини, години тощо. При такому підході поняття надійності ПЗ можна визначити як комплексну властивість правильно та своєчасно виконувати передбачені специфікації функції у процесі взаємодії з операційним середовищем.

Далі розглянемо надійність ПЗ під час їх експлуатації. При цьому оцінка та прогнозування надійності ПЗ здійснюється на основі математичних моделей надійності програм.

Модель Літтлвуда-Верраллома. Дана модель являє собою часову вісь (рис. 2), де знизу нанесені номери відмов програми, а зверху – інтервали часу між відмовами програми [13].

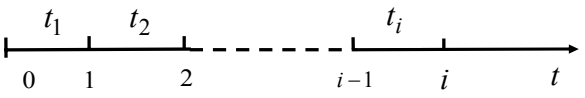

Рис. 2. Моменти відмов програми на осі часу

Згідно з цією моделлю в початковий момент часу t_0 програма працює і зберігає свою працездатність до закінчення інтервалу часу t_1, коли відбувається перша відмова програми. Далі відбувається виправлення програми, яка працює справно протягом часу t_2 і так далі. Нехай випадковий час між i-ю та $(i+1)$-ю відмовою має функцію густини розподілу $f(t \mid \lambda_i)$, де параметр λ_i – міра інтенсивності відмов. Чим менше λ_i, тим краще програма. Далі розглянемо параметр λ_i як довільну величину. Нехай $g_i(\lambda \mid \alpha)$ – функція густини розподілу λ_i, де α – параметр. В результаті цього сформулюємо умову для опису виправлення програми. Зв'яжемо зменшення інтенсивності відмов після виправлення програми з ймовірністю

$$P(\lambda_i < \lambda) > P(\lambda_{i-1} < \lambda) \qquad (1)$$

для всіх λ та i.

Позначимо функцію розподілу параметра λ_i через

$$G_i(\lambda \mid \alpha) = P(\lambda_i < \lambda) = \int_{-\infty}^{\lambda} g_i(\lambda \mid \alpha) d\lambda .$$

Тоді на підставі (1)

$$G_{i-1}(\lambda \mid \alpha) < G_i(\lambda \mid \alpha) .$$

Якщо λ_i – інтенсивність відмов, то $g_i(\lambda \mid \alpha) = 0$ для $-\infty < \lambda < 0$. У моделі передбачається також, що відновлення миттєве (тобто час відновлення набагато менше, ніж час безвідмовної роботи).

Припустимо, що є вихідні дані $t_1, t_2, ..., t_n$, що описують n перші відмови програми, і що виправляється i-а помилка, використовуючи той же метод, що і раніше. Необхідно оцінити розподіл часу до $(n+1)$ відмови, використовуючи попередній досвід (рис. 3).

Функції щільності розподілу, що задовольняють наведеним вище вимогам та поєднують близькі до реальних вирази зі зручним математичним апаратом, мають вигляд:

$$f(t \mid \lambda) = \begin{cases} \lambda e^{-\lambda t}, \text{ когда } t \ge 0; \\ 0, \quad \text{когда} \quad t < 0. \end{cases} \qquad (2)$$

$$g_i(\lambda \mid \alpha) = \begin{cases} \psi(i)(\psi(i)\lambda)^{\alpha-1} e^{-\psi(i)\lambda}, \ \lambda \ge 0; \\ 0, \qquad\qquad\qquad \lambda < 0, \end{cases} \qquad (3)$$

де $\psi(i)$ – деяка монотонно зростаюча від i функція.

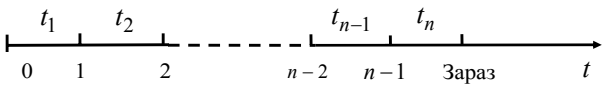

Рис. 3. Прогнозування відмови програми на осі часу

Оцінимо час до наступної відмови, починаючи з довільного часу згідно з рис. 3, на якому t_n означає час від відновлення після $(n-1)$-ї відмови до теперішнього часу «зараз» – нового початку відліку часу t до наступної відмови. Результати спостережень $t_1, t_2, ..., t_{n-1}$ позначають інтервали часу між відмовами (або, точніше, між відновленням після попередньої та настанням наступної відмови). Внаслідок низки перетворень на підставі (2) та (3) отримаємо вираз для інтенсивності відмов у момент часу («зараз»):

$$\lambda(t_1, t_2, ..., t_n) = \frac{n}{\psi(n)} \times \times \left[\log \prod_{i=1}^{n} \frac{\psi(i) + t_i}{\psi(i)} \right]^{-1} \qquad (4)$$

Пропонується вибрати функцію $\psi(i)$ у вигляді

$$\psi(i) = \exp(\beta_0 + \beta_1 i), \qquad (5)$$

де β_0, β_1 – коефіцієнти.

Функція щільності розподілу t – часу до наступної відмови (див. рис. 2) виражається як

$$\int_0^{\infty} f(t \mid \lambda) g_n(\lambda \mid \alpha) d\lambda = \left[\frac{\psi(n)}{t + \psi(n)} \right]^{\alpha} \frac{1}{\psi(n) + t} .$$

Розглянемо спочатку зміну $\lambda(t_1, t_2, ..., t_n)$ протягом одного періоду безвідмовної роботи між двома відмовами: при цьому мається на увазі, що n, а також $t_1, t_2, ..., t_{n-1}$ фіксовані, а t_n зростає. Отже, функція $\lambda(t) = \lambda(t_1, t_2, ..., t_n)$ при $t = t_1 + t_2 + ... + t_n$ зменшується монотонно, оскільки другий член в (4) зменшується монотонно. Вважається, система тоді більш надійна, коли більше часу вона відпрацювала після першої відмови. У момент відмови величина $\lambda(t)$ не визначена, але можна визначити її значення безпосередньо перед відмовою і після останнього виправлення програми, коли є ряд $t_1, t_2, ..., t_n, t_{n+1}$ при $t_{n+1} = 0$. Другий член (4) при цьому залишається незмінним, а перший перетворюється з $n/(\psi(n))$ на $(n+1)/(\psi(n+1))$. Це означає, що після виправлення програми миттєва інтенсивність відмов зменшиться стрибком, якщо $\frac{n+1}{\psi(n+1)} < \frac{n}{\psi(n)}$ або $\frac{\psi(n+1)}{n+1} > \frac{\psi(n)}{n}$. Остання нерівність задовольняється у разі, якщо $\psi(i)$ зростає від i швидше, ніж за лінійною залежністю. Таким чином, миттєва інтенсивність відмов (4) має тенденцію спадати, як при виправленнях з урахуванням швидкого зростання функції $\psi(\cdot)$, так і під час справної роботи між відмовами.

На рис. 4 наведено залежність $\lambda(t)$ для деякої ділянки експлуатації програми: 1 – момент першої та 2 – момент другої відмови.

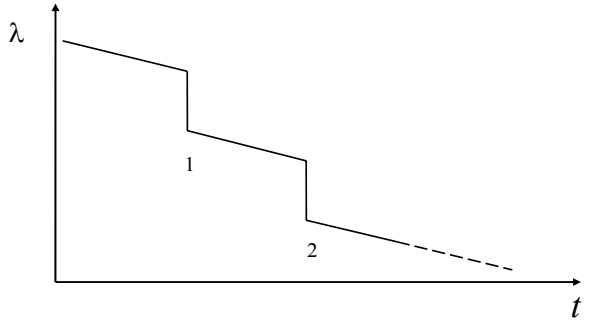

Рис. 4. Графік залежності інтенсивності відмов програми від часу

Наведена (4) модель може бути спрощена, якщо вважати, що $\psi(i) >> t_i$, що має місце при досить великих i. Тоді

$$\lambda(t_1, t_2, ..., t_n) = \frac{n}{\psi(n)}\left[\sum_{i=1}^{n}\log\left(1+\frac{t_i}{\psi(i)}\right)\right]^{-1} \approx \approx \frac{n}{\psi(n)\sum_{i=1}^{n}\left(t_i/\psi(i)\right)}, \quad (6)$$

тобто миттєва інтенсивність відмов обернено пропорційна деякій виваженій середній від інтервалів часу між відмовами. Коефіцієнти β_0, β_1 (5), що входять до $\psi(n)$ (6), можуть бути визначені за експериментальними даними, наприклад, методом максимуму правдоподібності.

Розглянута модель добре пояснює процеси, що відбуваються у ході формування послідовності відмов – відновлення. Однак для практичних розрахунків та прогнозів надійності програм достатньо і простих моделей надійності.

Модель Джелінського-Моранди. Дана модель заснована на наступних припущеннях [14]:

час до наступної відмови розподілено експонентно;

інтенсивність відмов програми пропорційна кількості помилок, що залишилися в програмі.

Згідно з цими припущеннями ймовірність безвідмовної роботи програми як функція часу t_i дорівнює

$$P(t_i) = e^{-\lambda_i t_i}, \quad (7)$$

де

$$\lambda_i = C_D(N-(i-1)). \quad (8)$$

Тут C_D – коефіцієнт пропорційності; N – початкова кількість помилок програми. У (7) відлік часу починається з моменту останньої $(i-1)$-ї відмови програми.

Модель Шумана. Представлена модель відрізняється від моделі Джелінського-Моранди лише тим, що періоди часу налагодження та експлуатації розглядаються окремо.

Модель Шика-Волвертона. Основою даної моделі є пропозиція про те, що інтенсивність прояву помилок програми пропорційна не тільки кількості помилок, що залишилися в програмі, але і часу, витраченому на налагодження.

Модель, запропонована в [15], ґрунтується на припущеннях про те, що потік виникнення ситуації, в яких можлива відмова програми, є пуассонівським з параметром λ, проте в міру виявлення та виправлення помилок відмови у цих випадках виникають з ймовірністю меншою за одиницю. Іншими словами, потік представляється у вигляді розрідженого потоку зі змінним коефіцієнтом розрідження p_i, де i – номер відмови.

У найпростішому випадку

$$p_i = 1-(1-p_н)q^{i-1}, \quad (9)$$

де q – деякий коефіцієнт $(0<q<1)$, $p_н = p_1$ – початковий коефіцієнт розрідження потоку. Зазначимо, що ця модель підходить не лише для прогнозування інтенсивності виникнення наступної відмови програми, але й для прогнозування параметра потоку відмов.

Із (9) випливає, що зображення параметра потоку відмов за Лапласом має вигляд:

$$\omega^*(s) = \sum_{r=1}^{\infty}\prod_{j=1}^{r}\frac{\lambda(1-p_1)q^{j-1}}{s+\lambda(1-p_1)q^{j-1}}. \quad (10)$$

Наближений вираз оригіналу (10) має вигляд:

$$\omega(t) = (1-p_1)\lambda - ((1-p_1)\lambda)^2(1-q)^t + +((1-p_1)\lambda)^3(1-q)(1-q^2)t^2/2! - ...(-1)^n \times \times((1-p_1)\lambda)^{n+1}).(1-q)...(1-q_n)t^n/n!$$

з похибкою не більше

$$\Delta\omega(t) = (-1)^{n+1}((1-p)\lambda)^{n+2} \times \times(1-q)\ ...\ (1-q^{n+1}t^{n+1})/(n+1)! \quad (11)$$

Експонентна модель зміни помилок залежно від часу налагодження. Дана модель враховує кількість команд R у програмі, що випробовуються. У цьому випадку змінюється не абсолютна кількість помилок у програмі, а кількість помилок на одну команду програми $P_0 = n_0/R$. Це дозволяє зручніше порівнювати характеристики програм, більш-менш близьких за обсягом. При цьому облік кількості помилок на одну команду P_0 згладжує, але не повністю компенсує відмінність статичних даних для програм різного обсягу і складності.

Модель, що враховує дискретно-знижувальну частоту появи помилок як лінійну функцію часу тестування та випробувань. Дана модель побудована на основі гіпотези про те, що частота прояву помилок лінійно залежить від часу випробувань t_i між моментами виявлення послідовних i-ї та $(i-1)$-ї помилок:

$$\lambda(t_i) = K\left[N_0-(i-1)\right]t_i, \quad (12)$$

де N_0 – початкова кількість помилок; K – коефіцієнт пропорційності, що забезпечує рівність одиниці площі під кривою ймовірності виявлення помилок. При цьому оцінка надійності (напрацювання на відмову) відповідає розподілу Релею:

$$T(t_i) = \exp\left\{-K\left[N_0 - (i-1)\right]\frac{t_i^2}{2}\right\}. \quad (13)$$

Звідси щільність розподілу часу напрацювання на відмову

$$f(t_i) = -T'(t_i) = K\left[N_0 - (i-1)\right]t_i \times \times \exp\left\{-K\left[N_0 - (i-1)\right]\frac{t_i^2}{2}\right\} \quad (14)$$

Далі, застосувавши функцію максимальної правдоподібності, отримаємо оцінку загальної кількості помилок N_0 та коефіцієнта K:

$$N_0 = \left[\frac{2n}{K} + \sum_{i=1}^{n}(i-1)\,t_i^2\right]\frac{1}{\sum_{i=1}^{n} t_i^2} \quad (15)$$

$$K = \left[\sum_{i=1}^{n}\frac{2}{N_0 - (i-1)}\right]\frac{1}{\sum_{i=1}^{n} t_i^2}. \quad (16)$$

В результаті обробки експериментальних даних моментів виявлення помилок t_i може бути побудована функція, що дозволяє прогнозувати надійність наступної помилки.

Модель, що базується на розподілі Вейбулла. Особливістю цієї моделі є облік ступінчастого характеру зміни надійності при усуненні чергової помилки. В якості основної функції розглядається розподіл часу напрацювання на відмову $T(t)$. Якщо помилки не усуваються то інтенсивність відмов є постійною, що призводить до експоненційної моделі для розподілу:

$$T(t) = \exp(-\lambda t). \quad (17)$$

Звідси щільність розподілу напрацювання на відмову визначається виразом

$$f(t) = \lambda e^{\lambda t}, \quad (18)$$

де $t > 0$, $\lambda > 0$ та $1/\lambda$ – середній час напрацювання на відмову.

Для апроксимації зміни частоти відмов від часу при виявленні та усуненні помилок використовується функція наступного вигляду:

$$\lambda(t) = \lambda\beta t^{\beta-1}. \quad (19)$$

Якщо $0 < \beta < 1$, то інтенсивність відмов знижується в міру налагодження або в процесі експлуатації. При такому вигляді функції $\lambda(t)$ щільність функції розподілу напрацювання на відмову описується двопараметричним розподілом Вейбулла:

$$f(t) = \lambda\beta e^{\beta-1}\exp\left(-\lambda t^{\beta}\right). \quad (20)$$

Дана математична модель використовується в процесі проєктування та експлуатації деяких систем, для яких визначені коефіцієнти λ та β. При цьому розподіл Вейбулла досить добре відображає реальні залежності при розрахунку функції напрацювання на відмову.

Таким чином, розглянуті моделі представляють інтерес, насамперед для прогнозування відмов у процесі експлуатації та налагодження програми. При цьому значення параметрів моделей визначають в процесі експлуатації або налагодження програми за даними про моменти виникнення відмов, використовуючи метод правдоподібності.

Способи забезпечення та підвищення надійності програмних засобів

На сьогодні способи забезпечення та підвищення надійності ПЗ можна поділити на такі основні категорії [15]:

– удосконалення технології програмування;

– вибір алгоритмів, не чутливих до різноманітних порушень обчислювального процесу (використання алгоритмічної надмірності);

– резервування програм – дуальне та N – варіантне програмування, інші методи введення структурної надмірності;

– контроль та тестування ПЗ з подальшою корекцією.

1. *Удосконалення технології програмування*. Нині рівень проникнення сучасної обчислювальної техніки у всі сфери діяльності людини підвищився настільки, що для створення програмного забезпечення нових типів АСУ, пакетів прикладних програм, програм мікропроцесорної техніки, а також їхнього ефективного використання потрібна програмна продукція іншого типу, вищого рівня організації її виробництва. Створення такого виробництва є зараз найбільш актуальним та основним завданням теорії та практики нового напряму у програмуванні – технології програмування.

Під технологією програмування розуміється сукупність узагальнених та систематизованих знань, або наука, про оптимальні способи (прийоми та процедури) проведення процесу програмування, що забезпечує в заданих умовах отримання програмної продукції із заданими властивостями. При цьому вдосконалення технології програмування здійснюється завдяки:

зведенню людського фактора до мінімуму внаслідок відторгнення програмного виробу від його розробника;

забезпеченню цілеспрямованої роботи, насамперед колективу розробників, а не окремих осіб, а також спонуканню колективу працювати

правильно та блокувати будь-які дії, які не санкціоновані технологією;

– переходу на безпаперові технології;

– впровадженню засобів автоматизації на всіх етапах створення програмної продукції;

– мінімізації залежності від мов програмування;

– простоті освоєння та ієрархічної прихильності до виробленого програмного продукту;

– наявності автоматичної фіксації у хронологічному порядку всіх дій, виконуваних у процесі колективного виготовлення програмного виробу.

2. *Вибір алгоритмів, не чутливих до порушень обчислювального процесу*, ґрунтується на дослідженні їхньої чутливості. Мірою чутливості можуть бути похибки, спричинені цими порушеннями. При цьому результати обчислень спотворюються похибками внаслідок вихідних даних, трансформованими під час обчислень, округлення; відмовами, збоями та помилками у програмах. Зауважимо, що помилки цієї категорії можуть бути виявлені та усунені шляхом ретельного налагодження та тестування програми. Розглянемо детальніше цю категорію.

Нехай заданий обчислювальний процес, що реалізує функцію $y = y(x)$, де x – вихідні дані, а y – результат. Тоді мірою похибки результату y є відстань $\rho(y, y^*)$, де y означає точне значення результату, а y^* – спотворене порушенням значення. У разі числового результату під відстанню $\rho(\dots,\dots)$ розуміється або евклідова відстань $\rho = \rho_1 = \sqrt{\sum_{i=1}^{n}\left(y_i - y_i^*\right)^2}$, де y_i, y_i^* – компоненти n-мірних векторів y_i та y_i^*, або вирази виду $\rho = \rho_2 = \sum_{i=1}^{n}\left|y_i - y_i^*\right|$.

У разі, якщо результати представлені у вигляді двійкових кодів, що не мають числового змісту, відстань виражається як $\rho = \rho_3 = \sum_{i=1}^{n}\left(y_i^{(2)} \oplus y_i^{(2)*}\right)$, де $y_i 2, y_i^{(2)*}$ – компоненти n-мірних двійкових кодів $y^{(2)}$ та $y^{(2)*}$, що являють собою неспотворений і спотворений результати.

У разі одновимірного числового результату $(n = 1)$ відстанню ρ служить різниця $\rho = \rho_4 = y - y^*$ або абсолютна різниця $\rho = \rho_5 = \left|y - y^*\right|$. Тоді мірою чутливості M може бути величина

$$M = \frac{\rho(y, y^*)}{\rho(z, z^*)}, \qquad (21)$$

де z, z^* – відповідно неспотворені та спотворені проміжні результати або вихідні дані. Залежно від причини спотворення говорять про чутливість обчислювального процесу до різних порушень. Подібна міра чутливості дозволяє виявити ті частини алгоритмів, які мають підвищену чутливість до різноманітних порушень, у тому числі і до похибок округлення і похибок вихідних даних.

3. *Резервування програм* – дуальне або N-варіантне (мультиверсійне) програмування, моделі відновлювальних блоків (проста, узагальнена та паралельна), а також метод контрольних функцій та перезавантаження.

В основі *N*-варіантного програмування лежить ідея *n*-разової реалізації програми у різний спосіб. Ефективність даного методу визначається насамперед ступенем несхожості програмних компонентів (диверситетом), що зводять до мінімуму появу однакових реакцій у разі порушення роботи обчислювальної системи чи наявності програмних помилок. Якщо розглядати модифіковане дуальне програмування, то ставка робиться на порівняння досить точної, але складної основної програми та простої резервної. Якщо при однакових вихідних даних результати роботи програм відрізняються на величину більшу, ніж допустима похибка, робиться припущення про те, що основна програма відмовила як менш надійна, і за правильний результат приймається результат, отриманий за допомогою резервної програми. Внаслідок цього середня похибка роботи двох програм дещо збільшується, але ймовірність відмови зменшується.

У свою чергу використання методу блоків відновлення зводиться до того, що в кожну версію вводиться модуль прийняття рішень. Якщо при виконанні першої версії виявляється помилка, то буде викликана відновлювальна функція, яка повертає систему в той стан, який було збережено до виконання першої версії. Потім виконуються та перевіряються наступні версії, доки не буде знайдена вірна відповідь або доки не закінчаться версії. Система повертає помилку, якщо жодна версія не дала прийнятних результатів.

При методі контрольних функцій і перезавантаження поряд з обчислюваною функцією за іншою програмою визначається друга функція, що перебуває з основною функцією, що обчислюється, у співвідношеннях, які називаються контрольними співвідношеннями. Ці співвідношення дозволяють не тільки виявити відмову однієї з програм, але також і відновити спотворений результат програми, що відмовила, на підставі результату, отриманого за безпомилково працюючою програмою.

4. *Контроль та тестування програм* з подальшою корекцією.

У спеціалізованих обчислювальних системах, що виконують обмежену кількість функціональних програм, широко застосовується контроль правильності виконання програм, який називається програмно-логічним контролем та включає в себе:

– контроль тривалості та послідовності виконання програми;

– метод контрольних функцій;

– контроль гладкості.

Контроль тривалості виконання програми заснований на тому, що для кожної програми відома максимальна тривалість виконання і будь-яке перевищення тривалості означає, що програма зациклилася, зупинилася або виконана неправильно. Перевищення тривалості виконання обов'язково пов'язане з помилками у програмі, часто причиною може бути спотворення адресної інформації збоями. Можливі навіть випадки, коли почне виконуватися одна програма, а потім результат збою виконання переходить помилково до іншої програми. Тому іноді застосовують контроль послідовності виконання підпрограм, порівнюючи номери фактично виконаних підпрограм з потрібними значеннями.

Метод контрольних функцій ґрунтується на тому, що результати роботи програми мають відповідати певним функціональним співвідношенням. Наприклад, рішення системи диференціальних рівнянь можуть бути перевірені за критерієм задоволення контрольного рівняння, утвореного у вигляді рівнянь вихідної системи.

Контроль гладкості заснований на тому, що якщо ряд результатів обчислень являє собою більш-менш гладку функцію, то будь-які різкі відхилення результату від екстрапольованого значення свідчать про помилку.

Отже, перелічені види контролю правильності виконання програм здійснюються переважно ПЗ. Вони дозволяють виявляти помилки в роботі спеціалізованих обчислювальних систем із затримкою, порівнянною з часом виконання програми або підпрограми.

У свою чергу, під тестуванням програм будемо розуміти перевірку роботи програми за результатами її виконання на спеціально підібраних наборах вихідних даних – тестах. Відповідно до [16] тестування програми поділяють на тестування чорної, білої та сірої скриньок.

Тестування чорної скриньки є стратегією, в якій тестування засноване виключно на вимогах і специфікаціях, при цьому не потрібні знання щодо внутрішніх шляхів, структури або реалізації програми, що перевіряється.

Тестування білої скриньки являє собою стратегію, в якій тестування засноване на внутрішніх зв'язках, структурі та реалізації програми, що перевіряється, а також детальних знань у галузі програмування.

Додатковим видом тестування є тестування сірої скриньки. При такому підході відбувається процес вивчення «скриньки» настільки, щоб зрозуміти, як вона була реалізована, а також обираються найефективніші тести чорної скриньки.

Слід зазначити, що тестування є невід'ємною частиною загальної методології розробки програмного забезпечення та характеризується високою трудомісткістю та відповідно вартістю. Серед багатьох проблем і завдань з тестування побудова тестів є одним із найскладніших інтелектуальних завдань, оскільки якість тестових послідовностей та трудомісткість їхньої побудови дуже впливають на розробку програмного забезпечення.

Зауважимо, що тести як важливий допоміжний елемент програмного забезпечення повинні будуватися на підставі деякої інформації, яка включає:

– структуру програми та/або вихідний код;

– специфікацію програмного забезпечення та/або моделі його проєктування;

– інформацію про простір даних вводу/виводу та інформацію, що динамічно отримується внаслідок виконання програми.

Розглянемо найпоширеніші методи тестування програмного забезпечення [17].

Метод символьного виконання заснований на аналізі коду програми для автоматичного генерування тестових даних програми та реалізує концепцію білої скриньки, тобто аналізує програму шляхом її вихідного коду чи двійкового коду. При цьому символьне виконання використовує символьні значення замість конкретних значень вхідних даних і представляє значення програмних змінних як символьні вирази цих вхідних даних.

Побудова тестів на основі моделі є формальною методологією, яка використовує різні моделі програмних систем для отримання тестових наборів. За такого підходу акцент робиться на поведінковий рівень програм, реалізуючи концепцію чорної скриньки, за якої тестується програма згідно з її поведінкою. Різновидом даної моделі є:

– аксіоматичний підхід побудови тестів, який використовує моделі, що базуються на певній формі логічного обчислення;

– підхід із використанням моделей кінцевих автоматів.

У рамках аксіоматичного підходу розроблені різні поняття тестових гіпотез, зокрема регуляр-

ності та однорідності. Відповідно до гіпотези про регулярність, всі можливі значення X для вхідних тестових впливів до певної складності N вважаються достатніми для досягнення необхідної ефективності тесту. Відповідно до гіпотези однорідності одне значення для кожного класу вхідних даних вважається достатнім для досягнення необхідної повноти покриття тесту.

Основою методів побудови тестів відповідно до їхніх моделей є кінцеві автомати та різні їхні модифікації. Кінцевий автомат являє собою систему з кінцевою множиною станів, серед яких виділяються початковий стан і кінцеві множини стимулів (вхідних даних), що сприймаються ззовні, а також реакцій, що формуються (вихідних даних). Крім цього, в кінцевому автоматі визначено набір переходів між станами, причому кожен перехід позначений його вхідним значенням (стимулом), що викликається, та вихідною реакцією, що генерується.

Комбінаторне тестування. Суть даного методу полягає в виборі вхідних параметрів чи реконфігурацій настройки, які охоплюють задану підмножину комбінацій елементів програм, які підлягають тестуванню.

Адаптивне ймовірнісне тестування. Даний метод реалізує концепцію чорної скриньки та є альтернативою ймовірнісному тестуванню. За таких умов структура та функціональність об'єкта тестування приймається невідомою або вважається, і ніяк не враховується при формуванні тестових наборів. Крім цього, ймовірнісне тестування не використовує інформацію, яка доступна в процесі генерування чергового тестового набору. Ця інформація може бути отримана з попередніх тестових наборів та в подальшому використана для формування чергового тактового впливу.

Пошукове тестування. В основі даного методу лежить використання програмного забезпечення, в якому застосовуються алгоритми оптимізації для автоматичного пошуку тестових даних, що максимізують досягнення цілей тестування, одночасно зводячи до мінімуму витрати на побудову тестів.

Підходи пошукового тестування охоплюють широке коло прикладних областей, включаючи: еволюційне тестування, що ґрунтується на програмних агентах; емпіричний підхід до пошукового тестування; структурний підхід до тестування взаємодії; мультиеволюційний підхід до інтеграційного тестування; високорівневий мутаційний підхід до формування тестових даних; формування тестових даних для стресового тестування; генерування тестів для web-додатків, а також регресійного тестування.

Таким чином, розглянуті способи забезпечення та підвищення надійності ПЗ сучасних обчислювальних систем дають можливість не тільки зберігати протягом необхідного інтервалу часу здатність правильно виконувати задані специфікацією правила переробки інформації в реальних умовах експлуатації, але також можуть бути основою для проєктування сучасних відмовостійких апаратно-програмних комплексів .

Підхід до відновлення правильного функціонування програмних засобів

Якщо розглядати логіку роботи та методи, закладені в основу засобів прийняття рішення на вибір тих чи інших стратегій відновлення правильного функціонування ПЗ, то основна схема, яка традиційно використовується у всіх системах, є схемою табличного типу. Інтерпретувати її можна таким чином: залежно від того, до якого типу виняткових ситуацій належить поточний стан системи, відповідно до таблиці обробки виняткових ситуацій вибирається та чи інша стратегія відновлення. Формально це можна представити наступним чином. Нехай $M = \{M_i\}$ – множина відмовних ситуацій. $P = \{P_i\}$ – множина процедур відновлення, причому $M \sim P$. N – ситуація нормальної роботи, $\varphi(j)$ – стан комплексу програм у разі виникнення j відмовної ситуації. Тоді $\forall M_i \; P : \varphi(M_i) \to \varphi(N)$. Подібні схеми прості, легко реалізовані, але їхній фундаментальний недолік полягає в тому, що вони працюють в умовах, коли типи відмовних ситуацій мають суто детермінований характер. В умовах значної невизначеності та неповноти відомостей про характер відмовної ситуації ці схеми фактично неприйнятні. Враховуючи даний факт, багато вчених [18; 19] пов'язують процес відновлення правильного функціонування ПЗ з концепцією інтелектуалізації, в основі якої лежать три основні складові:

– наявність бази даних (БД), бази знань (БЗ) та спеціальних мов подання знань;

– наявність спеціального механізму «машино логічного висновку», призначеного для уніфікації логічних процедур, що оперують знаннями стосовно даних;

– можливість заміни декларативного способу вирішення задач, наближених до людського мислення. Даний процес є, по суті, ситуаційним управлінням, коли рішення на продовження роботи ПЗ приймається на основі аналізу поточного стану всієї сукупності даних, які оброблюються з використанням деяких правил, які називаються стратегією виведення.

Так, згідно з [20] найбільш перспективним є підхід, заснований на використанні таких дедуктивно-адаптивних процедур відновлення правильного функціонування, які були б подібні до тих, якими користується людина для отримання висновків зі спостережень та вибору тих чи інших рішень. У формальному вигляді це можна уявити так. Нехай $M = \{M_i\}$ – множина відмовних ситуацій, N – ситуація нормальної роботи, $\varphi(j)$ – стан комплексу програм у разі виникнення j відмовної ситуації, $R = \langle S, L \rangle$ – дедуктивно-адаптивна процедура відновлення, де S – управляюча структура, L – механізм логічного висновку. Тоді $\forall M_i \; R : \varphi(M_i) \to \varphi(N)$. Реалізація цього підходу дозволить, по-перше, стандартизувати програмний апарат та запровадити єдину методику його розробки, тобто способи відновлення правильного функціонування стануть незалежними від відмовних ситуацій, а їх налаштування на конкретну ситуацію здійснюватиметься шляхом введення відповідних знань. По-друге, у разі зміни тих чи інших відмовних ситуацій виключається необхідність коригування програмного апарату, що супроводжується обов'язковим внесенням додаткових помилок, при цьому зміні підлягають лише відповідні знання.

На сьогодні розроблено велику кількість методів та теорій прийняття рішень, які могли б бути покладені в основу запропонованого підходу. Найбільш поширеними є: методи теорії дослідження операцій, теорії вибору та прийняття рішень, а також методи штучного інтелекту. Однак проведений аналіз показує, що дані методи відносяться до суворих кількісних методів, а це в свою чергу змушує чітко і однозначно визначати не лише цілі функціонування системи та різні варіанти її досягнення, але також визначати фактори, що враховуються при виборі однієї з альтернатив. Крім цього, необхідно врахувати також той факт, що побудова плану відновлення правильного функціонування відбувається в умовах значної невизначеності та неповноти відомостей про характер відмовної ситуації та ступеня її впливу на працездатність ПЗ.

Виходячи із зазначеного, можна зробити висновок, що традиційні методи кількісного моделювання та оптимізації виявляються малоефективними чи зовсім непридатними. Ця обставина і обумовлює необхідність поряд з подальшим розвитком кількісних методів використовувати інші методи для вирішення проблем, орієнтованих на оперування з факторами, що виражаються у нечислових формах.

Так, сьогодні згідно з [21] найбільш перспективними методами дослідження складних систем є методи логіко-лінгвістичного моделювання, які засновані на ідеях теорії штучного інтелекту.

Методи логіко-лінгвістичного моделювання не слід розглядати як альтернативу кількісним методам дослідження складних систем. Вони, швидше за все, є їх розширенням та доповненням, розвиваючись у слабо структурованих проблемних галузях, де застосування точних кількісних методів малоефективне чи неможливе. Зокрема, одним із таких методів є метод ситуаційного управління, в основі якого лежать уявлення знань про об'єкт управління та способи управління даним об'єктом [22]. Крім цього в логіко-лінгвістичних моделях реалізуються свої специфічні методи пошуку оптимальних рішень, орієнтовані на якісний опис компонентів рішень, зв'язків між ними та критеріїв вибору рішення. При цьому необхідно розуміти, що системи ситуаційного управління не призначені для оптимізації самого процесу управління, вони орієнтовані лише на таке управління, при якому результати будуть ідентичні тим результатам, які отримує людина, а в деяких випадках навіть краще. На рис. 5 представлена структурна схема, що реалізує метод ситуаційного управління.

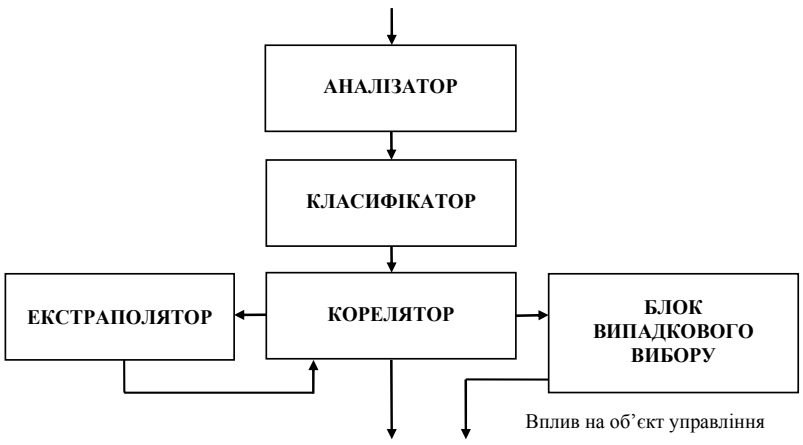

Рис. 5. Структурна схема, що реалізує метод ситуаційного управління

До складу даної схеми входить аналізатор, який призначений для зняття інформації та оцінки поточної ситуації на об’єкті управління, а також прийняття рішення щодо втручання в процес управління об’єктом. Класифікатор, який призначений для порівнювання поточної ситуації з інформацією про об'єкт управління та віднесення даної ситуації до одного або кількох класів, яким відповідають однокрокові рішення. Корелятор, в якому зберігаються всі логіко-трансформаційні правила та здійснюється вибір правила для подальшого виконання. Якщо таке правило єдине, воно видається до виконання, інакше вибір кращого правила проводиться після обробки попередніх рішень в екстраполяторі.

Якщо корелятор і класифікатор не можуть прийняти рішення щодо опису поточної ситуації, то спрацьовує блок випадкового вибору, який вибирає один з впливів, що впливають на об’єкт.

Розглянута схема, що реалізує метод ситуаційного управління, є плануючою системою, яка спочатку формує план відновлення правильного функціонування, перевіряє його виконання, і при необхідності коригує даний план при надходженні додаткової інформації від об’єкта управління або навколишнього середовища.

Так, згідно з [23] розрізняють два види планування. У першому випадку вводиться поняття стану, що складається зі стану об’єкта управління, стану довкілля та стану системи управління. Побудова плану відновлення правильного функціонування відбувається у просторі станів в такий спосіб, що кожне однокрокове рішення з управління переводить всю систему з одного стану до іншого у просторі станів. При цьому цей план є деякою траєкторією у просторі станів (рис. 6).

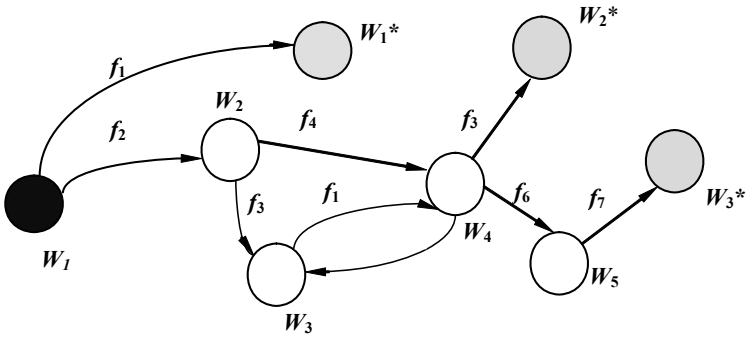

Рис. 6. Граф станів

У другому випадку передбачається, що є певний набір модулів, здатних вирішувати певні завдання. Процес планування полягає у пошуку такої декомпозиції вихідної задачі, елементами якої виявились би готові модулі. Тоді план відновлення правильного функціонування являє собою деревоподібну структуру, переміщаючись якою можна поступово отримати рішення більш часткових завдань вирішення вихідної задачі. При цьому зауважимо, що відмінною особливістю планування рішень, що спираються на висновок у формальній системі, є те, що план відновлення правильного функціонування ПЗ, знайдений за допомогою логічного висновку, реалізується в реальному часі та просторі, а особливості цього реального середовища необхідно враховувати при плануванні.

Таким чином, реалізація методу ситуаційного управління дає можливість побудувати таку дедуктивну процедуру відновлення правильного функціонування ПЗ, яка б адаптувалася на відмовну ситуацію не через вибір того чи іншого блоку відновлення, а шляхом введення відповідних знань. Крім цього, запропонований підхід до відновлення правильного функціонування ПЗ спільно із застосуванням процедури логічного висновку дозволить скоротити середній час відновлення правильного функціонування по відношенню до традиційного підходу.

Висновки

Розглянуто математичні моделі, які призначені для оцінки та прогнозування надійності ПЗ та є невід’ємною складовою сучасних обчислювальних систем. Представлено способи, що забезпечують та підвищують надійність ПЗ, а саме: вдосконалення технології програмування, використання алгоритмічної надмірності, резервування програм, контроль та тестування програм з подальшою корекцією. Запропоновано в якості основного методу прийняття рішення на відновлення правильного функціонування ПЗ використовувати схему побудови багатокрокових рішень, засновану на логіко-лінгвістичних моделях, які засновані на ідеях теорії штучного інтелекту. Це по-перше дозволить стандартизувати програмний апарат шляхом запровадження єдиної методики розробки, внаслідок чого засоби

відновлення ПЗ стануть незалежними від відмовних ситуацій, а адаптація їх до конкретної ситуації здійснюватиметься шляхом введення відповідних знань. А по-друге, скоротити час відновлення до деякого рівня, специфічного для кожної системи, відмови ПЗ перетворити на збої і тим самим підвищити надійність системи.

ЛІТЕРАТУРА

[1] A. Crespo, P. Albertos, J. Simó, Embedded control systems: from design to implementation, *Ifac Proceedings Volumes*, Volume 40, Issue 1, 2007, Pages 25–32, ISSN 1474-6670, ISBN 9783902661210. DOI: 10.3182/20070213-3-CU-2913.00006.

[2] Smit, Wim & Hendriksen, Wim. Embedded systems: Smart and intelligent tools in an increasingly interconnected globalised world. *International Journal of Technology Policy and Management*. 2004. Vol. 4, pp. 309–323. DOI:10.1504/IJTPM.2004.006614.

[3] Boudewijn R. Haverkort Challenges for modelling and analysis in embedded systems and systems-of-systems design. 1st Workshop on *Advances in System of System* (AiSoS-2013) EPTCS 133, pp. 40–46. DOI:10.4204/ EPTCS.133.5

[4] Palagin A. V., Boyun V. P., Yakovlev Yu. S. The problems of Creation the Computer Systems with Nanoelement Base Application. *Control Systems and Computers*, № 5 (271), 2017, pp. 3–15. DOI: 10.15407/usim.2017.05.

[5] Anil Arora, A. G. (2016). Software Reliability. A Review. International Journal of Scientific Research and Management, 4(7). URL: https://ijsrm.in/index.php/ijsrm/article/view/419

[6] Pinkevich V. Yu., Platunov A. E. Testing and debugging of embedded computing systems based on level models. *Scientific and Technical Journal of Information Technologies*, *Mechanics and Optics*. 2018. № 5 (117). T. 18. C. 801–808.

[7] Kleiman, L. A., & Freyman, V. I. Improving the functioning reliability of the information management system elements, using built-in diagnostic tools. *Radio Electronics, Computer Science, Control*, 2021, (1), pp. 158–171.

[8] Лукасевич Д. Б., Огнєвий О. В. Використання процедури POST в процесі тестового діагностування ЕОМ. *Вимірювальна та обчислювальна техніка в технологічних процесах*. 2010. № 2. С. 150–153.

[9] Lipaev V. V. , Software reliability (a review of the concepts), *Avtomat. i Telemekh.*, 1986, no. 10, 5–31; Autom. Remote Control, 47:10 (1986), pp. 1313–1335.

[10] Wilson G, Bryan J, Cranston K, Kitzes J, Nederbragt L, Teal TK. Good enough practices in scientific computing. *PLoS Comput Biol* 2017. 13(6): e1005510. https://doi.org/10.1371/journal.pcbi.1005510

[11] Kaffashi, Esmail. (2015). Evaluation Criteria for Reliability in Computer Systems. *Journal of Electrical and Electronic Engineering*. 3. 83. 10.11648/ j.jeee.s.2015030201.28.

[12] Wang, Cheng Cheng, et al. "Research on Reliability Analysis Method of Industrial Control System Based on Markov Process." *Applied Mechanics and Materials*, vol. 541–542.

[13] Littlewood, B., and J. L. Verrall. A Bayesian Reliability Growth Model for Computer Software. *Journal of the Royal Statistical Society*. Series C (Applied Statistics) 22, no. 3 (1973): 332–46. DOI:10.2307/2346781.

[14] Thomas Thayer, Myron Lipow, Eldred Carlyle Nelson. Software Reliability. North-Holland Publishing Company, 1978. p. 311. ISSN 0167-7888.

[15] Iyudu K. A. Reliability, control and diagnostics of computers and systems. M.: Higher school, 1989. 216 p.

[16] L. Copeland. A Practitioner's Guide to Software Test Design. Artech House. Publishers, 2003. p. 300.

[17] Yarmolik, V. N., & Shevchenko, N. A. (2022). Synthesis of test sequences with a given switching activity. *Automation and Remote Control*, 83(2), 291–302.

[18] Moskalenko V, Kharchenko V, Moskalenko A, Kuzikov B. Resilience and Resilient Systems of Artificial Intelligence: Taxonomy, Models and Methods. *Algorithms*. 2023; 16(3):165. https://doi.org/10.3390/a16030165

[19] Samokhvalov, Y. (2001). Automatic Theorem Proving and Fuzzy Situational Search for Decisions. *Cybernetics and Systems Analysis* № 37(4), pp. 509–514. DOI: 10.1023/A:1012725620069.

[20] Adrita, Mumtahina & Brem, Alexander & O'Neill, Patrick & Gorman, Eymard & O' Sullivan, Dominic & Bruton, Ken. (2020). Development of a Decision Support System to Enable Adaptive Manufacturing. *Smart and Sustainable Manufacturing Systems*. DOI: 10.1520/SSMS20190036.

[21] Samokhvalov, Y. Y. (1997). Decomposition of linguistic-logical decision models in distributed computing environments. *Cybernetics and Systems Analysis* № 33, pp. 44–49. DOI: 10.1007/BF02665939.

[22] Robert R. Blake, Jane Srygley Mouton. (1982). A comparative analysis of situationalism and 9,9 management by principle. *Organizational Dynamics*, Volume 10, Issue 4, pp. 20–43. ISSN 0090-2616. DOI: 10.1016/0090-2616(82)90027-4.

[23] Gladun, V., Vaschenko, N. Control on the Basis of Network Models. *IFAC Proceedings Volumes*, Volume 31, Issue 29, 1998, pp. 247–250. ISSN 1474-6670.

Штаненко С. С., Самохвалов Ю. Я., Толюпа С. В.

ПІДХІД ДО ВИЯВЛЕННЯ ПОМИЛОК ТА ВІДНОВЛЕННЯ ПРАВИЛЬНОГО ФУНКЦІОНУВАННЯ ПРОГРАМНИХ ЗАСОБІВ СУЧАСНИХ СИСТЕМ УПРАВЛІННЯ, РЕАЛІЗОВАНИХ ЗА ПРИНЦИПОМ «СИСТЕМА НА КРИСТАЛІ»

У статті проведено аналіз існуючих математичних моделей надійності програмних засобів сучасних управляючих систем, розкрито переваги та недоліки. Представлені математичні моделі дозволяють оцінювати характеристики помилок у програмах та прогнозувати їхню надійність при проєктуванні та експлуатації. Дані моделі мають імовірнісний характер, і достовірність прогнозів залежить від точності вихідних даних та глибини прогнозування за часом. Розглянуто способи, що забезпечують та підвищують надійність програмних засобів обчислювальних систем, які реалізують принцип «система на кристалі». Реалізація даних способів забезпечення та підвищення надійності надасть можливість сучасним обчислювальним системам зберігати протягом необхідного інтервалу часу здатність правильно виконувати задані специфікацією правила переробки інформації у реальних умовах експлуатації. Запропоновано план відновлення правильного функціонування програмних засобів, в основі якого лежить схема побудови багатокрокових рішень, що базується на логіко-лінгвістичних моделях та ідеях теорії штучного інтелекту. Зокрема, одним із таких методів є метод ситуаційного управління, в основі якого лежать уявлення знань про об'єкт управління та способи управління даним об'єктом. Крім цього в логіко-лінгвістичних моделях реалізуються свої специфічні методи пошуку оптимальних рішень, орієнтовані на якісний опис компонентів рішень, зв'язків між ними та критеріїв вибору рішення. При цьому системи ситуаційного управління не призначені для оптимізації самого процесу управління, вони орієнтовані лише на таке управління, при якому результати будуть ідентичні тим результатам, які отримує людина, а в деяких випадках навіть краще. Реалізація запропонованого плану, в основі якого лежить метод ситуаційного управління надасть можливість побудувати дедуктивну процедуру відновлення правильного функціонування програмних засобів, а це, у свою чергу, дозволить сучасній обчислювальній системі адаптуватися як до відмовних ситуацій, так і до навмисних або ненавмисних несприятливих впливів.

Ключові слова: контроль та тестування програм, надійність програмного засобу, відновлення правильного функціонування програм.

Shtanenko S., Samokhvalov Yu., Toliupa S.

APPROACH TO DETECTING ERRORS AND RESTORING THE CORRECT FUNCTIONING OF SOFTWARE IN MODERN CONTROL SYSTEMS IMPLEMENTED ACCORDING TO THE PRINCIPLE "SYSTEM ON A CRYSTAL"

The article analyzes the existing mathematical models of software reliability of modern control systems, reveals the advantages and disadvantages. The presented mathematical models make it possible to evaluate the characteristics of errors in programs and predict their reliability during engineering and maintenance. These models are probabilistic themselves, and the forecasts' reliability depends on the accuracy of the initial data and the depth of forecasting over time. Methods are considered as providing and improving the reliability of software tools for computing systems that implement the "system on a chip" principle. The implementation of these methods of providing and improving reliability will enable modern computing systems to maintain the ability to fulfill the rules of information processing specified by the specification in real operating conditions for the required time interval in a proper way. A plan for restoring the correct functioning of software tools is proposed, which is based on a scheme for constructing multi-step solutions based on logical-linguistic models and ideas of the theory of artificial intelligence. In particular, one of these methods is the method of situational control, which is based on the representation of knowledge about the control object and how to control this object. In addition, logical-linguistic models implement their own specific methods for finding optimal solutions, focused on a qualitative description of the components of the solutions, the relationships between them and the criteria for choosing a solution. At the same time, situational management systems are not designed to optimize the management process itself, they are focused only on such management, in which the results will be identical to those that a person receives, and in some cases even better. The implementation of the proposed plan, which is based on the method of situational management, will make it possible to build a deductive procedure for restoring the correct functioning of software, and this, in turn, will allow a modern computer system to adapt both to negative situations and to intentional or unintentional adverse effects.

Key words: control and testing of programs, reliability of software, restoration of the correct programs' functioning.

Стаття надійшла до редакції 03.05.2023 р.
Прийнято до друку 01.06.2023 р.