Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 1 (188) *ISSN 3083-7650 (0135-1710)(print)*

УДК 004.416 DOI: https://doi.org/10.30837/0135-1710.2026.188.145

Репіхов В. М., Чуприна А. С.

МЕТОД ПРЕВЕНТИВНОГО СУПРОВОДЖЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ НА ОСНОВІ КОМПАРАТОРНОЇ МОДЕЛІ

Предметом дослідження *є метод превентивного супроводження програмного забезпечення на основі компараторної моделі, який забезпечує формалізоване оцінювання експлуатаційного стану й підтримку прийняття рішень щодо експлуатаційної реакції.* ***Мета роботи*** *– підвищення ефективності превентивного супроводження програмного забезпечення способом формалізованого оцінювання його експлуатаційного стану, що забезпечує раннє виявлення деградаційних процесів, класифікацію експлуатаційної реакції та автоматизоване формування інцидентів супроводження за умов обмеженої історії інцидентів.* ***Основні завдання****: обґрунтувати вимоги до формалізованого оцінювання експлуатаційного стану програмного забезпечення в умовах превентивного супроводження та обмеженої доступності історичних даних; побудувати компараторну модель ідентифікації експлуатаційних станів способом формалізації взаємозв'язків між ознаками у вигляді системи логічних рівнянь і перевірки її функціональної визначеності; розробити метод компараторної ідентифікації експлуатаційних станів на основі побудованої моделі; виконати експериментальну валідацію запропонованого методу на експлуатаційних даних і порівняти з традиційними пороговими механізмами сповіщення.* ***Методи дослідження.*** *Для ідентифікації експлуатаційного стану застосовано метод компараторної ідентифікації в межах компараторної моделі з використанням предикатного подання стану. Взаємозв'язки між ознаками формалізовано системою логічних рівнянь із перевіркою функціональної визначеності. Формування ознак ґрунтується на поєднанні метрик тестування, агрегованих експлуатаційних показників, трендових характеристик і ознак аномальної поведінки. Емпіричну перевірку виконано на симуляційних експлуатаційних даних із використанням ковзних вікон агрегації та порівнянням із пороговими механізмами сповіщення.* ***Результати.*** *Продемонстровано, що запропонований метод забезпечує більш раннє виявлення деградаційних процесів порівняно з класичними пороговими алертами. Використання трендових і комбінованих ознак підвищує стабільність класифікації станів і зменшує кількість хибних спрацьовувань у дослідженому сценарії. На основі ідентифікованого класу стану реалізовано модель превентивного супроводження, що забезпечує автоматизоване формування інцидентів із зазначенням типу проблеми, локалізації та рівня терміновості.* ***Висновки.*** *Досягнуті результати розвивають формалізовані підходи до оцінювання експлуатаційного стану програмного забезпечення й демонструють можливість застосування компараторної моделі для превентивного супроводження за відсутності повної та репрезентативної історії інцидентів. Практична значущість роботи полягає в можливості впровадження інтерпретованих механізмів раннього виявлення деградації та автоматизації супроводження, зокрема формування інцидентів і підтримки вибору експлуатаційної реакції, без необхідності накопичення значних обсягів історичних даних.*

Ключові слова*: програмне забезпечення; превентивне супроводження; компараторна модель; компараторна ідентифікація; експлуатаційний стан; деградація програмного забезпечення; автоматизоване формування інцидентів.*

Вступ

Забезпечення стабільної експлуатації програмних продуктів упродовж усього життєвого циклу є одним із ключових завдань сучасної інженерії програмного забезпечення [1]. В умовах тривалої експлуатації програмні системи зазнають поступових змін, пов'язаних із модифікаціями коду, змінами конфігурацій, оновленням інфраструктури й зростанням навантажень, що призводить до появи деградаційних процесів. Традиційні підходи до супроводження програмного забезпечення, як правило, мають реактивний характер і ґрунтуються на усуненні вже зафіксованих інцидентів, що знижує ефективність експлуатації та підвищує ризик критичних відмов [2, 3].

З огляду на сказане актуальним є розвиток підходів превентивного супроводження програмних продуктів, орієнтованих на раннє виявлення ознак деградації та потенційних проблем ще до їх переходу в проблемний стан [1]. Реалізація такого підходу потребує формалізованого оцінювання експлуатаційного стану програмного забезпечення на основі сукупності різнорідних показників, що відтворюють поведінку системи в умовах експлуатації.

Водночас ідентифікація стану програмного забезпечення на практиці ускладнюється низкою обмежень. Зокрема здебільшого відсутні репрезентативні історичні дані експлуатації, а також апріорні оцінки проблемності можливих станів програмного продукту. Це унеможливлює використання класичних

ISSN 3083-7715 (online) Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188)
ISSN 3083-7650 (0135-1710)(print) Management Information System and Devices. 2026. No. 1 (188)

статистичних або навчальних підходів і зумовлює необхідність формування інформативної множини ознак у процесі поетапного накопичення інформації про систему.

Перспективним інструментом для розв'язання окресленого завдання є метод компараторної ідентифікації [4, 5], що дає змогу оцінювати стан об'єкта способом порівняння векторів ознак без обов'язкової наявності повних навчальних вибірок. У контексті превентивного супроводження програмного забезпечення це відкриває можливість поєднання апріорних властивостей, отриманих на етапах тестування, з апостеріорними експлуатаційними показниками, як-от тренди й ознаки аномальної поведінки.

Отже, ключовою науково-практичною проблемою є розроблення формалізованого методу превентивного супроводження програмного забезпечення, що поєднує поетапне формування множини ознак експлуатаційного стану з компараторною моделлю ідентифікації. Такий підхід має забезпечувати інтерпретовану класифікацію стану програмної системи з погляду експлуатаційної реакції та підтримувати автоматизоване формування інцидентів супроводження із зазначенням типу проблеми, її локалізації, порушеної умови або тенденції до її порушення, рівня терміновості та відповідальної команди. Зважаючи на сказане, сформулюємо основні питання дослідження.

*RQ*1. Яким чином може бути побудована компараторна модель формалізованої ідентифікації експлуатаційних станів програмного забезпечення в умовах обмеженої історії інцидентів?

*RQ*2. Яка структура й послідовність поетапного формування алфавіту ознак є достатньою для раннього виявлення деградаційних процесів і забезпечення функціональної визначеності моделі?

*RQ*3. Яким чином результати компараторної ідентифікації можуть бути використані для автоматизованого формування інцидентів супроводження з визначенням типу проблеми, локалізації та рівня терміновості?

Аналіз сучасних публікацій щодо проблеми превентивного супроводження програмного забезпечення

Превентивне супроводження програмного забезпечення в сучасних дослідженнях висвітлюється як стратегія проактивного втручання в життєвий цикл програмних систем з метою виявлення та усунення потенційних проблем до їх переходу у фазу відмов або критичних дефектів. У стандартах життєвого циклу превентивне супроводження визначається як модифікація програмного продукту після впровадження в експлуатацію, яка спрямована на ідентифікацію латентних дефектів і зниження ризику їх прояву в майбутньому [6]. Незважаючи на формалізацію цього поняття на нормативному рівні, його практична реалізація залишається неоднорідною та недостатньо систематизованою. Аналіз сучасних публікацій свідчить, що превентивне супроводження все частіше розглядається не як завершальна фаза, а як безперервний процес [7], тісно пов'язаний з розробленням і експлуатацією програмного продукту [8, 9]. Однак така інтеграція породжує нові методологічні й організаційні виклики, що досі не мають усталених рішень.

Однією з ключових проблем превентивного супроводження є складність раннього виявлення латентної деградації програмних систем. Дослідники доводять, що більшість відмов має накопичувальну властивість і супроводжується поступовими змінами метрик продуктивності, якості або структури системи, які не фіксуються традиційними механізмами тестування й моніторингу [1, 10]. У роботах, присвячених аналізу журналів подій, наголошено, що аномалії в логах можуть бути ранніми індикаторами майбутніх інцидентів, проте їх ідентифікація потребує складних методів оброблення даних і машинного навчання [11, 12].

Водночас огляди самонавчальних і самоадаптивних систем свідчать, що наявні підходи часто не здатні відрізнити тимчасові флуктуації від стабільних трендів деградації, що обмежує їх застосовність у превентивному супроводженні [13]. Як наслідок, превентивні дії ініціюються надто пізно, коли система вже потребує коригувального, а не профілактичного втручання.

Також суттєвою проблемою є відсутність сталого зв'язку між сигналами ризику, отриманими в процесі моніторингу, та управлінськими рішеннями щодо супроводження. Навіть коли потенційні проблеми ідентифікуються автоматизованими засобами, вони часто не трансформуються в конкретні завдання для команди розроблення або супроводження [1, 14]. У дослідженнях з управління супроводженням програмного забезпечення зазначено, що аналітичні

Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 1 (188) *ISSN 3083-7650 (0135-1710)(print)*

результати існують ізольовано від систем планування робіт, що призводить до ігнорування превентивних рекомендацій [15]. Систематичні огляди, присвячені поєднанню супроводження й машинного навчання, вказують на фрагментарність інтеграції аналітичних моделей у процеси прийняття рішень [16]. Це зумовлює домінування реактивного підходу, коли ресурси спрямовуються переважно на усунення вже наявних дефектів, а не на запобігання їх виникненню.

Проблема обґрунтованої пріоритизації превентивних заходів тісно пов'язана з управлінням технічним боргом. У низці систематичних оглядів наголошено, що, незважаючи на наявність численних метрик технічного боргу, відсутні універсальні методи визначення того, які превентивні дії мають найвищий пріоритет у конкретний момент часу [17]. Це ускладнює планування профілактичних робіт і знижує їх привабливість для менеджменту. Дослідження, присвячені прогнозуванню дефектів на основі історії змін, демонструють потенціал кількісного оцінювання ризиків, однак їх застосування потребує високої якості інформації та значних обчислювальних ресурсів [18]. Як наслідок, превентивні заходи часто розглядаються як другорядні порівняно з функціональним розвитком системи.

Хоча методи машинного навчання активно використовуються для підтримки превентивного супроводження, огляд джерел вказує на низку їх обмежень. Зокрема складні моделі глибокого навчання, застосовувані для виявлення аномалій або прогнозування дефектів, визначаються низькою інтерпретованістю, що ускладнює їх практичне впровадження [11, 13]. У студіях з експлуатації *ML*-систем також наголошено на проблемі деградації якості моделей у часі, зумовлену дрейфом даних і зміною контексту застосування [19]. У цьому контексті формування практик *MLOps* розглядається як відповідь на виклики превентивного супроводження *ML*-орієнтованих систем, однак систематичні огляди засвідчують відсутність єдиних стандартів і зрілих процесів у цій сфері [16]. Це обмежує можливість широкого впровадження превентивних підходів у проєктах з інтенсивним використанням машинного навчання.

Окрему групу проблем становлять організаційні та процесні чинники. У дослідженнях продемонстровано, що ефективність превентивного супроводження здебільшого залежить від структури команди й рівня взаємодії між розробленням і експлуатацією [20]. Попри існування міжнародних стандартів, вони мають переважно декларативну властивість і не пропонують детальних механізмів реалізації превентивного супроводження на практиці [6, 9].

Систематичний аналіз літератури демонструє, що превентивне супроводження програмного забезпечення є актуальним, але має низку методологічних прогалин. Основні питання зосереджені навколо раннього виявлення деградації, інтеграції аналітичних сигналів у процеси прийняття рішень, пріоритизації превентивних дій, а також організаційних бар'єрів. Аналіз наукових досліджень дав змогу обґрунтувати актуальність розроблення формалізованих моделей превентивного супроводження.

Мета й завдання дослідження

Метою статті є розроблення та обґрунтування методу превентивного супроводження програмного забезпечення на основі компараторної моделі, що уможливлює формалізоване оцінювання його експлуатаційного стану, раннє виявлення деградаційних процесів, класифікацію експлуатаційної реакції та автоматизоване формування інцидентів супроводження в умовах обмеженої історії інцидентів.

Для досягнення поставленої мети в роботі необхідно розв'язати певні завдання.

1. Обґрунтувати вимоги до формалізованого оцінювання експлуатаційного стану програмного забезпечення в умовах превентивного супроводження та обмеженої доступності історичних даних.

2. Побудувати компараторну модель ідентифікації експлуатаційних станів способом формалізації взаємозв'язків між ознаками у вигляді системи логічних рівнянь і перевірки її функціональної визначеності.

3. Розробити метод компараторної ідентифікації експлуатаційних станів програмного забезпечення, оснований на побудованій компараторній моделі.

4. Провести емпіричну валідацію запропонованого методу на експлуатаційних даних і виконати порівняння з традиційними пороговими механізмами сповіщення.

Матеріали й методи

Центральною проблемою розв'язання задачі превентивного супроводження програмного забезпечення є побудова процедур, здатних виявляти потенційні деградаційні стани до настання відмови або порушення якості. Класичні підходи машинного

ISSN 3083-7715 (online) Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188)
ISSN 3083-7650 (0135-1710)(print) Management Information System and Devices. 2026. No. 1 (188)

навчання, зокрема класифікатори й регресійні моделі прогнозування, зазвичай спираються на наявність історії інцидентів або принаймні на приклади негативних наслідків, що утворюють навчальну вибірку. У контексті превентивного супроводження ця передумова є центральною проблемою: ефективне превентивне супроводження програмної системи полягає в тому, що потенційно ризикова ситуація визначається завчасно й очікуваний негативний розвиток ситуації переривається вчасним утручанням, унаслідок чого спостережувані дані не містять оцінки негативних наслідків. За такого підходу ознаки, основані на подіях (інцидент / неінцидент, порушення/ непорушення *SLO*) перестають бути об'єктивним відтворенням латентного стану системи. Це знижує коректність прямого перенесення класичного *ML*-підходу на задачу превентивного супроводження й мотивує розгляд альтернативних ідентифікаційних парадигм, які не вимагають повної та репрезентативної історії.

Метод компараторної ідентифікації, розроблений у працях [4, 5, 21], пропонує таку альтернативу, замінюючи вимогу наявності числової цільової функції або "істинних" міток на використання лише бінарної інформації про попарну нерозрізненість або, навпаки, розрізненість ознак. Його вихідним положенням є те, що для відновлення латентної структури станів об'єкта достатньо мати доступ до компаратора, який на парі ознак повертає значення предиката, інтерпретованого як "еквівалентність" цих ознак з погляду внутрішнього стану [4]. Такий підхід принципово відрізняється від класичного *ML*: він не апроксимує безпосередньо ймовірність небажаної події та не вимагає спостереження цієї події в даних, натомість ідентифікує класи станів як фактор-множину за відношенням, що задається компаратором.

Формалізація методу здійснюється апаратом алгебри скінченних предикатів і систем логічних рівнянь. Метою формалізації є побудова такої системи логічних рівнянь, яка однозначно відтворює спостережувані значення предикатів у внутрішні стани системи. Ця система будується ітеративно способом послідовного уточнення алфавіту предикатів до досягнення функціональної визначеності рішення.

Нехай задано універсум об'єктів U. Предикатом називається відображення

$$P_i : U \to \{0,1\},\ i = 1,\dots,n.$$

Сукупність предикатів $A = \{P_1,...,P_n\}$ утворює алфавіт опису об'єктів. Кожному об'єкту $u \in U$ відповідає предикатний образ

$$\mathrm{P}(u) = \left(P_1(u),...,P_n(u)\right) \in \{0,1\}^n.$$

Алфавіт предикатів A формує на множині U відношення еквівалентності

$$\sim_A \subseteq U \times U,$$

$$u \sim_A v \Leftrightarrow \wedge_{i=1}^{n}\left(P_i(u) \leftrightarrow P_i(v)\right),\ u,v \in U.$$

Це відношення розбиває універсум U на класи еквівалентності, кожен з яких визначається унікальним предикатним образом. Нехай $D = \{d_1,...,d_m\}$ – скінченна множина можливих станів та існує невідома функція класифікації (прийняття рішення щодо певного стану) $Y : U \to D$. Для формалізації додано булеві предикати $D_j(u) = 1 \Leftrightarrow Y(u) = d_j$, $j = 1,\dots,m$. На ці предикати накладено структурні аксіоми:

1) аксіома повноти:

$$\vee_{j=1}^{m} D_j(u) = 1;$$

2) аксіома взаємонесумісності:

$$D_i(u) \wedge D_j(u) = 0,\ i \neq j.$$

Алфавіт предикатів є придатним для ідентифікації, якщо виконується умова функціональної визначеності:

$$u \sim_A v \Rightarrow Y(u) = Y(v). \qquad (1)$$

Умова (1) означає, що рішення $Y(u)$ однозначно визначається предикатним образом $P(u)$: якщо $P(u) = P(v)$, то $Y(u) = Y(v)$. Порушення цієї умови інтерпретується як неповнота алфавіту предикатів.

Алгебра предикатів дає змогу будувати логічні формули з предикатних змінних за допомогою операцій кон'юнкції, диз'юнкції та заперечення. Інформація, отримана від компаратора, фіксується у вигляді логічних обмежень двох основних типів: імплікації допустимості

$$\Phi(P_1,...,P_n) \Rightarrow D_j,$$

яка означає, що за виконання умови $\Phi(P_1,...,P_n) = 1$ можливе лише рішення d_j, та імплікації заборони

$$\Psi(P_1,...,P_n) \Rightarrow \neg D_j,$$

де $\Phi, \Psi : \{0,1\}^n \to \{0,1\}$ – булеві функції над предикатами $P_1(u),...,P_n(u)$. Сукупність усіх таких

Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 1 (188) *ISSN 3083-7650 (0135-1710)(print)*

імплікацій разом з аксіомами повноти й взаємонесумісності утворює систему логічних рівнянь, що визначає допустимі значення предикатів рішень:

$$\begin{cases} \Phi(u) \Rightarrow D_j(u), \\ \Psi(u) \Rightarrow \neg D_j(u), \\ \bigvee\limits_{j=1} D_j(u) = 1, \\ D_i(u) \wedge D_j(u) = 0,\ i \neq j. \end{cases}$$

За умови (1) задача зводиться до побудови булевих функцій f_j на просторі $\{0,1\}^n$:

$$D_j(u) = f_j\left(P_1(u),\ldots,P_n(u)\right),\quad j \in \{1,\ldots,m\},$$

тобто множини

$$X_j = \left\{ x \in \{0,1\}^n \,\middle|\, f_j(x) = 1 \right\}$$

утворюють розбиття простору предикатних образів.

У реальних умовах експлуатації алфавіт ознак A не є інформаційно повним. Тому поряд із функціональною моделлю прийняття рішення $Y(u) = f\left(P(u)\right)$ доцільно розглядати парну постановку, яка дає змогу визначати подібність нової ситуації до вже відомих. У парній постановці додається предикат порівняння

$$E(u,v) = 1 \Leftrightarrow Y(u) = Y(v)$$

або через предикати рішень

$$E(u,v) \Leftrightarrow \bigvee_{j=1}^{m} \left(D_j(u) \wedge D_j(v) \right).$$

Сукупність алфавіту A, предиката порівняння E та відповідне розбиття множини об’єктів на класи еквівалентності утворюють компараторну модель. Інформація про латентні стани задається відношенням еквівалентності між об’єктами. Якщо рішення однозначно визначається предикатним образом $P(u)$, то існує булева функція

$$\Theta : \{0,1\}^n \times \{0,1\}^n \to \{0,1\},$$

така, що

$$E(u,v) = \Theta\left(P(u), P(v)\right).$$

У загальному вигляді задача компараторної ідентифікації формулюється так [4, 21]. Нехай на множині M задано предикат E. Тоді існує відображення

$$F : M \to N,$$

таке, що

$$E(x,y) = 1 \Leftrightarrow F(x) = F(y),$$

тоді й лише тоді, коли E є відношенням еквівалентності на M, тобто для всіх $x, y, z \in M$ виконуються властивості рефлексивності, симетричності та транзитивності [4]:

$$E(x,x) = 1,$$

$$E(x,y) = 1 \Rightarrow E(y,x) = 1,$$

$$E(x,y) = 1 \wedge E(y,z) = 1 \Rightarrow E(x,z) = 1.$$

За виконання цих умов множина M факторизується на класи еквівалентності $[x] = y \in M : E(x,y) = 1$, і тоді можна покласти $F(x) = [x]$, отримуючи реалізацію рівності $E(x,y) \Leftrightarrow F(x) = F(y)$. Ця теорема одночасно визначає фундаментальне обмеження методу: якщо E не є еквівалентністю, то не існує жодного відображення F, узгодженого з E в наведеному сенсі, а отже, задача ідентифікації є некоректно поставленою незалежно від обсягу даних або складності алгоритмів.

Зазначене обмеження має принциповий характер. Зокрема предикати, що виражають відношення порядку або переваги (наприклад "гірше / краще", "більш ризикове / менш ризикове"), загалом не задовольняють симетричності й часто порушують транзитивність у присутності шуму, дрейфу та контекстної залежності. У цьому разі інформація компаратора не може бути інтерпретована як ознака рівності латентних станів, і метод компараторної ідентифікації в класичному вигляді не застосовується [1].

Другим, часто неявним, обмеженням є вимога нетривіальної розрізненості ознак. Навіть якщо E формально є еквівалентністю, можлива виродженість, коли

$$\forall x, y \in M\ E(x,y) = 1.$$

Тоді факторизація M містить рівно один клас і відображення F існує, але є інформативно тривіальним: воно не відображає жодної внутрішньої структури режимів об’єкта. Звідси випливає, що для отримання практично корисної моделі необхідна наявність принаймні двох експлуатаційно розрізнених класів, тобто нетривіальний компаратор, який інколи повертає 0 і цим формує непорожню структуру фактор-множини.

З огляду на ці формальні засади застосування компараторної ідентифікації до превентивного супроводження програмного забезпечення вимагає адаптації. По-перше, необхідно формувати

ISSN 3083-7715 (online) Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188)
ISSN 3083-7650 (0135-1710)(print) Management Information System and Devices. 2026. No. 1 (188)

й обґрунтовувати такі відношення попарної еквівалентності станів, які відображають операційно значущі інваріанти, зокрема експлуатаційні реакції. По-друге, важливо побудувати такий предикат, що забезпечує нетривіальну розрізненість станів (з погляду експлуатаційних реакцій).

Результати дослідження

Побудова системи ознак

Основою підходу до превентивного супроводження програмного забезпечення є набір вхідних ознак стану програмної системи. Ця система ознак містить різнорідні показники якості та поведінки програмного продукту, отримані під час його експлуатації й тестування: метрики тестування, агреговані показники, сигнали аномалій, динаміка й тренди показників тощо.

Метрики тестування відбивають результати й ефективність тестування (наприклад, відсоток пройдених тестів, кількість виявлених дефектів до релізу, показники продуктивності під тестовим навантаженням тощо). Ці метрики слугують об’єктивними індикаторами правильності функціювання системи відповідно до визначених вимог і можливих сценаріїв та можуть сигналізувати про відхилення ще на етапі контролю якості.

Агреговані показники за групами властивостей – це інтегральні індикатори, обчислені для основних якісних аспектів програмного забезпечення, зокрема функціональності, умов експлуатації, безпеки й зручності використання. Кожна група об’єднує низку конкретних метрик (ознак), які відповідають ключовим властивостям якості продукту. Агрегування за категоріями дає змогу отримати узагальнену оцінку стану за кожною важливою сферою якості.

Аномалією вважається будь-яка поведінка системи або значення показника, яке істотно відхиляється від очікуваного або нормального. Кожний сигнал аномалії є ознакою потенційної проблеми, що потребує уваги.

Аналіз динаміки дає змогу розрізняти одноразові відхилення від стійких тенденцій. На основі історичних даних відстежується тенденція (тренд) ключових параметрів, що допомагає сигналізувати про можливі подальші проблеми ще до того, як будуть перевищені граничні допустимі значення метрик.

Набір ознак забезпечує комплексний профіль поточного експлуатаційного стану програмної системи. Цей профіль формалізується у вигляді скінченного алфавіту булевих предикатів, де кожний предикат відображає наявність або відсутність певної ознаки поточного стану програмної системи. Важливо, що вхідні ознаки формалізуються у вигляді скінченного набору показників, які далі можуть бути перетворені для реалізації процедури компараторної ідентифікації.

Формалізація компараторної моделі

Алгебра скінченних предикатів (АСП) надає універсальну мову для опису скінченних множин ознак і відношень між ними у вигляді предикатних формул і рівнянь. На відміну від класичної булевої алгебри, яка оперує бінарними змінними й виразами, АСП працює з предикатами над скінченними універсумами, даючи змогу описувати будь-які відношення між об’єктами. У нашому випадку універсумом є множина можливих значень вектора ознак, а предикат відповідає класифікаційній функції, що встановлює належність стану до певного класу еквівалентності.

Реалізація методу превентивного супроводження на основі компараторної моделі передбачає такі етапи: 1) трансформація спостережуваних ознак у двійкові індикатори; 2) формування вектора стану програмної системи (предикатного образу); 3) побудова системи рівнянь для визначення класів еквівалентності; 4) порівняння векторів стану і встановлення еквівалентності (застосування компаратора); 5) вибір і реалізація відповідної експлуатаційної реакції системи.

На першому етапі кожен вхідний показник (метрика або сигнал), значення яких збираються під час моніторингу, перетворюється на логічний (двійковий) індикатор, що набуває значення 1 або 0 залежно від виконання певної умови. Для кількісних метрик додаються порогові значення або допустимі діапазони: якщо метрика перебуває в межах норми – відповідний індикатор 0 (немає відхилення), у разі перевищення порогу або виходу за межі норми – індикатор 1 (фіксується проблема). Для якісних або категоріальних ознак визначаються логічні умови або правила. Наприклад, сигнал аномалії безпосередньо є бінарним індикатором (1 – виявлено аномалію, 0 – ні), агрегований показник групи може розбиватися на кілька індикаторів (наприклад,

Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 1 (188) *ISSN 3083-7650 (0135-1710)(print)*

$Functionality\_OK = 1$, якщо всі функціональні тести пройдено, $Security\_Risk = 1$, якщо знайдено критичну вразливість, тощо). Унаслідок цього етапу вихідний різнорідний вектор даних перетворюється на набір логічних змінних $\{x_1, x_2, ..., x_n\}$, кожна з яких набуває значення 1 за наявності відповідного негативного симптому або 0 у разі його відсутності.

Отже, двійкові індикатори згруповуються у впорядкований набір – вектор стану $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)$, кожна компонента якого відповідає окремому індикатору. Вектор x є формальним відображенням поточного стану програмної системи в просторі ознак. Розмірність n вектора стану визначається повним переліком контрольованих ознак, таким чином метод легко розширюється додаванням нових індикаторів без зміни загального підходу.

На основі експертних знань і аналізу даних будується формальна логічна модель, що описує кожний допустимий клас стану системи. Для цього визначається множина класів еквівалентності $C_1, C_2, \ldots, C_m$, які відповідають різним ситуаціям експлуатації програмної системи. Для визначеності в межах цього дослідження запропоновано обрати три класи експлуатаційних реакцій: "норма", "проблема", "попередження". Кожний клас еквівалентності математично задається логічним виразом щодо компонент вектора x. Зручним канонічним способом подання таких умов є диз'юнктивна нормальна форма (ДНФ). Наприклад, формула для класу C_i може мати вигляд

$$F_i(x_1, \ldots, x_n) = \left(a_1^{(1)} \wedge a_2^{(1)} \wedge \ldots \wedge a_n^{(1)}\right) \vee \left(a_1^{(2)} \wedge \ldots \wedge a_n^{(2)}\right) \vee \ldots \vee \left(a_1^{(k)} \wedge \ldots \wedge a_n^{(k)}\right),$$

де $a_j^{(q)}$ дорівнює або x_j, або $\neg x_j$ (залежно від того, яким має бути значення ознаки j у q-му варіанті стану, що належить класу C_i). Отже, кожен кон'юнкт відповідає окремому поєднанню ознак, що належить цьому класу, а диз'юнкція таких кон'юнктів перелічує всі можливі варіанти в межах класу. Теоретично доведено, що виділення класів еквівалентності через ДНФ забезпечує повноту опису всіх можливих комбінацій ознак для кожного класу.

Компараторно-орієнтований підхід реалізує процедуру ідентифікації класу поточного стану способом порівняння вектора стану x, отриманого з поточних даних, із умовами (еталонами) для кожного класу C_i. Формально це означає перевірку виконання логічного рівняння $F_i(x) = 1$ для кожного i. Якщо для деякого класу C_k підставлення компонент x у формулу F_k задовольняє її (значення $F_k(x)$ істинне), то поточний вектор стану належить цьому класу еквівалентності. Практично таке зіставлення здійснюється як попарне порівняння бітів двох векторів: вектора x та еталонного вектора (або векторів) класу C_i. Еквівалентність двох станів установлюється, якщо всі відповідні компоненти збігаються. За більш складних умов (коли клас визначений через диз'юнкцію кількох кон'юнктів) перевірка еквівалентності зводиться до порівняння x з кожним допустимим шаблоном класу: якщо x збігається з хоча б одним шаблоном (варіантом кон'юнкту), вважаємо, що x належить класу. Таким чином здійснюється компараторна ідентифікація поточного стану: модель порівнює стан системи із заздалегідь визначеними еталонами нормального чи аномального функціювання й обирає еквівалентний клас. З погляду АСП, система рівнянь у ДНФ, побудована для класів стану, фактично є предикатною моделлю об'єкта (програмної системи), яка відтворює простір станів у термінах істинних значень предикатів (індикаторів).

Кожний виділений клас еквівалентності станів програмної системи пов'язаний із певною реакцією або заходом з боку експлуатаційної команди супроводження. Крім належності до категорії, для кожного класу може визначатися рівень критичності, який деталізує пріоритет реагування. Наприклад, клас "проблема" може мати підкласи за критичністю: високий (негайно виправити, падіння системи), середній (значна функціональна деградація), низький (некритичний збій окремого модуля). Критичність визначається як функція від значень індикаторів у векторі стану (наприклад, аномалія безпеки може автоматично підвищувати критичність). Отже, еквівалентні стани об'єднано в класи щодо їх впливу на функціювання програмного продукту, і кожному класу відповідає наперед визначена реакція підтримки й рівень пріоритетності. Це дає змогу автоматизувати ухвалення рішень: формально порівнявши вектор стану з класами, система супроводження одразу визначає, чи потрібні дії та як швидко.

ISSN 3083-7715 (online) Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188)
ISSN 3083-7650 (0135-1710)(print) Management Information System and Devices. 2026. No. 1 (188)

Метод компараторно-орієнтованого превентивного супроводження програмного забезпечення

Основна ідея методу полягає в побудові компараторної моделі, де кожне рівняння описує окремий клас еквівалентності. Клас еквівалентності визначається як множина станів програмного забезпечення, що потребують однакової реакції з боку експлуатаційної команди. Отже, метод зводить задачу превентивного супроводження до задачі логічної ідентифікації класу, до якого належить поточний стан системи.

У процесі експлуатації алгоритм періодично формує вектор індикаторів для поточного вікна спостереження й виконує компараторну ідентифікацію способом підставлення цього вектора в систему ДНФ-рівнянь. Як наслідок, визначається множина класів, предикати яких виконуються для поточного стану. Якщо спрацьовує рівно один клас, він вважається ідентифікованим. У разі спрацювання кількох класів застосовується правило розв'язання конфліктів. Якщо жоден клас не спрацьовує, стан вважається новим або раніше неописаним і обробляється як стан, що потребує додаткової уваги.

Після ідентифікації класу формується експлуатаційна реакція, що передбачає тип дії, рівень критичності та відповідальну команду. Додатково генерується пояснення результату у вигляді переліку індикаторів і диз'юнкта ДНФ, який спрацював. За необхідності автоматично створюється превентивне завдання або алерт для відповідної команди.

Алгоритм методу містить такі етапи й кроки.

Input :

$\mathcal{T}$ – специфікація тестів і критерії приймання (*acceptance rules*);

T – результати тестування й тестові метрики;

$M(t)$ – потік експлуатаційного моніторингу;

$A(t)$ – потік зафіксованих аномалій;

$\mathcal{W}$ – параметри вікон агрегування (розмір, крок);

$\mathcal{R}$ – модель ризику / критичності (правила домену, безпека, *SLO/SLA*);

$\mathcal{O}$ – правила локалізації (ознаки / артефакт $\rightarrow$ команда);

Π – політика ревізії моделі (період / умови).

Output :

$COPM = (I, P, F, \text{Map})$,

де I – множина бінарних індикаторів (скінченних предикатів);

P – правила / предикати бінаризації;

$F = \{F_c\}_{c \in C}$ – система ДНФ-рівнянь для класів еквівалентності;

Map – відображення класу $\rightarrow$ (*reaction, priority, owner*);

Γ – потік превентивних завдань / алертів.

Phase A. Побудова індикаторів із тестування (апріорна база)

1. $I \leftarrow \varnothing$, $P \leftarrow \varnothing$.

2. $S_{test} \leftarrow ExtractTestFeatures(\mathcal{T}, T)$.

3. Для кожної ознаки $s \in S_{test}$:

3.1. Побудувати предикат $p_s(\cdot)$ на основі *acceptance criteria* й $\mathcal{R}$.

3.2. Визначити індикатор $i_s := 1[\neg p_s]$ (1 – порушено, 0 – виконується).

3.3. $I \leftarrow I \cup \{i_s\}$, $P \leftarrow P \cup \{p_s\}$.

Phase B. Додавання експлуатаційних ознак (агрегати, тренди, аномалії)

4. Задати групи агрегування $G = \{Func, Env, Sec, Usab\}$.

5. Для кожного вікна $w \in \mathcal{W}$:

5.1. $Z_w \leftarrow Aggregate(M(t), w, G)$ (агреговані показники за групами).

5.2. $D_w \leftarrow Trends(Z_w)$ (дрейф / стабільність / повернення до еталона).

Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 1 (188) *ISSN 3083-7650 (0135-1710)(print)*

5.3. $A_w \leftarrow Anomalies\left(A(t), w\right)$ (наявність / тип / оцінка аномалії).

5.4. $S_{ops} \leftarrow ComposeFeatures\left(Z_w, D_w, A_w\right)$.

5.5. Для кожної нової ознаки $s \in S_{ops}$, що поки не модельована:

– побудувати $p_s(\cdot)$ із $\mathcal{R}$ і еталонів тестування;

– визначити $i_s := 1[\neg p_s]$;

– додати i_s до I, а p_s до P.

Phase C. Побудова класів еквівалентності (компараторна модель)

6. Визначити множину реакцій $R = \{NORM, ATTENTION, PROBLEM\}$.

7. Сформувати множину класів еквівалентності C за принципом:
стани еквівалентні тоді й лише тоді, коли потребують однакової експлуатаційної реакції.

8. Для кожного класу $c \in C$:

8.1. Визначити допустимі патерни індикаторів $X_c \subseteq 0,1^{|I|}$ (з тестів, аналізу ризиків тощо).

8.2. Побудувати ДНФ-предикат класу:

$$F_c(x) = \vee_{k=1}^{K_c}\left(\wedge_{l \in L_{c,k}} \lambda_l(x)\right),$$

де $\lambda_l(x)$ – літерал x_j або $\neg x_j$.

9. Визначити $\text{Map}(c) = (reaction, priority, owner)$ за $\mathcal{R}$ та $\mathcal{O}$.

10. (*Рекомендовано*) Провести валідацію:

– неперетинність $F_c \wedge F_d = 0$ для $c \neq d$ на множині відомих станів;

– покриття "відомих" станів $\vee_{c \in C} F_c = 1$.

Phase D. Онлайн-ідентифікація та формування превентивних завдань

11. Для кожного нового вікна w у процесі експлуатації:

12. Обчислити вектор індикаторів

$$x_w \leftarrow EvalIndicators\left(I, P, M(t), A(t), w\right) \in 0,1^{|I|}.$$

13. Знайти множину спрацьованих класів

$$S \leftarrow c \in C :: F_c(x_w) = 1.$$

14. Обрати клас c^*

якщо $|S| = 1$, то $c^* \leftarrow S$;

якщо $|S| > 1$, то $c^* \leftarrow Resolve(S)$ (правило: *safety-first, max priority, max specificity*);

якщо $|S| = 0$, то $c^* \leftarrow c_{new}$ і реакція за замовчуванням ATTENTION (*triage*).

15. $(r^*, p^*, o^*) \leftarrow \text{Map}(c^*)$.

16. Сформувати пояснення $E \leftarrow ExplainDNF\left(F_{c^*}, x_w\right)$ (який диз'юнкт ДНФ спрацював + активні індикатори).

17. Якщо $r^* \neq NORM$, сформувати завдання:

$$\gamma \leftarrow CreateTask\left(o^*, p^*, r^*, E, w\right), \quad \Gamma \leftarrow \Gamma \cup \{\gamma\}.$$

Phase E. Ревізія моделі (адаптація)

18. Якщо політика Π ініціює ревізію (періодично або за умови накопичення c_{new} / дрейфу / нових релізів):

19. Зібрати фідбек (результати реакцій, нові релізи, нові тестові метрики).

20. Оновити P (пороги / еталони), оновити I (вилучити шумові, додати нові індикатори).

21. Уточнити C і перебудувати відповідні F_c у ДНФ; повторити крок 10.

22. Повернути оновлену модель $COPM = (I, P, F, \text{Map})$.

ISSN 3083-7715 (online) Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188)
ISSN 3083-7650 (0135-1710)(print) Management Information System and Devices. 2026. No. 1 (188)

Отже, на відміну від методів машинного навчання, які ґрунтуються на статистичному узагальненні історичних даних, запропонований метод використовує апріорні знання, зафіксовані на етапі тестування, та формалізує їх у вигляді скінченної системи предикатів. Це дає змогу застосовувати метод за відсутності або недостатності статистики відмов, а також забезпечує інтерпретованість результатів ідентифікації.

Оскільки програмні системи еволюціонують у процесі експлуатації, метод передбачає регулярну ревізію компараторної моделі. Підставами для ревізії можуть бути накопичення нових невідомих станів, зміни вимог, нові релізи або результати аналізу інцидентів. У процесі ревізії оновлюються предикати, порогові значення й склад індикаторів, після чого уточнюються класи еквівалентності й перебудовуються відповідні ДНФ-рівняння. Це дає змогу підтримувати актуальність моделі та зберігати її здатність до превентивної ідентифікації деградацій програмного забезпечення.

Отже, метод превентивного супроводження програмного забезпечення на основі компараторної ідентифікації уможливлює формалізований, інтерпретований і незалежний від статистики підхід до раннього виявлення проблем у програмних системах. Поєднання тестових і експлуатаційних ознак у межах компараторної моделі створює науково обґрунтований базис для превентивного супроводження й автоматизованого формування експлуатаційних рішень.

Експериментальні дослідження: симуляційний приклад

Метою експерименту є емпірична перевірка здатності запропонованого методу компараторної ідентифікації виявляти деградаційні процеси раніше, ніж класичні порогові алерти, та формувати інтерпретовані стани експлуатаційної реакції: норма; ознаки деградації; проблема. Для забезпечення контрольованої "істини" про момент деградації застосовано симуляційний сценарій з керованим дрейфом метрик.

Як об’єкт спостереження розглядається мобільний застосунок, що

– зчитує вимірювання глюкози з глюкометра (через канал передачі даних);

– передає показники на сервер або виконує локальне обчислення;

– отримує / обчислює прогноз (наприклад, горизонтом 30–60 хв);

– демонструє результат користувачу.

Експлуатаційні проблеми, релевантні для превентивного супроводження в цьому прикладі: погіршення зв’язку з пристроєм, зростання затримок отримання прогнозу, деградація якості даних (пропуски або аномальні значення), що зрештою призводить до збоїв синхронізації або погіршення *UX*.

В експерименті використано три метрики, які є реалістичними для мобільного застосунку й спостережувані в *production* без доступу до медичних "істинних значень".

*M*1. Затримка отримання прогнозу (*Prediction latency*)

Визначення: час від моменту отримання нового вимірювання глюкози застосунком до моменту отримання прогнозу (або завершення локального *inference*). Познака – L_t (мс).

Очікувана поведінка: у нормі – стаціонарний розподіл із незначними коливаннями; деградація проявляється як тренд росту L_t або зростання хвостів розподілу.

*M*2. Частка невдалих синхронізацій або передач даних (*Sync failure rate*)

Визначення: відношення кількості невдалих спроб зчитування або передачі даних за певний інтервал до загальної кількості спроб. Познака – $F_t \in [0,1]$.

Очікувана поведінка: у нормі близька до нуля; деградація – поступове збільшення через проблеми каналу збору первинних даних, мережі або серверних залежностей.

*M*3. Частка пропусків вимірювань (*Missing measurement rate*)

Визначення: частка очікуваних вимірювань, що не надійшли у встановлене вікно (наприклад, якщо сенсор або пристрій має очікувану періодичність). Познака – $M_t \in [0,1]$.

Очікувана поведінка: у нормі мала; деградація може проявлятися як стабільне зростання або серії "провалів".

Необхідно зауважити, що метрика *M*3 не є медичною оцінкою коректності, а суто експлуатаційним індикатором якості потоку даних.

Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 1 (188) *ISSN 3083-7650 (0135-1710)(print)*

З метою зниження шуму й формування ознак для компаратора застосовано агрегацію на ковзних вікнах. Базове вікно спостереження визначено: $W = 60$ хв, що є достатнім для виявлення деградаційних трендів, але не надто великим, щоб ускладнити визначення переходу до проблемного стану. Крок оновлення $\Delta = 5$ хв (оновлення агрегатів кожні 5 хв) забезпечує оперативність реакції для мобільного застосунку.

Наступним етапом методу є формування агрегатних ознак і трендів. Для кожного вікна W обчислюються:

– $\mathrm{p95}(L)$ та $median(L)$ – для фіксації "хвоста" затримок;

– $\bar{F}$ – середнє значення F_t на вікні;

– $\bar{M}$ – середнє значення M_t на вікні.

Для раннього виявлення деградації додаються трендові властивості на двох масштабах часу:

коротке вікно: $W_s = 30$ хв;

довге вікно: $W_l = 6$ год.

Для кожної агрегованої метрики X (наприклад, $p95$ *latency*) оцінюється проста тенденція: $\Delta X = X(W_s) - X(W_l)$ (різниця короткого й довгого середнього / квантиля), або знак / величина нахилу лінійної апроксимації на W_s (за потреби).

Як порівняльний базовий підхід в експерименті застосовано статичні пороги, типові для реактивного моніторингу.

Baseline-Alert-1 (*Latency*): спрацьовує, якщо $p95(L) > 1700$ мс на вікні W.

Baseline-Alert-2 (*Sync failures*): спрацьовує, якщо $\bar{F} > 0.05$ (тобто понад 5% невдалих синхронізацій) на W.

Baseline-Alert-3 (*Missing measurements*): спрацьовує, якщо $\bar{M} > 0.10$ (понад 10% пропусків) на W.

Важливо зауважити, що числові пороги наведено як реалістичні стартові значення для мобільного застосунку; вони можуть бути параметризовані під конкретну систему.

Для класу "ознаки деградації" упроваджено правила, що спрацьовують до перетину жорстких порогів, використовуючи тренди й комбіновані умови.

*Rule-D*1 (ранній дрейф затримок прогнозу). Стан "ознаки деградації", якщо виконується

$$p95(L) \in (1200, 2000] \text{ мс}$$

та одночасно

$$\Delta p95(L) > 300 \text{ мс},$$

де $\Delta p95(L)$ оцінюється як різниця між коротким і довгим вікном.

Інтерпретація: затримка ще не критична, але помітно "повзе" вгору.

*Rule-D*2 (комбінована деградація каналу даних). Стан "ознаки деградації", якщо

$$\bar{F} \in (0.02, 0.05] \text{ та } \bar{M} \in (0.05, 0.10].$$

Інтерпретація: ще немає масового фейлу, але канал збору / передачі стає нестабільним і з'являються пропуски.

*Rule-D*3 (аномальна нестабільність без порогового перевищення). Стан "ознаки деградації", якщо протягом останніх $K = 3$ оновлень (тобто ~15 хв) спостерігаються "спайки" $p95$ *latency*: $p95(L)$ перевищує власну медіану на W_l понад 1.5 раза у двох із трьох кроків.

Інтерпретація: нестабільність зростає, хоча середній рівень може ще бути прийнятним.

Класифікація експлуатаційної реакції реалізується ієрархічно:

"проблема", якщо спрацював будь-який *baseline-alert* (перетин жорсткого порогу);

"ознаки деградації", якщо *baseline-alert* не спрацював, але істинне хоча б одне правило *Rule-D*1–*D*3;

"норма", якщо не спрацювали ні *baseline-alert*, ні правила деградації.

Ця схема відповідає практичному процесу: "проблема" – це вже інцидент, "деградація" – підстава для превентивного тикета (або задачі на дослідження), "норма" – відсутність дій (продовжується експлуатація).

Для симуляційного дослідження задається момент початку деградації t_0 і керовані зміни:

– повільний дрейф $p95(L)$ вгору протягом 2–6 год;

– поступове збільшення F_t (наприклад, через погіршення зв'язку);

– зростання M_t (пропуски вимірювань).

ISSN 3083-7715 (online) *Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188)*
ISSN 3083-7650 (0135-1710)(print) *Management Information System and Devices. 2026. No. 1 (188)*

Оцінювання проводиться за такими показниками:

– *Lead time* – різниця між моментом, коли система вперше класифікувала "ознаки деградації", та моментом першого *baseline-alert* ("проблема");

– *False alarms* – кількість хибних спрацювань "ознаки деградації" на стабільних сегментах без деградації;

– *Churn* – кількість перемикань класу стану (стабільність класифікації).

Очікуваний результат експерименту: запропоновані правила деградації забезпечують позитивний *lead time* (раннє попередження) за прийнятного рівня *false alarms* та без надмірної нестабільності класифікації.

Для симуляційного експерименту згенеровано псевдоісторичні дані за два дні моніторингу з дискретністю вимірювання 1 хв. Приклад показників наведено на рис. 1. На цьому періоді штучно задано період деградації та параметри симуляції (інтенсивність деградації, базові рівні, а також алерти). На рис. 2 зображено графік станів (для наочності обрано фрагмент, де експлуатаційні стани явно змінюються). На рис. 3–5 продемонстровано динаміку агрегованих ознак навколо періоду деградації.

	latency_ms	sync_fail_rate	missing_rate
00:00:00	700.098412	0.008409	0.013660
00:01:00	724.074175	0.006765	0.008826
00:02:00	678.418033	0.007744	0.017609
00:03:00	629.276237	0.007835	0.014891
00:04:00	664.324433	0.008510	0.015530

Рис. 1. Генерація псевдоісторичних даних (фрагмент)

Рис. 2. Графік експлуатаційних станів (фрагмент)

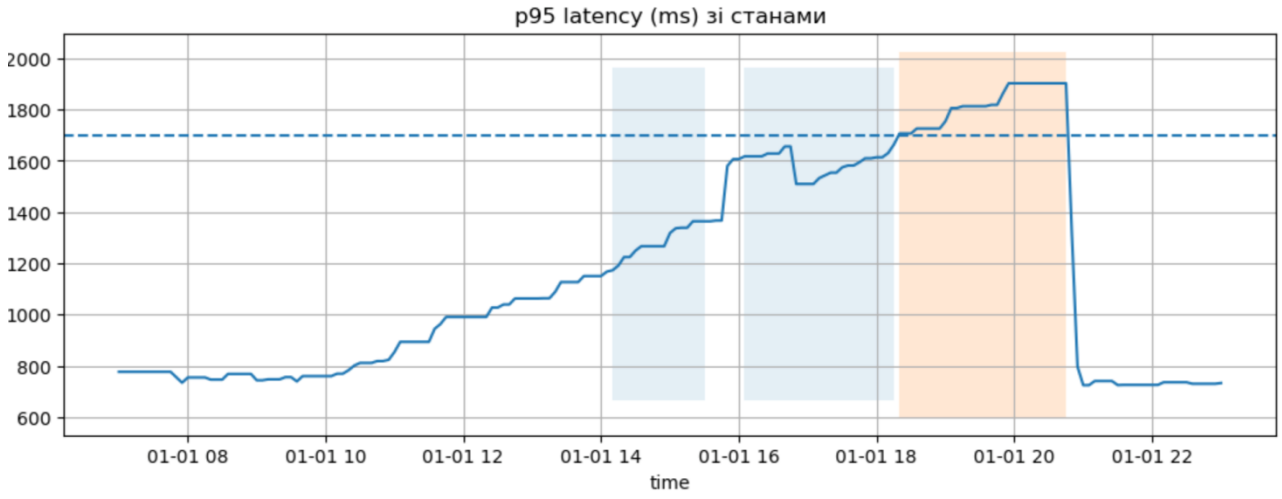

Рис. 3. Тренд затримки отримання прогнозу

Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 1 (188) *ISSN 3083-7650 (0135-1710)(print)*

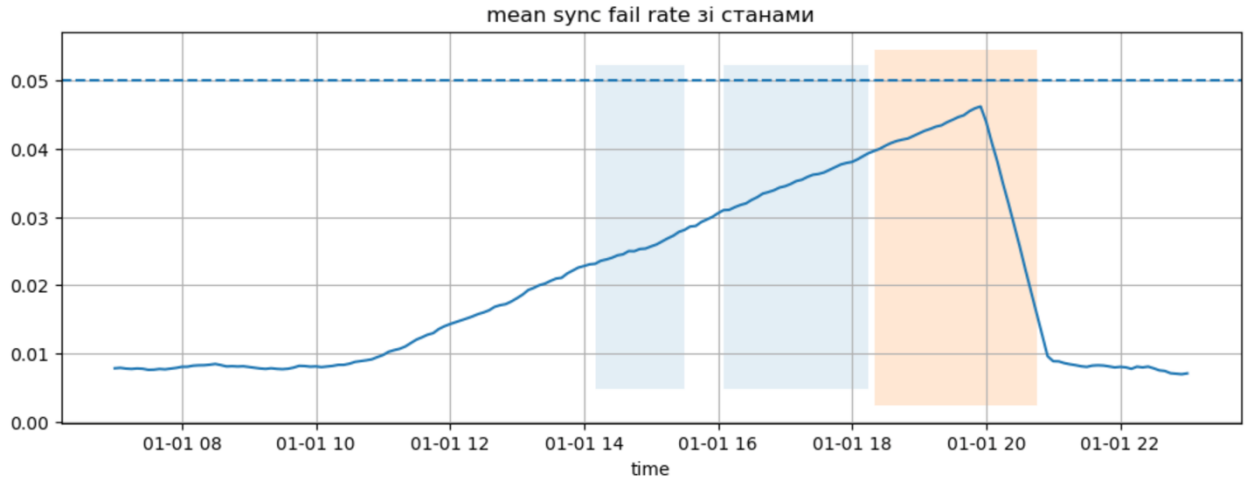

Рис. 4. Тренд частки невдалих синхронізацій або передач даних

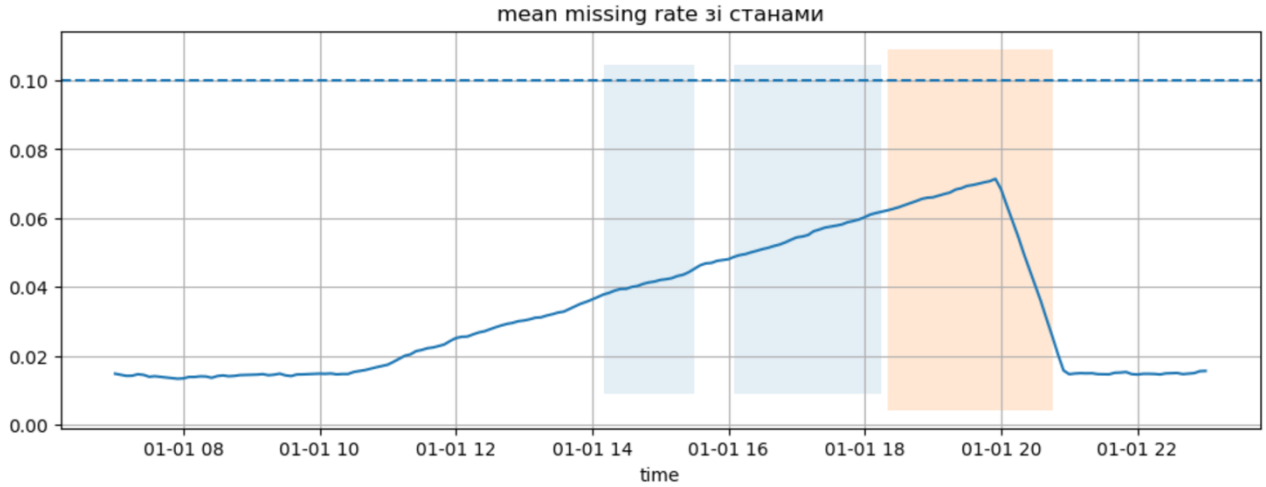

Рис. 5. Тренд частки пропусків вимірювань

Унаслідок експерименту:

– перша деградація (за визначеними правилами й ознаками) зафіксована після 14 год 10 хв від початку симуляції;

– початок проблемного стану – після 18 год 20 хв симуляції;

– *Lead time* (*min*): 250.0;

– *False alarms* (кількість): 0;

– *Churn* (кількість змін): 5.

Отже, отримане значення *Lead time* кількісно підтверджує здатність запропонованого методу виявляти деградаційні процеси до моменту досягнення критичних порогів окремих показників. Випередження в часі зумовлено використанням комбінованих і трендових ознак, що дають змогу фіксувати латентні зміни динаміки стану ще до їх явного прояву. Це свідчить про реалізацію саме превентивного, а не реактивного підходу до супроводження програмного забезпечення та підтверджує практичну ефективність компараторної моделі в задачах раннього реагування.

Обговорення

Запропонований метод превентивного супроводження програмних систем спрямований на раннє виявлення деградації. Цей метод надає формальну основу розв’язання задачі проактивного моніторингу, що актуально з огляду на сучасні тенденції переходу від реактивного до прогнозувального обслуговування програмних систем [5, 21]. Зазначимо, що в традиційних *DevOps*-підходах спостерігається нестача систематичного контролю якості програмного забезпечення на етапі експлуатації [20]. Це створює розрив між етапом тестування й реальними умовами експлуатації, який і покликаний заповнити запропонований підхід. Його новизна полягає

ISSN 3083-7715 (online) Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188)
ISSN 3083-7650 (0135-1710)(print) Management Information System and Devices. 2026. No. 1 (188)

в поєднанні інформації, отриманої під час тестування із результатами моніторингу в експлуатації (агреговані показники, тренди, аномалії) для формування цілісної моделі станів програмної системи. Отже, компараторна модель дає змогу класифікувати стани програмної системи на основі еквівалентності експлуатаційної реакції.

З погляду науки, використання методу компараторної ідентифікації є перспективним, оскільки він забезпечує повне математичне відображення об'єкта дослідження [4, 5]. Виявлені закономірності поведінки системи записуються у вигляді системи логічних рівнянь [21], що підвищує об'єктивність і точність моделі. На відміну від статистичних чи машинних методів, які потребують великого масиву історичних даних для навчання [16, 19], компараторний підхід спирається на експертно визначені критерії та предикати. Це набуває особливого значення за відсутності розгорнутої історії збоїв, де методи машинного навчання є менш ефективними. Замість побудови "чорної скриньки" прогнозування формуються явні логічні умови належності стану до того чи іншого класу. Отже, модель є пояснюваною: команда експлуатації може зрозуміти, чому стан належить до класу "проблема" чи "потребує уваги", оскільки це випливає з порушення конкретних метрик або умов. У цьому сенсі запропонований підхід відповідає сучасним вимогам щодо пояснюваності рішень [22].

Очікуване наукове значення запропонованого методу полягає в об'єднанні теорії компараторної ідентифікації, метрик якості програмного забезпечення та *DevOps*-практик для проактивного супроводження. Наукова новизна методу – це формалізація процесу раннього виявлення проблем: за допомогою використання апріорно визначених класів еквівалентності станів можна математично обґрунтувати момент, коли нормальний стан переходить у аномальний чи проблемний на основі порушення логічних умов.

Практична цінність превентивного моніторингу полягає в здатності виявляти деградацію на ранній стадії – ще до того, як вона переросте в повномасштабний збій. Це досягається внаслідок контролю трендів і відхилень від еталонних значень. Такий проактивний підхід відповідає загальній концепції прогнозувального обслуговування, яка покликана мінімізувати незаплановані простої та відмови [3]. По-друге, розбиття потенційно проблемних ситуацій на класи з огляду на критичність (низька, середня, висока важливість) і локалізацію (відповідальний домен чи команда) забезпечує пріоритизацію реакції на інциденти. Замість однакової реакції на всі тривожні сигнали команда отримує структуровану інформацію: де виникла проблема, наскільки вона серйозна і хто має втрутитися. Це дає змогу спрямувати зусилля лише туди, де є необхідність, уникаючи зайвих утручань у незначних випадках, що відповідає принципу зниження витрат унаслідок точкових інтервенцій [9]. Крім того, опис умов, за яких система перейшла в проблемний стан, надає команді експлуатації фактичний контекст для діагностики. Такий рівень обґрунтованості дій підвищує довіру до системи моніторингу й полегшує комунікацію між командами.

Запропонована компараторна модель фактично є розвитком *rule-based*-підходу, але на якісно новому рівні: замість ізольованих порогів вона використовує систему логічних умов для всього вектора ознак стану, що дає змогу фіксувати більш складні закономірності. Методи машинного навчання, зокрема алгоритми виявлення аномалій, визначають складні багатовимірні відхилення [11, 12]. Якщо порівнювати з *ML*-моделями, запропонований підхід не потребує великих масивів даних для навчання, може працювати відразу й виграє в інтерпретованості. З іншого боку, гібридні підходи намагаються виявляти нові закономірності й одночасно використовують експертні правила. У сучасних системах діагностики відмов, зокрема в промисловості, вже впроваджують багаторівневі архітектури з використанням нейронних мереж і нечіткої логіки для самодіагностики обладнання [23]. Порівняно з ними, запропонований підхід менш "автоматичний" – його адаптація до нових умов відбувається не способом машинного самонавчання, а внаслідок регулярного перегляду моделі на основі накопичених даних.

Критично оцінюючи запропонований підхід, можемо визначити кілька потенційних обмежень. По-перше, виродженість класифікації: якщо програмна система тривалий час працює безвідмовно й усі спостережувані стани еквівалентні "нормі", модель може не мати достатньої чутливості для виокремлення передвісників проблем. Частково цю проблему усуває долучення показників динаміки й аномалій. Однак налаштування порогів чутливості є

Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188) *ISSN 3083-7715 (online)*
Management Information System and Devices. 2026. No. 1 (188) *ISSN 3083-7650 (0135-1710)(print)*

складним завданням: надто вимогливі критерії призведуть до надлишкових сповіщень, тоді як надто м'які – до пропущення реальних проблем. По-друге, повнота еталонного класу "норма" залежить від якості тестування. Якщо сценарії тестів не охопили певні режими роботи або конфігурації системи, під час експлуатації можуть виникнути стани, які не вписуються у визначені класи. Це вказує на необхідність безперервного процесу валідації моделі: нові показники моніторингу й інциденти мають періодично аналізувати експерти для уточнення меж класів і умов. Підхід передбачає такий циклічний перегляд моделі, проте на практиці це вимагатиме налагодження відповідного процесу. По-третє, інтеграція запропонованого рішення в наявні інфраструктури моніторингу може спричинити складнощі. Необхідно забезпечити збір усіх потрібних метрик з експлуатаційного середовища, їх коректну агрегацію та вчасний аналіз.

Отже, метод компараторної ідентифікації в застосуванні до превентивного супроводження програмних систем є перспективним, оскільки поєднує строго формалізовану модель станів із практичною спрямованістю на раннє виявлення й класифікацію проблем. Очікується, що впровадження такого підходу сприятиме підвищенню надійності та безпеки програмних систем завдяки вчасній ідентифікації проблемних ситуацій і цілеспрямованому плануванню дій з їх усунення.

Висновки

У роботі розглянуто задачу превентивного супроводження програмного забезпечення з позицій формалізованого оцінювання його експлуатаційного стану в умовах обмеженої доступності історичних даних та відсутності апріорних оцінок проблемності можливих станів. Продемонстровано, що за таких умов доцільним є впровадження методів ідентифікації, орієнтованих на аналіз багатовимірних ознак і порівняння поточних станів із формалізованими еталонами.

Запропоновано поетапний підхід до формування множини ознак експлуатаційного стану програмного продукту, який базується на поєднанні метрик тестування, агрегованих експлуатаційних показників, трендових компонентів і ознак аномальної поведінки. Формалізація взаємозв'язків між ознаками у вигляді системи логічних рівнянь дає змогу аналізувати достатність і повноту сформованої множини ознак для задачі ідентифікації стану.

Продемонстровано можливість застосування методу компараторної ідентифікації для оцінювання експлуатаційного стану програмного забезпечення та класифікації експлуатаційної реакції програмного продукту з виокремленням нормального стану, стану з ознаками деградації та проблемного стану. Це створює основу для раннього виявлення деградаційних процесів і вчасного реагування на потенційні проблеми під час експлуатації.

Запропонований метод превентивного супроводження забезпечує автоматизоване формування інцидентів супроводження з визначенням типу проблеми, її локалізації, порушеної умови або тенденції до її порушення, рівня терміновості та відповідальної команди. Отже, досягається підвищення ефективності супроводження програмного забезпечення внаслідок переходу від реактивного до превентивного підходу.

Результати дослідження можуть бути використані для побудови систем моніторингу й супроводження програмних продуктів різного призначення. Подальші випробування доцільно спрямувати на адаптацію запропонованого підходу до різних класів програмних систем, а також на експериментальне вивчення його ефективності в умовах промислової експлуатації.

Конфлікт інтересів

Автори не мають конфлікту інтересів, зокрема фінансового, особистого, авторського чи будь-якого іншого характеру, який міг би вплинути на дослідження, а також на результати, опубліковані в цій статті.

Фінансування

Дослідження проводилося без фінансової підтримки.

Доступність даних

Дані будуть надані за обґрунтованим запитом.

Використання засобів штучного інтелекту

Автори застосовували технології штучного інтелекту (*GPT*-5.2, *DeepL*) для перевірки граматики й орфографії. Після використання цих інструментів автори перевірили й відредагували контент і несуть

повну відповідальність за зміст публікації. Автори підтверджують, що не застосовували технології штучного інтелекту для досягнення результатів, поданих у роботі.

References

1. Chupryna, A., Repikhov, V. (2025), "Reference model for preventive software maintenance", *Management Information Systems and Devices*, No. 4 (187), P. 254–277. DOI: https://doi.org/10.30837/0135-1710.2025.187.254

2. Thakur, P. S., Chouhan, S. S., Rathore, S. S., Parmar, J. (2026), "Systematic literature review on software code smell detection approaches", *Journal of Systems and Software*, Vol. 235, P. 112784. DOI: https://doi.org/10.1016/j.jss.2026.112784

3. Liu, P. S., Chin, J. F., Ab-Samat, H., Muhammad, N. A. (2025), "Simulation-based maintenance systems: A systematic review", *Journal of Simulation*, P. 1–39. DOI: https://doi.org/10.1080/17477778.2025.2579125

4. Cherednichenko, O., Vovk, M., Sharonova, N., Vorzhevitina, A. (2025), "Comparator-based identification of food edibility from natural language description", *Proceedings of the 9th International Conference on Computational Linguistics and Intelligent Systems (COLINS), CEUR Workshop Proceedings*, Vol. 4015, P. 185–199. DOI: https://doi.org/10.31110/COLINS/2025-3/014

5. Karataiev, O., Sitnikov, D., Sharonova, N. (2023), "A method for investigating links between discrete data features in knowledge bases in the form of predicate equations", *CEUR Workshop Proceedings*, Vol. 3387, P. 224–235. Available at: https://ceur-ws.org/Vol-3387/paper17.pdf

6. ISO/IEC/IEEE 14764:2022 (2022), Software engineering – Software life cycle processes – Maintenance, International Organization for Standardization.

7. Shahin, M., Babar, M. A., Zhu, L. (2017), "Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices", *IEEE Access*, Vol. 5, P. 3909–3943. DOI: https://doi.org/10.1109/ACCESS.2017.2685629

8. Giordano, G., Della Porta, A., Ferrucci, F., Palomba, F. (2025), "An evidence-based study on the relationship of software engineering practices on code smells in Python ML projects", *Proceedings of the 51st Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, P. 105–120. DOI: https://doi.org/10.1007/978-3-032-04207-1_8

9. Oliveira Carvalho, L. de, Biazotto, J. P., Feitosa, D., Nakagawa, E. Y. (2024), "Technical debt in continuous software engineering: An overview of the state of the art and future trends", *Proceedings of the 27th Ibero-American Conference on Software Engineering (CIbSE)*, P. 313–326.

10. Jiang, L., Xu, G. (2007), "Modeling and analysis of software aging and software failure", *Journal of Systems and Software*, Vol. 80, No. 4, P. 590–595. DOI: https://doi.org/10.1016/j.jss.2006.06.029

11. Du, M., Li, F., Zheng, G., Srikumar, V. (2017), "DeepLog: Anomaly detection and diagnosis from system logs through deep learning", *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, P. 1285–1298. DOI: https://doi.org/10.1145/3133956.3134015

12. Jia, T., Li, Y., Yang, Y., Huang, G., Wu, Z. (2022), "Augmenting log-based anomaly detection models to reduce false anomalies with human feedback", *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD)*, P. 3081–3089. DOI: https://doi.org/10.1145/3534678.3539106

13. Saputri, T. R. D., Lee, S.-W. (2020), "The application of machine learning in self-adaptive systems: A systematic literature review", *IEEE Access*, Vol. 8, P. 205948–205967. DOI: https://doi.org/10.1109/ACCESS.2020.3036037

14. Buse, R. P. L., Zimmermann, T. (2012), "Information needs for software development analytics", *Proceedings of the 34th International Conference on Software Engineering (ICSE)*, P. 987–996. DOI: https://doi.org/10.1109/ICSE.2012.6227122

15. Yalçıner, A., Dikici, A., Gökalp, E. (2024), "Data-driven software engineering: A systematic literature review", *Systems, Software and Services Process Improvement (EuroSPI 2024), Communications in Computer and Information Science*, Vol. 2179. DOI: https://doi.org/10.1007/978-3-031-71139-8_2

16. Serban, A., van der Blom, K., Hoos, H., Visser, J. (2024), "Software engineering practices for machine learning – Adoption, effects, and team assessment", *Journal of Systems and Software*, Vol. 209, P. 111907. DOI: https://doi.org/10.1016/j.jss.2023.111907

17. Codabux, Z., Williams, B. (2013), "Managing technical debt: An industrial case study", *Proceedings of the 4th International Workshop on Managing Technical Debt (MTD)*, P. 8–15. DOI: https://doi.org/10.1109/MTD.2013.6608672

18. Hall, T., Beecham, S., Bowes, D., Gray, D., Counsell, S. (2012), "A systematic literature review on fault prediction performance in software engineering", *IEEE Transactions on Software Engineering*, Vol. 38, No. 6, P. 1276–1304. DOI: https://doi.org/10.1109/TSE.2011.103

19. Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., Dennison, D. (2015), "Hidden technical debt in machine learning systems", *Proceedings of the 29th International Conference on Neural Information Processing Systems (NIPS)*, Vol. 2, P. 2503–2511.

20. Lwakatare, L. E., Kilamo, T., Karvonen, T., Sauvola, T., Heikkilä, V., Itkonen, J., Kuvaja, P., Mikkonen, T., Oivo, M., Lassenius, C. (2019), "DevOps in practice: A multiple case study of five companies", *Information and Software Technology*, Vol. 114, P. 217–230. DOI: https://doi.org/10.1016/j.infsof.2019.06.010

21. Sharonova, N., Doroshenko, A., Cherednichenko, O. (2018), "Issues of fact-based information analysis", *Proceedings of the 2nd International Conference on Computational Linguistics and Intelligent Systems (COLINS), CEUR Workshop Proceedings*, Vol. 2136, P. 11–19. Available at: http://ceur-ws.org/Vol-2136/10000011.pdf

22. Guidotti, R., Monreale, A., Ruggieri, S., Turini, F., Pedreschi, D., Giannotti, F. (2018), "A survey of methods for explaining black box models", arXiv:1802.01933. Available at: https://arxiv.org/abs/1802.01933

23. Lei, Z., Shi, J., Luo, Z., Cheng, M., Wan, J. (2024), "Intelligent manufacturing from the perspective of Industry 5.0: Application review and prospects", *IEEE Access*, Vol. 12, P. 167436–167451. DOI: https://doi.org/10.1109/ACCESS.2024.3496697

Received (Надійшла) 19.01.2026
Accepted for publication (Прийнята до друку) 25.02.2026
Publication date (Дата публікації) 12.03.2026

Відомості про авторів / About the Authors

Репіхов Вадим Миколайович – Харківський національний університет радіоелектроніки, аспірант кафедри програмної інженерії; Харків, Україна;

Vadym Repikhov – Kharkiv National University of Radio Electronics, Postgraduate Student, Department of Software Engineering; Kharkiv, Ukraine;

e-mail: vadym.repikhov@nure.ua

ORCID ID: https://orcid.org/0000-0002-1274-4205

Чуприна Анастасія Сергіївна – кандидат технічних наук, доцент, Харківський національний університет радіоелектроніки, доцент кафедри програмної інженерії; Харків, Україна;

Anastasiya Chupryna – PhD (Engineering Sciences), Associate Professor, Kharkiv National University of Radio Electronics, Associate Professor at the Department of Software Engineering; Kharkiv, Ukraine;

e-mail: anastasiya.chupryna@nure.ua

ORCID ID: https://orcid.org/0000-0003-0394-9900

METHOD OF PREVENTIVE SOFTWARE MAINTENANCE BASED ON A COMPARATOR MODEL

The subject of the study *is a method of preventive software maintenance based on a comparator model that provides formalized evaluation of the operational state and supports decision-making regarding the corresponding operational response.* ***The purpose of the study*** *is to improve the efficiency of preventive software maintenance through a formalized assessment of the operational state, enabling early detection of degradation processes, classification of the operational response, and automated generation of maintenance incidents under conditions of limited historical incident data.* ***The main objectives*** *are: to substantiate the requirements for the formalized assessment of the operational state of a software system under conditions of preventive maintenance and limited availability of historical data; to construct a comparator-based model for identifying operational states by formalizing the relationships between features in the form of a system of logical equations and verifying its functional determinacy; to develop a comparator-based method for identifying operational states based on the constructed model; and to perform experimental validation of the proposed method using operational data with comparison against traditional threshold-based alerting mechanisms.* ***Research methods.*** *Operational state identification is performed using the comparator-based identification method within a comparator model with predicate-based state representation. Relationships between features are formalized as a system of logical equations with verification of functional determinacy. Feature construction is based on a combination of testing metrics, aggregated operational indicators, trend characteristics, and anomaly-related features. Empirical validation is conducted on simulated operational data using sliding-window aggregation and comparison with traditional threshold-based alerting mechanisms.* ***Results.*** *The proposed method provides earlier detection of degradation processes compared to classical threshold-based alert mechanisms. The use of trend-based and combined features increases the stability of state*

ISSN 3083-7715 (online) *Автоматизовані системи управління та прилади автоматики. 2026. № 1 (188)*
ISSN 3083-7650 (0135-1710)(print) *Management Information System and Devices. 2026. No. 1 (188)*

classification and reduces false positives in the studied scenario. Based on the identified state class, a preventive maintenance model is implemented to automatically generate maintenance incidents specifying the type of problem, its localization, and priority level. ***Conclusions.*** *The obtained results advance formalized approaches to evaluating the operational state of software systems and demonstrate the applicability of the comparator model to preventive maintenance tasks in the absence of comprehensive historical incident data. The practical significance lies in enabling interpretable mechanisms for early degradation detection and automation of maintenance processes, including incident generation and support of operational response selection, without the need for large volumes of historical data.*

Keywords: *software; preventive maintenance; comparator model; comparator identification; operational state; software degradation; automated incident generation.*

Бібліографічні описи / Bibliographic descriptions

Репіхов В. М., Чуприна А. С. Метод превентивного супроводження програмного забезпечення на основі компараторної моделі. *Автоматизовані системи управління та прилади автоматики*. 2026. № 1 (188). С. 145–162. DOI: https://doi.org/10.30837/0135-1710.2026.188.145

Repikhov, V., Chupryna, A. (2026), "Method of preventive software maintenance based on a comparator model", *Management Information System and Devices*, No. 1 (188), P. 145–162. DOI: https://doi.org/10.30837/0135-1710.2026.188.145