УДК 629.735
DOI https://doi.org/10.32782/2663-5941/2025.1.2/18

Корнута В.А.
Івано-Франківський національний технічний університет нафти і газу

Меренько Б.І.
Івано-Франківський національний технічний університет нафти і газу

Катамай Ю.В.
Івано-Франківський національний технічний університет нафти і газу

Дмитрів І.Я.
Івано-Франківський національний технічний університет нафти і газу

Іванців Н.Т.
Івано-Франківський національний технічний університет нафти і газу

Дячук А.В.
Івано-Франківський національний технічний університет нафти і газу

МЕТОДИ УБЕЗПЕЧЕННЯ ВІД ПОМИЛОК ІНТЕЛЕКТУАЛЬНИХ АВТОМАТИЗОВАНИХ СИСТЕМ НАФТОГАЗОВОЇ ГАЛУЗІ

У статті йдеться про сучасні підходи до зменшення ризиків і помилок у функціонуванні інтелектуальних автоматизованих систем (ІАС), які застосовуються в нафтогазовій галузі. Основна увага приділяється використанню тестування, верифікації та аналізу покриття коду як ефективних інструментів для забезпечення надійності та відповідності системи галузевим стандартам безпеки. Верифікація дозволяє ідентифікувати помилки на етапі проектування та розробки програмного забезпечення, забезпечуючи відповідність алгоритмів, коду та логіки системи вимогам проекту. Формальні методи, такі як перевірка моделей і синтез правил керування, застосовуються для підтвердження коректності системи та оптимізації процесу її розробки. Забезпечується тестування всіх можливих сценаріїв функціонування системи, дозволяючи ідентифікувати нефункціональні або недокументовані частини коду. Також досліджено ефективність модульного та інтеграційного тестування, що дозволяє забезпечити безперебійну роботу системи в екстремальних умовах і запобігти аварійним ситуаціям. Приділено увагу тестуванню на основі властивостей, які вирішують відповідність нефункціональним вимогам та виявляють критичні перевірки, які можуть залишитися непоміченими іншими видами тестування. У роботі також описано підходи до моделювання поведінки автоматизованих систем із використанням формальних мов і методик моделей, що дозволяють перевірити недоліки на етапах планування. Розглянуто переваги застосування перевірки обмеженої моделі, яка дозволяє спростити пошук контрприкладів. Запропоновані рекомендації спрямовані на оптимізацію тестових сценаріїв, підвищення структурної закритості системи та використання інструментів автоматизації процесу перевірки. Такі підходи сприяють зниженню витрат на розробку та випробування, а також забезпечують відповідність ІАС сучасному стандарту.

Ключові слова: *нафтогазова галузь, інтелектуальні автоматизовані системи, тестування, верифікація, формальні методи.*

Постановка проблеми. Специфіка роботи в нафтогазовій сфері, а саме, складні умови експлуатації, значний обсяг оброблюваних даних і високий рівень автоматизації, вимагає використання спеціалізованих підходів до управління ризиками. Зменшення ймовірності виникнення помилок можливо завдяки застосуванню сучасних методів аналізу ризиків, превентивного обслуговування обладнання та алгоритмічних підходів виявлення та прогнозування відхилень у роботі системи. Ефективне управління ризиками в інтелектуальних автоматизованих системах (ІАС) забезпечує використання як традиційних, так і інноваційних підходів. Серед найпоширеніших методів можна виділити системний аналіз, моніторинг стану обладнання в реальному часі та машинне

навчання. Застосування цифрових двійників дозволяє моделювати сценарії розвитку подій та оцінювати наслідки можливих відхилень у роботі системи. Це, у свою чергу, допоможе розробляти ефективні стратегії реагування на ризики. Крім того, велике значення мають технології предиктивної аналітики, які дають можливість прогнозувати несправності до їх виникнення, що сприяє зниженню аварійності та посиленню продукту. У цій статті розглянуто основні методи мінімізації ризиків і помилок у роботі ІАС, що застосовуються в нафтогазовій галузі.

Аналіз останніх досліджень і публікацій. Зменшення ризиків і помилок у роботі ІАС має вирішальне значення для забезпечення безпеки та ефективності у складних промислових галузях, таких як нафтогазова промисловість. Одним із ключових етапів є тестування програмного забезпечення, а також вибір і оцінка тестових кейсів, які здатні виявити специфічні помилки, що можуть залишатися непоміченими при використанні стандартних підходів [1]. Верифікація та перевірка покриття коду є ефективними інструментами, що дозволяють оцінити коректність, надійність і повноту системи. Верифікація допомагає виявити помилки ще на етапі розробки алгоритмів, коду або логіки системи, тим самим мінімізуючи ризики серйозних збоїв у майбутньому. Наприклад, у нафтогазовій промисловості це може включати контроль критичних параметрів, таких як тиск, температура або витрати у трубопроводах, а також прогнозування несправностей обладнання на основі алгоритмів штучного інтелекту [2, 3]. Застосування верифікації дозволяє також перевірити відповідність галузевим стандартам безпеки, тестувати поведінку системи в екстремальних умовах та перевіряти функціональність у сценаріях, що максимально наближені до реальних умов експлуатації [4]. Наприклад, верифікація сценаріїв аварійного відключення обладнання дозволяє оцінити готовність системи до реагування на відмови сенсорів чи інші позаштатні ситуації [5]. Верифікація може проводитися за допомогою формальних та неформальних методів. Формальні методи, такі як перевірка моделі, дозволяють виконувати аналіз на рівні абстрактних моделей системи, використовуючи формальні мови [6]. Це забезпечує перевірку відповідності системи визначеним вимогам на ранніх етапах проєктування, знижуючи витрати на усунення помилок після впровадження. Формальні підходи, як-от перевірка моделей, дозволяють зосередитися на ключових аспектах системи, використовуючи алгебраїчні методи або алгоритми диспетчерського керування [7, 8]. Перевірка покриття гарантує, що всі можливі шляхи виконання та сценарії використання системи були належним чином протестовані. Це дозволяє виявляти недокументовані або нефункціонуючі частини коду, підвищуючи надійність і відповідність системи технічним вимогам. У галузі автоматизації важливо досягати повного покриття для перевірки коректної реакції системи на критичні помилки, такі як вихід із ладу сенсорів чи неправильні зовнішні сигнали [9].

Постановка завдання. Формулювання цілей статті – розробка, аналіз та удосконалення методів зменшення ризиків і помилок у роботі інтелектуальних автоматизованих систем, які використовуються в нафтогазовій галузі, шляхом впровадження ефективних підходів до тестування, верифікації та перевірки покриття коду з метою забезпечення їхньої надійності, функціональності та відповідності галузевим стандартам.

Виклад основного матеріалу. Верифікація інтелектуальних автоматизованих систем (ІАС) є критичним процесом, що забезпечує відповідність компонентів і підсистем проектним вимогам. У цьому контексті застосовуються як формальні методи, так і підходи тестування, що дозволяють максимально точно перевіряти системи. Одним із ключових аспектів є побудова моделі поведінки, яка складається з автоматичних переходів та операцій, що включають формальні та неформальні елементи. Цей підхід дозволяє використовувати формальні методи на етапі планування моделі, а також виконувати перевірку поточної моделі з урахуванням її драйверів, інтерфейсів, симуляцій тощо.

Зокрема, у розробці ІАС застосовуються інструменти та мови програмування, орієнтовані на автоматизацію. Наприклад, операція сканування вікна може бути реалізована за допомогою мови ROS (Robot Operating System). Приклад коду для виконання такої операції подано нижче:

```
operation:scan_box
deadline:10seconds
pre:start_scan_box    ->precondition
g:scan_req_state==initial&&->planningguard
scan_req_trigger==false&&
box_is_scanned==false
gr:true->runningguard
a:[scan_req_trigger<-true]   ->planningactions
ar:[] ->runningactions
```

```
post:complete_scan_box        ->postcondition
g:true ->planningguard
gr:scan_req_state==succeeded ->runningguard
a:[scan_req_state<-initial,  ->planningactions
scan_req_trigger<-false,
box_is_scanned<-true]
ar:[] ->runningactions
```

Цей приклад демонструє використання формального опису умов та дій для виконання конкретної операції в IAC. Такі моделі дозволяють детально описувати поведінку системи та забезпечують гнучкість у використанні як для симуляцій, так і для реального впровадження.

Формальні методи у верифікації інтелектуальних автоматизованих систем (IAC) дозволяють забезпечити відповідність системи визначеним специфікаціям і виявити можливі недоліки на ранніх етапах. Одними з найважливіших властивостей, що перевіряються, є властивості безпеки та властивості життєздатності.

- Властивості безпеки гарантують, що жодна небажана подія не станеться, наприклад, уникнення одночасного доступу до спільної зони або запобігання досягненню заборонених станів.
- Властивості життєздатності, навпаки, забезпечують, що система врешті-решт досягне бажаного цільового стану, наприклад, виконає задане завдання без зупинок через взаємоблокування [10].

Специфікації безпеки мають обмежену кількість контрприкладів, у той час як специфікації життєздатності – необмежену.

Окрім перевірки відповідності моделі специфікаціям, можливо застосовувати підхід синтезу, що дозволяє автоматично генерувати правила керування. Наприклад: теорія наглядового контролю [5] використовується для обчислення додаткових обмежень, що запобігають потраплянню системи до небажаних станів; техніка вилучення захисту [11] вдосконалює моделі планування, гарантуючи безпечність і відповідність високорівневим специфікаціям. Інший формальний підхід, перевірка моделі [12, 13], використовується для перевірки відповідності моделі заданим властивостям шляхом дослідження простору станів. Тимчасові властивості формулюються у термінах часової логіки, зокрема: лінійна часова логіка (LTL) включає оператори, такі як:

- o "наступний стан" (○),
- o "завжди" (□),
- o "зрештою" (◇),
- o "до" (U).

Наприклад, формула: □((scan_req_trigger ∧ ¬box_is_scanned) → ◇box_is_scanned) означає, що якщо сканування запущено, а коробку ще не просканировано, то врешті-решт коробку обов'язково буде просканировано.

Щоб скористатися перевагами методів розв'язання задачі задовільності пропозицій, модель і специфікації можна сформулювати як задачу з обмеженим розміром. До такого формулювання можна застосувати метод перевірки обмеженої моделі [14], де границя визначає, за скільки кроків від початкового стану шукати контрприклади. Подібно до ітераційного кодування переходів, скасовані специфікації лінійної часової логіки кодуються до межі, яка представлена поточною ітерацією. Якщо задача розв'язується, це означає, що специфікація була порушена, і отримане присвоєння може бути використане для реконструкції контрприкладу.

Слабкою стороною таких систем моделювання шляхом розділення поведінки планування та виконання є те, що не забезпечується загального способу уникнення тупикових ситуацій під час виконання. Хоча на моделі планування можна виконати вилучення захисту та перевірку моделі, перевірка працюючої моделі, а також зв'язку, інтерфейсів, драйверів тощо залежить саме від дій тестування.

Модульне тестування. Щоб мати можливість переконатися, що певний код поводитиметься так, як це було задумано розробником, звичайною практикою є тестування такого коду за допомогою написаного вручну модульних тестів [15]. Такі тести призначені для верифікації чи виявлення хибності коду для обраного та повністю визначеного вузла набору вхідних даних, а також перевірки раніше відомих проблемних наборів вхідних даних, які спричиняли помилки в минулому.

У методології тест-орієнтованої розробки модульні тести зазвичай пишуться перед фактичним кодом, що означає, що тести можуть бути невдалими, доки розробники не реалізують код правильно. Кожний модуль тестується незалежно в ізольованому середовищі, щоб переконатися у відсутності залежностей у коді.

Модульне тестування зосереджується виключно на аспектах, які є важливими для блоку, який досліджується. Цей підхід надає розробнику можливість вносити зміни у вихідний код, не впливаючи на інші модулі чи загальну функціональність програми. Наприклад, під час розробки IAC доцільно провести наступне модульне тестування: симуляції (симуляція отримує команди

та моделює поведінку ресурсу, як очікувалося); функції (функції та алгоритми, що керують ІАС, працюють як очікувалося); комунікація (при обробленні ресурсів їхні відповідні інтерфейси та апаратне забезпечення обмінюються інформацією, як очікувалося, використовуючи правильні типи повідомлень, обробка всіх полів повідомлень, обробка команд та вчасна відповідь тощо); драйвери (драйвери ресурсів здатні керувати обладнанням, як очікувалося).

Для модульного тестування поведінки симуляторів, інтерфейсів і драйверів під час розробки ІАС пропонується створити та використовувати прості фіктивні вузли. Ці вузли дозволяють тестувати конкретні команди та перевіряти очікувані відповіді від симуляторів, інтерфейсів і драйверів. Якщо їх поведінка відповідає очікуваним результатам для перевірених вхідних даних, тестування можна продовжувати. В іншому випадку треба змінювати компоненти, поки не буде досягнена бажана продуктивність.

Інтеграційне тестування. Коли кожен блок у системі буде перевірено в задовільній кількості разів, можна переходити до оцінки більших конфігурацій системи та взаємодії компонентів за допомогою інтеграційного тестування. Інтеграційне тестування виконується поетапно, коли компоненти поступово інтегруються, а потім тестуються як група. В ІАС інтеграційне тестування виконується під час віртуального введення в експлуатацію, де цифровий двійник (який по суті є віртуальним представленням відповідного фізичного об'єкта) забезпечує спільну основу для тестування зв'язку, контролерів, симуляторів і драйверів для всіх ресурсів. Після незалежної перевірки таких блоків їх взаємодія та сукупна функціональність перевіряються разом шляхом інтеграції контролера і моделі поведінки та тестування конкретних сценаріїв.

Нарешті, повна модель включається в тест, який містить завантажену поточну модель та автоматичні переходи. Такі інтеграційні тести потенційно виявлять, чи щось поводиться неправильно в конкретному випадку, визначеному користувачем.

Тестування на основі властивостей. На відміну від модульного тестування, яке перевіряє систему на окремі тестові випадки, та інтеграційного тестування, яке перевіряє взаємодію одиниць в окремих випадках сценарію, тестування на основі властивостей [16] перевіряє відповідності властивостей нефункціональним вимогам. Коли знайдено вхід, який порушує властивість, можна використати такі методи, як скорочення, щоб автоматично зменшити його до мінімального контрприкладу. На практиці такий контрприклад є дуже корисним, оскільки він прямо вказує на причину та яким чином властивість порушено, надаючи розробникам точну інформацію про те, як змінити програму.

Природа такого тестування зазвичай є випадковою [17] і без знання попередніх невдалих тестових прикладів або початкових значень таке тестування може пропустити конкретні невдалі крайні випадки. Крім того, тестування спостерігає лише за кінцевим набором виконань кінцевої програми, оскільки зазвичай дуже дорого або навіть неможливо перевірити код для всіх можливих вхідних значень. Таким чином, тестування на основі властивостей найкраще використовувати як доповнення до традиційного модульного тестування.

Щоб перевірити властивості ІАС, варто почати з визначення певних властивостей, які мають зберігатися під час виконання таких тестів, наприклад:

1. Якщо мета полягає в тому, щоб об'єкт сканувався, то врешті об'єкт має бути відсканована.

2. Якщо сканування тричі поспіль не вдається, сканування слід перервати, інакше сканування об'єкта відбувається повторно.

3. Якщо загалом сканування не вдається п'ять разів, сканування слід припинити, інакше об'єкт сканувати повторно.

Тестування покриття. Вимірювання структурної охоплюваності передбачає кількісну оцінку адекватності процесу тестування та надання розуміння повноти набору тестів. Цього можна досягти, визначивши набір показників покриття, які вказують на ступінь використання системи під час тестування. Наприклад, під час процесу тестування системи на заданий набір вимог можна відслідковувати та кількісно оцінювати частоту та ступінь застосування моделі поведінки. Ця оцінка може запропонувати цінну інформацію про те, як оптимізувати початковий набір тестів і покращити загальну охоплюваність. Використовуючи цей відгук для вдосконалення тестування, можна гарантувати, що система всебічно перевірена на відповідність необхідним стандартам.

Відомий критерій модифікованого покриття умов/рішень [18] не варто безпосередньо застосувати для оцінки охоплення моделей поведінки. Тому ми зосереджуємося на програмі виконання операцій і складаємо огляд різних станів виконання операцій (рис. 1):

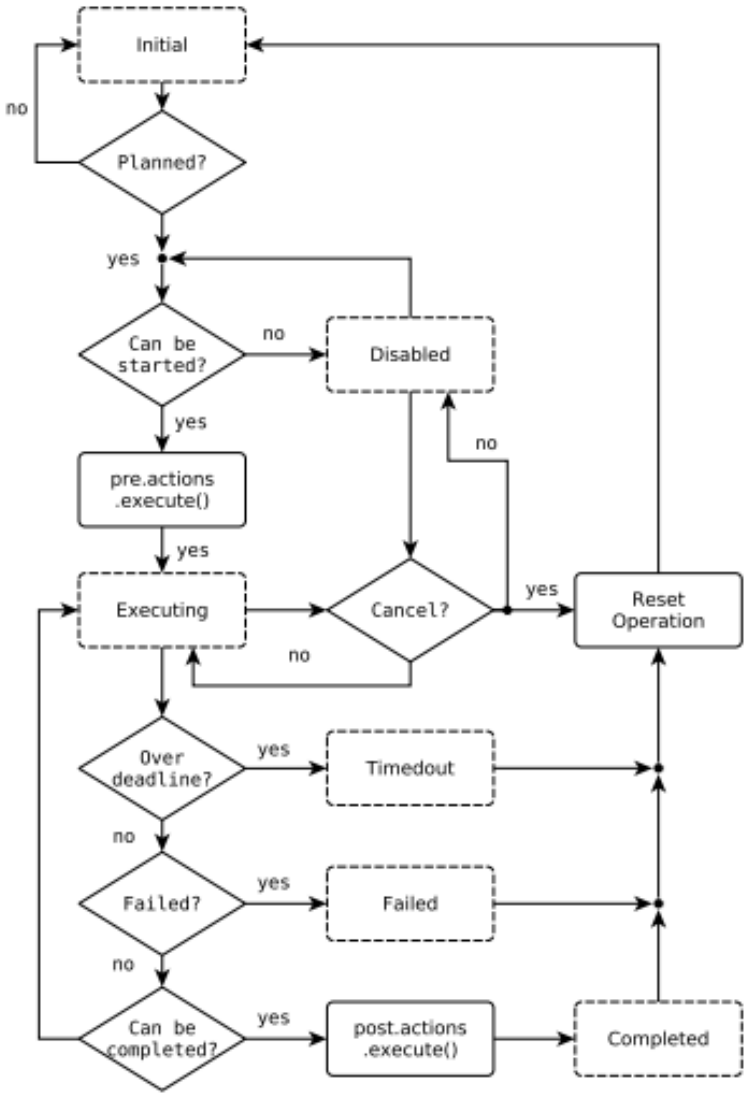

Рис. 1. Операції та стани під час виконання програми

- *Initial: операція не є наступною згідно плану.*
- *Disabled: операція є наступною в плані для виконання, але її захист попередніх умов ще не ввімкнено.*
- *Executing: захист попередніх умов увімкнено, дії передумови виконуються.*
- *Timedout: операція перебувала в стані виконання більше часу, ніж дозволяє її кінцевий термін*
- *Failed: операція не виконана через помилку.*
- *Completed: захист постумови ввімкнено, і дії післяумови виконуються. Операція успішно завершена.*

Таким чином, можна визначати критерії структурної здатності покриття IAC для набору тестів наступним чином. Кожна запланована операція в моделі поведінки принаймні один раз перебувала у стані Disabled, Executing, Timedout, Failed та Completed. Кожна операція була включена в план принаймні один раз. Кожен автоматичний перехід виконано принаймні один раз.

Виконання цих критеріїв для набору вимог гарантує, що модель поведінки була здійснена та може використовуватися як перевірка. Як і у випадку з критерієм модифікованого покриття умов/рішень, відповідність вимогам цього критерію не гарантує відсутності дефектів. Однак невиконання цього критерію вказує на те, що певні частини моделі не були достатньо використані, висвітлюючи потенційні проблемні області.

Покращення здатності до покриття є ітеративним процесом. Якщо критерії не відповідають, необхідно повернутися до моделі та визначити пропущені частини, а потім створити спеціальні тестові випадки для їх покриття. Крім того, тестувальнику можна дозволити впливати на вузли моделювання, щоб швидше охопити пропущені аспекти. Після додаткових тестів можна повторно

оцінити здатність до покриття та повторити процес, доки не буде досягнуто бажаного рівня покриття.

Покращення покриття є ітеративним процесом. Коли критерії не відповідають, необхідно повернутися до моделі та визначити пропущені частини, а потім створити спеціальні тестові випадки для їх покриття. Крім того, тестувальнику можна дозволити впливати на вузли симуляції, щоб швидше охопити пропущені аспекти. Після додаткових тестів можна повторно оцінити рівень покриття та повторити процес, доки бажаного рівня покриття не буде досягнуто.

Висновки. Результати дослідження підтверджують важливість інтеграції формальних методів і тестування для забезпечення надійності та функціональності ІАС. Формальні методи дозволяють виявити критичні помилки на етапі моделювання, тоді як тестування, зокрема модульне та інтеграційне, забезпечує перевірку коректності роботи системи в реальних умовах. Оцінка покриття допоможе оптимізувати тестові ресурси та підвищити ефективність тестування. Застосування запропонованих підходів сприяє мінімізації ризиків аварій та забезпечує безперебійність функціонування ІАС.

Список літератури:

1.Cai, Xia & Lyu, Michael. The effect of code coverage on fault detection under different testing profiles. *ACM Sigsoft Software Engineering Notes*. 2005. № 30(4). P.1-7. URL: https://doi.org/10.1145/1082983.1083288 (дата звернення: 28.01.2025).

2. Hametner R., Kormann B., Vogel-Heuser B., Winkler D., Zoitl A. Test case generation approach for industrial automation systems. *5th International Conference on Automation, Robotics and Applications*. 2011. P. 57–62.

3. Winkler D., Hametner R., Östreicher T., Biffl S. A framework for automated testing of automation systems. *IEEE 15th Conference on Emerging Technologies and Factory Automation (ETFA 2010)*. 2010. P. 1–4.

4. Clarke E. M., Emerson E. A., Sifakis J. Model checking: Algorithmic verification and debugging. *Communications of the ACM*. 2000. № 52 (11). P.74–84, 200.

5. Ramadge P. J. Wonham W. M. Supervisory control of a class of discrete event processes. *SIAM J. Control Optim*. 1987. vol. 25. no. 1. P. 206–230.

6. Orso A., Rothermel G. Software testing: A research travelogue (2000–2014). *Future of Software Engineering Proceedings*. 2014. P. 117–132.

7. Buzhinsky I., Pang C., Vyatkin V. Formal modeling of testing software for cyber-physical automation systems. *5 IEEE Trustcom/BigDataSE/ISPA, IEEE*. 2015. Vol. 3. P. 301–305.

8. Zander J., Schieferdecker I., Mosterman P. J. Model-based testing for embedded systems. CRC press, 2017. 122 p.

9. Rival X., Yi K. Introduction to static analysis: an abstract interpretation perspective. Mit Press, 2020. 320 p.

10. Sistla A. Safety, liveness and fairness in temporal logic. *Formal Aspects of Computing*. 1999. vol. 6. URL: https://surl.li/xdvnkm (дата звернення: 28.01.2025).

11. Dahl M., Bengtsson K., Fabian M., Falkman P. Guard extraction for modeling and control of a collaborative assembly station. *IFACPapers On Line*. 2020.Vol. 53. No. 4. P. 223–228.

12. Grumberg O., Clarke E., Peled D. Model checking. MIT Press, 1999. 314 p.

13. Pnueli A. The temporal logic of programs. *18th Annual Symposium on Foundations of Computer Science*. 1977. P. 46–57.

14. Biere A., Cimatti A., Clarke E., Zhu Y. Symbolic model checking without bdds. *International conference on tools and algorithms for the construction and analysis of systems*. 1999. P. 193–207.

15. Runeson P. A survey of unit testing practices. *IEEE software*. 2006. vol. 23. no. 4. P. 22–29.

16. Fink G., Bishop M. Property-based testing: A new approach to testing for assurance. *SIGSOFT Softw. Eng. Notes.* 1997. vol. 22. no. 4. P. 74–80.

17. Claessen K., Hughes J. Quickcheck: A lightweight tool for random testing of haskell programs. *SIGPLAN Not*. 2000. Vol. 35. No. 9. P. 268–279.

18. Hayhurst K., Veerhusen D. A practical approach to modified condition/decision coverage. *Digital Avionics Systems Conference (Cat. No.01CH37219)*: 20th DASC. 2001. Vol. 1. 1B2/1–1B2/10.

Kornuta V.A., Merenko B.I., Katamay Yu.V., Dmytriv I.Ya., Ivantsiv N.T., Dyachuk A.V. METHODS OF PROTECTING INTELLIGENT AUTOMATED SYSTEMS FROM ERRORS IN THE OIL AND GAS INDUSTRY

The article discusses modern approaches to reducing risks and errors in the operation of intelligent automated systems (IAS) used in the oil and gas industry. The main focus is on the use of testing, verification and code coverage analysis as effective tools to ensure the reliability and compliance of the system with industry safety standards. Verification allows identifying errors at the software design and development stage,

ensuring that the system's algorithms, code and logic meet the design requirements. Formal methods, such as model checking and control rule synthesis, are used to confirm the correctness of the system and optimise the development process. Testing of all possible scenarios of system operation is ensured, allowing identification of non-functional or undocumented parts of the code. The effectiveness of unit and integration testing is also investigated, which allows to ensure uninterrupted operation of the system in extreme conditions and prevent emergencies. Attention is paid to property-based testing, which addresses compliance with non-functional requirements and identifies critical checks that may go unnoticed by other types of testing. The paper also describes approaches to modelling the behaviour of automated systems using formal languages and modeling techniques that allow for checking defects at the planning stages. The advantages of using limited model checking, which simplifies the search for counterexamples, are considered. The proposed recommendations are aimed at optimising test scenarios, increasing the structural closure of the system and using tools to automate the verification process. Such approaches help to reduce development and testing costs and ensure that IAS complies with the modern standard.

Key words: *oil and gas industry, intelligent automated systems, testing, verification, formal methods.*