DOI 10.34185/1562-9945-2-157-2025-05
УДК: 629.735

В.А. Корнута, Ю.В. Катамай, Б.І. Меренько,
І.Я. Дмитрів, Н.Т. Іванців, О.В.Корнута, А.В. Дячук

МЕТОДИ ЗМЕНШЕННЯ РИЗИКІВ І ПОМИЛОК В РОБОТІ ІНТЕЛЕКТУАЛЬНИХ АВТОМАТИЗОВАНИХ СИСТЕМ НАФТОГАЗОВОЇ ГАЛУЗІ

Анотація. Сучасні інтелектуальні автоматизовані системи (ІАС) забезпечують високий рівень автоматизації та покращують ефективність виробничих процесів в нафтогазовій галузі. Водночас функціонування таких систем супроводжується ризиками та ймовірністю помилок, що можуть призводити до фінансових втрат та аварійних ситуацій. У статті розглянуто основні методи зменшення ризиків та помилок у роботі ІАС нафтогазової галузі, зокрема, вдосконалені підходи до верифікації та тестування. Досліджено застосування формальних та неформальних методів верифікації, а також модульного, інтеграційного та тестування на основі властивостей у ІАС. Визначено підходи для вимірювання структурного покриття та оптимізації процесу тестування. Отримані результати спрямовані на зменшення ймовірності помилок та забезпечення відповідності системи галузевим стандартам.
Ключові слова: інтелектуальні автоматизовані системи, нафтогазова галузь, верифікація, перевірка покриття, тестування.

Постановка проблеми. Сучасні інтелектуальні автоматизовані системи (ІАС) відіграють важливу роль у розвитку нафтогазової галузі, забезпечуючи ефективність та безпеку ключових виробничих процесів. Завдяки їх використанню досягається підвищення продуктивності, оптимізація видобутку та транспортування вуглеводнів, а також мінімізація впливу на довкілля. Водночас функціонування таких систем супроводжується певними ризиками та ймовірністю виникнення помилок, які можуть призводити до значних фінансових втрат, порушення безпеки та аварійних ситуацій.

Особливості роботи в нафтогазовій галузі, такі як екстремальні умови експлуатації, великий обсяг даних та високий рівень автоматизації, потребують застосування спеціалізованих підходів до управління ризиками. Ефективне зменшення ймовірності помилок можливе завдяки впровадженню сучасних методів аналізу ризиків, превентивного обслуговування обладнання, а також алгоритмічних підходів до виявлення та прогнозування відхилень у роботі систем. У цій статті розглянуто основні методи зменшення ризиків і помилок у роботі ІАС, що застосовуються в нафтогазовій галузі.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

Аналіз останніх досліджень і публікацій. Ключову роль у зменшенні ризиків і помилок в інтелектуальних автоматизованих системах (IAC), особливо в складних галузях, таких як нафтогазова **промисловість** відіграє тестування та вибір і оцінка тестових кейсів. Ефективний набір тестів повинен виявляти програмні помилки , які не можуть бути виявлені іншими тестовими кейсами [1]. Для покращення розподілу тестових ресурсів можна запропонувати використовувати верифікацію та перевірку покриття коду як показник ефективності та повноти тестування. Ці методи забезпечують перевірку коректності, надійності та повноти системи ще до її впровадження або під час її роботи. Так, верифікація допомагає виявленню помилок на ранніх етапах **(**дозволяє виявити помилки в алгоритмах, коді або логіці системи ще до її впровадження, зменшуючи ризики дороговартісних збоїв), перевіряє відповідність галузевим нормам і стандартам безпеки , допомагає перевірити, як система працює в екстремальних умовах або за несприятливих обставин, використовує скрипти для перевірки функціоналу і продуктивності системи в різних сценаріях. Зокрема, в нафтогазовій галузі використовується верифікація програмного забезпечення для контролю тиску, температури чи витрат у трубопроводах, перевірка алгоритмів ШІ, для прогнозування несправностей обладнання тощо.

Покриття забезпечує, щоб усі можливі шляхи виконання, сценарії використання або частини системи були протестовані. А саме, виявляє частини коду, логіки або функціоналу, які можуть залишитися невипробуваними без повного покриття, гарантує, що немає недокументованих або нефункціонуючих частин системи та допомагає забезпечити коректну реакцію системи на помилки, наприклад, аварійне відключення обладнання або відмову сенсорів, дає змогу ідентифікувати і покращити слабкі місця системи.

Верифікація - це сімейство методів, метою яких є переконатися, що система задовольняє певні вимоги користувача. Процес верифікації є вирішальним елементом у розробці керуючого програмного забезпечення та відіграє ключову роль у забезпеченні надійності, функціональності та продуктивності систем автоматизації [2, 3]. Верифікація може бути формальною (містить такі методи, як перевірка моделі [4] і теорію диспетчерського керування [5]), або неформальною (містить такі методи, як тестування [6]).

Формальна верифікація виконується на рівні моделі, а не на рівні системи, що означає, що модель системи має бути побудована. Модель системи створюється за допомогою формальної мови, і це зазвичай передбачає певний рівень абстракції, оскільки побудувати модель, яка повністю повторює поведінку системи, яку вона моделює, зазвичай неможливо. Замість цього практичним компромісом є моделювання та офіційна перевірка певних аспектів системи, а також застосування тестування для перевірки більших частин або повних систем [7].

На відміну від формальних методів, таких як перевірка моделі, тестування рідко буває вичерпним, оскільки зазвичай досить дорого охопити розумну кількість простору для введення. Це означає, що неможливо довести відсутність помилок, але при цьому можна збільшити довіру до системи. За допомогою відповідних методів тестування, що

супроводжуються критеріями покриття, тестування можна масштабувати та використовувати для складних промислових систем [8].

Крім того, тестування може бути застосоване до програми в умовах, в яких вона передбачається виконувати, наприклад, на цільовому обладнанні або з цільовою операційною системою та драйверами [9].

Мета дослідження - вдосконаленні підходів до верифікації та тестування інтелектуальних автоматизованих систем, що використовуються в нафтогазовій галузі для зниження ризиків, підвищення надійності та забезпечення відповідності сучасним стандартам безпеки.

Викладення основного матеріалу дослідження. Верифікація інтелектуальних автоматизованих систем - це процес, який гарантує, що конкретні компоненти або підсистеми відповідають своїм проектним вимогам. Для верифікації IAC можна застосовувати як формальні методи, так і тестування. Модель поведінки побудована з автоматичних переходів і операцій, що складаються з формальної та не-формальної частин. Такий вибір моделі дозволяє використовувати формальні методи до планування моделі, а також перевірити поточну модель, а саме, її драйвери, інтерфейси, симуляції тощо. Приклад операції сканування вікна з використанням С-подібної мови ROS (Robot Operating System) наведено нижче:

```
    operation: scan_box
deadline: 10 seconds
    pre: start_scan_box                  -> precondition
g:  scan_req_state == initial &&   -> planning guard
    scan_req_trigger == false &&
box_is_scanned == false
    gr: true                             -> running guard
a:  [scan_req_trigger <- true]           -> planning actions
ar: []                                   -> running actions
    post: complete_scan_box              -> postcondition
g:  true                                 -> planning guard
    gr: scan_req_state == succeeded      -> running guard
a:  [scan_req_state <- initial,          -> planning actions
    scan_req_trigger <- false,
box_is_scanned <- true]
    ar: []                               -> running actions
```

Формальні методи. Одними з найважливіших груп властивостей, які варто перевіряти, є властивості безпеки (властивості «життєздатності»). При дотриманні властивостей безпеки ніколи не повинно статися чогось поганого, наприклад, одночасного доступу до спільної зони, неправильного порядку складання або досягнення забороненого стану. З іншого боку, властивості життєздатності декларують, що врешті-решт станеться щось хороше, наприклад, завжди можна досягти бажаного цільового стану [10], не застрягнувши в динамічному взаємоблокуванні. Специфікації безпеки

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

мають скінчену кількість контрприкладів, тоді як специфікації життєздатності – нескінченну.

Замість перевірки того, що модель відповідає специфікації, можна застосувати підхід синтезу для автоматичного обчислення правил керування. Для розробки IAC можна використовувати теорію наглядового контролю [5] для прямого обчислення додаткових обмежень для системи з використанням техніки вилучення захисту [11], яка гарантує, що згенерований план не буде у жодному з небажаних станів. Така техніка використовується для вдосконалення моделі планування на основі специфікацій високого рівня.

Іншим формальним методом, який можна застосувати до моделі планування, є перевірка моделі, яка використовується для відповіді на таке запитання: чи задовольняє модель системи заданим властивостям? У перевірці моделі [12] тимчасові властивості перевіряються шляхом дослідження простору станів з використанням набору початкових станів і переходів. Тимчасові властивості виражаються за допомогою розширень логіки висловлювань, наприклад лінійної часової логіки [13]. Лінійна часова логіка містить тимчасові оператори, такі як «наступний стан» (O), «завжди» (□), "зрештою" (◊) та «до» (U). Наприклад, формула □(x→O y) виражає, що для всіх досяжних станів завжди вірно, що якщо x зберігається в поточному стані, у буде зберігатися в наступному стані.

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. Розглянемо, наприклад, властивість, яку можна перевірити на моделі планування роботи робота: завжди, після того, як сканер отримує команду на сканування, а коробка не сканується, коробку все одно слід сканувати. У часовій логіці це записується так:

$$((scan\_req\_trigger \wedge \neg box\_is\_scanned) \rightarrow \Diamond box\_is\_scanned)$$

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 ІАС 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

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

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 IAC, 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 відскановано.

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):

– 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.

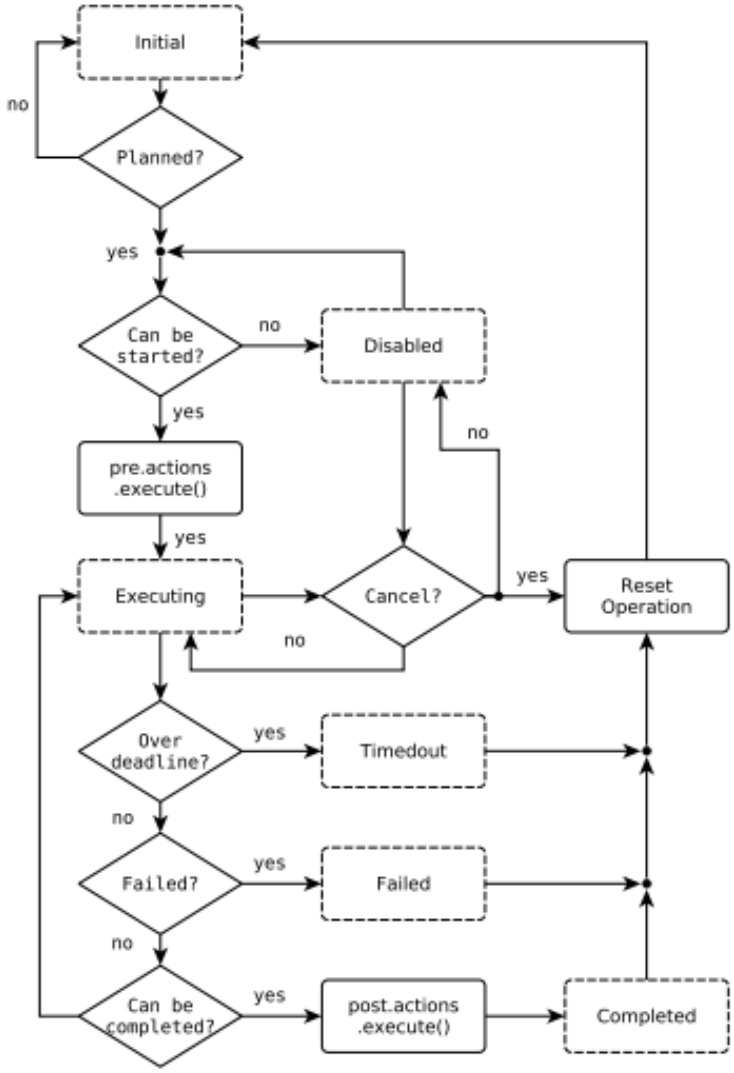

Рисунок 1 - Операції та стани під час виконання програми

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.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

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 структурної охоплюваності допоможе оптимізувати тестові ресурси та підвищити ефективність тестування. Застосування запропонованих підходів сприяє мінімізації ризиків аварій та забезпечує безперебійність функціонування ІАС.

ЛІТЕРАТУРА

1. Cai, Xia & Lyu, Michael. (2005). The effect of code coverage on fault detection under different testing profiles. ACM Sigsoft Software Engineering Notes. 30. 10.1145/1082983.1083288.

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

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

4. E. M. Clarke, E. A. Emerson, and J. Sifakis, “Model checking: Algorithmic verification and debugging,” Communications of the ACM, vol. 52, no. 11, pp. 74–84, 200.

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

6. A. Orso and G. Rothermel, “Software testing: A research travelogue (2000–2014),” in Future of Software Engineering Proceedings, 2014, pp. 117–132.

7. I. Buzhinsky, C. Pang, and V. Vyatkin, “Formal modeling of testing software for cyber-physical automation systems,” in 2015 IEEE Trustcom/BigDataSE/ISPA, IEEE, vol. 3, 2015, pp. 301–30.

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

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

10. A. Sistla, “Safety, liveness and fairness in temporal logic,” Formal Aspects of Computing, vol. 6, Sep. 1999.

11. M. Dahl, K. Bengtsson, M. Fabian, and P. Falkman, “Guard extraction for modeling and control of a collaborative assembly station,” IFACPapers On Line, vol. 53, no. 4, pp. 223–

228, 2020, 15th IFAC Workshop on Discrete Event Systems WODES 2020 — Rio de Janeiro, Brazil, 11- 13 November 2020, issn: 2405-8963.
12. O. Grumberg, E. Clarke, and D. Peled, Model checking, 1999.
13. A. Pnueli, "The temporal logic of programs," in 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), IEEE, 1977, pp. 46–57.
14. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without bdds," in International conference on tools and algorithms for the construction and analysis of systems, Springer, 1999, pp. 193–207.
15. P. Runeson, "A survey of unit testing practices," IEEE software, vol. 23, no. 4, pp. 22–29, 2006.
16. G. Fink and M. Bishop, "Property-based testing: A new approach to testing for assurance," SIGSOFT Softw. Eng. Notes, vol. 22, no. 4, pp. 74–80, Jul. 1997, issn: 0163-5948.
17. K. Claessen and J. Hughes, "Quickcheck: A lightweight tool for random testing of haskell programs," SIGPLAN Not., vol. 35, no. 9, pp. 268– 279, Sep. 2000, issn: 0362-1340.
18. Hayhurst and D. Veerhusen, "A practical approach to modified condition/decision coverage," in 20th DASC. 20th Digital Avionics Systems Conference (Cat. No.01CH37219), vol. 1, 2001, 1B2/1–1B2/10 vol.1.

REFERENCES

1. Cai, Xia & Lyu, Michael. (2005). The effect of code coverage on fault detection under different testing profiles. ACM Sigsoft Software Engineering Notes. 30. 10.1145/1082983.1083288.
2. R. Hametner, B. Kormann, B. Vogel-Heuser, D. Winkler, and A. Zoitl, "Test case generation approach for industrial automation systems," in The 5th International Conference on Automation, Robotics and Applications, 2011, pp. 57–62
3. D. Winkler, R. Hametner, T. Östreicher, and S. Biffl, "A framework for automated testing of automation systems," in 2010 IEEE 15th Conference on Emerging Technologies and Factory Automation (ETFA 2010), 2010, pp. 1–4.
4. E. M. Clarke, E. A. Emerson, and J. Sifakis, "Model checking: Algorithmic verification and debugging," Communications of the ACM, vol. 52, no. 11, pp. 74–84, 200.
5. P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," SIAM J. Control Optim., vol. 25, no. 1, pp. 206–230, 1987.
6. A. Orso and G. Rothermel, "Software testing: A research travelogue (2000–2014)," in Future of Software Engineering Proceedings, 2014, pp. 117–132.
7. I. Buzhinsky, C. Pang, and V. Vyatkin, "Formal modeling of testing software for cyber-physical automation systems," in 2015 IEEE Trustcom/BigDataSE/ISPA, IEEE, vol. 3, 2015, pp. 301–30.
8. J. Zander, I. Schieferdecker, and P. J. Mosterman, Model-based testing for embedded systems. CRC press, 2017.
9. X. Rival and K. Yi, Introduction to static analysis: an abstract interpretation perspective. Mit Press, 2020.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)

10. A. Sistla, “Safety, liveness and fairness in temporal logic,” Formal Aspects of Computing, vol. 6, Sep. 1999.
11. M. Dahl, K. Bengtsson, M. Fabian, and P. Falkman, “Guard extraction for modeling and control of a collaborative assembly station,” IFACPapers On Line, vol. 53, no. 4, pp. 223–228, 2020, 15th IFAC Workshop on Discrete Event Systems WODES 2020 — Rio de Janeiro, Brazil, 11- 13 November 2020, issn: 2405-8963.
12. O. Grumberg, E. Clarke, and D. Peled, Model checking, 1999.
13. A. Pnueli, “The temporal logic of programs,” in 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), IEEE, 1977, pp. 46–57.
14. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, “Symbolic model checking without bdds,” in International conference on tools and algorithms for the construction and analysis of systems, Springer, 1999, pp. 193–207.
15. P. Runeson, “A survey of unit testing practices,” IEEE software, vol. 23, no. 4, pp. 22–29, 2006.
16. G. Fink and M. Bishop, “Property-based testing: A new approach to testing for assurance,” SIGSOFT Softw. Eng. Notes, vol. 22, no. 4, pp. 74–80, Jul. 1997, issn: 0163-5948.
17. K. Claessen and J. Hughes, “Quickcheck: A lightweight tool for random testing of haskell programs,” SIGPLAN Not., vol. 35, no. 9, pp. 268– 279, Sep. 2000, issn: 0362-1340.
18. Hayhurst and D. Veerhusen, “A practical approach to modified condition/decision coverage,” in 20th DASC. 20th Digital Avionics Systems Conference (Cat. No.01CH37219), vol. 1, 2001, 1B2/1–1B2/10 vol.1.

Received 10.03.2025.
Accepted 13.03.2025.

Methods for reducing risks and errors in the operation of oil and gas industry intellectual automated systems

Modern intelligent automated systems (IAS) provide a high level of automation and improve the efficiency of production processes in the oil and gas industry. At the same time, the operation of such systems is accompanied by risks and the probability of errors, which can lead to financial losses and emergencies. The article considers the main methods for reducing risks and errors in the operation of IAS in the oil and gas industry, in particular, improving approaches to verification and testing. The application of formal and informal verification methods and modular, integration, and property-based testing in specific IAS systems is investigated. Approaches for measuring structural coverage and optimizing the testing process are identified. The results obtained are aimed at reducing the probability of errors and ensuring the system's compliance with industry standards. The study's results confirm the importance of integrating formal methods and testing to ensure the reliability and functionality of the IAS. Formal methods allow you to detect critical errors at the modeling stage, while testing, in particular modular and integration, ensures the correctness of the system's operation in real conditions. The application of the proposed approaches helps minimize the risks of accidents and ensures the smooth functioning of the IAS.

Keywords: intelligent automated systems, oil and gas industry, verification, coverage verification, testing

Корнута Володимир - кандидат технічних наук, доцент кафедри інженерії програмного забезпечення, Івано-Франківський національний технічний університету нафти і газу.

Катамай Юлія - аспірантка, Івано-Франківський національний технічний університет нафти і газу.

Меренько Богдан - аспірант, Івано-Франківський національний технічний університет нафти і газу.

Дмитрів Ігор - аспірант, Івано-Франківський національний технічний університет нафти і газу.

Іванців Назарій - аспірант, Івано-Франківський національний технічний університет нафти і газу.

Корнута Олена - кандидат технічних наук, доцент кафедри технічної механіки, інженерної та комп'ютерної графіки, Івано-Франківський національний технічний університету нафти і газу.

Дячук Андрій - аспірант, Івано-Франківський національний технічний університет нафти і газу.

Kornuta Volodymyr - Ph.D, Associate Professor of Ivano-Frankivsk National Technical University of Oil and Gas.

Katamai Yuliia - PhD student, Ivano-Frankivsk National Technical University of Oil and Gas.

Merenko Bogdan - PhD student, Ivano-Frankivsk National Technical University of Oil and Gas.

Dmytriv Ihor - PhD student, Ivano-Frankivsk National Technical University of Oil and Gas.

Ivantsiv Nazar - PhD student, Ivano-Frankivsk National Technical University of Oil and Gas.

Kornuta Olena - Ph.D, Associate Professor of Ivano-Frankivsk National Technical University of Oil and Gas.

Diachuk Andriy - PhD student, Ivano-Frankivsk National Technical University of Oil and Gas.

ISSN 1562-9945 (Print)
ISSN 2707-7977 (Online)