ДОДАТОК А

Графічний матеріал кваліфікаційної роботи

Харківський національний університет радіоелектроніки
Кафедра ЕОМ

Кваліфікаційна робота

Другий (магістерський) рівень

Методи автоматизації тестування та верифікації системного програмного забезпечення

Автор:
Василенко Д.В.,
студ. гр. СПм-23-4

Керівник:
Сорокін А.Р.,
доц. каф. ЕОМ

Мета і задачі роботи

Мета кваліфікаційної роботи:

- розробка та вдосконалення методів автоматизації тестування і верифікації системного програмного забезпечення з використанням сучасних інструментів і практик програмної інженерії.

Об’єкт дослідження:

- процес тестування та верифікації системного програмного забезпечення.

Предмет дослідження:

- методи автоматизації, інструменти та підходи до підвищення ефективності тестування та верифікації СПЗ.

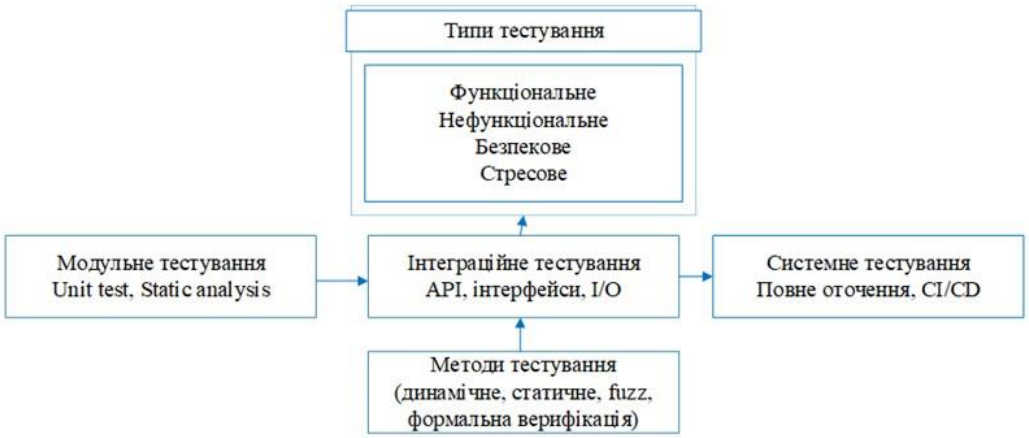
Мета і задачі роботи

Задачі:

- провести аналіз сучасних підходів до тестування і верифікації СПЗ;
- розробити класифікацію типів системного ПЗ та відповідних методів тестування;
- побудувати модель автоматизованої верифікації з використанням формальних та емпіричних підходів;
- реалізувати програмний комплекс для автоматизованої перевірки СПЗ;
- провести експериментальне дослідження ефективності запропонованого методу.

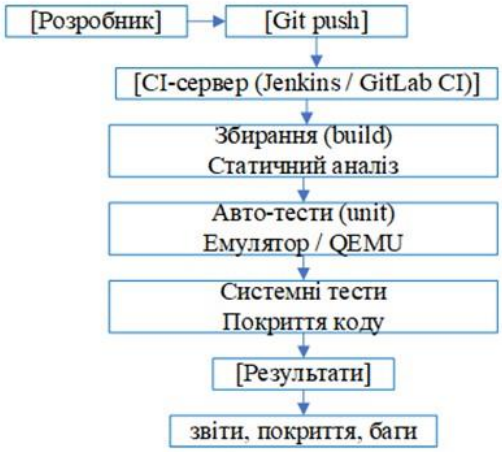
3

Узагальнена модель тестування СПЗ



4

Вбудовування тестування в DevOps-пайплайн



5

Верифікація системного ПЗ

6

Особливості тестування системного ПЗ

7

Застосування стандартів для верифікації СПЗ

Стандарт	Основна область застосування	Специфічність до СПЗ	Ключові вимоги
ISO/IEC/IEEE 29119	Універсальний	Часткова	Формалізовані процеси тестування
IEEE 1012	V&V у ПЗ	Так	Верифікація на всіх фазах ЖЦ
DO-178C	Авіаційне СПЗ	Так	Сертифікація, формальні методи
MISRA C	Вбудовані системи (C)	Так	Кодекс правил, синтаксичні обмеження
ISO 25010	Оцінка якості	Ні	Метрики якості ПЗ

8

Порівняння методів тестування за принципом доступу до внутрішньої структури програми

Ознака	Black Box	White Box	Grey Box
Знання коду	Відсутнє	Повне	Часткове
Приклади у СПЗ	IOCTL, dev-тести	KUnit, static tools	e2e з логами, strace
Глибина покриття	Низька	Висока	Середня
Складність реалізації	Низька	Висока	Середня
Використання в CI/CD	Легко інтегрується	Потребує емул.	Частково автоматизується

9

Порівняння інструментів автоматизації тестування СПЗ

Інструмент	Тип аналізу	Сфера застосування	Інтеграція в CI	Придатн. до СПЗ	Переваги	Обмеження
KLEE	Символічний	Модульне тестування (user-space)	Часткова	Непряма	Глибоке покриття шляхів	Обмежена підтримка kernel-space
CBMC	Формальна верифікація	Вбудовані компоненти	Так	Так	Виведення властивостей, MISRA	Високе навантаження на ресурси
AFL	Fuzzing	Системні виклики, драйвери	Так	Так	Швидке знаходження edge-case	Не гарантує повноту покриття
Valgrind	Динамічний	User-space утиліти	Так	Частково	Глибокий аналіз пам'яті	Не працює з kernel-mode кодом
QEMU	Емуляція	Ядро, системні образи	Так	Так	Апаратна незалежність, CI-ready	Не точно моделює апаратну поведінку
Clang Static Analyzer	Статичний	Системні бібліотеки, ядро	Так	Так	Інтеграція з LLVM, швидкість	Хибні спрацювання, обмежена глибина
Jenkins / GitLab CI	Автоматизація	Весь процес	Так	Так	Повна інтеграція тестів	Потребує налаштування інфраструктури

10

Основні інструменти перевірки моделей

Найменування	Властивості
SPIN	- мова моделювання: PROMELA (Process Meta Language); - орієнтований на моделювання паралельних процесів; - перевіряє властивості, задані в LTL; - підтримує генерацію контрприкладів, трасування помилок [57]
NuSMV / nuXmv	- призначений для перевірки скінченних станів; - підтримує CTL і LTL; - використовується в індустрії для формального аналізу цифрових систем і ПЗ
UPPAAL	- спеціалізований інструмент для перевірки таймованих автоматів; - застосовується в аналізі реального часу - наприклад, перевірка затримок у планувальниках ОС
TLA+	- заснований на темпоральній логіці; - підходить для моделювання алгоритмів керування станами, протоколів ядра.

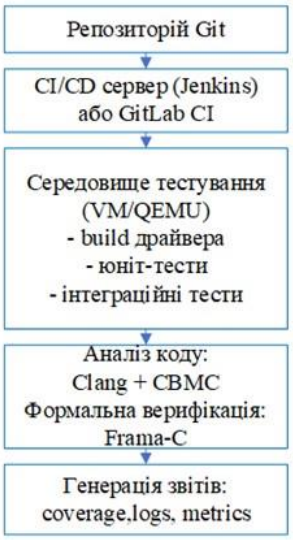
11

Практичні кейси подолання проблеми масштабованості

Проєкт / Ядро	Підхід	Інструмент(и)
seL4	інкрементальне доведення	Isabelle, AutoCorres
Linux scheduler	slicing + WP	Frama C
Bao hypervisor	BMC + контрактне доведення	CBMC + ACSL
Verus + NrOS	SMT аналіз обмежених структур	Verus, Z3
Dafny формалізація	модульний аналіз	Dafny, Boogie

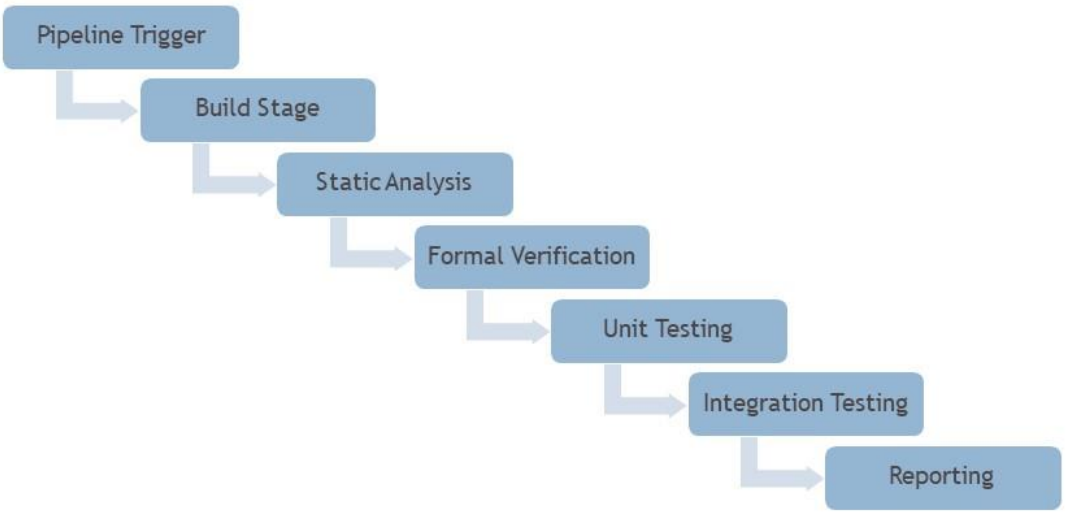
12

Модель програмного комплексу



13

Запропонований пайплайн CI/CD з елементами формального аналізу



14

Приклади шаблонів сценаріїв

```
name: parallel_race_check
description: Паралельний запис у /dev/demochar
threads:
  - script:
      - open
      - write "THREAD_1"
      - close
  - script:
      - open
      - write "THREAD_2"
      - close
expect:
  - no_crash: true
  - buffer_consistency: true
```

```
name: demochar_read_buffered
description: Читання з пристрою із заповненим буфером
steps:
  - action: write
    input: "ABCDEF"
  - action: read
    size: 4
    expect: "ABCD"
  - action: read
    size: 4
    expect: "EF"
```

15

Проблеми, виявлені під час впровадження

Проблема	Причина	Рішення
Відсутність покриття певних гілок	Неактивні блоки при нормальній роботі	Генерація тестів з інструментами gcovr + кастомні сценарії
CBMC зависав на складних функціях	Занадто глибока вкладеність циклів	Обмеження unwinding depth та розбиття функцій
Frama-C не розумів нестандартні макроси	Макроси ядра container_of, get_user()	Додавання псевдо-реалізацій для аналізатора
Відсутність підтримки KUnit в дистрибутиві	Мінімальне ядро без включеного CONFIG_KUNIT	Побудова окремого ядра з потрібними параметрами
Нестабільність під час тестування в QEMU	Проблеми з таймінгом	Встановлення обмежень часу виконання через timeout, watchdog

16

Оцінка ефективності

Етап	Ручне тестування	Автоматизоване
Компіляція + запуск	5-10 хв	15 сек
Виконання сценаріїв	30-60 хв	2-3 хв
Перевірка формальних властивостей	майже недосяжно	5-10 хв
Генерація звітів	вручну	автоматично

Параметр	Ручне тестування	Автоматизоване тестування та верифікація
Кількість тестів	14	78
Покриття коду	~40%	91.3%
Помилок виявлено	4	9 (включно з 3 критичними)
Середній час перевірки	~1 год	4 хв
Фальшиві спрацювання	0	1 (з боку CBMC)
Потреба в розробнику	100% взаємодії	<10% (тільки аналіз звітів)

17

Висновки

1. Теоретичні результати.

- Проведено аналіз основних типів системного програмного забезпечення та специфіки його тестування.
- Розглянуто методи верифікації, включно з формальними (model checking, доказ коректності), статичними (аналіз коду) та динамічними (інструментальне тестування).
- Систематизовано та класифіковано інструменти автоматизації, серед яких особливу увагу приділено KLEE, CBMC, Frama-C, Jenkins, QEMU, а також CI/CD-системам.

2. Практичні результати.

- Розроблено систему автоматизації тестування та верифікації на прикладі драйвера символьного пристрою ядра Linux.
- Реалізовано повний CI/CD-пайплайн із використанням статичного аналізу (clang, cppcheck), модульних та інтеграційних тестів (KUnit, QEMU), формальної верифікації (CBMC, Frama C).
- Розроблено сценарії тестування та шаблони, що охоплюють ключові функції драйвера, підтримують параметризацію, паралелізм та fault-injection; проведено експериментальні дослідження.

18

Висновки

3. Наукова новизна.

- Запропоновано інтегровану модель перевірки системного ПЗ, яка поєднує: автоматизоване юніт- та інтеграційне тестування; формальну верифікацію з ACSL-контрактами; CI/CD-підхід до забезпечення якості системного коду.
- Для верифікації модуля ядра було поєднано CBMC та Frama C у межах однієї автоматизованої збірки.

4. Практична значущість. Запропоновану систему можна адаптувати для:

- перевірки модулів ядра ОС;
- тестування низькорівневого ПЗ (драйверів, прошивок);
- впровадження у промислові проєкти з підвищеними вимогами до безпеки (RTOS, медичні пристрої, автомобільні системи).

5. Підготовлена публікація: Vasylenko D., Sorokin V., Rosinskiy D. «Strategic Planning in the Context of Combined Software Testing».

19