DOI: 10.32347/2412-9933.2021.45.75-81
УДК 004.415.532

Кудрявцев Олег Аркадійович
Аспірант кафедри програмного забезпечення автоматизованих систем, *orcid.org/0000-0002-9890-9444*
Черкаського державного технологічного університету, Черкаси

АКТУАЛЬНІСТЬ ПРОВЕДЕННЯ МОДУЛЬНОГО ТЕСТУВАННЯ ПРИ РОЗРОБЛЕННІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Анотація. *Процес тестування при створенні програмного забезпечення, як окремий етап, має свій життєвий цикл. Життєвий цикл тестування є частиною всього циклу програмного забезпечення, і вони мають синхронізуватися один з одним. Проєктування та розроблення тестування в створенні нових програмних систем складне та трудомістке завдання. Робота будь-якої програмної системи організовується як сукупність модулів, які можуть виконувати різноманітні функції. Для забезпечення правильності роботи системи загалом, необхідно спочатку окремо протестувати кожен модуль програмного забезпечення. У разі виникнення дефектів така процедура допоможе простіше виявити проблему в модулях і повністю усунути відповідні недоліки. Тестування окремо всіх модулів називається модульним тестуванням. У статті проведено аналіз основних аспектів модульного тестування. Розглянуто життєвий цикл програмного забезпечення та графічно представлені стадії циклу розроблення комп'ютерних програм. Доведено, що однією з найважливіших стадій є стадія тестування програмного забезпечення. Детально проаналізовано основні визначення поняття «тестування» і перелічені основні види процесу тестування. Встановлено, що модульне тестування є одним з найвагоміших видів тестування. Модульне тестування вважається найбільш простим етапом тестування всієї системи, тому що модулі, які піддаються тестуванню, зазвичай невеликі за розміром. При використанні такого методу підвищується складність створення тестових прикладів. Для приведення в робочий стан всієї системи потрібно правильно встановити значення тестових змінних, а для приведення в необхідний стан частини реальної системи необхідно виконати цілий сценарій. Однак з модульним тестуванням пов'язані значні проблеми, які досліджено в статті. Оцінено основні завдання та підходи до модульного тестування, ґрунтовно розкрито характеристики модульного тестування як спосіб структурного тестування.*

Ключові слова: стадія тестування; структурне тестування; заглушка; драйвер; програма

Вступ

Життєвий цикл програмного забезпечення – це умовна схема, що включає окремі етапи, які представляють стадії процесу створення програмного забезпечення. При цьому на кожному етапі виконуються зовсім різні дії.

На рис. 1 представлені основні стадії циклу розроблення програмного забезпечення.

Розглянемо детальніше етап тестування. У фазі тестування виявляються пропущені в процесі розроблення помилки або дефекти. При виявленні дефекту тестувальник складає звіт про помилку, який передає розробникам. Ці дефекти виправляють, після чого тестування повторюється. На цьому етапі проведення тестів відбувається для перевірки, що проблема виправлена, і саме виправлення не стало причиною появи нових дефектів у продукті.

Тестування повторюється допоки не будуть досягнуті критерії його закінчення.

Рисунок 1 – Стадії циклу розроблення програмного забезпечення

Аналіз останніх джерел

Розглянемо детальніше визначення поняття «тестування».

У різний час в різних літературних джерелах тестування отримало різні визначення. Розглянемо декілька з них.

© О. А. Кудрявцев

Лайза Кріспін та Джанет Грегорі розглядали тестування як процес виконання програми з метою знаходження помилок [9].

Канер Кем визначив, що тестування – це інтелектуальна дисципліна, що має за мету одержання надійного програмного забезпечення без зайвих зусиль на його перевірку [8].

Гленфорд Майєрс, Том Баджетт, Корі Сандлер зазначили, що тестування – це перевірка відповідності між реальною поведінкою програми та її очікуваною поведінкою на кінцевому наборі тестів, які виконуються певним чином [6].

Калбертсон Роберт, Браун Кріс та Кобб Гері в своїй книзі «Швидке тестування» описали процес тестування, як процес спостереження за виконанням програми в спеціальних умовах і винесення на цій основі оцінки будь-яких аспектів її роботи [7].

У підручнику «Верифікація програного забезпечення» тестування представлено, як процес, який має за мету виявлення ситуацій, в яких поведінка програми є неправильною, небажаною або не відповідає специфікації [14].

Бейзер у своїй праці відмітив, що тестування це процес, який містить у собі всі активності життєвого циклу (як динамічні, так і статичні). Вони стосуються планування, підготовки, оцінки програмного продукту та пов'язаних з цим результатів роботи, з метою визначити відповідності описаним вимогам та показати, що вони підходять для заявлених цілей для визначення дефектів [4].

Мета статті

Мета статті – дослідження модульного тестування в процесі розроблення програмної системи за рахунок покриття дугами інформаційного графа програми.

Опис предметної області

Процес тестування, як окремий етап, має свій життєвий цикл. Життєвий цикл тестування є частиною життєвого циклу програмного забезпечення, і вони мають синхронізуватися один з одним. Проєктування та розроблення тестування може бути настільки ж складним і трудомістким завданням, як і розробка самого програмного продукту. У разі невиконання тестування разом з першими версіями програмного забезпечення, багато неполадок будуть виявлятися на більш пізніх стадіях розроблення.

Часто внаслідок таких дій випуск продукту відкладається через довгий період виправлення помилок програми, що практично зводить нанівець переваги ітеративної розробки. Ітерація – завершений цикл розробки, що приводить до випуску кінцевого продукту або деякої скороченої версії, яка розширюється від ітерації до ітерації, щоб, врешті-решт, стати закінченою системою.

Детальна схема життєвого циклу тестування представлена на рис. 2.

Рисунок 2 – Життєвий цикл тестування програмного забезпечення

Стадія 1 (планування тестування) – необхідна для того, щоб отримати відповіді на такі питання: що не належить тестувати; як багато буде роботи; які є складності; чи все необхідне є для проведення роботи. Як правило, отримати відповіді на ці запитання неможливо без аналізу вимог, тому що саме вимоги є первинним джерелом відповідей.

Стадія 2 (уточнення критеріїв) – дає змогу сформулювати або уточнити метрики та ознаки можливості або необхідності початку тестування, припинення та поновлення тестування, завершення або припинення тестування.

Стадія 3 (уточнення стратегії тестування). На цій стадії розглядаються і уточнюються частини стратегії тестування, які актуальні для поточної ітерації.

Стадія 4 (розроблення тест-кейсів) – присвячена розробленню, перегляду, уточненню, доопрацюванню, переробленню та іншим діям з тест-кейсами, наборами тест-кейсів, тестовими сценаріями та артефактами, які будуть використовуватися при безпосередньому виконанні тестування.

Стадія 5 (виконання тест-кейсів) та *стадія 6* (фіксація знайдених дефектів) тісно пов'язані між собою і фактично виконуються паралельно. Дефекти фіксуються відразу за фактом їх виявлення в процесі виконання тест-кейсів. Однак найчастіше після виконання всіх тест-кейсів і написання всіх звітів про знайдені дефекти проводиться явно виділена стадія уточнення, на якій всі звіти про дефекти розглядаються повторно з метою формування єдиного розуміння проблеми та уточнення таких характеристик дефекту, як важливість і терміновість.

Стадія 7 (аналіз результатів тестування) та стадія 8 (звітність) також тісно пов'язані між собою та виконуються практично паралельно.

Сформульовані на стадії аналізу результатів висновки безпосередньо залежать від плану тестування, критеріїв приймання та уточненої стратегії, отриманих на стадіях 1, 2 і 3. Отримані результати оформлюються на *стадії 8* та є основою для *стадій 1, 2 і 3* для наступної ітерації тестування. Отже, цикл замикається.

Тестування програмного забезпечення охоплює цілу низку видів діяльності. До них належать: постановка задачі для тесту, проєктування, написання тестів, тестування тестів і, нарешті, виконання тестів і перегляд результатів тестування. Вирішальну роль відіграє проєктування тесту.

Є декілька основ, за якими прийнято проводити класифікацію видів тестування.

1. За об'єктом тестування:
   - навантажувальне тестування (performance/load/stress testing);
   - функціональне тестування (functional testing);
   - тестування інтерфейсу користувача (UI testing);
   - тестування зручності використання (usability testing);
   - тестування безпеки (security testing);
   - тестування сумісності (compatibility testing);
   - тестування локалізації (localization testing).
2. За доступом до коду системи:
   - тестування методом «сірого ящика» (grey box);
   - тестування методом «білого ящика» (white box);
   - тестування методом «чорного ящика» (black box).
3. За рівнем автоматизації:
   - автоматизоване тестування (automated testing);
   - ручне тестування (manual testing).
4. За ступенем ізольованості:
   - системне тестування (system testing);
   - інтеграційне тестування (integration testing);
   - модульне тестування (unit testing).
5. За рівнем готовності:
   - приймальне тестування (acceptance testing);
   - альфа-тестування (alpha testing);
   - бета-тестування (beta testing).
6. Пов'язане зі змінами:
   - регресивне тестування (regression testing);
   - димове тестування (smoke testing).
7. Тестування збірки (build verification test) [5].

Викладення основного матеріалу

Дослідимо детальніше такий вид тестування, як модульне тестування. Для початку розглянемо основні визначення.

Модульне тестування – тестування, мета якого перевірити працездатність окремих модулів (функції або класу) [11]. Модульне тестування зазвичай виконується незалежно для кожного програмного модуля і є, напевно, найбільш поширеним видом тестування, особливо для систем малих і середніх розмірів.

Драйвер – конкретний модуль тесту, який виконує елемент тестування [1].

Заглушка – частина комп'ютерної програми, яка імітує обмін інформацією з компонентом тестування і виконує проєктування робочої системи.

Кожна складна програмна система складається з окремих частин – модулів, які виконують різні функції в складі системи. Для впевненості у коректній роботі системи в цілому потрібно спочатку протестувати окремо кожен модуль програмної системи. У разі виникнення дефектів це допоможе набагато простіше виявити модулі, які викликали проблему, та повністю усунути відповідні дефекти. Тестування окремо модулів отримало назву модульного тестування (unit testing).

Головна ціль модульного тестування – точно впевнитись у відповідності висунутим вимогам для окремого кожного модуля програмної системи перед тим, як проводити інтеграцію до складу всієї системи. При цьому під час модульного тестування вирішуються основні чотири завдання.

1. Документування та пошук невідповідностей визначеним вимогам – класичне завдання тестування, яке включає не тільки створення тестового оточення та тестових прикладів, а й протоколювання результатів виконання, виконання тестів, складання звітів про дефекти.

2. Підтримка рефакторингу розробки та низькорівневої архітектури програмної системи та міжмодульної взаємодії – завдання властиве методологіям типу, в яких використовується принцип тестування перед розробкою (test-driven development). Основне джерело вимог для програмного модуля – тест, який написаний до самого модуля. Проте, навіть використовуючи класичні схеми тестування, модульні тести можуть виявити дефекти в дизайні системи та заплутані або нелогічні механізми роботи з даним модулем.

3. Підтримка рефакторингу модулів – завдання, яке пов'язане з підтримкою процесу модифікації системи. Досить часто в процесі розроблення необхідно проводити рефакторинг модулів або їх цілих груп – повне перероблення або оптимізацію програмного коду з метою підвищення швидкості роботи, надійності або супроводження. При цьому модульні тести є потужним інструментом для перевірки нового варіанта програмного коду, який має працювати аналогічно старому.

4. Підтримка усунення проблем і налагодження – завдання пов'язане зі зворотним зв'язком, який одержують розробники від тестувальників у вигляді звітів про дефекти. Деталізовані звіти, складені на етапі модульного тестування, допомагають усунути та локалізувати багато проблем в програмній системі на ранніх стадіях її розроблення або розроблення деякої нової функціональності.

Модулі, які піддаються тестуванню, зазвичай невеликі за розміром, тому модульне тестування вважається найпростішим, але дуже трудомістким етапом тестування всієї системи. Проте, незважаючи на простоту у виконанні, з модульним тестуванням пов'язано дві основні проблеми:

1. Відмінності в трактуванні поняття модульного тестування. Під визначенням можуть розуміти відокремлений модуль тестування, робота якого підтримується тільки тестовим оточенням, або перевірку коректності роботи окремого модуля у складі вже створеної системи.

2. Не існує єдиних принципів для визначення окремого модуля.

Останнім часом сам термін «модульне тестування» частіше використовується в іншому сенсі, хоча в цьому випадку, мова скоріше йде про інтеграційне тестування.

Традиційне визначення поняття модуля з точки зору тестування: «модуль – компонент мінімального розміру, який може бути незалежно протестований в процесі верифікації системи». У реальності дуже часто виникають проблеми з тим, що вважати за модуль. Є декілька підходів до цієї проблеми:

– модуль – програмний модуль, тобто мінімальний компільований елемент системи;

– модуль – частина коду, що виконує єдину функцію з точки зору функціональних вимог;

– модуль – частина коду, який може поміститися на одному аркуші паперу або одному екрані;

– модуль – завдання, з точки зору менеджера, у списку завдань проєкту;

– модуль – один клас або їх множина з одним інтерфейсом.

В основному за модуль приймається програмний модуль (одиниця компіляції) у разі, якщо програмна система розробляється процедурною мовою програмування, або клас, якщо програмна система розробляється об'єктно-орієнтованою мовою.

Як зазначалося вище, модульні тести – потужний інструмент для перевірки коректності внесених змін у вихідний програмний код при рефакторінгу. Проте, в результаті рефакторингу тільки одного класу, не змінюється зовнішній інтерфейс з іншими різними класами (зміна інтерфейсу відбувається при рефакторінгу одночасно декількох класів). В результаті простих еволюційних змін програмної системи у класі може змінюватися зовнішній інтерфейс, причому як за функціональними ознаками (при збереженні інтерфейсу змінюється логіка роботи методів), так і за формальними (змінюється склад та назви методів, їх характеристики). Для проведення тестування класу після таких змін, необхідна зміна драйвера та, ймовірно, заглушок. В цьому випадку недостатньо тільки модульного тестування, а також необхідно проводити інтеграційне тестування даного класу разом з усіма класами, які пов'язані інформацією або управлінням.

Незалежно від тестування модулів, на які розбивається система, рекомендується виявити принципи акцентування протестованих модулів у стратегії та плані тестування, а також створити на основі структурної схеми архітектури системи новітню схему, на якій зазначити всі протестовані модулі. Такий підхід дасть змогу спрогнозувати складність та склад драйверів і заглушок, які необхідні для модульного тестування програмної системи. Цю схему можна використовувати пізніше, на етапі модульного тестування для виокремлення великих груп модулів, які необхідно інтегрувати.

Незважаючи на те, яка мінімальна одиниця вихідних кодів програмної системи вибирається за мінімальний протестований модуль, є ще одна різниця в підходах до модульного тестування.

Перший підхід ґрунтується на припущенні, що функціональність всіх, знову розроблених модулів, зобов'язана перевірятися без інтеграції з програмною системою в автономному режимі. Для кожного нового розробленого модуля створюються заглушки та тестовий драйвер, за допомогою яких виконується необхідний набір тестів. Тільки після усунення всіх проблем в автономному режимі, відбувається інтеграція модуля в систему та проводиться тестування на наступному рівні.

Перевагою такого підходу є набагато простіша локалізація дефектів у модулі, оскільки при автономному тестуванні виключається вплив інших частин програмної системи, що може викликати маскування дефектів.

Головний недолік такого методу – підвищена трудомісткість написання заглушок і драйверів, оскільки заглушки повинні відповідно моделювати поведінку програмної системи в різних ситуаціях, а драйвер – створювати тестове оточення та імітувати внутрішній стан програмної системи, в складі якої функціонує модуль.

Другий підхід організовано на припущенні, що модуль все одно працює у складі програмної системи

та інтеграції модулів у систему по одному, може протестувати поведінку даного модуля у складі всієї системи. Такий підхід властивий великій кількості сучасних методологій розробки.

При використанні такого підходу різко зменшуються витрати на розроблення драйверів і заглушок. Як заглушка виступає вже протестована частина програмної системи, а драйвер тільки виконує функції прийому і передачі інформації, який не моделює внутрішній стан системи.

При застосуванні такого методу зростає складність написання тестових прикладів. Для приведення в необхідний стан системи заглушок, як правило, потрібно тільки встановити значення тестових змінних, а для приведення в необхідний стан частини реальної програмної системи потрібно виконати цілий сценарій. У цьому випадку кожен тестовий приклад повинен містити такий сценарій.

Крім того, у разі такого підходу не завжди вдається обмежити помилки, приховані всередині модуля, які можуть виявитися при інтеграції наступних модулів [10].

Дослідимо модульне тестування як спосіб структурного тестування. За способом виконання структурного тестування або тестування «білого ящика», модульне тестування відрізняється тим, що тести покривають або виконують логіку програми. Тести, пов'язані зі структурним тестуванням, будуються в такий спосіб:

На базі аналізу потоку управління. В такому разі елементи, які повинні бути покриті при проходженні тестів, визначаються на основі структурних критеріїв тестування. До них належать: вершини, дуги, шляхи керуючого графа програми, який керує умовами або комбінацією умов.

На базі аналізу потоку даних. В такому випадку елементи, які повинні бути покриті, визначаються за допомогою потоку даних, тобто інформаційного графа програми.

Тестування на основі потоку даних спрямоване на знаходження аномалій потоку даних. Запропонована стратегія вимагає тестування всіх взаємозв'язків, тобто необхідне покриття дуг інформаційного графа програми.

Процес побудови набору тестів при структурному тестуванні прийнято розподіляти на такі фази:

- конструювання керуючого графа програми;
- вибір тестових шляхів;
- генерація тестів, відповідних тестовим шляхам.

Перша фаза відповідає статичному аналізу програми, завдання якої полягає в отриманні графа програми та залежного від критерію тестування множини елементів, які потрібно покрити тестами.

Друга фаза забезпечує вибір тестових шляхів. До побудови тестових шляхів розрізняють три підходи:

- динамічні методи;
- статичні методи;
- методи реалізованих шляхів.

На третій фазі за відомими шляхами тестування відбувається пошук відповідних тестів, що реалізовують проходження цих шляхів.

Статичні методи. Найпростіший метод – побудова кожного шляху за допомогою поступового його подовження за рахунок додавання дуг, поки не буде досягнута вихідна вершина графа програми. Ця ідея може бути посилена в адаптивних методах, які кожного разу додають лише один тестовий шлях, використовуючи попередні шляхи, як керівництво для вибору подальших доріг, відповідно до деякої стратегії. Головний недолік статичних методів – не враховується можливість реалізації побудованих тестових шляхів.

Динамічні методи. Такі методи передбачають побудову повної системи тестів, які задовольняють необхідному критерію, шляхом одночасного розв’язання задачі побудови покриваючої множини шляхів і формування тестових даних. При цьому можна автоматично враховувати можливість реалізації раніше розглянутих шляхів або їх частин. Головною ідеєю динамічних методів є додавання до початкових реалізованих відрізків шляхів їх подальших частин так, щоб:

- не втрачати знову отриманих шляхів;
- покрити потрібні елементи структури програми.

Методи реалізованих шляхів. Така методика полягає у виділенні з множини шляхів підмножини всіх шляхів, які реалізовуються. Потім покриваюча множина шляхів будується з отриманої підмножини шляхів.

Перевага статичних методів полягає в порівняно малій кількості потрібних ресурсів як при використанні, так і при розробленні. Проте їх реалізація може містити непередбачуваний відсоток браку. Крім того, в таких системах перехід від покриваючої великої кількісті шляхів до повної системи тестів, користувач має здійснити вручну, а цей перехід доволі трудомісткий.

Динамічні методи потребують значно більших ресурсів як при розробленні, так і при експлуатації, однак збільшення витрат відбувається за рахунок розроблення і експлуатації апарату визначення шляхів, що реалізуються (символічний інтерпретатор).

Перевага цих методів полягає в тому, що їх продукція має деякий якісний рівень, який реалізовується за допомогою шляхів. Методи реалізованих шляхів дають найкращий результат [11].

Висновки і перспективи подальшого розвитку

Весь життєвий цикл програмного забезпечення включає окремі етапи, які представляють процес створення програм з виконанням різних дій на кожному етапі. Фаза тестування допомагає виявити пропущені помилки або дефекти. Основна мета модульного тестування полягає в перевірці відповідності встановлених вимог для кожного модуля системи перед інтеграцією до складу всієї системи. Використання модульного тестування, як спосіб структурного тестування, дасть змогу повністю покривати тестами або виконувати логіку програми. Процес побудови набору розділиться на такі фази: конструювання керуючого графа програми; вибір тестових шляхів; генерація тестів, відповідних тестовим шляхам. Запропонована стратегія допоможе тестувати всі взаємозв'язки, тобто повністю покривати дугами інформаційний граф програми.

Проблема тестування програмних систем і комунікації між розробниками та тестувальниками, які дадуть змогу підвищити рівень програмного забезпечення, є актуальним та перспективним напрямом подальших досліджень.

Список літератури

1. Fewster M, Graham D. Software Test Automation. ACM Press, 2019. 600 p.
2. Sangwan O. P. Automated software test optimization using test language processing: *The international arab journal of information technology,* vol. 16. 2019. № 3. P. 348–356.
3. Автоматизация тестирования методом программных приложений. URL: https://www.dissercat.com/content/avtomatizatsiya-testirovaniya-programmnykh-prilozhenii-metodom-klyuchevykh-sostoyanii. (дата звернення: 10.11.2020).
4. Бейзер Б. Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем. Питер : Санкт-Петербург, 2004. 320 с.
5. Види тестування та відмінності між ними. URL: https://www.quality-assurance-group.com/vydy-testuvannya-ta-vidminnosti-mizh-nymy-shpargalka-z-testuvannya-chastyna-4/ (дата звернення: 02.12.2020).
6. Майерс Гленфорд, Баджетт Том, Сандлер Кори Искусство тестирования программ. Москва : Диалектика, 2012. 272 с.
7. Калбертсон Роберт, Браун Крис, Кобб Гэри Быстрое тестирование. Москва : Вильямс, 2002. 374 с.
8. Канер Кем, Фолк Джек, Нгуен Енг Кек Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. Киев : ДиаСофт, 2001. 544 с.
9. Криспин Лайза, Грегори Джанет Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд. Москва : Вильямс, 2010. 464 с.
10. Липаев В. В. Тестирование программ. Москва : Радио и связь, 2016. 292 с.
11. Модульне тестування. URL: https://msn.khnu.km.ua/pluginfile.php/208290/mod_resource/content/2/%D0%9B%D0%A0%20%E2%84%964.pdf (дата звернення: 25.11.2020).
12. Модульне тестування. URL: https://studfile.net/preview/14533269/ (дата звернення: 05.11.2020).
13. Принципы тестирования. URL: https://qalight.com.ua/baza-znaniy/pochemu-testirovanie-neobhodimo/ (дата звернення: 10.11.2020).
14. Синицын С. В., Налютин Н. Ю. Верификация программного обеспечения. Москва : БИНОМ, 2008. 368 с.
15. Системный контекст программного обеспечения. URL: https://stepik.org/lesson/106620/step/1?unit=81144 (дата звернення: 27.11.2020).

Стаття надійшла до редколегії 23.02.2021

Kudryavtsev Oleg
Postgraduate student, Department of software for automated systems, ORCID: 0000-0002-9890-9444
Cherkasy State Technological University, Cherkasy

ACTUALITY OF CONDUCTING MODULAR TESTING IN SOFTWARE DEVELOPMENT

Abstract. *The testing process in creating software, as a separate stage, has its own life cycle. The testing lifecycle is part of the entire software cycle, and they must be synchronized with each other. Designing and developing testing in creating new software systems is a complex and time consuming task. The work of any software system is organized as a set of modules that can perform various functions. To ensure the correct operation of the system as a whole, you must first test each software module separately. In case of defects, this procedure will make it easier to identify the problem in the modules and completely eliminate the relevant shortcomings. Testing all modules separately is called modular testing. The article examines the main aspects of modular testing. The software life cycle is considered and the stages of the computer program development cycle are graphically presented.*

It is investigated that one of the most important stages is the stage software testing. The main definitions concept of "testing" are analyzed in detail and the main types of testing process are listed. It is established that modular testing is one of the most important types of testing. Modular testing is considered to be the simplest step in testing the entire system because the modules to be tested are usually small in size. When using this method, the complexity of creating test cases increases. To put the whole system into operation, you need to set the values of the test variables correctly, and to bring part of the real system to the required state, you need to run the whole scenario. However, modular testing is associated with significant problems, which are explored in the article. The main tasks and approaches to modular testing are evaluated. The article thoroughly reveals the characteristics of modular testing as a method of structural testing.

Keywords: testing stage, structural testing, stub, driver, program

References

1. Fewster, M., & Graham, D., (2019). Software Test Automation. ACM Press.
2. Sangwan, O.P., (2019). Automated software test optimization using test language processing. *The International Arab Journal of Information Technology*, 16, 348–356.
3. Avtomatyzatsyia testyrovanyia metodom prohrammnukh prylozhenyi [electronic source]. Available: https://www.dissercat.com/content/avtomatizatsiya-testirovaniya-programmnykh-prilozhenii-metodom-klyuchevykh-sostoyanii.
4. Bejzer, B., (2004). Functional testing technologies for software and systems. Piter: SPb.
5. Vydy testuvannia ta vidminnosti mizh nymy [electronic source]. Available: https://www.quality-assurance-group.com/vydy-testuvannya-ta-vidminnosti-mizh-nymy-shpargalka-z-testuvannya-chastyna-4.
6. Majers, Glenford, Badzhett, Tom, & Sandler, Kory, (2012). The art of software testing. Moscow: Dialektika.
7. Kalbertson, Robert, Braun, Krys, & Kobb, Gery, (2002). Rapid testing. Moscow: Viliams.
8. Kaner, Kem, Folk, Dzhek, Nguen, Eng Kek, (2001). Software testing. Fundamental Concepts of Business Application Management. Kiiv: DiaSoft
9. Kryspyn, Lajza, (2017). Agile Testing: A Practical Guide for Software Testers and Agile Teams. Moscow: Viliams.
10. Lypaev, V. V., (2016). Testing programs. Moscow: Radyo y sviaz.
11. Modulne testuvannia [electronic source]. Available: https://msn.khnu.km.ua/pluginfile.php/208290/mod_resource/content/2/% D0%9B%D0%A0%20%E2%84%964.pdf.
12. Modulne testuvannia [electronic source]. Available: https://studfile.net/preview/14533269.
13. Pryntsypu testyrovanyia [electronic source]. Available: https://qalight.com.ua/baza-znaniy/pochemu-testirovanie-neobhodimo.
14. Synyczin, S. V., & Nalyuty`n, N. Yu., (2008). Software verification. Moscow: Binom.
15. Systemnui kontekst prohrammnoho obespechenyia [electronic source]. Available: https://stepik.org/lesson/106620/step/1?unit=8114.

Посилання на публікацію

APA Kudryavtsev, Oleg, (2021). Actuality of conducting modular testing in software development. *Management of Development of Complex Systems*, 45, 75–81, dx.doi.org\10.32347/2412-9933.2021.45.75-81.

ДСТУ Кудрявцев О. А. Актуальність проведення модульного тестування при розробленні програмного забезпечення. *Управління розвитком складних систем.* Київ, 2020. № 45. С. 75 – 81, dx.doi.org\10.32347/2412-9933.2021.45.75-81.