ТЕХНОЛОГІЇ ТА ДИЗАЙН
ISSN 2304-2605
№ 1 (18) 2016 р.

УДК 004.42

ОСОБЛИВОСТІ ТИПІВ ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Писанець К. К., Вовчук А. В.

Київський національний університет технологій та дизайну

Стаття присвячена розвитку тестування, контролю якості в розробці програмного забезпечення. Досліджені наявні методи тестування. Обґрунтовано актуальність тестування на даний момент через великий попит на нього.
Ключові слова: *тестування, типи тестування, якість*

Тестування програмного забезпечення – процес дослідження, випробування програмного продукту, що має дві різні мети: продемонструвати розробникам і замовникам, що програма відповідає вимогам та виявити ситуації, в яких поведінка програми є неправильною, небажаною або не відповідає специфікації.

Також тестування програмного забезпечення – перевірка відповідності між реальною і очікуваною поведінкою програми, здійснювана на кінцевому наборі тестів, яка виконана певним чином. У більш широкому сенсі, тестування – це одна з технік контролю якості, що включає в себе активності з планування робіт (Test Management), проектування тестів (Test Design), виконанню тестування (Test Execution) та аналізу отриманих результатів (Test Analysis).

Постановка завдання

Сучасні процеси світового економічного розвитку вказують на підвищення ролі тестування для підвищення якості розробки програмного забезпечення. Одним із сегментів ринку ПЗ, який найбільш динамічно розвивається, є аутсорсинг. В Україні така ситуація почала спостерігатись із появою міжнародних компаній по ПЗ, або створення українських з введенням міжнародних стандартів. Ринок аутсорсингу та послуг тестування в Україні розвивається дуже стрімко. Хоча 10 років назад ніхто и не чув про таку професію, як тестувальник програмного забезпечення (ПЗ) – то на сьогодні вона дуже потрібна в сфері економіки та програмування. Про це свідчить те, що порівняно з 2013 роком тестувальників по Україні збільшилося в 4 рази, та компаній представників послуг тестування вже близько 20, що є конкурентноспроможними на весь світ. Разом з тим, залишається питання чи дійсно тестування являється ефективним економічним рятівником компаній розробки ПЗ.

Результати досліджень

Існуючі на сьогодні методи тестування програмного забезпечення не дозволяють однозначно і повністю виявити всі дефекти і встановити коректність функціонування аналізованої програми, тому всі існуючі методи тестування діють в рамках формального процесу перевірки досліджуваного або розроблюваного програмного забезпечення.

Такий процес формальної перевірки, або верифікації, може довести, що дефекти відсутні з погляду використовуваного методу. Тобто немає ніякої можливості точно встановити або гарантувати відсутність дефектів у програмному продукті з урахуванням людського чинника, присутнього на всіх етапах життєвого циклу програмного забезпечення.

Існує безліч підходів до вирішення завдання тестування та верифікації програмного забезпечення, але ефективне тестування складних програмних продуктів – це процес у вищій мірі творчий, що не зводиться до слідування строгим і чітким процедурам або створенню таких.

Перші програмні системи розроблялися в рамках програм наукових досліджень і програм для потреб міністерств оборони. Тестування таких продуктів проводилося строго формалізованно із записом всіх тестових процедур, тестових даних, отриманих результатів. Тестування виділялося в окремий процес, який починався після завершення кодування, але при цьому, як правило, виконувалося тим же персоналом.

У 1960-х багато уваги приділялося «вичерпного» тестуванню, яке повинно проводитися з використанням усіх шляхів у коді або всіх можливих вхідних даних. Було відзначено, що в цих умовах повне тестування програмного забезпечення неможливо, тому що, по-перше, кількість можливих вхідних даних дуже велике, по-друге, існує безліч шляхів, по-третє, складно знайти проблеми в архітектурі і специфікаціях. З цих причин «вичерпне» тестування була відхилена і визнано теоретично неможливим [2].

На початку 1970-х років тестування програмного забезпечення позначалося як «процес, спрямований на демонстрацію коректності продукту» або як «діяльність із підтвердженню правильності роботи програмного забезпечення». У зародженні програмної інженерії верифікація ПО значилася як «доказ правильності». Хоча концепція була теоретично перспективною, на практиці вона вимагала багато часу і була недостатньо всеохоплюючою. Було вирішено, що доказ правильності –

неефективний метод тестування програмного забезпечення. Проте, в деяких випадках демонстрація правильної роботи використовується і в наші дні, наприклад, приймально–здавальні випробування. У другій половині 1970-х тестування уявлялося як виконання програми з наміром знайти помилки, а не довести, що вона працює. Успішний тест – це тест, який виявляє раніше невідомі проблеми. Даний підхід прямо протилежний попередньому. Зазначені два визначення являють собою «парадокс тестування», в основі якого лежать два протилежних твердження: з одного боку, тестування дозволяє переконатися, що продукт працює добре, а з іншого – виявляє помилки в програмах, показуючи, що продукт не працює. Друга мета тестування є більш продуктивною з точки зору поліпшення якості, так як не дозволяє ігнорувати недоліки програмного забезпечення.

У 1980-і роки тестування розширилося таким поняттям, як попередження дефектів. Проектування тестів – найбільш ефективний з відомих методів попередження помилок. В цей же час почали висловлюватися думки, що необхідна методологія тестування, зокрема, що тестування має включати перевірки на всьому протязі циклу розробки, і це повинен бути керований процес. У ході тестування треба перевірити не тільки зібрану програму, але й вимоги, код, архітектуру, самі тести. «Традиційне» тестування, яке існувало до початку 1980-х, відносилося тільки до компільованою, готовій системі (зараз це зазвичай називається системне тестування), але надалі тестувальники стали залучатися в усі аспекти життєвого циклу розробки. Це дозволяло раніше знаходити проблеми у вимогах та архітектури і тим самим скорочувати терміни і бюджет розробки. У середині 1980-х з'явилися перші інструменти для автоматизованого тестування. Передбачалося, що комп'ютер зможе виконати більше тестів, ніж людина, і зробить це більш надійно. Спочатку ці інструменти були вкрай простими і не мали можливості написання сценаріїв на скриптових мовах.

На початку 1990-х років у поняття «тестування» стали включати планування, проектування, створення, підтримку та виконання тестів і тестових оточень, і це означало перехід від тестування до забезпечення якості, щоохоплює весь цикл розробки програмного забезпечення. У цей час починають з'являтися різні програмні інструменти для підтримки процесу тестування: більш просунуті середовища для автоматизації з можливістю створення скриптів і генерації звітів, системи управління тестами, ПО для проведення навантажувального тестування. У середині 1990-х років з розвитком Інтернету і розробкою великої кількості веб–додатків особливу

популярність стало отримувати «гнучке тестування» (за аналогією з гнучкими методологіями програмування) [1].

Всі види тестування програмного забезпечення, залежно від переслідуваних цілей, можна умовно розділити на наступні групи: функціональні, нефункціональні, пов'язані зі змінами.

Розглянемо функціональне тестування. Функціональне тестування розглядає заздалегідь задану поведінку і грунтується на аналізі специфікацій функціональності компонента або системи в цілому.Функціональні тести грунтуються на функціях, виконуваних системою, і можуть проводитися на всіх рівнях тестування (компонентному, інтеграційному, системному, приемному). Як правило, ці функції описуються у вимогах, функціональних специфікаціях або у вигляді випадків використання системи (use cases).

Тестування функціональності може проводитися у двох аспектах:

– вимоги;

– бізнес процеси.

Тестування в перспективі «вимоги» використовує специфікацію функціональних вимог до системи як основу для дизайну тестових випадків (Test Cases). У цьому випадку необхідно зробити список того, що буде тестуватися, а що ні, пріорітезувати вимоги на основі ризиків (якщо це не зроблено в документі до вимог), а на основі цього пріорітезувати тестові сценарії (test cases). Це дозволить сфокусуватися і не упустити при тестуванні найбільш важливий функціонал.

Тестування в перспективі «бізнес–процеси» використовує знання цих самих бізнес–процесів, які описують сценарії щоденного використання системи. У цій перспективі тестові сценарії (test scripts), як правило, грунтуються на випадках використання системи (use cases).

Переваги функціонального тестування:

– імітує фактичне використання системи.

Недоліки функціонального тестування:

– можливість упущення логічних помилок в програмному забезпеченні;

– ймовірність надлишкового тестування.

Досить поширеною є автоматизація функціонального тестування.

Функціональні види тестування розглядають зовнішню поведінку системи. Далі перераховані одні з найпоширеніших видів функціональних тестів:

– тестування безпеки (Security and Access Control Testing).

Тестування безпеки – це стратегія тестування, використовувана для перевірки безпеки системи, а також для аналізу ризиків, пов'язаних із забезпеченням цілісного підходу до захисту додатки, атак хакерів, вірусів, несанкціонованого доступу до конфіденційних даних.

Принципи безпеки програмного забезпечення.

Загальна стратегія безпеки грунтується на трьох основних принципах:

– конфіденційність;

– цілісність;

– доступність.

Конфіденційність – це приховування певних ресурсів або інформації. Під конфіденційністю можна розуміти обмеження доступу до ресурсу деякої категорії користувачів, або іншими словами, за яких умов користувач авторизований отримати доступ до даного ресурсу.

Цілісність. Існує два основні критерії при визначенні поняття цілісності:

Довіра. Очікується, що ресурс буде змінений тільки відповідним способом певною групою користувачів.

Пошкодження і відновлення. У разі коли дані пошкоджуються або неправильно змінюються авторизованим або авторизованим користувачем, ви повинні визначити на скільки важливою є процедура відновлення даних.

Доступність являє собою вимоги про те, що ресурси повинні бути доступні авторизованому користувачеві, внутрішньому об'єкту або пристрою. Як правило, чим більш критичний ресурс тим вище рівень доступності повинен бути.

– тестування взаємодії (Interoperability Testing).

З розвитком мережевих технологій та інтернету взаємодія різних систем, сервісів і додатків один з одним набуло значної актуальності, оскільки будь–які пов'язані з цим проблеми можуть призвести до падіння авторитету компанії, що як наслідок спричинить фінансові втрати. Тому до тестування взаємодії варто підходити з усією серйозністю.

Тестування взаємодії (Interoperability Testing) – це функціональне тестування, що перевіряє здатність програми взаємодіяти з одним і більше компонентами або системами і включає в себе тестування сумісності (compatibility testing) і інтеграційне тестування (integration testing).

Програмне забезпечення з гарними характеристиками взаємодії може бути легко інтегровано з іншими системами, не вимагаючи якихось серйозних модифікацій. У цьому випадку, кількість змін і час, необхідний на їх виконання, можуть бути використані для вимірювання можливості взаємодії [3].

Розглянемодругий тип – нефункціональні види тестування.

Нефункціональне тестування описує тести, необхідні для визначення характеристик програмного забезпечення, які можуть бути виміряні різними величинами. В цілому, це тестування того, «Як» система працює. Далі перераховані основні види нефункціональних тестів:

1.Тестування продуктивності (Performance testing).

Завданням тестування продуктивності є визначення масштабованості програми під навантаженням, при цьому відбувається:

– вимір часу виконання обраних операцій за певних интенсивностях виконання цих операцій

– визначення кількості користувачів, щоодночасно працюють з додатком

– визначення меж прийнятної продуктивності при збільшенні навантаження (при збільшенні інтенсивності виконання цих операцій)

– дослідження продуктивності на високих, граничних, стресових навантаженнях

2.Стресове тестування (Stress Testing).

Стресове тестування дозволяє перевірити наскільки додаток і система в цілому працездатні в умовах стресу і також оцінити здатність системи до регенерації, тобто до повернення до нормального стану після припинення впливу стресу. Стресом в даному контексті може бути підвищення інтенсивності виконання операцій до дуже високих значень або аварійне зміна конфігурації сервера. Також одним із завдань при стресовому тестуванні може бути оцінка деградації продуктивності, таким чином цілі стресового тестування можуть перетинатися з цілями тестування продуктивності.

3.Об'ємне тестування (Volume Testing).

Завданням об'ємного тестування є отримання оцінки продуктивності при збільшенні обсягів даних в базі даних програми, при цьому відбувається:

– вимір часу виконання обраних операцій за певних интенсивностях виконання цих операцій

– може проводитися визначення кількості користувачів, щоодночасно працюють з додатком

4.Тестування стабільності або надійності (Stability / Reliability Testing).

Завданням тестування стабільності (надійності) є перевірка працездатності програми при тривалому (багатогодинному) тестуванні із середнім рівнем навантаження. Час виконання операцій може грати в даному вигляді тестування другорядну роль. При цьому на перше місце виходить відсутність витоків пам'яті, перезапусків серверів під навантаженням та інші аспекти впливають саме на стабільність роботи.

5.Тестування Установки або Installation Testing.

Тестування установки направлено на перевірку успішної інсталяції і налаштування, а також оновлення або видалення програмного забезпечення.

На даний момент найбільш поширена установка ПЗ за допомогою інсталяторів.

У реальних умовах інсталяторів може не бути. У цьому випадку доведеться самостійно виконувати установку програмного забезпечення, використовуючи документацію у вигляді інструкцій або readme файлів,які крок за кроком описують всі необхідні дії та перевірки [1].

У розподілених системах, де програма розгортається на вже працюючому оточенні, простого набору інструкцій може бути мало. Для цього, найчастіше, пишеться план установки (Deployment Plan), що включає не тільки кроки по інсталяції програми, але і кроки відкату (roll-back) до попередньої версії, у разі невдачі. Сам по собі план установки також повинен пройти процедуру тестування для уникнення проблем при видачі в реальну експлуатацію. Особливо це актуально, якщо установка виконується на системи, де кожна хвилина простою – це втрата репутації та великої кількості засобів, наприклад: банки, фінансові компанії або навіть банерні мережі. Тому тестування установки можна назвати однією з найважливіших завдань щодо забезпечення якості програмного забезпечення.

Саме такий комплексний підхід з написанням планів, покрокової перевіркою установки і відкату інсталяції, повноправно можна назвати тестуванням установки або Installation Testing

6. Тестування зручності користування або Usability Testing.

Іноді ми стикаємося з незрозумілими, нелогічними програмами, багато функцій і способи використання яких часто не очевидні. Після такої роботи рідко виникає бажання використовувати цю програму знову, і ми шукаємо більш зручні аналоги. Для того щоб програма було популярною, їй мало бути функціональною – вона має бути ще

й зручною. Якщо задуматися, інтуїтивно зрозумілі програми економлять нерви користувачам і витрати роботодавця на навчання. А значить вони більш конкурентоспроможні! Тому тестування зручності використання, про який піде мова далі є невід'ємною частиною тестування будь–яких масових продуктів.

Тестування зручності користування – це метод тестування, спрямований на встановлення ступеня зручності використання, навченості, зрозумілості та привабливості для користувачів розроблювального продукту в контексті заданих умов.

Тестування зручності користування дає оцінку рівня зручності використання додатка за наступними пунктами:

– продуктивність, ефективність (efficiency) – скільки часу і кроків знадобиться користувачеві для завершення основних завдань програми, наприклад, розміщення новини, реєстрації, покупка і т.д.? (менше – краще);

– правильність (accuracy) – скільки помилок зробив користувач під час роботи з програмою? (менше – краще);

– активізація в пам'яті (recall) – як багато користувач пам'ятає про роботу програми після призупинення роботи з ним на тривалий період часу? (повторне виконання операцій після перерви має проходити швидше ніж у нового користувача);

– емоційна реакція (emotional response) – як користувач себе почуває після завершення завдання – розгублений, випробував стрес? Порекомендує користувач систему своїм друзям? (позитивна реакція – краще).

7. Тестування на відмову та відновлення (Failover and Recovery Testing).

Тестування на відмову та відновлення (Failover and Recovery Testing) перевіряє тестовану програму з точки зору здатності протистояти і успішно відновлюватися після можливих збоїв, що виникли у зв'язку з помилками програмного забезпечення, відмовами обладнання або проблемами зв'язку (наприклад, відмова мережі). Метою даного виду тестування є перевірка систем відновлення (або дублюючих основний функціонал систем), які, у разі виникнення збоїв, забезпечать збереження і цілісність даних тестованого продукту.

Тестування на відмову і відновлення дуже важливо для систем, що працюють за принципом «24х7». Якщо Ви створюєте програму, яка буде працювати, наприклад в інтернеті, то без проведення даного виду тестування вам просто не обійтися.Кожна хвилина простою або втрата даних у разі відмови обладнання може коштувати вам грошей, втрати клієнтів і репутації на ринку.Методика подібного тестування полягає в

симулюванні різних умов збою і наступному вивченні та оцінці реакції захисних систем. У процесі подібних перевірок з'ясовується, чи була досягнута необхідна ступінь відновлення системи після виникнення збою.

Для наочності розглянемо деякі варіанти подібного тестування і загальні методи їх проведення. Об'єктом тестування в більшості випадків є вельми ймовірні експлуатаційні проблеми, такі як:

– відмова електрики на комп'ютері–сервері;

– відмова електрики на комп'ютері–клієнті;

– незавершені цикли обробки даних (переривання роботи фільтрів даних, переривання синхронізації);

– оголошення або внесення в масиви даних неможливих або помилкових елементів;

– відмова носіїв даних.

Дані ситуації можуть бути відтворені, як тільки досягнута деяка точка в розробці, коли всі системи відновлення або дублювання готові виконувати свої функції. Технічно реалізувати тести можна наступними шляхами:

– симулювати раптову відмову електрики на комп'ютері (знеструмити комп'ютер);

– симулювати втрату зв'язку з мережею (вимкнути мережевий кабель, знеструмити мережевий пристрій);

– симулювати відмову носіїв (знеструмити зовнішні носії даних);

– симулювати ситуацію наявності в системі невірних даних (спеціальний тестовий набір або база даних).

При досягненні відповідних умов збою і за результатами роботи систем відновлення, можна оцінити продукт з погляду тестування на відмову. У всіх перерахованих вище випадках, по завершенні процедур відновлення, має бути досягнуто певний необхідний стан даних продукту:

– втрата або псування даних в допустимих межах;

– звіт або систему звітів з зазначенням процесів або транзакцій, які не були завершені в результаті збою [3].

Варто зауважити, що тестування на відмову і відновлення – це вельми програмно–специфічне тестування. Розробка тестових сценаріїв повинна проводитися з урахуванням всіх особливостей тестованої системи. Беручи до уваги досить жорсткі

методи впливу, варто також оцінити доцільність проведення даного виду тестування для конкретного програмного продукту.

8. Конфігураційне тестування або Configuration Testing

Конфігураційне тестування (Configuration Testing) – спеціальний вид тестування, спрямований на перевірку роботи програмного забезпечення при різних конфігураціях системи (заявлених платформах, підтримуваних драйверах, при різних конфігураціях комп'ютерів і т.д.)

Залежно від типу проекту конфигурационное тестування може мати різні цілі:

– проект по профілізації роботи системи;

Мета тестування: визначити оптимальну конфігурацію устаткування, що забезпечує необхідні характеристики продуктивності та часу реакції тестованої системи.

– проект з міграції системи з однієї платформи на іншу;

Мета тестування: Перевірити об'єкт тестування на сумісність з оголошеним в специфікації обладнанням, операційними системами та програмними продуктами третіх фірм.

Рівні проведення тестування.

Для клієнт–серверних додатків конфигурационное тестування можна умовно розділити на два рівні (для деяких типів програм може бути актуальний тільки один):

– серверний;

– клієнтський.

На першому (серверному) рівні, тестується взаємодія випускаємого програмного забезпечення з оточенням, в яке воно буде встановлено:

– апаратні засоби (тип і кількість процесорів, обсяг пам'яті, характеристики мережі/мережевих адаптерів і т.д.);

– програмні засоби (ОС, драйвера і бібліотеки, стороннє ПЗ, що впливає на роботу програми і т.д.).

Основний упор тут робиться на тестування з метою визначення оптимальної конфігурації обладнання, що задовольняє необхідним характеристикам якості (ефективність, портативність, зручність супроводу, надійність).

На наступному (клієнтському) рівні, програмне забезпечення тестується з позиції його кінцевого користувача і конфігурації його робочої станції. На цьому етапі будуть протестовані наступні характеристики: зручність використання,

функціональність. Для цього необхідно буде провести ряд тестів з різними конфігураціями робочих станцій:

– тип, версія і бітность операційної системи (подібний вид тестування називається крос-платформенне тестування);

– тип і версія Web-браузеру, у випадку якщо тестується Web програма (подібний вид тестування називається крос-браузерні тестування);

– тип і модель відео адаптера (при тестуванні ігор це дуже важливо);

– робота програми при різних дозволах екрану.

Версії драйверів, бібліотек тощо (для JAVA додатків версія JAVA машини дуже важлива, теж можна сказати і для .NET додатків щодо версії .NET бібліотеки).

Розглянемо третій тип – види тестування,які пов'язані зі змінами.

Після проведення необхідних змін, таких як виправлення багу/дефекту, програмне забезпечення повинне бути перетестовано для підтвердження того факту, що проблема була дійсно вирішена. Нижче перераховані види тестування, які необхідно проводити після установки програмного забезпечення, для підтвердження працездатності програми або правильності виправлення дефекту:

1. Димове тестування або Smoke Testing.

Поняття димове тестування пішло з інженерної середовища:«При введенні в експлуатацію новогообладнання («заліза») вважалося, що тестування пройшло вдало, якщо з установки не пішов дим. »

В області ж програмного забезпечення, димове тестування розглядається як короткий цикл тестів, виконуваний для підтвердження того, що після складання коду (нового або відредагованого) встановлюване ПЗ стартує і виконує основні функції. Висновок про працездатності основних функцій робиться на підставі результатів поверхневого тестування найбільш важливих модулів програми на предмет можливості виконання необхідних завдань та наявності швидконаходжуваних критичних і блокуючих дефектів. У разі відсутності таких дефектів димове тестування оголошується пройденим, і ПЗ передається для проведення повного циклу тестування, в іншому випадку, димове тестування оголошується проваленим, і додаток йде на доопрацювання [1].

Аналогами димового тестування є Build Verification Testing і Acceptance Testing, що виконуються на функціональному рівні командою тестування, за результатами яких

робиться висновок про те, приймається чи ні встановлена версія програмного забезпечення в тестування, експлуатацію або на поставку замовнику.

Для полегшення роботи, економії часу і людських ресурсів рекомендується впровадити автоматизацію тестових сценаріїв для димового тестування.

2. Регресійне тестування або Regression Testing.

Регресійне тестування – це вид тестування спрямований на перевірку змін, зроблених в ПЗ або навколишньому середовищі (виправлення дефекту, злиття коду, міграція на іншу операційну систему, базу даних, веб сервер або сервер ПЗ), для підтвердження того факту, що існуюча раніше функціональність працює як і колись. Регресійними можуть бути як функціональні, так і нефункціональні тести.

Як правило, для регресійного тестування використовуються тест кейси, написані на ранніх стадіях розробки і тестування. Це дає гарантію того, що зміни в новій версії програми не пошкодили вже існуючу функціональність. Рекомендується робити автоматизацію регресійних тестів, для прискорення подальшого процесу тестування і виявлення дефектів на ранніх стадіях розробки програмного забезпечення.

Сам по собі термін «Регресійне тестування», залежно від контексту використання може мати різний зміст. Сем Канер, наприклад, описав 3 основних типи регресійного тестування:

– регресія багів (Bug regression) – спроба довести, що виправлена помилка насправді не виправлена;

– регресія старих багів (Old bugs regression) – спроба довести, що нещодавня зміна коду або даних зламало виправлення старих помилок, тобто старі баги стали знову відтворюватися;

– регресія побічного ефекту (Side effect regression) – спроба довести, що нещодавня зміна коду або даних зламало інші частини розроблюваного ПЗ.

3. Санітарне тестування і перевірка узгодженості/справності або Sanity Testing.

Санітарне тестування – це вузьконаправлене тестування для доказу того, що конкретна функція працює згідно заявленим в специфікації вимогам. Є підмножиною регресійного тестування. Використовується для визначення працездатності певної частини програми після змін вироблених в ній або навколишньому середовищу. Зазвичай виконується вручну.

Відмінність санітарного тестування від димового (Sanity vs Smoke testing).

У деяких джерелах помилково вважають, що санітарне та димове тестування – це одне і теж. Я вважаю, що ці види тестування мають «вектора руху», напрямки в різні боки. На відміну від димового (Smoke testing), санітарне тестування (Sanity testing) направлено вглиб перевіряємої функції, в той час як димове направлено вшир, для покриття тестами якомога більшого функціоналу в найкоротші терміни.

Висновки

На сьогодні професія тестувальника ПЗ є високо затребуваною. Існує більше 3 типів тестування, кожен з яких має свої підвиди. Найбільш поширені з них: функціональне, нефункціональне та повязане зі змінами тестування. Ця техніка контролю якості є невід'ємною частиною розробки програмного забезпечення, але нажаль скільки б часу воно не тривалось – усіх «багів» в програмі знайти неможливо.

ЛІТЕРАТУРА

1. Бейзер Б. Тестирование чёрного ящика. Технологии функционального тестирования программного обеспечения и систем. / Бейзер Б. – М. : Питер, 2014. – 320 с.
2. Йордан Е. Шлях камікадзе. Як розробнику програмного забезпечення вижити в безнадійному проекті. / Йордан Е. – М. : Лорі, 2012. – 290 с.
3. Канер К. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнес-приложений. / Канер Кем, Фолк Джек – К. : ДиаСофт, 2013. – 544 с.
4. Криспин Л. Гибкое тестирование: практическое руководство для тестировщиков ПО и гибких команд. / Лайза Криспин, Джанет Грегори – М. : «Вильямс», 2010. – 464 с.

Особенности типов тестирования программного обеспечения
Писанець К. К., Вовчук А. В.
Киевский национальный университет технологий и дизайна
Статья посвящена развитию тестирования, контроля качества в разработке программного обеспечения. Исследованы существующие методы тестирования. Обоснована актуальность тестирования на данный момент, из-за большого спроса на него.
Ключевые слова: *тестирование, типы тестирования, качество*

Features and types of software testing
Pisanets K. K., Vovchuk A. V.
Kiev National University of Technology and Design
The article is devoted to development testing, quality assurance in software development. We studied the existing methods and types of testing. The urgency of testing at the moment, due to the high demand for it.
Keywords: t*esting, testing type, quality*

