control of glass furnaces. Ceramics-Silikáty, № 50(1), 1–56.**3.** *Hill, J. H.* (2004). Adaptive control of an industrial float glass process. Int. Journal of Adaptive Control and Signal Processing.**4.** *Sulikova, V. A.* (2014). Fuzzy control algorithm glass melting process. VESTNIK OGU, № 3(164), 173–179.**5.** *Moon, U.* (2003). Hybrid algorithm with fuzzy system and conventional PI control for the temperature control of TV glass furnace. Control Systems Technology, IEEE Transactions, № 11(4), 548–554. **6.** *Jianling, Q.* (2010). Design of Glass Furnace Control System Based on Model-Free Adaptive Controller. Computer Modeling and Simulation, № 4, 130–133. **7.** *Rajarathinam, K.* (2014). Decentralised PID control tuning for a multivariable glass furnace by genetic algorithm. Automation and Computing (ICAC), 14–19. **8.** *Sinitsyin, I. N.* (2006). Kalman and Pugachev filters. M.: Universitetskaya kniga, Logos, 640 s.**9.** *Kolos, M. V.* (2000). Methods of optimal linear filtering. Pod red. V. A. Morozova. – Moscow.: Izd-vo MGU, 102 s.**10.** Balakrishnan, A. V. (2012). Kalman filtering theory. Izd-vo «Kniga po Trebovaniyu», 164 p.

Надійшла (received) 29.04.2015

УДК 665.9

Т. Б. ШАТОВСЬКА, канд. техн. наук, доц., ХНУРЕ, Харків
І. В. КАМЕНЄВА, канд. техн. наук, доц., ХНУРЕ, Харків

ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ ЗАСТОСУВАННЯ BDD-ФРЕЙМВОРКІВ У ТЕСТУВАННІ БЕЗПЕКИ WEB-ОРІЄНТОВАНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

На сьогоднішній день особливо важливі технології та методології, які дозволяють оперативно та ефективно створювати нові інструменти, при цьому на перше місце виходять технології та методології, які дозволяють з мінімальними затратами досягати бізнес-цілей замовника. Дана тема особливо актуальна в умовах поточної економічної кризи, істотно обмежує IT-бюджети і підсилило конкуренцію на ринку розробки замовленого програмного забезпечення. Виявлені деякі слабкості bdd-фреймворків у тестуванні безпеки web-орієнтованого програмного забезпечення

Ключові слова: bdd-фреймворк, web-додаток, Agile, модель, програмне забезпечення.

Вступ. Проекти розробки систем автоматизації, як і будь-які інші проекти, мають стандартний набір обмежень: терміни, ресурси, бюджет, якість. Як і в інших проектах, у проектах розробки програмного забезпечення стоїть завдання оптимізації цих показників.

Аналіз оптимізації процесу розробки систем автоматизації здійснюється за рахунок більш ефективного використання ресурсів, підвищення якості виконання робіт та зниження термінів шляхом застосування методології Agile, пропонованої як розширення методологій, побудованих за ітеративному принципом і націленої на швидку і ефективну розробку інноваційних продуктів.

Аналіз ефективності моделей розробки. Проекти розробки систем автоматизації, як і будь-які інші проекти, мають стандартний набір обмежень: терміни, ресурси, бюджет, якість. Як і в інших проектах, у проектах розробки програмного забезпечення стоїть завдання оптимізації цих показників.

Дана тема особливо актуальна в умовах поточної економічної кризи, істотно обмежує IT-бюджети і підсилило конкуренцію на ринку розробки замовленого програмного забезпечення.

Зараз особливо важливі технології та методології, які дозволяють оперативно та ефективно створювати нові інструменти, при цьому на перше місце виходять

технології та методології, які дозволяють з мінімальними затратами досягати бізнес-цілей замовника. Зміни що відбуваються на ринку вимагають впровадження нових підходів до створення програмного забезпечення. Якщо раніше представники замовника працювали з виділеними аналітиками, які передавали їхні вимоги розробникам, то тепер компанії прагнуть безпосередньо залучати зацікавлені сторони в дискусію з перших же етапів планування та розробки ПЗ. Активні узгодження на підставі бізнес-цілей дають можливість оперативно виявляти потреби замовника. Таким чином, виробляється нова стратегія, що передбачає спільну роботу над вимогами. Чим раніше - на етапах планування архітектури, розробки дизайну - виявляються помилки, тим дешевше обходиться їх виправлення і вартість всієї розробки ПЗ. Аналіз оптимізації процесу розробки систем автоматизації здійснюєеться за рахунок більш ефективного використання ресурсів, підвищення якості виконання робіт та зниження термінів шляхом застосування методології Agile, пропонованої як розширення методологій, побудованих за ітеративному принципом і націленої на швидку і ефективну розробку інноваційних продуктів. Agile характеризується полегшеними, неформальними і високо адаптивними процесами розробки, що беруть основу в теорії хаосу, теорії систем, системному мисленні, системної динаміці, теорії подвійних циклів навчання, навчальних організацій, адаптивних систем і базується на чотирьох основних принципах: ітеративна розробка, отримання зворотнього зв'язку, обмеження числа учасників команд розробки, гнучкість технологій розробки.

В даний час методологія Agile досить популярна і може сприйматися як шлях вирішення більшості проблем, що існують в проектах розробки ПЗ, проте, це далеко не так - дана методологія ефективна далеко не завжди.У розумінні і застосуванні ключових принципів Agile існують серйозні проблемні зони. При впровадженні необхідно уникнути ризику погоні за неможливим і спрямувати зусилля в напрямку, що дає реальні результати.

Мета аналізу - вироблення критеріїв, що дозволяють визначити ефективність застосування методології Agile в проектах розробки систем автоматизації. В роботі проводиться порівняльний аналіз методологій розробки з точки зору застосування в різних умовах, дані конкретні рекомендації із застосування методології Agile в проектах розробки система автоматизації та розібрані типові помилки і проблеми, що виникають при впровадженні Agile.

Методологічною базою для досліджень стали публікації в літературі, методики з впровадження методологій розробки програмного забезпечення, а також публікації у відкритих джерелах і статистичні дослідження, дані про хід виконання проектів провідних компаній світу. Хоча спектр методологій розробки програмного забезпечення та систем автоматизації дуже широкий, але так чи інакше, всі вони зводяться до 2 моделей: Водоспадної і ітеративної. Якщо розглядати методології не з точки зору розбиття на стадії виділення ролей і артефактів, що виробляються в процесі розробки, а з точки зору отримання значущих результатів та ефективності, має сенс порівнювати методології по мірі ітеративності, а також ступеня формальності в оформленні розробки.

Якщо розглянути найбільш поширені на даний момент у галузі методології розробки, то можна отримати наступний розподіл (табл. 1).

Таблиця 1 – Аналіз рівня ітеративності та формалізації методологій розробки ПЗ

Методології	Ступінь ітеративності	Ступінь формалізації
Структурні методології	Низька	Висока
ГОСТи 19, 34	Низька	Висока
ГОСТ 12207	Середня	Середня
RUP	Висока	Середня
MSF	Висока	Середня
Гнучкі (Agile) методології	Висока	Низька

Іллюстрація статистики поліпшень показників якості у результаті застосування Agile практик приведена у наступному блоці (табл. 2):

Таблиця 2 – Ефективність застосування Agile

Джерело	Результати застосування	Кіл-сть респондентів
2008 Version One	76% Agile-проектів є успішними 56% - приріст більш ніж 25% показника time-to-market 55% - зниження більш 25% вартості проектів 30% - зниження більш 25% дефектів	2319
2007 University of Maryland University college, UMUC	26% - приріст якісних показників на рівні 50%	250
2006 Version One	86% - приріст показника time-to-market 87% - підвищення показників продуктивності 92% - висока готовність до зміни приоритетів	722
2006 Amby Soft	44% - поліпшення показників вартості, продуктивності, якості 38% - ріст показника задоволеності клієнтів	4232

Классифікація можливих загроз WEB-додатків. Виділимо до дослідження необхідну групу классів можливих загроз для проведення подальшого прототипування та тестування базуючись на классифікації WEB-загроз від WEB-appsec. Та поділимо їх на дві групи як приведено у таблиці нижче (табл. 3):

Таблиця 3 – Класи загроз

Уразливості	Слабкості
1	2
Зловживання функціональними можливостями	Недостатня аутентифікація
Підбір	Недостатня авторизація
Підмна вмісту	Відсутність таймаута сессії
Передбачуване значення ідентифікатора сессії	Недостатня протидія автоматизації
Міжсайтове виконання сценаріїв	Недостатня перевірка процесу

Продовження таблиці 3

1	2
Відмова в обслуговуванні	Індексування директорій
Зворотній шлях в директоріях	Витік інформації
Передбачуване розташування ресурсів	-
Фіксація сессії	-
Ін’єкція SQL	-
Ін’єкція SSI	-
Ін’єкція XPath	-
Атака на функції форматування рядків	-
Ін’єкція LDAP	-
Виконання команд операційної системі	-
Логічні атаки	-
Небезпечне відновлення паролів	-
Переповнення буферу	-

Аналіз засобів об’єктно-поведінкових специфікацій. Застосування методів формального опису вимог є невід'ємною частиною сучасного процесу розробки складних програмних систем. Тестування не гарантує виявлення всіх помилок у продукті та не дає однозначної відповіді, чи відповідає продукт поставленим вимогам. Зважаючи на те, що обсяги і складність розробок постійно зростають, а кількість вимог досягає десятків тисяч, проблема автоматичної верифікації (перевірки правильності) набуває значної актуальності в індустріальних проектах. У цьому сенсі формальні методи мають найважливіше значення. Вони дозволяють створювати формальні вимоги й моделі поведінки систем з подальшою автоматичною верифікацією, генерацією програмного коду та набору тестів за заданими критеріями. Коректність отриманих результатів гарантується математичним апаратом, що опирається на досягнення алгебри, логіки і дискретної математики.

Перевірка на моделі – техніка, яка застосовується до моделей із скінченою множиною станів і перевіряє, що задані властивості виконуються на цій моделі. Тобто перевірка здійснюється як повний перебір простору станів, який має гарантовано закінчитися на скінченій моделі.

Сучасні розробки поєднують перевірку на моделі з доведенням теорем, а також застосовують статичний аналіз (аналіз програмного коду або моделі системи без моделювання її поведінки) для побудови абстракцій та тверджень певної логіки з подальшими доведеннями. Такими розробками є PVS (символьна перевірка на моделі, автоматичні доведення з використанням BDD, техніка переписування), STeP (перевірка на моделі з інтерактивним доведенням теорем на базі BDD), SLAM (перевірка на моделі з використанням методу предикатних абстракцій), Bandera (аналіз анотованих Java програм, для доведення тверджень використовуються алгоритми Нельсона – Оппена), Verisoft (аналіз програм на мовах C, C++, Java, використовується скорочення часткового порядку).

Основна мета формальних методів – допомога інженерам у розробці більш якісних систем. Але формалізми, з якими працюють наведені інструменти, досить часто є більш математичними і складними для розуміння інженерами, що стає перешкодою для їх впровадження в процес розробки програмного забезпечення. Тому актуальною проблемою є використання інженерних мов як вхідних формалізмів для систем верифікації. У даній роботі ця проблема вирішується завдяки розробленим алгоритмам трансляції таких розповсюджених стандартизованих мов моделювання, як MSC, SDL та UML, в мову базових протоколів, яка є основним формалізмом у роботі. Для розв'язання задач верифікації моделей, описаних базовими протоколами, використовуються методи алгебраїчного та інсерційного програмування, засоби автоматичного доведення теорем та перевірки на моделі. Велика ступінь ітеративності сприяє мінімізації ризиків, пов'язаних з часомвиходу продукту і відповідності очікуванням замовника. Велика ступінь формалізації сприяє отриманню заздалегідь відомого результату, дає можливість здійснення формального контролю, однак, не дає гарантій відповідності результатів реальним потребам на момент закінчення розробки. Тому BDD-концепт обраний як можливий шлях оптимізації вимог та максимальної відповідності вимог до рельної поведінки системи (рис. 1).

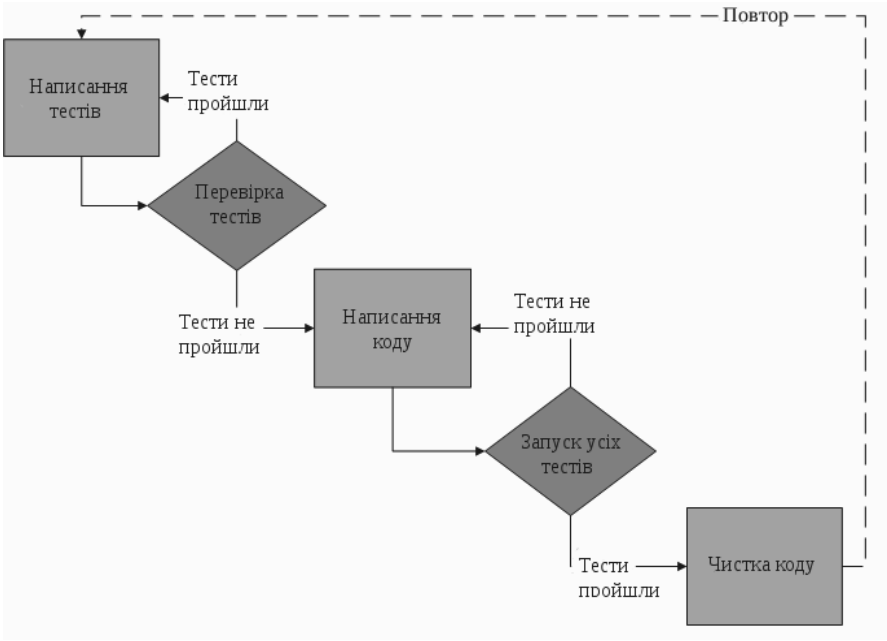

Рис. 1 – Графічна презентація циклу розробки TDD

Знайомство із BDD-концептом. Тестування, яке виконується розробником - це очевидно корисна річ. Тестування на ранній стадії, наприклад, під час написання коду - ще краще, особливо коли воно призводить до підвищення якості коду. Напишіть тести заздалегідь - і ви маєте шанс виграти "блакитну стрічку" переможця регати. Додаткові можливості для перевірки функціонування коду та його попередньої налагодження, без всякого сумніву, підвищують швидкість розробки. Але навіть знаючи все це, ми ще дуже далекі від того часу, коли написання тестів до написання коду стане загальним стандартом. Точно так само як TDD стало наступним етапом еволюції розвитку екстремального програмування (XP) і висунуло на перший план інфраструктури для Unit-тестування, наступний стрибок еволюції буде зроблено з того рівня, де знаходиться TDD. У цьому місяці пропонується зробити подібний стрибок в еволюції від TDD до його інтуїтивного родичу: behavior-driven development (BDD) - розробці, заснованої на функціонуванні.

Хоча підхід з попереднім тестуванням працює у багатьох, він підходить не для всіх. На кожного розробника програм, з успіхом застосовує TDD, знайдеться декілька розробників, які активно заперечують цей підхід. Незважаючи на числен-

ність інфраструктур тестування, таких як TestNG, Selenium і FEST, все одно знаходиться багато причин не виконувати тестування коду. Дві звичайні причини відмови від TDD - "у нас недостатньо часу для тестування" і "код занадто складний і важко перевіряється". Іншою перепоною для програмування з попередніми написанням тестів є сама концепція "тест пишеться до коду". Більшість розглядає тестування як відчутну дію, швидше конкретне, ніж абстрактне. Досвід підказує, що неможливо перевірити те, що ще не існує. Для деяких розробників, що залишаються в рамках цієї концепції, ідея попереднього тестування - просто оксюморон (рис. 2).

Рис. 2 – Діаграма TDD-методології

Але що якщо замість того, щоб думати в термінах написання тестів і тестування компонентів, почати думати про функціональність? Говорячи про функціональність, я маю на увазі як додаток повинен вести себе, фактично його специфікацію (рис. 3).

Рис. 3 – Діаграма BDD-методології

Якщо відволіктися від конкретної предметної області (структури даних) і замінити цю область іншою, наприклад, додатком для call-центру, суть процесу залишиться тією ж. Замовник або експерт в предметній області - каже, що саме система, функція або додаток повинні робити, а хтось на кшталт розробника використовує BDD для перевірки, що він правильно почув і реалізував вимоги клієнта.

Для багатьох розробників перехід від розробки, заснованої на тестах, до BDD виявиться досить розумним кроком. У розробці, заснованої на поведінці, немає необхідності думати про тести, досить сконцентруватися на вимогах до додатка і зробити так, щоб поведінка додатка відповідала цим вимогам. У цьому випадку використання BDD і JBehave допомогло легко реалізувати працюючу версію стека, засновану на специфікації замовника. Потрібно було просто слухати, що він говорить, а потім розробити стек, що відповідає його вимогам, думаючи при цьому в термінах поведінки. В ході процесу також вдалося виявити кілька аспектів стека, про які замовник забув. Беручі до уваги увесь рядок BDD-фремворків розроблений засновником концепції (Деном Норсом у 2003 році) можна виділити найбільш швидкий та прийнятний для опису терминів поведінки (табл. 4).

Таблиця 4 – Аналіз BDD-фреймворків

Технологія	Компіляція / Інтерпритація	Легкість кастомізації
RSpec / RBehave / Cocumber (Ruby)	Інтерпритація	Середня
JBehave (Java)	Компіляція	Низька
Lettuce (Python)	Інтерпритація	Висока
CBehave (C)	Компіляція	Низька
Spock (Groovy)	Компіляція	Середня
Speckflow (.NET)	Інтерпритація	Низька
Behat (PHP)	Інтерпритація	Середня

Висновки. У час вивчення ефективність застосування bdd-фреймворків були виявлені такі уразливості: зловживання функціональними можливостями, підбір, підміна вмісту, передбачуване значення ідентифікатора сесії, міжсайтове виконання сценаріїв, відмова в обслуговуванні, зворотній шлях в директоріях, передбачуване розташування ресурсів, фіксація сесії, ін'єкція SQL, ін'єкція SSI, ін'єкція XPath, атака на функції форматування рядків, ін'єкція LDAP, виконання команд операційної системі, логічні атаки, небезпечне відновлення паролів, переповнення буферу.

Також, під час тестування були виявлені деякі слабкості bdd-фреймворків у тестуванні безпеки web-орієнтованого програмного забезпечення, такі як: недостатня аутентифікація, недостатня авторизація, відсутність таймаута сесії, недостатня протидія автоматизації, недостатня перевірка процесу, індексування директорій, витік інформації.

Список літератури: 1. *Бек К.* Екстремальне програмування [Текст] / К. Бек. – Санкт-Петербург: Видавництво "Питер", 2003. – 224 с.**2.** *Фаулер М., Маккавеев С.* Рефакторинг. Покращення існуючого коду [Текст] / М. Фаулер, С.Маккавеев – М.: Символ-плюс, 2008. – 432 с.**3.** *Бек К., Чеботарев А.* Шаблони реалізації корпоративних додатків [Текст] / К. Бек, А. Чеботарев – М.: Вильямс, 2008. – 176с.**4.** *Попендик М., Попендик Т., Меженной О.* Безперервна інтеграція. Поліпшення якості програмного забезпечення і зниження ризику [Текст] / М. Попендик, Т. Попендик, О. Меженной – М.: Вильямс, 2008. – 240 с.**5.** *Попендик М., Попендик Т., Меженной О.* Бережливе виробництво програмного забезпечення. Від ідеї до прибутку [Текст] / М. Попендик, Т. Попендик, О. Меженной – М.: Вильямс, 2010. – 256 с.**6.** *Кон М., Красиков И.* Scrum. Гнучка розробка ПЗ [Текст] / М. Кон, И. Красиков – М.: Вильямс, 2011. – 576 с.**7.** *Хамбл Д., Фарли Д, Сисонюк А.* Безперервне розсортування ПЗ. Автоматизація процесів збору, тестування та поставки нових версій програм [Текст] / Д. Хамбл, Д. Фарли, А.Сисонюк – М.: Вильямс, 2011. – 432 с.**8.** *Субраманиам В., Хант Є., Лукач Е.* Этюди на тему швидкої розробки програмного забезпечення [Текст] / В. Субраманиам, Є. Хант, Е. Лукач – М.: Вильямс, 2009. – 208 с.**9.** Відкрита вікі-енциклопедія, "Всесвітня павутина" - Режим доступу: www/ url: http://uk.wikipedia.org/wiki/WWW**10.** Відкрита вікі-енциклопедія, "Програмний каркас" - Режим доступу: www/ url: http://uk.wikipedia.org/wiki/Каркас_(програмування)**11.** Відкрита вікі-енциклопедія, "Структури вирішення комплексних задач" - Режим доступу: www/ url: http://uk.wikipedia.org/wiki/Фреймворк

Bibliography (transliterated):1. *Beck, K.* (2003). Ekstremalne programuvanny. St. Petersburg: Vidavnitstvo "Peter", 224.**2.** *Fowler, M., Refactoring, S. Maccabees* (2008). Improving existing code. Moscow: Symbol Plus, 432.**3.** *K. Beck, Chebotarev* (2008). Templates implementation of enterprise applications. Moscow.: Williams, 176.**4.** *Popendik, M. Popendik T., O. Mezhennaya* (2008). Continuous Integration. Improving software quality and reduce the risk. M.: Williams, 240.**5.** *Popendik, M. Popendik, T., O. Mezhennaya* (2010). Lean manufacturing software. From idea to. Moscow: Williams, 256.**6.** *Cohn, M., Krasikov, I. Scrum.* (2011). Flexible *software development* Moscow: Williams, 576.**7.** *Humble, D., Farley, D., Sisonyuk, A.* (2011). Continuous deployment of software. Process automation assembly, testing and implementation of new versions of software. Moscow: Williams, 432.**8.** *Subramaniam, V., Hunt, Je., Lukacs, E.* (2009). Study on rapid software development. M.: Williams, 208.**9.** Vidkrita viki-Enzyklopädie "Vsesvitnya pavutina" - Mode of access: www / url: http://uk.wikipedia.org/wiki/WWW.**10.** Vidkrita viki-Enzyklopädie "software framework" - Mode of access: www / url: http://uk.wikipedia.org/wiki/Karkas_(prigramming).**11.** Vidkrita viki-Enzyklopädie, "Structure virishennya complex tasks" - Mode of access: www / url: http://uk.wikipedia.org/wiki/ framework.

Надійшла (received) 29.04.2015