

УДК 004.415.53(045)

Боровик В.М., канд. техн. наук,
Труш О.І., канд. техн. наук,
Супрунова Т.В.

ТЕСТУВАННЯ У СИСТЕМАХ ТА МЕТОДОЛОГІЯХ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Національного авіаційного університету

Проведено огляд найпоширеніших у світі методологій розробки великих програмних продуктів та досліджено роль процесу тестування в них

Вступ

Тестування є невід'ємним етапом процесу розробки будь-якого програмного продукту. Компанії-розробники, як невеликі так і зі світовим ім'ям, все більшої уваги приділяють якості своїх продуктів, їх зручності та надійності. Менеджери компаній детально вивчають існуючі системи розробки з метою виявлення найбільш вдалої, яка дозволить організувати взаємодію між програмуванням і тестуванням.

Мета статті

Дослідити існуючі методології розробки та тестування програмного забезпечення та проаналізувати роль тестування в кожній з цих систем.

Постановка проблеми

Розглянемо методології, які об'єднуються під назвою «гнучка розробка програмного забезпечення» (*Agile Software Development – Agile*). Під цим терміном розуміють клас методологій розробки програмного забезпечення, що базується на ітеративних процедурах. Термін з'явився в 2001 році, коли був написаний так званий «Маніфест гнучкої розробки».

Більшість таких методологій націлені на мінімізацію ризиків шляхом зведення розробки до серії коротких циклів, що мають назву ітерацій, які, зазвичай, тривають один-два тижні. Кожна ітерація сама по собі виглядає як програмний проєкт в мініатюрі, що включає всі етапи: планування, аналіз вимог, проєктування, кодування, тестування і документування. Хоча окрема ітерація, як правило, недо-

статня для випуску нової версії продукту, мається на увазі, що гнучкий програмний проєкт готовий до випуску в кінці кожної ітерації. Після закінчення кожної ітерації проєктувальна група виконує переоцінку пріоритетів розробки.

Agile акцентує увагу на безпосередньому спілкуванні членів проєктувальної групи, більшу частину, якої складають програмісти та тестувальники. Зазвичай, сюди включаються і кілька осіб «замовника», які безпосередньо ставлять задачу перед групою, визначають перелік вимог до продукту і оцінюють якість розробки та її відповідність поставленій задачі. В ролі «замовника» можуть виступати менеджери проєкту, бізнес-аналітики або клієнти, тощо.

Принципи Agile

Agile – сімейство процесів розробки, а не єдиний підхід у розробці програмного забезпечення, що визначається «*Agile Manifesto*». *Agile* не включає практики, а визначає цінності та принципи, якими керуються проєктувальні групи:

- забезпечення замовника за рахунок раннього та безперебійного постачання програмного забезпечення (ПЗ);
- реалізація зміни вимог;
- часте постачання робочого ПЗ;
- щоденне спілкування замовника з розробниками подовж проєкту;
- включення в проєкт мотивованих співробітників;
- постійно надання уваги поліпшенню технічної майстерності і організація зручного дизайну;

– реалізація адаптацій, з можливістю змін .

Серед методологій, які притримуються принципів *Agile Manifesto*, є такі як *Scrum*, Екстремальне програмування (*Extreme programming* чи *XP*).

Досить поширеною у світі та популярною серед розробників веб-орієнтованого ПЗ є система *Scrum*, якою успішно користується для організації розробки і тестування своїх продуктів *Microsoft*.

Scrum-методологія

Scrum-методологія управління проектами для гнучкої розробки програмного забезпечення. *Scrum* чітко робить акцент на якісному контролі процесу розробки (рис.1).

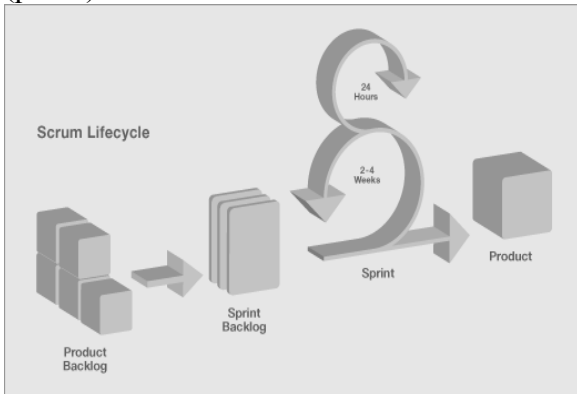


Рис.1.Життєвий цикл Scrum

Цей підхід вперше описали проєктувальники Гіротака Такеучі та Ікуджіро Нонака. Вони відзначили, що це проєкти, над якими працюють невеликі крос-функціональні проєктувальні групи .

Вперше метод *Scrum* було представлено на загальний огляд задокументованим, чітко сформульованим та описаним спільно Сазерлендом та Швабером, які протягом декількох років працювали разом, щоб обробити та представити весь досвід і найкращі практичні зразки для індустрії, як одне ціле.

Scrum – це основа процесу, який включає набір методів і попередньо визначених ролей. Головна складова методології – *Scrum Master*, що опікується процесами, веде їх і працює як керівник

проєкту, що представляє інтереси кінцевих користувачів та ін (рис.2 , рис.3).



Рис.2.Scrum–дошка зі завданнями

Протягом кожного спринту (15-30 денного періоду, тривалість якого визначається командою), працівники здійснюють функціональний ріст програмного забезпечення.

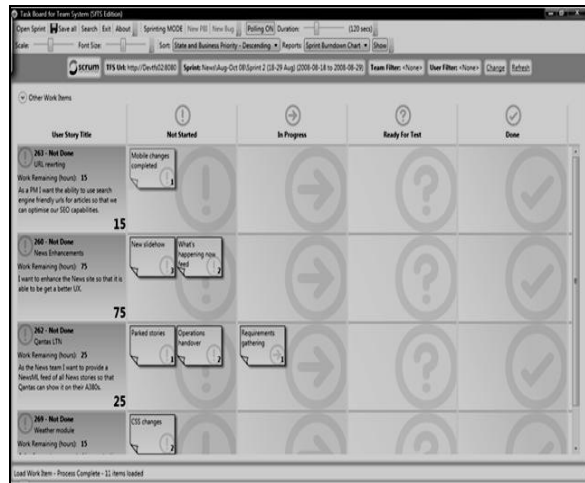


Рис. 3. Електронна Scrum-дошка

Набір можливостей які реалізуються кожного спринту, надходять із етапу життєвого циклу, що визначає документацію запитів на виконання робіт (*Product backlog*), який має найвищу пріоритетність за рівнем вимог до роботи, що повинна бути виконана.

Запити на виконання робіт (*Backlog items*), що визначені протягом наради-планування спринту переміщуються в наступний етап спринту.

Протягом спринту проєктувальна група виконує визначений фіксований

список завдань – *Backlog items*. Впродовж цього періоду ніхто не має права змінювати перелік запитів на виконання робіт, що слід розуміти, як заморожування вимог протягом спринту.

Екстремальне програмування

Екстремальне програмування (*Extreme Programming – XP*) – популярна методологія розробки програмного забезпечення.

Технологія XP була розроблена Кентом Бекем, Уардом Каннінгхемом та Роном Джеффріесом.

Головною метою XP є скорочення вартості неочікуваних змін. У традиційних методах розробки вимоги до розвитку системи визначаються на початку роботи над проектом і часто виправляються пізніше. Це приводить до того, що вартість проекту через зміни буде більшою за заплановану, традиційна особливість для програмного забезпечення, що проектується.

XP використовується для скорочення вартості змін, завдяки представленню простих значень, принципів і методів. При використанні системи «Екстремальне Програмування», проект повинен стати гнучкішим щодо змін і представляється різними складовими.

Парне програмування

Парне програмування припускає, що весь код створюється парами програмістів, які працюють за одним АРМ. Один з них працює безпосередньо з текстом програми, формує код, другий оглядає його роботу і спостерігає за загальним кодом програміста. При необхідності клавіатура вільно передається від одного програміста до іншого.

Протягом роботи над проектом рекомендується пари програмістів змінювати, щоб кожен з них мав повне уявлення про всю систему, що посилює взаємодію.

Колективне володіння кодом

Колективне володіння означає, що кожен член групи несе відповідальність за весь вихідний код. Таким чином, кожен має право вносити зміни в будь-яку час-

тину програми. Важлива перевага колективного володіння кодом в тому, що прискорюється процес розробки, оскільки при появі помилки її може усунути будь-який програміст. Хоча при цьому збільшується ризик появи помилок, що вносяться програмістами, які вважають, що знають, що роблять. Взагалі існують UNIT-тести, що вирішують цю проблему: якщо не розглядати залежності, що породжують помилки.

Практично роль замовника в XP відтворює не платник рахунків, а користувач системи, тобто в XP основним принципом є те, що замовник постійно на зв'язку.

Канбан

Канбан (Kanban Development) система візуалізації технологічних процесів компанії «Тойота».

Мета Kanban – зменшення виконуваної в даний момент роботи (*Work in progress*).

Система Kanban - це ще більш гнучка методологія, ніж Scrum і XP, тому проектувальні групи повинні бути ще більш готовою до гнучкої роботи, ніж навіть групи, що використовують Scrum і XP.

Kanban відрізняється від Scrum в першу чергу орієнтацією на завдання, а в Scrum основна орієнтація проектувальної групи - це успішне виконання спринтів.

В Kanban спринтів немає, група працює над завданням з самого початку і до завершення. Закриття завдання робиться тоді, коли воно готове. Презентація виконаної роботи – теж. Все, що потрібно від менеджера - управління проектом тобто додавати завдання або змінювати їх пріоритет.

Проектувальна група для роботи використовує Канбан-дошку, структура якої представлена в матричному вигляді:

1. Стовець «Цілі проекту». Розміщуються загальні цілі проекту, їх загальна наглядна візуалізація.

2. Стовець «Черга (порядок) завдань». Тут зберігаються завдання, які готові до початку їх виконувати. Як пра-

вило завжди для виконання береться верхня, пріоритетна задача та її картка переміщується в наступний стовпець.

3. Стовпець «Опрацювання дизайну». Цей та інші стовпці до «Закінчено» можуть змінюватися, тому що саме проєктувальна група вирішує, які кроки проходить завдання до стану «Закінчено». Наприклад, в цьому стовпці можуть перебувати завдання, для яких дизайн коду або інтерфейсу ще не ясний і обговорюється, по закінченню обговорення, завдання пересувається в наступний стовпець.

4. Стовпець «Розробка». Тут завдання знаходиться до завершення розробки функціональності. Після завершення завдання пересувається в наступний стовпець, якщо архітектура не вірна чи не точна, то повертається в попередній стовпець.

5. Стовпець «Тестування». Особливо важливий процес. У цьому стовпці завдання знаходиться, поки проходить процес тестування. Якщо знайдені помилки - повертається в стовпець «Розробка». Якщо ні - пересувається далі.

6. Стовпець «Завантаження». У кожного з проєктів цей термін має різне значення. В одному випадку це значить викласти нову версію продукту на сервер, а у іншому - просто помістити код в репозиторій.

7. Стовпець «Закінчено». Сюди картка переводиться тільки після повністю закінчення всіх робіт по завданню.

У будь-якому проєкті трапляються термінові завдання. Заплановані чи ні, але такі, які потребують негайного виконання. Для таких екстрених випадків існує можливість виділення спеціального місця (наприклад, як зазначено в оригіналі опису - «Expedite»). У «Expedite» можна помістити одну термінову задачу і проєктувальна група повинна почати її виконувати негайно з терміновим завершенням. Якщо з'являється ще одна - вона повинна бути додана в стовпець «Черга завдань».

Конкретна кількість підбираються експериментально, але вважається, що

вони повинні залежати від числа розробників в проєктувальній групі. Наприклад, якщо в групі 8 програмістів, то в стовпець «Розробка» можна помістити цифру 4. Це означає, що одночасно програмісти будуть робити не більше 4-х завдань, а значить у них буде забагато часу для спілкування та обміну досвідом.

Якщо поставите цифру 2, то 8 програмістів, які займаються цими двома завданнями, можуть втрачати занадто багато часу на обговореннях.

Якщо поставити 8, то кожен буде займатися своїм завданням і деякі завдання будуть затримуватися на дошці надовго, що протиречить меті Kanban - зменшення часу проходження завдання від початку до стадії готовності.

Визначити ці ліміти можна апріорі, для початку розділити число розробників на 2 і виявити, як це спрацьовує у даній проєктній групі. Потім ці числа можна підігнати під групу з конкретним проєктом.

Під «розробниками» розуміються не тільки програмісти, а й інші фахівці. Наприклад, для стовпця «Тестування» розробники - це тестувальники, і їх основне обов'язкове завдання це тестування вже розробленої частини проєкту.

Завдання на такій дошці - це не просто завдання, а так звана мінімальною маркетинговою одиницею, тобто одиниця, яку можна запропонувати замовникам.

Переваги Kanban-дошки з лімітами:

По-перше, зменшення числа паралельно виконуваних завдань, що значно зменшує час виконання кожної окремої задачі. Немає потреби перемикає контекст між завданнями, відслідковувати різні сутності, планувати їх і т.д., за принципом, робиться тільки те, що заплановано. Немає потреби влаштовувати спринти, планінги, тому що планування вже зроблено в стовпці «Черга (порядок) завдань», а детальне опрацювання завдання починається після початку виконання завдання.

По-друге, відразу реалізується візуалізація проблеми. Наприклад, якщо тес-

тувальники не справляються з тестуванням, то вони дуже швидко заповняють весь свій стовпець і програмісти, які закінчили нове завдання, вже не зможуть перемістити його в стовпець тестування, так як він заповнений. В цьому випадку шляхи вирішення цієї проблеми очевидні. Наприклад, програмісти можуть допомогти тестувальникам завершити одну з задач тестування і тільки тоді пересунути нове завдання на звільнене місце. Це дозволить виконати обидва завдання швидше.

По-третє, можна обчислити час на виконання усередненого завдання. На картці можна позначати дату, коли вона потрапила в чергу завдань, потім дату, коли її взяли в роботу і дату, коли її завершили. По цим трьом точкам для для 10 завдань можна вже порахувати середній час очікування в чергу завдань і середній час виконання завдання. А з цих цифр менеджер або замовник може вже провести необхідні розрахунки.

Взагалі система Kanban описується трьома основними правилами:

Візуалізацією виробництва – декомпозицією проекту на завдання, кожне фіксується на картці.

Обмеженням WIP (Work in progress), або роботи, виконуваною одночасно на кожному етапі виробництва.

Вимірюванням часу циклу (середній час на виконання однієї задачі).

В Scrum, наприклад, – 9 основних правил, а в Екстремальному програмуванні XP – 13 основних правил.

Висновки

З наведеного огляду можна зробити висновок, що навіть найвідоміші і найдосконаліші системи розробки і тестування програмного забезпечення мають серйозні недоліки.

Наприклад, загальновідома проблема Scrum - це великі витрати від обговорень, зустрічей і великі втрати часу на стиках спринтів (коли як мінімум один день йде на закриття одного спринту, а потім ще один на відкриття нового спринту). І якщо, як правило спринт - 2 тижні,

то 2 дні з 2 тижнів - це 20%. У результаті мало не 30-40% часу при застосуванні Scrum витрачається на підтримку самого процесу, часто на щоденні наради-планування спринту.

Загалом, при виборі підходящої методології важливо ознайомитись із принципами системи, дослідити всі її переваги, недоліки і врахувати потреби, особливості конкретної проектувальної групи та проекту.

Список літератури

1. Луиза Твмре. Введение в тестирование программного обеспечения.—М.: Изд-во Вильямс, 2003. – 368 с.

2. Скотт Амблер. Гибкие технологии:экстремальное программирование и унифицированный процесс. – СПб.: Питер, 2005.– 416 с.

3. Рекс Блэк. Ключевые процессы тестирования.Планирование, подготовка, проведение, совершенствование. – М.:Изд-во Лори, 2006. – 544 с.

4. Кент Бек, Мартин Фаулер. Экстремальное программирование: планирование. – СПб.:Питер, 2003. – 144 с.

5. Хенрик Книберг. Scrum и XP: заметки с передовой. М.:– InfoQ,2007. – 168 с.

Подано до редакції 17.11.2010