УДК 004.051
DOI https://doi.org/10.32838/TNU-2663-5941/2020.6-1/13

Киричек Г.Г.
Національний університет «Запорізька політехніка»

Чубіч А.І.
Національний університет «Запорізька політехніка»

ВИКОРИСТАННЯ KANBAN-МЕТОДУ В ОРГАНІЗАЦІЇ РОЗРОБЛЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Використання Kanban у розробці програмного забезпечення щороку набирає популярності. Гнучкість методу дозволяє пристосовуватися до постійних змін у процесі роботи та швидко переключатися між різними етапами розробки. У роботі проведено аналіз різновидів Kanban-дошок, які використовуються у розробці програмного забезпечення, та наведено процес організації роботи команди за основними принципами Kanban-методу. Метою роботи є проведення досліджень і реалізація методу роботи команди з розробки програмного забезпечення за принципами Kanban-методу. Об'єктом дослідження є процес організації роботи команди у розробці програмного забезпечення. Предметом є моделі, методи, алгоритми та інструменти реалізації Kanban-методу. Авторами розглянуто реалізацію загальних Kanban-підходів agile-методології та lean-мислення, без жорстких правил, але із принципами, на які можна посилатися. Для Kanban існують тільки спеціальні методи управління, але вони не є стандартизованими. Отже, удосконалення процесу lean-розробки є достатньо актуальним за його використання для реалізації програмного забезпечення. Kanban передбачає послідовні дії для удосконалення процесу організації розробки й отримання гарного результату. Система Kanban передбачає максимальне скорочення Lead Time, а саме часу роботи над завданням на всіх етапах. Тому підхід до виконання задач реалізується з орієнтацією на дошку, потік завдань і виявлення проблемних місць. Основним завданням є вдосконалення і реалізація процесу роботи таким чином, щоб команда була готова до того, що вимоги до системи та пріоритети задач можуть змінюватися декілька разів на день. Процес підтримки комунікацій між різними командами також потребує чіткої схеми функціонування для швидкого обміну інформацією про виконану роботу. У роботі реалізована дошка візуалізації завдань з оптимальним набором статусів для ефективної роботи. Наведено графік, що демонструє ефективність методу залежно від певних факторів.

Ключові слова: *Kanban-дошка, метод, алгоритм, пріоритет, програмне забезпечення.*

Постановка проблеми. У кожного програмного забезпечення є життєвий цикл – етапи, через які воно проходить із початку створення до кінця розробки та впровадження. Методологія розробки включає набір методів з управління розробкою: правила, техніки та принципи, які роблять її більш ефективною. Останнім часом набирають популярності так звані гнучкі моделі та методи розробки, що передбачають розбиття проекту на невеликі робочі частини, які мають назву історії [1]. Одним із найпопулярніших прикладів гнучких підходів є Kanban-метод. Однак для ефективної роботи з використанням цієї системи потрібно реалізувати чітку схему циклів розробки програмного забезпечення.

Аналіз останніх досліджень і публікацій. Сьогодні існує велика кількість методів і способів розробки [2; 3]. Одними з найпопулярніших є Scrum і Waterfall. Scrum передбачає розробку через невеликі ітерації (два-чотири тижні), за які команда виконує чітко визначений об'єм задач [2]. Scrum забезпечує високий рівень взаємодії між членами команди та мотивування до самоорганізованої плідної роботи. В основі методу Waterfall лежить послідовний перехід від одного етапу на другий без пропусків і можливості повернутися назад. Це забезпечує зрозумілу і просту структуру процесу розробки та стабільність завдань, що виконуються [4]. Система Kanban передбачає використання дошки: фізичної чи електронної. Дошка є обов'язковим елементом для гнучкої методології. Кожен член команди отримує до неї доступ в будь-який час і бачить, на якому етапі перебуває те чи інше завдання. Наявні приклади дошок передбачають розподіл на основні категорії: планування, розробку, тестування та реліз [1]. Такий підхід дозволяє чітко розуміти, на якому етапі знаходиться команда, але у процесі розробки дуже часто виникає необхідність повернутися назад на будь-який етап [5]. У роботі представлено цикл розробки програмного забезпечення, що передбачає вільний перехід між основними етапами розробки,

а також описано процедури, які потрібно виконувати команді для ефективної роботи за цим методом. Також авторами визначені основні ролі спеціалістів та задачі, які повинен виконувати кожен із них [6].

Задачі постійно передаються від однієї команди розробників до іншої. Крім цього, Kanban реалізує ідею потоку як виробничого процесу, де немає простою незавершених завдань [7]. Тобто над однією задачею можуть працювати послідовно декілька спеціалістів. У цьому разі попередні помилки очевидні та виявляються миттєво. Це дозволяє уникнути зайвих витрат, підвищує якість розробки та скорочує терміни її виконання [8].

Постановка завдання. Метою роботи є проведення досліджень і реалізація методу роботи команди з розробки програмного забезпечення за принципами Kanban-методу. Об'єктом дослідження є процес організації роботи команди при розробці програмного забезпечення. Предметом є моделі, методи, алгоритми та інструменти реалізації Kanban-методу. Основним завданнями роботи є реалізація методу ефективної роботи над розробкою програмного забезпечення з написанням вимог до впровадження системи. Під час розробки необхідно вирішити такі задачі: визначити основні принципи роботи, розробити статуси для реалізації Kanban-дошки, визначити основні типи завдань і реалізувати цикл змін статусів для задач і процесу взаємодії команди [9].

Виклад основного матеріалу дослідження. Kanban-метод передбачає обговорення продуктивності в режимі реального часу і повну прозорість робочих процесів. Етапи роботи візуально представлені на Kanban-дошці, що дозволяє членам команди бачити стан кожного завдання у реальному часі. Зазвичай команда складається з таких спеціалістів: власника продукту (Product Owner), програмістів і головного розробника (Developer Lead), спеціалістів із тестування (QA) і головного QA (QA Lead), скрам-майстра (Scrum Master), бізнес-аналітика (Business Analyst) і продукт-архітектора (Product Architect).

Після затвердження проектних рішень, прийнятих на етапі проектування, команда починає розробку. Весь функціонал розподіляться між програмістами, котрі реалізують алгоритми, пишуть вихідний код, виконують компіляцію і налагодження коду. У проекті є план процесу робіт. Спочатку він аналізується і поділяє дошку на об'єкти, які відображають етапи [9].

Виділимо статуси для завдань: To do, In Progress, Code Review, Ready QA Testing, QA Under Test, Ready UAT Testing, UAT Under Test, UAT Done, Expected та Done (рис. 1).

Система Kanban передбачає максимальне скорочення Lead Time, тобто часу роботи над завданням на всіх етапах. У зв'язку з цим підхід до виконання задач реалізується з орієнтацією на дошку, фокусуванням на потоці завдань і виявленням проблемних місць. Тобто команда пересувається по об'єктах із завданнями справа наліво й обговорює, яким чином можна швидше перевести задачу на наступний етап. Kanban-дошка з візуалізацією ситуації на проекті дозволяє обговорювати не конкретні дії членів команди, а й поточний стан завдань і їх статус у потоці виконання.

Виділимо п'ять основних типів завдань, які використовуються під час роботи над проектом: завдання (task), історію (story), епік (epic), поліпшення (improvement) і баг (bug). На початку циклу розробки створюється проектна документація із вимогами до системи. Вона повинна чітко та детально відображати весь функціонал системи та після погодження бути оформлена бізнес-аналітиками у вигляді задач. Спочатку створюється епік – велике завдання, яке повністю описує окремий новий функціонал. Кожен епік складається з історій (описів вимог). Кожна історія реалізує окремий функціонал і може не входити до складу епіків, а бути окремим самостійним об'єктом, що описує новий функціонал або зміни поточного.

Крім того, створюються покращення (improvements) – вимога до оновлення або невеликої зміни частини функціоналу. Вони можуть додаватися на етапі розробки або тестування. Тестувальники вносять свої пропозиції, які можуть спростити роботу із системою та зробити її більш комфортною для кінцевого користувача. Також часто баги, що знаходяться на етапі тестування, перекваліфіковують у поліпшення. Баг є помилкою в системі, через яку програма видає неправильну

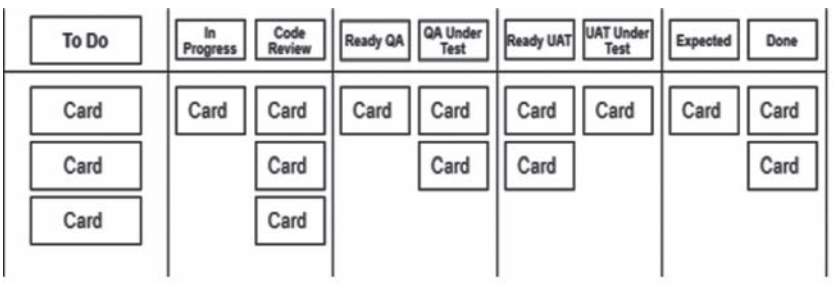

Рис. 1. Схема дошки за методом Kanban

поведінку і, як наслідок, результат. Більшість багів виникають через помилки, допущені розробниками у вихідному коді або при реалізації дизайна. Також деякі баги виникають через некоректну роботу компілятора, що виробляє некоректний код.

Баги розрізняють за ступенем критичності та пріоритетом. За ступенем критичності виділяють: блокуючі (blocker), які роблять неможливою подальшу роботу з додатком; важливі (major), через які система не функціонує належним чином; нормальні (normal), що призводять до некоректної роботи окремих компонентів системи; малозначущі (minor) або невеликі баги.

За пріоритетністю виділяють баги: fix in release – виправити в новій версії продукту; must fix – терміново виправити (блокуючі, які усувають до виходу нової версії); fix if time – виправити, якщо дозволяє час; never fix – ніколи не виправляти (є неактуальними або прийнятні для системи). Оскільки Kanban не передбачає жорстких, обмежених певних часом етапів, під час яких потрібно реалізувати ту чи іншу задачу, рішення про початок роботи із завданням приймається на командних мітингах (основа зворотного зв'язку). Регулярність задає ритм, за яким проходить потік роботи. Виділимо основні типи мітингів, такі як:

− щоденна зустріч, де обговорюється статус поточних завдань;

− зустріч по плану робіт (раз на два тижні);

− зустріч по покращенню сервісу (раз на два тижні) – обговорюється якість системи та її покращення;

− зустріч із організації процесу роботи (раз на два тижні) – обговорюються проблеми, що виникли під час організації процесу роботи (займається Agile-мастер, котрий координує роботу команди);

− зустріч по обзору стратегії (раз на місяць-квартал) – обговорюються зміни у стратегії.

Взаємодія команди відбувається за схемою, зображеною на рис. 2.

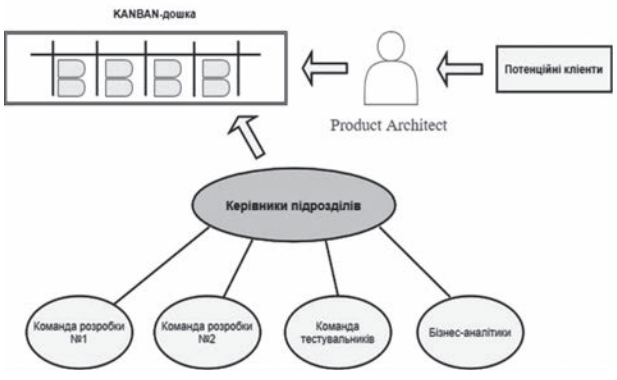

Рис. 2. Взаємодія Kanban-команди

Така модель забезпечує особисте спілкування членів команд кожного напряму з керівником підрозділу. Члени команди самостійно організовують свою роботу, використовуючи Kanban-дошку. Керівник займається пріоритезацією завдань і вирішенням виникаючих проблем.

Визначимо процес розробки на прикладі невеликої частини додатку, що виконує просте математичне ділення двох чисел із плаваючою крапкою.

```
int main()
{
float x = 0;
float y = 0;
std::scanf("%f %f", &x, &y);
float z = -x / y;
std::printf("Result: %f \n", z);
}
```

Product Architect аналізує побажання потенційних клієнтів щодо функціоналу та інтерфейсу додатку. На основі отриманих даних створюється проектна документація з описом вимог. Далі оформлюється Kanban-дошка, створюються задачі та додається команда.

Для нових створених задач автоматично призначається статус To do. Цей статус означає, що задача створена, але поки не розпочата робота над нею. Цикл змін статусів відображено на рис. 3.

Спеціаліст, на якого призначена задача, змінює статус на In Progress, коли починає виконання. На етапі розробки програміст, який розробляє або виправляє функціонал, описаний у завданні, вносить зміни на dev-оточенні та залишає свої коментарі про результати виконання.

Розробники розроблюють окремо невеликі частини функціоналу та проводять юніт-тестування, тобто переконуються, що окремо взята частина додатку працює. Після цього статус завдання потрібно змінити на Code Review [9]. Після встановлення цього статусу на задачу автоматично призначається головний розробник, який перевіряє розроблений код. Якщо текст коду задовольняє вимоги, зміни додаються до тестового оточення. Задачі призначається статус Ready QA Testing.

Розробник може призначити задачу на тестувальника, котрий її створював або вже працював із нею [10]. Також задачу можна залишити непризначеною (Unassigned), і вільний тестувальник зможе почати роботу над нею. Коли починається етап тестування, QA-спеціаліст змінює статус на QA Under Test. Головна задача тестувальника – перевірити, що нова частина додатку працює та коректно взаємодіє з усією системою. Якщо під

час перевірки виникають питання або тестувальник знаходить баги, вони оформлюються в окремих задачах зі статусом To do.

Далі продукт-архітектори та розробники вивчають проблему та приймають рішення про доцільність її вирішення. Коли тестування задачі закінчено, проблеми вирішені та функціонал відповідає поставленим до нього вимогам, тестувальник переміщає задачу у статус Ready UAT Testing.

Коли починається цикл UAT тестування, задачі переводяться спеціалістами у статус UAT Under Test, і призначаються відповідні тестувальники. UAT тестування – це приймальне, призначене для користувача тестування. Будь-яка розробка або доопрацювання програмного забезпечення проходить завершальну стадію UAT-тестування. Тестування проводиться бізнес-користувачами прийнятої системи, тобто потенційними клієнтами, але на практиці часто його проводять QA-спеціалісти. Перевірки проводять на спеціальному оточенні, яке містить у собі всі зміни та доповнення, що відбувалися на етапах розробки та QA-циклу. Якщо знаходяться проблеми на етапі UAT-тестування, приймається рішення про їх виправлення у поточному релізі, перенесення на наступний або визнання проблеми особливою можливістю системи та внесення відповідних правки в релізну документацію. Після успішного завершення перевірки задачі переводяться до UAT Done.

Також передбачені статуси Expected і Done. Expected призначаються створеному багу, якщо він визнається не багом, а особливою можливістю (feature). Тобто мається на увазі поведінка системи, що явно не відображена у вимогах, але є правильною з погляду бізнес-логіки.

Статус Done використовується для багів, які вирішено не виправляти. Окрім цього, цей статус виставляється для задач типу Task, коли вони виконані тестувальником або розробником. Основний результат ефективності методу можна відстежити на діаграмі, наведеній на рис. 4.

На діаграмі відображена динаміка задач залежно від часу та кількості задач. Зі збільшенням часу та кількості задач відповідно змінюється кількість задач у процесі та закритих задач.

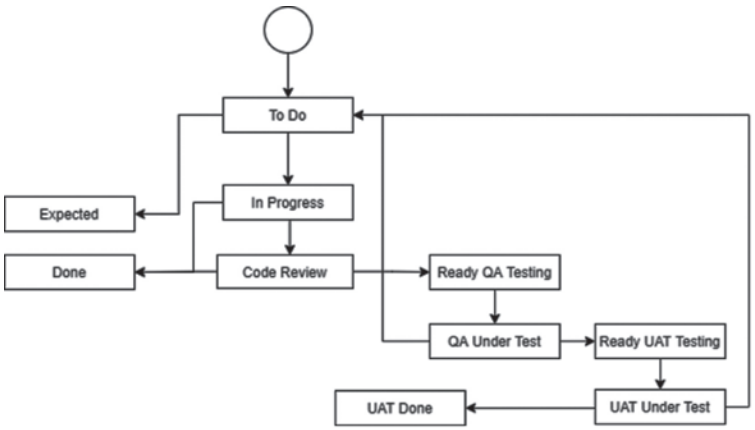

Рис. 3. Цикл змін статусів для задач

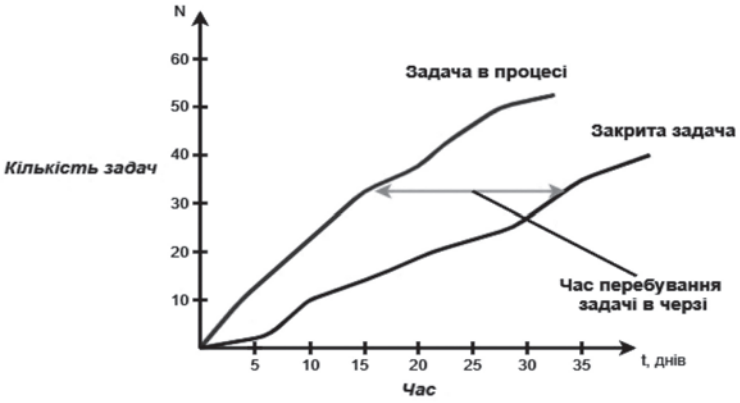

Рис. 4. Діаграма залежності

На ці показники насамперед впливають такі фактори: час на формування вимог, розробку, тестування та виправлення багів, зміна пріоритетів по задачам, зміна вимог до функціоналу. Саме від цих факторів залежить час на виконання задачі та перебування задачі в черзі. Завдяки орієнтованості методу на зменшення кількості одночасно виконуваних завдань час на закриття задачі відповідно зменшується. Дуже часто виникають зависання задач. Виділимо основні причини:

– велику кількість заблокованих завдань. Така ситуація може виникати через баги, які блокують функціонал, затримки на етапі підтвердження від продукт-архітектор, затримки у процесі виправлення багів;

– постійну зміну пріоритетів задач. Завдання доходять до останніх стадій, раптом стають не дуже важливими, команда перемикається на нові завдання. Виходить, що змінилися пріоритети, і робота не завершена;

– додавання нових поліпшень і змін у вимогах до системи. Kanban передбачає можливість

постійних змін вимог до системи, тому часто виникають ситуації, коли розроблений функціонал доповнюють або навіть змінюють.

Висновки. У роботі вдосконалено процес організації роботи команди з розробки програмного забезпечення за системою Kanban. Реалізована дошка для візуалізації завдань з оптимальним набором статусів для ефективної роботи, приведені принципи роботи з нею та цикл змін статусів для задач. Розглянуто на практичному прикладі процес розробки та взаємодії команди. Наведено графік, що демонструє ефективність методу залежно від певних факторів. Розробка за системою Kanban є однією з найпопулярніших як для невеликих, так і для малих проектів завдяки орієнтованості на задачі, а саме: на зменшення кількості одночасно виконуваних задач і зменшення часу виконання кожної задачі, гнучкість у прийняті рішень і частоту змін пріоритетів, тісну взаємодію між усіма членами команди. Наукова цінність наведеного методу полягає у підвищенні ефективності та покращенні роботи наявних методів організації розробки програмного забезпечення. У ході подальших досліджень планується вдосконалення розробленої моделі з метою підвищення ефективності системи.

Список літератури:

1. Corona E., Pani F.E. A Review of Lean-Kanban Approaches in the Software Development. 2013. URL: https://pdfs.semanticscholar.org/0725/ 482b6ced393863440f7e063c268e3790d18c.pdf. (дата звернення: 07.11.2020)
2. Reddy A. The Scrumban [R] Evolution: Getting the Most Out of Agile, Scrum, and Lean Kanban. Addison-Wesley Professional, 2015.
3. Стелман Е., Грін. Д. Осягаючі Agile: Цінності, принципи, методології. Москва : Манн, Іванов і Фербер, 2019. 448 с.
4. Киричек Г.Г., Киричек О.О. Модель оцінки плагіату програмного коду на основі системи контролю версій. *Восточно-Европейский журнал передовых технологий*. 2012. № 2 (2). С. 25–28.
5. Кон М. Scrum: гибкая разработка ПО. Москва : Вільямс, 2011. 576 с.
6. Kirichek, G., Tymoshenko, V., Rudkovskyi, O., Hrushko, S.: Decentralized System for Run Services. In: CEUR Workshop Proceedings 2353, 2019. P. 860–872.
7. Murino T., Naviglio G., Romano E. Optimal size of Kanban board in a single stage multi product system. 2010. URL: https://www.researchgate.net/ publication/234790654_Optimal_size_of_Kanban_board_in_a_single_stage_multi_product_system (дата звернення: 07.11.2020)
8. Kirichek, G., Skrupsky, S., Tiahunova, M., Timenko, A.: Implementation of web system optimization method. In: CEUR Workshop Proceedings 2608, 2020. P. 199–210.
9. Ahmad, M.O., Markkula, J., Oivo, M. Kanban in software development: A systematic literature review. In Software Engineering and Advanced Applications (SEAA). 2013. URL: https://www.researchgate.net/publication/260739586_Kanban_in_Software_Development_A_Systematic_Literature_Review (дата звернення: 07.11.2020)
10. Мартин Р., Ньюкирк Дж, Косс Р. Быстрая разработка программ. Принципы, примеры, практика. Москва : Вильямс, 2004. 752 с.

Kyrychek H.H., Chubich A.I. KANBAN-METHOD USE FOR SOFTWARE DEVELOPMENT ORGANIZATION

The use of Kanban in software development is gaining popularity every year. Flexible method allows you to adapt to constant changes in the process and quickly switch between different stages of development. The paper analyzes the existing examples of Kanban boards used for software development and elaborates the process of organizing teamwork based on the basic principles of Kanban methodology. The purpose of the work is to conduct research and implementation of the method of team software development for principles Kanban-method. The research object is the organization process of team's work in software development. The subject is models, methods, algorithms and tools for implementing Kanban method. The authors consider the implementation of general Kanban approaches of agile-methodology and lean thinking, without strict rules, but with principles that can be referred to. Only special management methods exist for Kanban, but they are not standardized. Therefore, improving the Lean development process is relevant for software development. Kanban provides consistent and smooth improvements for good results. The Kanban system provides the maximum reduction of Lead Time, and the time of work on the task at all stages. Therefore, the approach to the implementation of tasks is implemented with a focus on the board, the flow of tasks and the identification of problem areas. The work process needs to be designed so that the team is ready so that the requirements for the system and priorities of the tasks can change several times a day. The process of supporting communications between different departments also requires a clear flow chart to quickly share information about work being done. The paper implements a task visualization board, with an optimal set of statuses, for effective work. The graph showing the effectiveness of the method depending on certain factors is given.

Key words: *Kanban board, method, algorithm, priority, software.*