ВСЕУКРАЇНСЬКИЙ КОНКУРС СТУДЕНТСЬКИХ НАУКОВИХ РОБІТ

з галузі знань «Управління проектами і програмами»

ШИФР «Ігрова індустрія»

УПРАВЛІННЯ ІТ-ПРОЕКТАМИ В ІГРОВІЙ ІНДУСТРІЇ

2019 р.

ЗМІСТ

ВСТУП

Актуальність теми. Проектний контекст у безперервному розвитку індустрії інформаційних технологій зробив програмне забезпечення невід'ємним елементом діяльності людини та бізнесу.

Сучасні напрямки інформаційного удосконалення, такі як Internet of Things, націлені на побудову середовища, у якому пристрої під управлінням програмних продуктів керуватимуть усілякими повсякденними процесами суспільного та особистого життя. У цих умовах можна сказати, що необхідність створити якісний продукт набуває нового сенсу, зростає вплив якості на економічний результат ІТ-проекту.

Наразі інформаційні технології досягли високого рівня та супроводжують людство не тільки в освітньому і науковому спектрі, а також в розважальному. Ігрові ІТ-проекти представляють не тільки комунікативно-розважальну функцію, а так само виробляють певний заробіток їх творцям. Кожен ІТ-проект є окремим цілим зі своєю ідеологією і функціональною відмінністю. І, відповідно, до управління такими проектами слід підходити неординарно.

Комп'ютеризація дала можливість автоматизувати численну кількість професій та позбавити продукти їх праці негативного впливу «людського фактору», скоротити кількість виробничих дефектів, підвищити якість кінцевого продукту проекту.

У той же час виробництво програмного забезпечення є суто «мануальною» справою. Програмний код розробляють люди, а кінцевий продукт у повної мірі потрапляє під дію того самого «фактору».

У роботі розглядається перспектива впливу інновацій тренду Internet of Things на повсякденне життя людини. Наприклад, програмно регульовані світлофори, реверсивний рух транспорту, «розумне» дорожнє покриття, що надає інформацію про свій стан (пошкодження, ожеледь) на бортові комп'ютери автомобілів та у відповідні муніципальні структури тощо.

Спектр застосування систем, які збирають сенсорні дані із середовища та переробляють їх на інформацію чи дії, корисні для споживача, є віртуально необмеженим.

Можливо припустити, що масштаб подібних IT-проектів буде великим, привабливим для інвесторів, а продукти істотно впливатимуть на якість повсякденного життя людини. Забезпечення належних стандартів якості програмного продукту може бути вирішальним фактором успіху чи провалу такого проекту.

Світ інформаційних технологій дуже динамічний. Щодня з'являються все нові і нові технології і продукти. Звичайно ж, не обходиться і без розважальної сфери. Інформаційні технології – узагальнена назва технологій, що відповідають за зберігання, передачу, обробку, захист та відтворення інформації з використанням комп'ютерів. Використовують ці технології для автоматизації, полегшення та прискорення видів діяльності, що певним чином пов’язані з обробкою інформації, наприклад, прикладні програми чи Інтернет-сервіси.

Ігри є лідером розважальної інфраструктури. Щомісяця в світі стає все більше і більше нових ігрових продуктів зі своєю унікальною ідеєю та реалізацією. Більшість ігрових компаній намагаються розробляти нові режими для залучення більшого попиту до їх продукції.

Ігри діляться на 2 типу: одиночні і розраховані на багато користувачів. Одиночна гра представляє собою певний сюжет з прописаними системними гравцями (ботами). Ігри, розраховані на багато користувачів, містять певну кількість реальних гравців на 1 згенерованій карті. Всі гравці системно підключаються до сервера, на якому і зберігається потрібна інформація.

Ще один сценарій розвитку подій: зібрана команда з ентузіастів-професіоналів, які об'єднані єдиною метою – написанням власної гри. У них є непогані технічні навички, але немає досвіду роботи і розуміння в маркетингу та менеджменті самого підприємства. Вони не знають, як працює власне сам ринок, як можна випустити продукт і при цьому вийти в "плюс". Для такої команди кращий вихід – це пошук спонсора або ж видавця. Видавці створюють і

поширюють ігри. Для цього наймають команду розробників, маркетологів і керівників, встановлюють жорстку структуру проектної роботи. Буває так, що видавництво бере розробників у штат і починає саме робити гри.

Компанія "Electronic Arts" починала як видавництво, яке разом з іграми просувало їх творців: дизайнерів і програмістів. В кінці 80-х "EA" купили команди розробки і зробили серії ігор "FIFA", "Need for Speed", "Medal of Honor", "Command & Conquer". Такі типи команд називають змішаними [1, 2].

Ігровий сервер — програмний компонент обчислювальної системи, що забезпечує зв'язок між різними клієнтами, надаючи їм можливість комунікації один з одним в рамках програмної оболонки конкретної гри. У роботі ігрового сервера можна виділити три основних механізми:

- Зв'язок з клієнтом.
- Синхронізація отриманих даних.
- Надсилання даних клієнтові.

Ігровий сервер може розробляти як офіційний розробник (при цьому обмежувати кількість гравців на сервері і саму кількість серверів), так і сторонній розробник. Сторонні розробники намагаються вдосконалити гру на рівні сервера, і в більшості випадків у них це виходить. На даний момент, ігрових серверів створених сторонніми розробниками набагато більше, ніж офіційних.

Ігрові сервера представляють не тільки комунікативно-розважальну функцію, а так само виробляють певний заробіток їх творцям завдяки залученості самих гравців.

Предметом залученості гравця стають:

1. Прокачування ігрових показників (рівень, гроші, скіли і т.і.).
2. Знаходження в сеттінгу, схожому з реальним світом, з обмеженням відповідальності за дії.
3. Можливість підйому по соціальних сходах (в ігровому режимі присутня сувора соціальна ієрархія, подібна до реальної).

4. Можливість потрапити в адміністраторський склад (гравець зможе спостерігати і безпосередньо брати участь в процесі управління сервером, а можливо – і проектом).

Для того, щоб контролювати всю команду ігрового проекту та поширювати заробіток самим розробникам, треба правильно налагодити управління самої компанії. При організації роботи у більшості молодих компаній є певні помилки та недоліки, через які на перших стадіях розробки найчастіше команда розпадається, а продукт так і не виходить до кінцевого релізу. І, щоб налагодити усі процеси виробництва, продукту потрібно дотримуватися певних моделей і достіджень. Ці проблеми певною мірою досліджували М. Аліменков, С. Амблер, С. Крамський, Д. Ларсен, С. Муравецький, Д. Шреєр та інші. Проте, жоден з цих високоповажних авторів детально не аналізував можливу методологію розробки програмного продукту.

Це і зумовило обрання саме цієї теми конкурсної роботи – «Управління ІТ-проектами в ігровій індустрії».

Мета дослідження. Метою дослідження є вдосконалення моделі та підходу, завдяки яким керуючий ігровим ІТ-проектом може якісно налагодити усю структуру організації, компанії.

Завдання, що вирішуються в роботі:

1. Провести аналіз існуючих методологій.
2. Вирішити, яка із методологій найбільш ефективна при роботі з ігровими проектами.
3. Впровадити результат дослідження.
4. Розробити рекомендації щодо впровадження методології розробки програмного продукта у ігрову компанію.

Об'єкт дослідження – структура управління ІТ-проектами в ігровій індустрії.

Предмет дослідження – існуючі методології розробки програмного забезпечення.

Методи дослідження. При дослідженні використані методи структурно-логічний, історичний, порівняльний, аналітико-синтетичний. При написанні конкурсної роботи були досліджені методології “Agile”, а саме: “Scrum” та “Kanban”.

Теоретичне значення конкурсної роботи полягає у можливості моделювання гнучкої роботи ІТ-компанії в ігровій індустрії.

Практичне значення – в таких напрямках: Результати дослідження різних методологій для управління ІТ-проектами. Впровадження результатів у ігрову компанію та написання рекомендацій до існуючих компаній. Теоретичні висновки роботи можна використати у наступних наукових дослідженнях.

Структура роботи. Робота складається зі вступу, трьох розділів, поділених на підрозділи, висновків та списку використаних джерел, загальний обсяг складає – 30 друкованних сторінок основного тексту і один додаток.

РОЗДІЛ 1. РОЗГЛЯД МЕТОДОЛОГІЙ AGILE

1.1. Аналіз методу Agile

У перекладі з англійської мови «agile» означає «живий, рухливий», але переводять його частіше як «гнучкий». В галузі розробки програмного забезпечення цей термін з'явився на початку 2000-х років, коли в штаті Юта (США) було видано «Маніфест гнучкої розробки ПО». З тих пір під «Agile» розуміють набір підходів за «гнучкою» розробкою програмного забезпечення.

Основоположні принципи Agile-маніфесту:

- Найвищим пріоритетом для нас є задоволення потреб замовника, завдяки регулярному постачанню цінного програмного забезпечення.
- Зміна вимог вітається, навіть на пізніх стадіях розробки. Agile-процеси дозволяють використовувати зміни для забезпечення замовнику конкурентної переваги.
- Працюючий продукт слід випускати якомога частіше, з періодичністю від кількох тижнів до кількох місяців.
- Протягом всього проекту розробники і представники бізнесу повинні щодня працювати разом.
- Над проектом повинні працювати мотивовані професіонали, щоб робота була зроблена, створіть умови, забезпечте підтримку і повністю довіртеся їм.
- Безпосереднє спілкування є найбільш практичним і ефективним способом обміну інформацією як з самою командою, так і всередині IT-команди.
- Працюючий продукт – основний показник прогресу.
- Інвестори, розробники і користувачі повинні мати можливість підтримувати постійний ритм нескінченно. Agile допомагає налагодити такий стійкий процес розробки.

- Постійна увага до технічної досконалості і якості проектування підвищує гнучкість проекту.
- Простота, мистецтво мінімізації зайвого клопоту вкрай необхідна.
- Найкращі вимоги, архітектурні та технічні рішення народжуються у самоорганізованих команд.
- Команда повинна систематично аналізувати можливі способи поліпшення ефективності та відповідно коригувати стиль своєї роботи.

Отже, дані 12 принципів призначені для створення і підтримки робочого середовища, орієнтованого на клієнта, що відповідає діловим цілям, і який може швидко реагувати, змінивши потреби користувачів та ринкові сили.

Цінності, які викладені у маніфесті: Люди і взаємодія – важливіше процесів та інструментів. Працюючий продукт – важливіше вичерпної документації. Співпраця з замовником – важливіше узгодження умов контракту. Готовність до змін – важливіше проходження за попереднім планом.

Прихильники методів Agile стверджують, що ці чотири значення, викладені в "Agile Manifesto", сприяють процесу розробки програмного забезпечення, який зосереджується на якості, створюючи продукти, що задовольняють потреби та очікування споживачів, стейкхолдерів.

1.2. Інноваційний підхід до імплементації методології SCRUM в IT-проектах

Перш за все, ми розглянемо методологію, яка використовується найчастіше у розробці програмного продукту в IT-індустрії – це SCRUM.

SCRUM – не абревіатура, цей термін взятий з регбі, який позначає сутичку навколо м'яча. Сам термін Scrum – це методологія управління проектами, яка побудована на принципах тайм-менеджмента. Основною її особливістю є залученість в процес усіх учасників, причому, у кожного учасника є своя певна роль. Суть в тому, що не тільки команда працює над вирішенням завдання, але

все ті, кому цікаво рішення задачі, не просто поставили її, а постійно «працюють» з командою, і ця робота не означає тільки постійний контроль [1].

Далі, перейдемо до ролей, які залучені до самого процесу:

Власник продукту (Product owner) – людина, яка має безпосередній інтерес в якісному кінцевому продукті, яка розуміє, як цей продукт повинен виглядати та працювати. Ця людина не працює в команді, а працює на стороні замовника або засновника самої ідеї, але ця людина працює з командою. Власник продукту і виставляє пріоритети задач.

Scrum-майстер – це людина, яку можна назвати керівником проекту, хоча це не зовсім так. Він стежить за тим, щоб усі принципи Scrum дотримувалися у роботі. Scrum-мастером може бути також і Project Manager проекту, який, крім своїх основних задач, контролює якісне виконання роботи команди, дотримуючись методу SCRUM.

Scrum-команда – це команда, яка приймає всі принципи Scrum і готова з ними працювати. Згідно The Scrum Guide (документу, що є офіційним описом Scrum від його авторів), команда повинна володіти такими якостями і характеристиками: вміти само організовуватися, ніхто не може вказувати команді, як перетворити опис продукту в працюючий продукт; бути багатофункціональними, володіти всіма необхідними навичками для випуску працюючого продукту; за виконану роботу відповідає вся команда, а не індивідуальні члени команди [1].

Ще у давнину знали, що «число друзів за дружнім столом не повинне бути меншим від числа грацій (3) і більшим за число муз (9)». А науковець Дж. Міллер у 1956 році опублікував роботу «Магічне число 7 ± 2» [4], в якій на основі експериментів довів, що людина взмозі одночасно утримувати в пам'яті число смислових одиниць (так званих «сем»), рівне 7 ± 2. Цим же числом обмежується число завдань, які може одночасно утримувати в пам’яті підлеглий. Цим же числом Міллера обмежується і кількість людей у IT-команді, якщо ми хочемо, щоб кожний постійно взаємодіяв із кожним членом команди. Отже, рекомендований розмір IT-команди $= 7 \pm 2$ осіб. Згідно ідеологем Scrum, команди

більшого розміру вимагають занадто великих ресурсів на комунікації, в той час, як команди меншого розміру підвищують ризики і зменшують розмір роботи, який команда може виконати в одиницю часу [4]. Основою Scrum є Sprint, впродовж якого виконується робота над продуктом.

Sprint – відрізок часу, який береться для виконання певного (обмеженого) списку завдань. Рекомендується брати 2-4 тижні (тривалість визначається командою один раз). По закінченню Sprint має бути отримана нова робоча версія продукту. Перед початком кожного Sprint здійснюється “Sprint Planning”, на якому проводиться оцінка вмісту “Product Backlog” і формування “Sprint Backlog”, який містить завдання (Story, Bugs, Tasks), що повинні бути виконані в поточному спринті.

Беклог (backlog) – це список усіх робіт. Можна сказати, що це щоденник загального користування. Розрізняють 2 види беклогов:

1. **Product - беклог** – це повний список усіх робіт, при реалізації яких ми отримуємо кінцевий продукт.
2. **Sprint - беклог** – це список робіт, який визначає і узгоджує з Власником продукту сама команда, на найближчий звітний період (спринт). Завдання в Sprint-беклог беруться з Product-беклога.

Планування спринту (або Sprint Planning) – це нарада, на якій присутні всі (команда, Scrum-майстер, Власник продукту). Протягом цієї наради Власник продукту визначає пріоритети завдань, які він хотів би побачити виконаними після закінчення спринту. Команда оцінює за часом, скільки з бажаного вона може виконати. У підсумку виходить список завдань, який не може змінюватися протягом спринту і до кінця спринту повинен бути повністю виконаний [5].

Кожен спринт повинен мати мету, яка є мотивуючим фактором і досягається за допомогою виконання завдань з Sprint Backlog. Нижче наведен наглядний приклад виконання спринту на Рисунку 1.

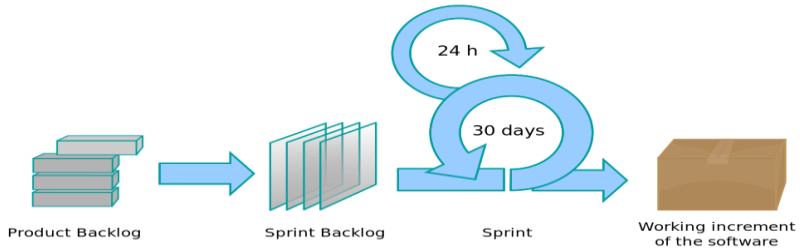

Рис. 1. Виконання спринту

На Рисунку 1 ми наочно бачимо, як підготовлюється та виконується спринт. Спочатку команда проекту на плануванні складає задачі з Product Backlog до Sprint BackLog, Далі, спринт запускається, і задачі, які були заплановані у Sprint BackLog переходять до статусу виконання [6]. Наочно представлено час виконання цього спринту, а саме 30 днів (30 days). Команда та Scrum Master не може розривати тимчасові рамки, тобто додавати ще день або два. Якщо Sprint не виконали у срок, то він закривається і переходить до статусу “Провалений”, після цього вже не можна створювати новий спринт та аналізувати, чому саме команда не встигла у встановлений термін.

Також, на Рисунку 1 показано час, який виділяється на щоденні обговорення з командою, а саме – 24 години з 30 днів. До щоденних “мітингів” входять обговорення кожного члена команди, на якій стадії він знаходиться, а також якісь нагальні проблеми. І якщо спринт пройшов добре, на виході ми получаємо готову частину продукту IT-проекту [6].

Переваги та недоліки методології Scrum такі:

Scrum володіє досить привабливими якостями. Scrum орієнтований на клієнта, дуже гнучкий. Scrum дає клієнтові можливість зробити зміни у вимогах в будь-який момент часу (але не гарантує того, що ці зміни будуть виконані). Можливість зміни вимог приваблива для багатьох замовників ПО. Scrum досить простий у вивченні, дозволяє економити час, за рахунок виключення не критичних активностей. Scrum дозволяє отримати потенційно робочий продукт в кінці кожного Sprint'a. Scrum робить упор на багатофункціональну IT-команду, здатну вирішити необхідні завдання з мінімальною координацією. Це особливо

привабливо для малих компаній і стартапів, оскільки позбавляє від необхідності наймати або навчати спеціалізованих керівників IT-проектів [1].

Звичайно, у Scrum є і важливі недоліки. З огляду на простоту і мінімалістичність, Scrum задає невелику кількість досить жорстких правил. Однак це вступає у конфлікт з ідеєю клієнтоорієнтованості в принципі, бо клієнтові не важливі внутрішні правила команди розробки, особливо якщо вони обмежують клієнта. Наприклад, в разі необхідності, за рішенням клієнта Sprint backlog може бути змінений, не дивлячись на явне протиріччя з правилами Scrum. Проблема є більшою, ніж здається, оскільки Scrum відноситься до сімейства Agile, в Scrum не прийнято, наприклад, створення плану комунікацій та реагування на ризики, таким чином роблячи складним або неможливим формальну протидію порушенням правил Scrum.

Іншою слабкою особливістю Scrum є упор на багатофункціональну команду. При уявному зниженні витрат на координацію команди, це призводить до підвищення витрат на відбір персоналу, його мотивацію, навчання. При певних умовах ринку праці, формування повноцінної, ефективної Scrum-команди може бути неможливим [1].

1.3. Аналіз методології Kanban

Канбан вперше з'явилася в Японії. Там це слово перекладається як «рекламний щит» або «рекламна вивіска» («кан» - видимий, візуальний, «бан» - картка). Цю методику винайшла в 1959-му році і почала використовувати в 1962-му році компанія Toyota для виробництва автомобілів. Пізніше, використовуючи концепцію «Тойоти», з'являється Канбан як підхід до розробки програмного забезпечення. Методологія використовує ті самі, але трохи видозмінені, принципи для створення програмного забезпечення, при цьому вводить їх в існуючі методи розробки [7].

Канбан часто порівнюють зі Scrum і зараховують до Agile-методологій. Методику можна назвати гнучкою, якщо говорити про розробку ПО, але сама по собі Канбан лише частково дотримується принципів гнучких методологій. Якщо порівнювати з «Скрам»: в Kanban відсутні спринти, ролі, призначені для користувача історії, необов'язкові. При цьому методологію часто вважають більш «гнучкою», так як робочий процес практично не керується, мало регламентується, і результат на 90 % залежить від команди і повідомлення всередині неї, а не від менеджера. Виходячи з цього, робочий колектив може бути як істотним недоліком, так і великою перевагою Kanban. Якщо співробітники зацікавлені і хочуть працювати, то жорсткі рамки будуть тільки шкодити, і в умовах Kanban команда покаже високу продуктивність [15]. Якщо ж колектив сам по собі не надто ефективний без управління зверху, то при Kanban робочий процес взагалі ризикує розвалитися. Те саме можна сказати і про керівництво, і про ролі. Впроваджувати методологію буде легше в самовмотивований колектив, знижуючи таким чином витрати на менеджера, не витрачаючись на Scrum-майстрів, і складніше туди, де керівник забезпечує успіх роботи [8].

Весь робочий процес візуалізується, щоб команда завжди розуміла, які завдання можуть почекати, а над якими необхідно працювати прямо зараз. Найбільш підходящий інструмент для цього: Канбан-дошка.

Канбан-дошка – це таблиця з кількома стовпцями. Усередині стовпців знаходяться стікери з завданнями. У процесі роботи над проектом кожен член команди може переносити стікери у відповідне місце. За кожним членом команди може бути закріплені певні стікери з завданнями.

Як вже вказувалося вище, на Канбан-дошці є певні стовбці. Буває різна кількість цих стовпців. В основному це залежить від задач та планування проекту. Нижче ми розглянемо декілька з цих стовпців [9].

1. **Глобальні цілі.** Стовпець варто розміщувати на самому початку, завдяки йому команда бачить, яким повинен стати продукт за певний проміжок часу. Наприклад, знищення продуктивності команди на 10 %.

2. **Завдання в черзі.** Тут розташовуються ті завдання, які можна почати виконувати. Чим вище в стовпці завдання, тим вище їх пріоритет, починати потрібно з самого верху. Пріоритет задачі визначає команда на загальних зборах.

3. **Колонки з етапами роботи над завданням.** Завдання переміщується по них, поки не дійде до завершального стовпчика. Їх кількість залежить від типу роботи, іноді можна обійтися і одним-двома. Стікер переходить як вперед, якщо черговий етап виконаний успішно, так і назад, якщо були виявлені помилки на попередній стадії. Етапи мають плануватися у логічній послідовності.

4. **Останній стовпець.** Цей стовпець містить стікери з виконаними завданнями. Перед ним корисно ставити колонку «Тест» або «Перевірка». І вже після тестування продукту на цій стадії його треба пересувати на останній стовпець. Якщо продукт не пройде певну перевірку, він може залишитися помилковим, і далі – вже на більш пізніших етапах – створити певний хаос у роботі над продуктом.

5. **Помилки та недоліки.** Це відокремлений стовпець, на який пересуваються стікери з помилковими задачами. Зазвичай пересуванням таких карток займаються тестери проекту.

6. **Термінові завдання.** Їм буде відразу ж виділятися пріоритет у всієї команди. При цьому не може бути більше однієї невідкладної задачі одночасно – тоді інші теж стоять в черзі. Також на кожному стовпці може бути певна кількість завдань [10].

Максимальна кількість завдань – це цифра, яка ставиться під кожною колонкою, крім «Цілей» і «Виконано». Виходячи з чисельності команди, визначається, над скількома завданнями вони можуть працювати одночасно. Не можна перенести стікер у наступний стовпець, якщо під ним стоїть цифра «6», а їх там вже шість. Так, якщо члени команди виявляють, що робочий процес встав, вони приходять на допомогу своїм товаришам, щоб ті швидше відправили завдання на наступний етап. Ще стікер може мати свій особливий колір. Колір

кожного стікера теж може нести додаткову інформацію. Наприклад, ступінь важливості, терміновості або необхідність пропустити кілька етапів [8].

Проаналізуємо переваги та недоліки Kanban в порівнянні зі Scrum. Канбан часто порівнюють зі Scrum і зараховують до Agile-методологій. Методику можна назвати гнучкою, якщо говорити про розробку ПО, але сама по собі Канбан лише частково дотримується принципів гнучких методологій. Якщо порівнювати з «скрам»: в Kanban відсутні спринти, ролі, призначені для користувача історії необов'язкові. При цьому методологію часто вважають більш «гнучкою», так як робочий процес практично не управляється, мало регламентується, і результат на 90 % залежить від команди і повідомлення всередині неї, а не від менеджера.

Виходячи з цього, робочий колектив може бути як істотним недоліком, так і великою перевагою Kanban. Якщо співробітники зацікавлені і хочуть працювати, то жорсткі рамки будуть тільки шкодити, і в умовах Kanban команда покаже високу продуктивність. Якщо ж колектив сам по собі не надто ефективний без управління зверху, то при Kanban робочий процес взагалі ризикує розвалитися. Те ж можна сказати і про керівництво.

Впроваджувати методологію буде легше в самовмотивований колектив, знижуючи таким чином витрати на менеджера, не витрачаючись на Scrum-майстри, і складніше туди, де керівник забезпечує успіх роботи [10].

У доданій до конкурсної роботи таблиці ми розглянемо порівняння методології Scram та Kanban. Варто відмітити, що ці дві методології відносяться до Angail (Додаток А). Основна різниця між Scrum і Канбан – в довжині ітерацій. У Scrum ітерації – 2 тижні, в Kanban завдання програмісту можна «підсовувати» хоч кожен день. Kanban дає більше гнучкості, якщо під гнучкістю розуміти частоту зміни пріоритетів. У Scrum завдання прийнято оцінювати в Story points або в годинах. Без оцінки не вийде сформувати спринт: адже нам потрібно знати, чи встигнемо ми зробити завдання за 2 тижні [11].

Після аналізу обох методологій, можно зробити висновок: кожна з них спрямована під певний IT-проект, та певну IT-команду.

РОЗДІЛ 2. ПРАКТИЧНА РЕАЛІЗАЦІЯ ОБРАНИХ МЕТОДОЛОГІЙ

2.1. Вибір інструмента реалізації IT-проекта

У попередньому розділі ми розглянули практичну частину роботи методологій, які ґрунтуються на маніфесті Agile. В даному розділі ми розглянемо безпосередньо застосування даних методологій. Для того, щоб дохідливо зрозуміти, як працює кожна з них, ми будемо впроваджувати одну методологію в роботу іншої, а саме Scrum в работу Kanban. Робити ми це будемо на прикладі відкриття комп'ютерної аудиторії [11, 12].

По-перше, потрібно ознайомиться з інструментами для реалізації методологій. Провівши власний аналіз усіх існуючих аналогів, я вибрав для себе найбільш ефективний продукт – TRELLO.

Trello – програма для управління проектами невеликих груп, розроблене Fog Creek Software. Trello використовує парадигму для управління проектами, відому як канбан, метод, який спочатку популяризувався Toyota в 1980-х для управління ланцюгами поставок, про що ми вже сказали вище. Картки Trello підтримують коментарі, вкладення, терміни виконання і контрольні списки. В даний час підтримуються мобільні платформи додатків iPhone і Android. Також був розроблений веб-сайт, щоб бути доступним у більшості мобільних веб-браузерів.

Головні переваги, які дозволили Trello домогтися популярності – це: простий інтерфейс; майже необмежений безкоштовний доступ; зручність в роботі і можливість інтеграції з іншими популярними інструментами для онлайн-роботи; велика кількість інструкцій до роботи з дошками.

2.2. Створення основних компонентів дошки

Основними компонентами простору нашої дошки є картки та колонки.

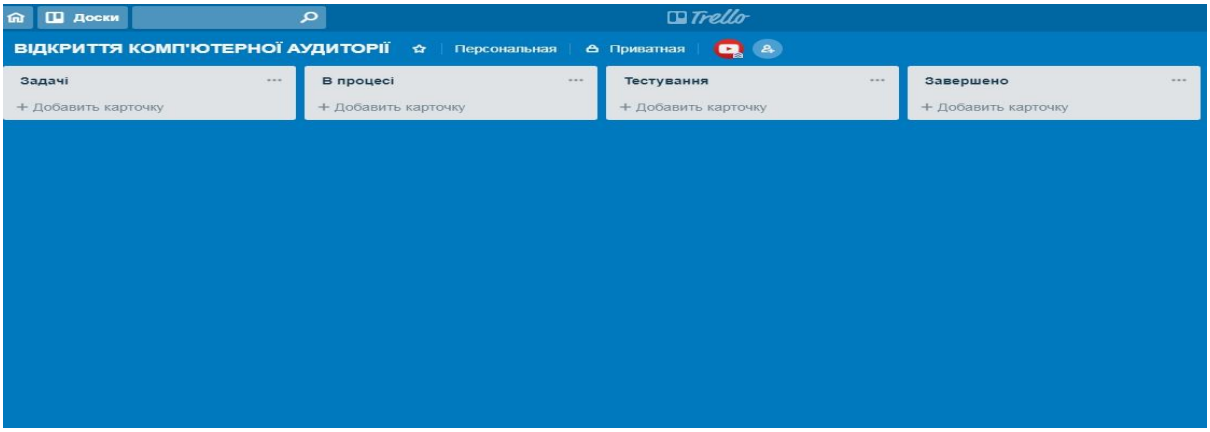

Рисунок 2.1. – Дошка IT-проекту

Отже, на рисунку 2.1. ми бачимо дошку нашого проекту, на якій розташовуються певні колонки, які ми розглянемо детальніше.

- Колонка “Задачі”. На ній буде розташований список задач, які потребують виконання IT-командою. Ці задачі будуть поставлені на спеціальних зборах усіх учасників даного IT-проекту.
- Колонка “В процесі”. Ця колонка включає до себе список активних завдань, які пересуваються з задач IT-проекту.
- Колонка “Тестування”. Має у собі задачі, які вже реалізовані командою, але результат реалізації ще не до кінця стабільний.
- І на останок, колонка “Завершено”. Це, остання стадія розробки нашого IT-проекту. На цю колонку пересуваються вже відтестовані задачі, які мають позитивний результат.

Також, у нашому проекті має бути одна запасна колонка на випадок, якщо у процесі реалізації проекту будуть виникати проблеми або складності. Ця колонка називається “Помилки”. Побачити її можно на Рисунку 2.2.

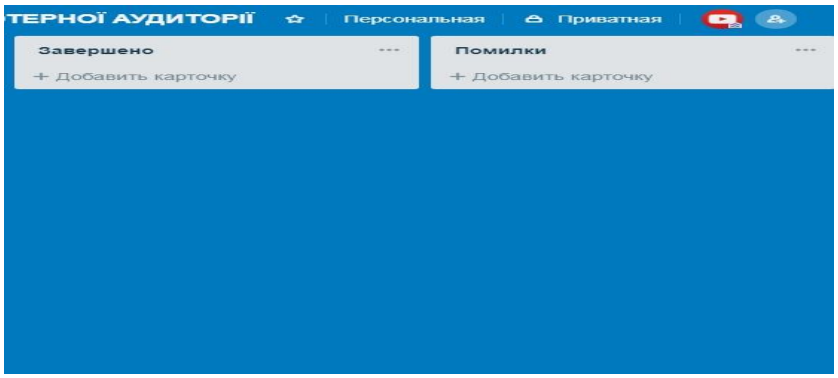

Рисунок 2.2. – Колонка “Помилки”

Після того, як ми створили усі необхідні колонки, ми проводимо збори, на яких усі члени IT-команди виясняють, які задачі треба додати до першої колонки нашої дошки. Кожен член IT-команди має своє слово, завдяки чому Project Manager (керуючий) має загальну картину розробки IT-проекту.

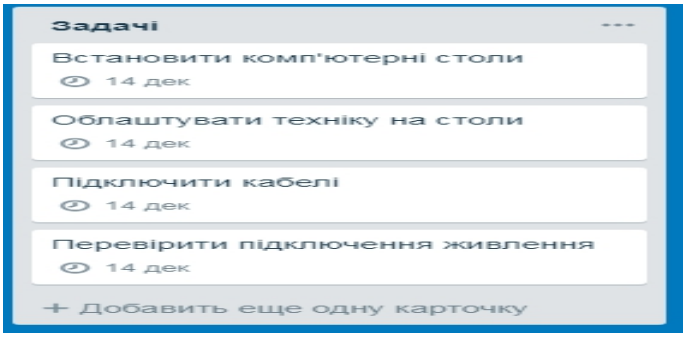

Рисунок 2.3. – Поточні завдання

На Рисунку 2.3. ми бачимо, що після обговорення IT-проекту команда вирішила, які задачі слід поставити у першу чергу. На зборах IT-команда загалом вирішує пріоритет завдань, і виставляє їх у певній послідовності. Керуючий, відповідно вніс ці картки до певної колонки і встановив термін, за яким потрібно ці завдання виконати (Спринт).

2.3. Процесс розробки IT-проекту

Після проведення перших зборів керівник проекту повинен вирішити декілька питань:

1. За який час приблизно цей проект може дійти до стадії активного тестування та релізу.
2. Який член команди буде відповідати за яку діяльність.
3. Які задачі для проекту мають більший пріоритет, а які меньший.
4. Які приблизні зміни можуть статися в процесі роботи

У нашому випадку все було вирішено у такому форматі:

1. Процес розробки буде поділено на 3 спринта. Кожен спринт несе за собою двотижневий термін. Якщо, після виконання першого спринта задачі, які були поставлені, не вирішується, то спринти будуть розширені до 3 тижнів.

Кожен з спринтів має свою назву: “Встановлення обладнання”, “Підключення до мережі Інтернет”, “Встановлення ОС та кінцеве тестування”.

2. Керуючий “розподілив” усю команду на 2 типи ролей. Перший тип буде займатися безпосередньо “зовнішньою” роботою, а саме – підключенням обладнання та монтажем усіх систем. А вже другий тип людей буде займатися “внутрішньою” роботою. Це беспосередне встановлення операційної системи, перевірка працездатності техніки тощо.

3. Найбільш термінові завдання для реалізації – це монтаж самого обладнання та підключення цього обладнання до загальної системи.

4. На протязі усього проекту може передбачатися багато змін, починаючи від загального відключення електроенергії та закінчуючи природними катаклізмами. Тому кожен шаг проекту розплановано у такій послідовності, що найголовніше буде виконуватись терміново.

Після обгрунтування усіх цих тез проект починає свій перший спринт. На Рисунку 2.4. наочно видно активну роботу на протязі першого тижня.

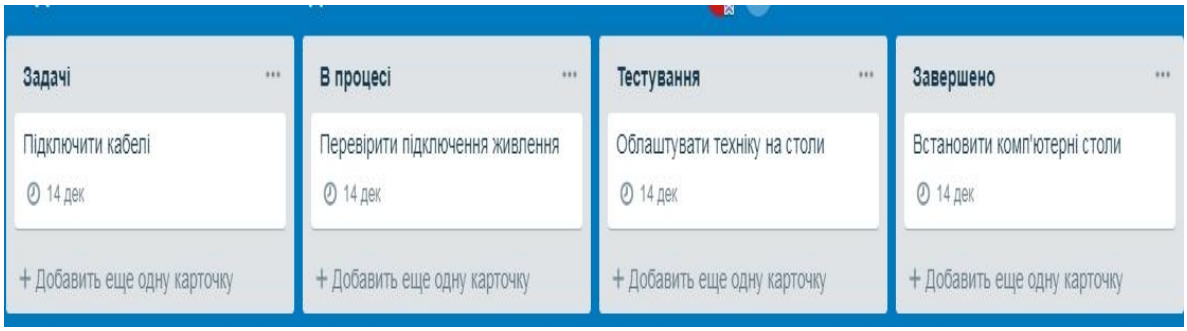

Рисунок 2.4. Робота на першому спринті

На протязі кожних 3-х днів проходять збори, де з’ясовується, на якій стадії зараз робота над IT-проектом, чи потребується достроково закінчувати спринт, або обговорення зауважень та інші поточні проблеми.

Після закінчення першого спринту команда проводить загальні збори, на яких підводяться підсумки роботи за тиждень.

У нашому випадку спринт завершився дуже успішно, і усі процеси були виконані ще до загального завершення спринту. Було достатньо часу, щоб протестувати готові задачі. Загалом і керівник і IT-команда остаточно вирішили,

що план, який вони разом спроектували, працює. Тому термін наступного спринту було змінено з 7 до 6 днів [9]. Це зроблено для того, щоб замовник мав позитивний погляд на роботу команди, а команда мала певну відповідальність за свої дії. Також, на зборах було вирішено, що другий спринт найменш складний. Після усіх дискусій, керівник IT-проекту знову заповнив колонку “Задачі” та дав старт другого спринту.

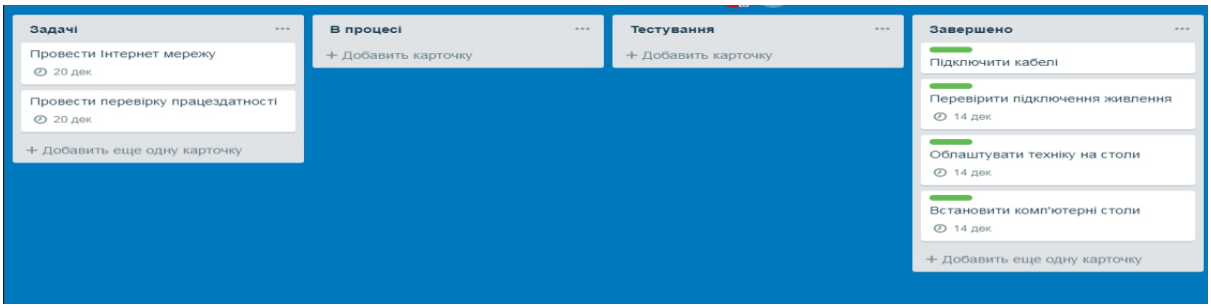

Рисунок 2.5. Старт другого спринту

На Рисунку 2.5. можна побачити робочу дошку на етапі старту другого спринту. Задачі, які були виконані на першому спринті, з дошки не видаляються, а остаються задля того, щоб бачити повну картину розробки проекту на фінальній стадії. Задачі, які вже виконані, помічаються “зеленим стікером”.

Після проведення ще двух спринтів IT-команда прийшла до того, що замість запланованих 21 дня робота вклалася в 13 днів, що безпосередньо доводить, що обрані методології практично працюють ефективно [13].

Надалі перейдемо до фіналізації аналізу роботи над проектною дошкою. Дивиться нижче рисунок 2.6.

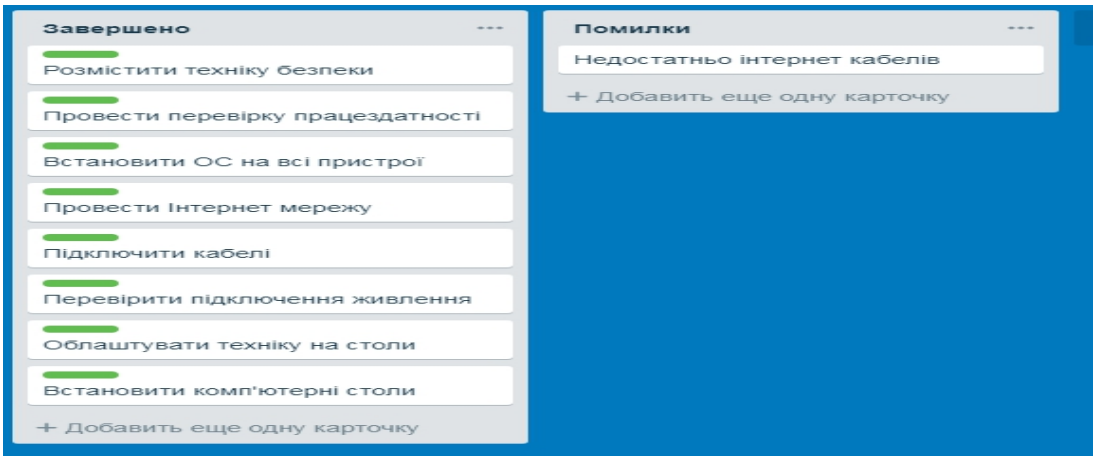

Рисунок 2.6. Остаточний вигляд дошки

Підводячи підсумки роботи над IT-проектом, команда знову провела остаточні збори, на яких було розглянуто питання помилок, а як можно побачити на Рисунку 2.6., помилки над проектом були, але не дуже критичні. Також на зборах були вирішенні остаточні питання по тестуванню (3 спринт) та здачі завершеного IT-проекту замовнику.

Загалом, це легка модель розробки програмного IT-продукту. Таку гібрідну модель можно використовувати керівникам, коли команда вже згуртована і може швидко і якісно виконувати усі процеси.

Для розробки нових ігрових проектів за статистикою більш підійде окрема методологія, тому що члени IT-команди не можуть настільки швидко на перших етапах розуміти один одного, і керівник IT-проекту не розуміє, що саме буде представляти продукт IT-проекту.

Після практичної реалізації описаних методологій можно визначити головний підсумок – ці інструменти для менеджменту працюють і мають свої певні позитивні сторони. Тому ігрова IT-індустрія є однією з наймолодших, і може поглинати у себе такі методи керівництва. Ці методології, до речі, також є інновайними розробками.

РОЗДІЛ 3. ТЕХНІЧНА ІМПЛЕМЕНТАЦІЯ IT-ПРОЕКТУ

3.1. Загальне розуміння технічної частини IT-проекту

Звичайно ж, для того, щоб створити сам продукт, недостатньо тільки скласти план роботи, тобто вибрати дошку, закріпити завдання і встановити терміни. Крім усього іншого, треба правильно вибрати технічну складову IT-проекту. У більшості випадків на IT-проекті є технічний директор, в завдання якого входить переклад завдань замовника або головного менеджера в завдання для розробників.

Оскільки більшість замовників не розуміються в технічній частині, вони не можуть до кінця пояснити, що вони хочуть, самим розробникам. Тому технічний директор збирає всі завдання, обмірковує їх і підводить до спільного знаменника, далі складає наочне технічне завдання і передає його розробникам [14].

Що стосується безпосередньо написання технічної частини, то у даному питанні є багато підпунктів, починаючи з розгляду платформи для написання гри і закінчуючи аналізом синхронізації, фізикою та й штучним інтелектом.

Платформ для написання повноцінної гри існує безліч. Є навіть спеціальні інструменти, які дозволяють створити власну гру навіть без знання мови програмування і без уміння розуміти технічну частину питання. Але зараз ми розглянемо одних з найбільш популярних мов програмування: C # і Java.

3.2. Мова програмування Java

Java швидко стала найпопулярнішою мовою програмування для Інтернету, тому що програми, створені на ній, можуть запускатися на різних операційних системах, завдяки встановленій віртуальній машині Java на комп'ютері

користувача. Програми Java створюються в проміжному коді, а віртуальна машина перетворює цей код у виконючий код для конкретної операційної системи. Наприклад, при запуску деяких програм з’являється пропозиція встановити додаток із зображенням чашки кави – це і є віртуальна машина Java. Програми Java без неї працювати не будуть.

Створенням мови Java займалася компанія Sun Microsystems, яка далі була куплена компанією Oracle. Слово Java вимовляється і як "Джава" і як "Ява". На цій мові можна створювати будь-які додатки для різних операційних систем, писати ігри, додатки для Інтернету і для мобільних пристроїв. Іншими словами, на ній можна створювати все, крім операційних систем [15, 16].

Програми, написані на Java, мають репутацію більш повільних і займають більше оперативної пам'яті, ніж написані на мові С. Проте, швидкість виконання програм, написаних на мові Java, була істотно поліпшена з випуском в 1997-1998 роках так званого JIT-компілятора в версії 1.1 на додаток до інших особливостей мови для підтримки кращого аналізу коду. Крім того, була проведена оптимізація віртуальної машини Java – з 2000 року для цього використовується віртуальна машина HotSpot.

Код Java 7 приблизно в 1,8 рази повільніше коду, написаного на мові Сі. Деякі платформи пропонують апаратну підтримку виконання для Java. Наприклад, мікроконтролери, що виконують код Java на апаратному забезпеченні замість програмної JVM, а також засновані на ARM процесори, які підтримують виконання байткода Java через опцію Jazelle [17].

3.3. Аналіз мови програмування C#

C # – це об'єктно-орієнтована мова програмування, є C-подібною (сі-подібною) мовою, тобто має явну схожість з мовою програмування C (Сі). Її знання не є обов'язковою умовою для освоєння мови C #. Мова C # перейняла гідності таких мов як: C ++; Pascal; Smalltalk; Java; Модула.

Мова поставляється з гігантською бібліотекою. Тому якщо задумали щось написати на C #, то швидше за все реалізація такого завдання вже існує в світі, її потрібно лише знайти і настроїти під себе [18].

Що стосується плюсів розробки на цій мові, то їх можно звести до таких пунктів:

1. У C # робиться більший упор на керований і безпечний код;
2. Тут присутній збирач сміття, тому можна не так сильно турбуватися за «витік» пам'яті при розробці програм;
3. Відсутнє множинне спадкування;
4. У C # існують тільки об'єкти, тобто завжди робота ведеться тільки з об'єктами;
5. Легка (в якійсь мірі) в освоєнні;
6. Швидка розробка;
7. Сучасна і повсюдно використовується;
8. Досить висока надійність (на відміну від C ++);

На C # розробляються ігри, сайти, мобільні додатки (Windows Phone), практикується створення штучного інтелекту. Як не складно здогадатися по списку вище, дана мова програмування використовується в самих різних галузях та для реалізації самих різних завдань [19, 20].

3.4. Розрахунки вартості продукту IT-проекту

Слід здійснити практичні розрахунки вартості IT-проекту.

1) Чиста приведена вартість проекту (NPV):

- інвестиційні витрати становлять – 9953,880 тис. грн.;
- річний економічний ефект від впровадження інвестиційних заходів становить – 5680,438 тис. грн.;
- ставка дисконтування – 6,5 %;
- нормативний період експлуатації проекту – 2 роки.

Чиста приведена вартість розраховується за формулою, тис.грн:

NPV=∑CFk/(1+r)k - ∑ Ik/(1+r)k,

де CFk – потік коштів (доходів) (річний економічний ефект) від впровадження інвестиційного заходу у k-му році, тис. грн.,

r – ставка дисконтування,

Ik – інвестиційні витрати у k-му році, тис. грн.

NPV = *((5680,438/(1+0,065)1)+((5680,438/(1+0,065)2)-*

*(9953,880/(1+0,065)1)=5333,744+5008,210-9346,366=**995,588 тис. грн.***

2) Внутрішня норма дохідності (IRR):

Для розрахунку внутрішньої норми дохідності інвестиційної програми використовуємо функцію ВСД (внутрішня ставка дохідності) програмного комплексу EXEL за таким алгоритмом:

IRR=функція ВСД(-9953,880 +5680,438+5680,438)=14,1%

3) Дисконтований період окупності (DPP):

Перераховуємо грошові потоки у вигляді поточних вартостей для кожного року, тис.грн.:

PVk= CFk/(1+r)k,

PV1=(5680,438 /(1+0,065)1=5333,744 тис. грн.,

PV2=(5680,438 /(1+0,065)2=5008,210 тис. грн.

Сума дисконтованих доходів складе, тис. грн.:

*5333,744+5008,210 =**10341,954 тис. грн.*** що більше розміру дисконтованих інвестицій (**9346,366 тис. грн.**). Це означає, що відшкодування первісних інвестиційних витрат відбудеться раніше 2-х років: *DPP=∑CFk/(1+r)k≥∑ Ik/(1+r)k*

Залишок 2-го року становить: *1-(10341,954 -9346,366)/ 5008,210=1-0,198=0,802*

Таким чином, дисконтований період окупності складе менше 2-х років, а саме: *DPP=1+0,802=1,802 роки*

4) Індекс прибутковості проекту (Інвестиційної програми):

PI=(∑CFk/(1+r)k)/(∑ Ik/(1+r)k)

PI=10341,95/9346,37=1,11

Отже, продукт є окупним, строк створення його незначний, після двох років з моменту первісного інвестування, можна очікувати прибуткок.

ВИСНОВКИ

Проведене дослідження дозволяє зробити наступні висновки. У результаті впроваджень декількох методологій управління проектом був розроблений план керівництва проектом. Як показує практика, найбільш ефективною є гібридна методологія, яка складається з Kanban та Scrum.

В ігровій індустрії тестування представляє з себе вже завершальну стадію розробки продукту. Останнім часом вакансія тестувальника може оплачуватися вище, ніж самого розробника, тому що на моменті тестування продукту будуть заново пройдені всі алгоритми, які планувалися при розробці. І якщо в одному з цих алгоритмів буде не виявлена критична "діра", то вся подальша робота розробника буде позбавлена сенсу.

Ігрова індустрія сама по собі є тією ж самою IT-індустрією. Основна відмінність у тому, що кінцевий продукт буде нести в собі розважальну функцію. На етапі розробки сама команда розробників може поділитися на кілька типів, але результат їх роботи буде один – це кінцевий продукт, продуманий ще на етапі планування.

Одними з важливих елементів процесу управління і розробки ігрового продукту є: набір персоналу, план розробки, маркетинг і тестування продукту.

Кожний з цих етапів слід розглядати "під мікроскопом", щоб не упустити будь-яку невелику дрібницю, через яку може обрушитись весь процес.

По-друге,, управляти ігровою компанією – досить цікавий і захоплюючий процес. Найголовніше – мати точне розуміння, що ти хочеш донести до кінцевого користувача, щоб після релізу самого продукту всі зрозуміли, що мету було досягнуто.

В процесі конкурсного дослідження нами складені та впроваджені схеми розподілу робіт і вирахувані інвестиційні витрати за проектом. Таким чином, інвестиційні витрати становлять – 995,588 тис. грн., сума дисконтованих доходів

складає 10341,954 тис. грн. Слід зазначити, що дисконтований період окупності проекту складає менше 2-х років.

Таким чином, доцільно рекомендувати для впровадження в практику створення ІТ-продуктів застосовані нами і описані в цьому дослідженні методології управління проектами і програмами.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойченко К. В. Управління IT-проектами в ігровій індустрії // Актуальні проблеми сучасного управління в соціально-економічних, гуманітарних та технічних системах : *Зб. наук. праць за матеріалами XIV Міжнародної наук.-практ. конф., присвяченої 30-річчю МАУП (21 лист., 2018 р., м. Одеса)* / Міжрег. Академія управління персоналом. Одеськ. ін-т. – Одеса: Лерадрук, 2018. – С. 216-222.

2. Блог компании Center-Game // [Електронний ресурс] / Режим доступу : http://blog.center-game.com

3. Шрейер Джейсон. Кровь, пот и пиксели. Обратная сторона индустрии видеоигр / Джейсон Шрейер – издательство “Эксма”, 2018.

4. Антоненко С. В., Малий В. В., Мазуркевич О. С. Психологія управління проектами : національні особливості / С. В. Антоненко, В. В. Малий, О. С. Мазуркевич. –Дніпропетровськ : Пороги, 2008. – 139 с.

5. Agile Development with the ICONIX Process: People, Process and Pragmatism / Rosenberg Doug, Stephens Matt и Collins-Cop Mark. – 2005.

6. Муравецький С. А. Планування процесів забезпечення якості у великих та географічно розподілених гібридних IT–проектах / С. А. Муравецький, С. О. Крамський // VII Міжнар. наук.-практ. конф. «Інтегроване стратегічне управління, управління портфелями, програмами, проектами». Вісник НТУ «ХПІ». Серія : Стратегічне управління, управління портфелями, програмами та проектами. – Харків : НТУ «ХПІ», 2016. – № 1 (1173). – С.106-109.

7. Agile Metrics [Книга] / Алименков Николай // [Електронний ресурс] – Режим доступу :

8. Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process [Книга] / Ambler Scott W. – 2002 // [Електронний ресурс] – Режим доступу :

9. Agile Project Management with Scrum [Книга] / Schwaber Ken // [Електронний ресурс] – Режим доступу :

10. Agile Retrospectives: Making Good Teams Great [Книга] / Esther Derby, Diana Larsen // [Електронний ресурс] – Режим доступу :

11. Agile Software Development with Scrum (Series in Agile Software Development) [Книга] / Ken Schwaber, Mike Beedle. – 2001 // [Електронний ресурс] – Режим доступу :

12. Cause-effect diagrams / Книберг Хенрик // Нью-Йорк, США – 2009 // [Електронний ресурс] – Режим доступу :

13. Crystal Clear – Alistair Cockburn (2005) // [Електроний ресурс] / Tomas. – Режим доступу : http://www.crisp.se/rd/Crystal_Clear.pdf

14. Crystal Clear : A Human-Powered Methodology for Small Teams [Текст] / Cockburn Alistair. – 2004 // [Електронний ресурс] – Режим доступу :

15. Захарченко О. В. Проджект менеджмент / О. В. Захарченко, С. О. Крамський // Навчальний посібник з "Менеджменту". – Одеса : «Екологія», 2018. – 227 с.

16. DSDM: Business Focused Development [Книга] / DSDM Consortium // [Електронний ресурс] – Режим доступу :

17. Коршунов О. П. Библиография : теория, методология, методика / О. П. Коршунов. – М. : Книга, 2012. – 287 с.

18. Бухгалтерский учет финансово-хозяйственной деятельности организаций. Методология, задачи, ситуации, тесты / З. Д. Бабаева и др. – М. : Финансы и статистика, 2015. – 544 с.

Додаток А

Аналіз методологій SCRUM та KANBAN

НАЗВА	SCRUM	KANBAN
Гнучкість	+	++
Комунікація в команді	+	+
Додавання нових коректив на етапі розробки	+	++
Самостійність команди	-	+
Контроль команди	+	-
Швидкість ітерацій	-	+
Розпланування усього процесу розробки продукту	++	+

Примітка: + - присутні; – - відсутні; ++ - подвійний ефект.

Власна розробка автора