Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Системотехніки
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістрський)

Оптимізація прогресу команди та технологій управління проектами в IT-компаніях на платформі Jira

(тема)

Виконав:
студент II курсу, групи СПРм-22-1
Добудько А. М.
(прізвище, ініціали)
Спеціальність 122 Комп'ютерні науки
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітня програма
Системне проєктування
(повна назва освітньої програми)

Керівник доц. каф. СТ Тітов С. В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ проф. Гребеннік І. В.
(підпис) (прізвище, ініціали)

20 24 р.

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Системотехніки

Рівень вищої освіти другий (магістрський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)

Освітня програма Системне проєктування
(повна назва)

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)

«13» 06 2024 р.

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Добудьку Андрію Миколайовичу
(прізвище, ім'я, по батькові)

1. Тема роботи Оптимізація прогресу команди та технологій управління проектами в IT-компаніях на платформі Jira

затверджена наказом університету від «13» 06 2024 р. № 687Ст

2. Термін подання студентом роботи до екзаменаційної комісії «13» 06 2024 р.

3. Вихідні дані до роботи оптимізація процесів в команді, використання Agile методологій, визначення пріоритетів

4. Перелік питань, що потрібно опрацювати в роботі огляд і аналіз сучасного стану розглянутої задачі, опис та аналіз можливих методологій для управління проєктами, опис і аналіз інструментів для впровадження методологій, опис сучасних практик управління проєктами, опис методів оптимізації прогресу команди, визначення пріорітетів в управлінні проєктами, опис функцій Jira, які корисні для оптимізації прогресу команди, виявлення найкращих практик щодо використання Jira для

управління проєктами, виявлення можливостей вдосконалення процесів в команді за допомогою інструментів Jira, впровадження Agile методологій в Jira для покращення процесу управління проєктами, розробка звітів для аналізу прогресу команди та його оптимізації

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Пошук необхідної літератури	02.05.2022 – 07.05.2022	Виконано
2	Аналіз сучасного стану розглянутої задачі	08.05.2022 – 11.05.2022	Виконано
3	Практичне дослідження	12.05.2022 – 17.05.2022	Виконано
4	Постановка задачі атестаційної роботи	18.05.2022	Виконано
5	Опис і аналіз методологій	19.05.2022	Виконано
6	Опис і аналіз інструментів для впровадження методологій	20.05.2022 – 22.05.2022	Виконано
7	Формування сучасних практик управління проєктами	23.05.2022 – 25.05.2022	Виконано
8	Формування методів оптимізації прогресу команди	26.05.2022	Виконано
9	Оформлення пояснювальної записки	25.05.2024 – 08.06.2024	Виконано
10	Захист кваліфікаційної роботи	13.06.2024	

Дата видачі завдання «02» 05 20 22 р.

Студент _____
(підпис)

Керівник роботи _____ доц. каф. СТ Тітов С. В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка складається з 74 сторінок, 7 рисунків та 26 джерел.

ОПТИМІЗАЦІЯ, AGILE, МЕТОДОЛОГІЯ, JIRA, УПРАВЛІННЯ ПРОЄКТАМИ, УПРАВЛІННЯ КОМАНДАМИ.

Метою роботи є вивчення та аналіз впливу використання Agile методологій та платформи Jira на оптимізацію робочих процесів, підвищення продуктивності та якості управління проєктами в IT-компаніях.

Об'єктом дослідження є управління проєктами в IT-компаніях, а предметом - вплив використання Agile методологій (зокрема Scrum та Kanban) разом з платформою Jira на робочі процеси та управління проєктами.

Методами дослідження є аналіз літератури, інтерв'ю з управлінцями та співробітниками, спостереження за робочими процесами, аналіз статистичних даних, а також порівняльний аналіз результатів впровадження Agile методологій та використання платформи Jira.

Робота містить результати виявлення переваг та недоліків використання Agile методологій та платформи Jira для управління проєктами в IT-компаніях, визначення оптимальних підходів до використання цих інструментів та розроблення рекомендацій для покращення робочих процесів.

ABSTRACT

The explanatory note consists of 74 pages, 7 images and 26 sources.

OPTIMIZATION, AGILE, METHODOLOGY, JIRA, PROJECT MANAGEMENT, TEAM MANAGEMENT.

The purpose of the work is to study and analyze the impact of the use of Agile methodologies and the Jira platform on the optimization of work processes, the improvement of productivity and the quality of project management in IT companies.

The object of the research is project management in IT companies, and the subject is the impact of using Agile methodologies (in particular, Scrum and Kanban) together with the Jira platform on work processes and project management.

Research methods include literature analysis, interviews with managers and employees, observation of work processes, analysis of statistical data, as well as a comparative analysis of the results of the implementation of Agile methodologies and the use of the Jira platform.

The work contains the results of identifying the advantages and disadvantages of using Agile methodologies and the Jira platform for project management in IT companies, determining optimal approaches to using these tools, and developing recommendations for improving work processes.

ЗМІСТ

ВСТУП 10
1 ПОСТАНОВКА ЗАДАЧІ 11
2 МЕТОДОЛОГІЇ ТА ЇХ ЗАСТОСУВАННЯ 12
2.1 Методологія Waterfall 12
2.2 Методологія Agile 15
2.3 Методологія Scrum 20
2.4 Методологія Kanban 26
2.5 Методологія XP 28
2.6 Методологія Lean 30
3 ІНСТРУМЕНТИ ДЛЯ ВПРОВАДЖЕННЯ МЕТОДОЛОГІЙ 35
4 СУЧАСНІ ПРАКТИКИ УПРАВЛІННЯ ПРОЄКТАМИ В IT-КОМПАНІЯХ НА ПЛАТФОРМІ JIRA 39
4.1 Користувацькі історії 39
4.2 Оцінювання (Estimation) 40
4.3 Епіки 42
4.4 Дорожні карти 43
4.5 Вимоги 44
4.6 Беклог 44
4.7 Спринт 47
4.8 Планування спринта 49
4.9 Огляд спринта 52
4.10 Ретроспектива 53
5 МЕТОДИ ОПТИМІЗАЦІЇ ПРОГРЕСУ КОМАНДИ 57
5.1 OKRs 57
6.2 Звітність та метрики Scrum 62

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ..... 71
ДОДАТОК А ..... **Ошибка! Закладка не определена.**

ВСТУП

У сучасному світі, де IT-технології стають все більш ключовими для успішного функціонування бізнесу, управління проєктами та координація роботи команд стають надзвичайно важливими складовими успіху компаній. IT-компанії постійно шукають способи покращення продуктивності та результативності управління проєктами, зокрема на платформі Jira.

Актуальність теми дослідження полягає в пошуку оптимальних стратегій управління проєктами та оптимізації робочих процесів в IT-компаніях через використання платформи Jira. Здійснення ефективного управління проєктами є критичним для надання конкурентної переваги та досягнення успіху в галузі IT.

Метою даного дослідження є аналіз сучасних практик управління проєктами в IT-компаніях на платформі Jira, виявлення проблем та невирішених аспектів управління проєктами, розробка та апробація оптимальних стратегій управління проєктами та командною роботою на платформі Jira для підвищення ефективності та результативності.

Обґрунтування значущості вибраної теми полягає в тому, що ефективне управління проєктами на платформі Jira може забезпечити підвищення продуктивності, зменшення затрат та покращення якості продукту, що є ключовими факторами конкурентоспроможності для IT-компаній.

Таким чином, дане дослідження має велике значення для практики управління проєктами в області IT та може призвести до значних покращень в цій сфері.

1 ПОСТАНОВКА ЗАДАЧІ

Проведення огляду сучасних методів управління проєктами в IT-компаніях, включаючи техніки та підходи, що використовуються для організації процесів управління проєктами. Це включатиме аналіз використання Agile методологій та платформи Jira у сучасних умовах діяльності IT-компаній.

Аналіз впливу використання Agile методологій та платформи Jira на робочі процеси та ефективність управління проєктами в IT-компаніях. Це включає вивчення того, як використання цих методологій та інструментів впливає на продуктивність команд, управління завданнями та процесами розробки.

Визначення переваг та недоліків використання Agile методологій та платформи Jira для управління проєктами в IT-компаніях. Це включатиме ідентифікацію позитивних та негативних аспектів використання цих методологій та рішень для управління проєктами.

Розроблення рекомендацій з оптимізації використання Agile методологій та платформи Jira для підвищення продуктивності та результативності управління проєктами в IT-компаніях. Це включатиме створення конкретних стратегій та методів використання Agile та Jira для поліпшення управління проєктами з урахуванням результатів дослідження.

2 МЕТОДОЛОГІЇ ТА ЇХ ЗАСТОСУВАННЯ

2.1 Методологія Waterfall

В світі управління проектами і розробки програмного забезпечення існує декілька популярних методологій, що слугують як настанови та маршрутні картки для команд і організацій. Дві з них, такі як Waterfall та Agile, стали класикою в даній галузі.

Спочатку розглянемо методологію Waterfall, визначимо її переваги та нодоліки.

Waterfall методологія - це традиційний лінійний підхід до управління проектами. Вона набула значної популярності в розробці програмного забезпечення через свою послідовність та легкість у використанні.

На самому початку проекту, що використовує Waterfall методологію, збираються детальні вимоги та специфікації. На цьому етапі фокусується на розумінні потреб клієнта та визначенні масштабу проекту. Все документується, і встановлюється початковий часовий графік, що надає чіткий шлях вперед.

Після того, як ці вимоги визначено, команда проекту розпочинає етап системного проектування. Мета тут полягає в створенні всебічного плану проекту, який використовує вимоги як зразок. Це планування допомагає розробникам зрозуміти архітектуру програмного забезпечення та передбачити потенційні проблеми.

За ним слідує етап реалізації, під час якого відбувається власне розробка програмного забезпечення або застосунку. Він базується на плані проектування, розробленому на попередньому етапі. Основний акцент робиться на переклад дизайну в функціональне програмне забезпечення.

Після цього програмне забезпечення тестується на наступному етапі, щоб переконатися, що воно працює як очікується та відповідає первинним вимогам. Це передбачає різні методи тестування, такі як модульне тестування, інтеграційне тестування та системне тестування.

Після успішних тестів відбувається етап розгортання, під час якого програмне забезпечення нарешті постачається замовнику або стає доступним для публічного використання.

Після релізу починається етап обслуговування. Під час цього етапу розробники роблять необхідні налаштування, оновлення та зміни за відгуками користувачів або еволюцією потреб.

Останній етап складається з відгуків та роздумів над завершеною роботою. Часто на цю стадію припадає обговорення потенційних пропозицій та вдосконалень для майбутніх проектів розробки програмного забезпечення.

Характерною особливістю Waterfall методології є її строгость: кожен етап повинен бути повністю завершений, перш ніж починається наступний. Прогрес відбувається послідовно від початку до кінця, подібно до водоспаду, що й дало назву цій методології. Однією з головних особливостей цієї моделі є те, що важко повернутися до попередніх етапів, як тільки вони завершені.

Модель Waterfall зазвичай розглядається як більш консервативний тип моделі, в якій вимоги встановлюються і передаються спочатку і тільки після релізу продукту, зворотній зв'язок надається. [1]

Нижче наведена ілюстрація стандартного Waterfall проєкту з жорстко сегментованими блоками часу. Це створює менталітет «використовуй або втрачай», який заохочує розробників, власників продукту та зацікавлених сторін вимагати якомога більше часу в кожному часовому вікні, оскільки в майбутньому може не бути можливості для повторення. Зазвичай команди, які використовують Waterfall, намагаються контролювати розповзання обсягу через «контроль змін», де всі погоджуються, що початковий контракт не змінено. [2]

Рис. 2.1.1 – Таймлайн стандартного Waterfall проєкту

Модель Waterfall може загострити деякі з відомих проблем, пов’язаних зі створенням продукту:

Блокувальники та керування залежностями: традиційні стилі управління проектами часто створюють «критичні шляхи», де проект не може рухатися вперед, доки не буде вирішено проблему блокування.

Труднощі з отриманням відгуків користувачів і валідацією продукту: на жаль, кінцевий клієнт не може взаємодіяти з продуктом, доки він не буде повністю готовий. Таким чином, важливі проблеми в дизайні та коді продукту залишаються нерозкритими до моменту релізу. [2]

Переваги Waterfall:

- вимагає менше координації через чітко визначені послідовні фази процесів;
- чітка фаза проекту допомагає чітко визначити залежності роботи;
- вартість проекту можна оцінити після визначення вимог;
- краще зосередження на документації проєктів і вимог;
- фаза проектування є більш методичною та структурованою перед написанням програмного забезпечення.

Недоліки Waterfall:

- важче розділити роботу через більш сувору послідовність фаз, команди більш спеціалізовані
- ризик втрати часу через затримки та збої під час фазових переходів

- додаткові вимоги до найму для виконання спеціалізованих етапів, тоді як гнучкість заохочує більш багатофункціональний склад команди.
- додаткові витрати на зв'язок під час передачі між фазовими переходами
- керування продуктом і залучення можуть бути не такими сильними порівняно з agile, оскільки фокус зосереджений на поточній фазі. [2]

2.2 Методологія Agile

У сучасному світі розробки програмного забезпечення та управління проектами Agile-методологія стає все більш популярною та визнаною. Підхід Agile забезпечує гнучкість, адаптивність та можливість швидко реагувати на зміни під час розробки проектів, що робить його особливо цінним у сфері технологій та програмування. У цьому розділі буде більш детально розглянуто основні принципи Agile-методології, її особливості та вплив на управління проектами в IT-компаніях. Також буде досліджено впровадження Agile-підходу на платформі Jira та його вплив на результативність команд та технологій управління проєктами в сфері IT [3].

Agile – це гнучка методологія управління проектами, яка розроблена для покращення ефективності та швидкості процесу розробки. Agile зосереджується на підтримці постійного діалогу між учасниками команди і, за потреби, готовий до миттєвого реагування та адаптації до змін.

Agile підхід розроблено для роботи в умовах, коли фінальні вимоги до продукту можуть змінюватися або постійно розвиваються. Він дозволяє командам виявляти та усувати питання на ранніх етапах, а також поступово і стабільно додавати цінність.

В Agile прийнято працювати в коротких циклах, які називаються ітераціями або спринтами. Кожен спринт триває від одного до чотирьох тижнів, і метою є розробка готового до перевірки користувачем продукту. Після

кожного спринта, команда аналізує результати своєї роботи, вузькі місця в процесі розробки, а також можливі зміни у вимогах.

Agile підтримує активну участь власника продукту та подалі від розробників в процесах прийняття рішень. Протягом всього життєвого циклу проекту рекомендується постійний контакт із зацікавленими сторонами. Agile підтримує культуру постійного вдосконалення та реагування на зміни, надаючи вищий пріоритет реакції на потреби клієнтів, ніж слідуванню початковому плану.

Agile спирається на принципи маніфесту Agile, який включає у себе такі цінності як індивіди та взаємодія більше, ніж процеси та інструменти; робочий продукт більше, ніж документація; співпраця з клієнтом більше, ніж угода про надання послуг; реагування на зміни більше, ніж виконання плану.

Agile розбиває зусилля розробки на фіксовані часові рамки, які називаються ітераціями. Через це бізнес має право очікувати, що розробники працюватимуть над найважливішими речами в будь-який момент часу, і що кожна ітерація забезпечить їм максимально можливу корисну бізнес-цінність. Цей пріоритет цінності визначається замовником під час сеансів планування на початку кожної ітерації. Клієнти обирають історії, які дають їм найвищу віддачу від інвестицій і які відповідають оцінці розробника для ітерації. [4]

Управління проектами за допомогою Agile використовує ітеративний підхід до розробки, створюючи кілька поетапних кроків із регулярними інтервалами зворотнього зв’язку. Це сприяє адаптації, оскільки команда може адаптуватися протягом усього процесу розробки продукту, а не обмежуватися лінійним шляхом. Це також дозволяє регулярно випускати результативні релізи, які дозволяють командам з часом досягати серії перемог.

Ітераційні випуски відкривають команді кілька можливостей:

- адаптуватися до обставин, що змінюються, від нещодавно виявлених вимог до заблокованої частини роботи.
- збирати відгуки від зацікавлених сторін під час процесу та повторюйте реакцію без напруги кінцевого терміну доставки.

- будувати стосунки та зв’язки між ролями, які полегшують людям зв’язок та ефективне спілкування.

Agile дозволяє командам бути більш стійкими до змін, які неминуче відбуваються під час проекту. [2]

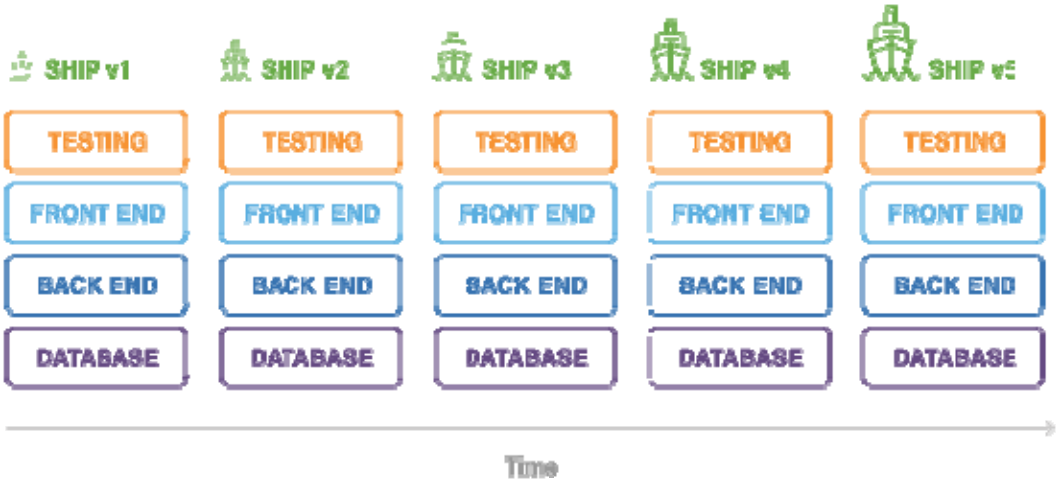

Рис. 2.2.1 – Таймлайн стандартного Agile проєкту

Agile принципи:

- гнучкий проект сегментується на кілька поступових кроків, які включають регулярні інтервали зворотного зв’язку.
- вимоги проекту сегментується на менші частини, які потім встановлюються за важливістю.
- сприяє співпраці, особливо з клієнтом.
- коригується через регулярні проміжки часу, щоб забезпечити задоволення потреб клієнта
- поєднує планування з виконанням, що дозволяє команді ефективно реагувати на мінливі вимоги

Переваги гнучкого управління проектами:

- швидші цикли зворотного зв'язку
- проблеми виявляються на початковому етапі
- вищий потенціал для задоволення клієнтів
- час виходу на ринок значно скорочено
- краща видимість / підзвітність
- віддані команди з часом підвищують продуктивність

- гнучке встановлення пріоритетів, зосереджене на доставці цінності

Недоліки agile:

- критичний шлях і міжпроєктні залежності можуть бути нечітко визначені, як у Waterfall
- існує організаційна вартість кривої навчання
- справжнє гнучке виконання з конвеєром безперервного розгортання має багато технічних залежностей і витрат на розробку

Гнучке управління проектами — це інноваційний підхід не лише для програмних проектів, а й для проектів усіх мастей. Забезпечуючи гнучкість реагування на зміни протягом життєвого циклу розробки, гнучкість дозволяє командам постачати продукти вищої якості, які відповідають потребам клієнтів. Agile розширює можливості команд, створює підзвітність і заохочує інновації, сприяючи постійному вдосконаленню. Agile дає вам можливість реагувати на зміни, не сходячи з колії. І це чудово для будь-якої програми. [2]

Ніхто не хоче поставляти програмне забезпечення з багатьма помилками, проблемами з продуктивністю та низьким рівнем задоволеності клієнтів. Постійна інтеграція та перегляд коду допомагають запобігти цьому.

Agile розробники програмного забезпечення зосереджуються на сталому розвитку, а не на героїзмі. Стійкість — це хороша оцінка, ефективні стратегії розгалуження для керування кодом, автоматизоване тестування для захисту якості та постійне розгортання для швидкого отримання відгуків від користувачів.

Застосування практик сталого розвитку потребує дисципліни, до якої більшість із нас прагне, але часто важко усвідомити це як особи. Це тому, що ніхто не може розвивати agile або DevOps у вакуумі. За цим має згуртуватися культура всієї організації, і іноді потрібен агент змін, наприклад інженер DevOps. Це означає змусити керівників проектів прийняти думку про те, що якість важливіша за обсяг чи графік, що часто є найскладнішою частиною впровадження гнучкості.

Розробники отримують свободу та підзвітність для стабільної розробки програмного забезпечення, зберігаючи при цьому чудові стосунки з бізнесом. І

бізнес отримує можливість виводити на ринок продукт вищої якості, що ще більше зміцнює ці чудові стосунки з технікою. Більше того, Agile розробники програмного забезпечення рідко стикаються з «маршами смерті». Коли розробка відстає від графіка, тому що підтримання високої якості потребує більше зусиль, ніж очікувалося, сторона залізного трикутника може зігнутися, щоб пристосуватися до цієї реальності – і нічиї вихідні не будуть викрадені.

Усі розробники програмного забезпечення знають «залізний трикутник» управління проектами: обсяг, графік і якість. І більшість із нас працювали над проектами, масштаби яких були негнучкими, графік спотворювався, а розробка була перевантажена зростаючим технічним боргом. Іноді – просто щоб додати образи до травми – кінцевий продукт був навіть не таким, яким хотів ринок. Це неприємне і трагічно знайоме місце, де варто бути.

Завдяки Agile розробці програмного забезпечення обсяг стає динамічною змінною, тож команди можуть захищати якість, створювати динамічну культуру розробки та залишатися тісно пов’язаними з бізнесом.

Agile — це командна робота, що не дивно, оскільки сьогодні більшість програмного забезпечення створюється командами. Розробники будують міцні стосунки з управлінням продуктом, дизайном, контролем якості та операціями, тому що написання стабільного коду означає залишатися на зв’язку з усіма аспектами проекту.

Наставництво теж важливе. Agile команди перехресно навчають одна одну, щоб забезпечити поширення знань про кодову базу серед команди. Один із способів зробити це — перевірка коду, яка не тільки захищає якість, але й сприяє знайомству з кодом у команді. Незалежно від того, як поширюються знання, Agile команди не закінчуються розробниками критичного шляху, які не можуть піти у відпустку, тому що вони єдині, хто розуміє певну область коду. Тому що ніхто не хоче бути таким розробником.

Agile розробники також можуть працювати вгору та вниз у технологічному стеку свого продукту з більшою легкістю, ніж їхні колеги з водоспаду, оскільки Agile команди самоорганізовуються, даючи членам можливість отримати нові навички. Фактом є те, що розробники, які надають

цілі функції – від інтерфейсу користувача до БД – більше володіють своїм кодом.

До популярних методологій Agile відносяться Scrum, Kanban, Extreme Programming (XP) та Lean, кожна з яких має свої власні принципи та підходи. Agile дозволяє швидко реагувати на зміни у вимогах проекту, зменшувати ризики та прискорювати темпи розробки. Цей підхід часто знаходить своє використання в сфері розробки програмного забезпечення, а також у багатьох інших галузях, де важливе бути гнучким та реагувати на зміни в середовищі виконання проектів. [5]

Розглянемо кожну методологію окремо та виділимо переваги і недоліки.

2.3 Методологія Scrum

Scrum – це підтип Agile-методології, орієнтований на виконання складних проектів в короткі терміни. Scrum був розроблений у 1990-х роках та отримав свою назву від позиції гравців в регбі під час змагань за м'яч.

Scrum є однією з найпоширеніших Agile-методологій управління проектами, спряморованою на ітераційний та інкрементальний підхід до розробки програмного забезпечення. Вона базується на невеликих, фіксованих за часом ітераціях, називаних "спрінтами" (Sprints), які зазвичай тривають від 1 до 4 тижнів. У кожному спрінті визначаються конкретні завдання та цілі для досягнення.

Основні складові у методології Scrum:

1. Ролі:
    - Product Owner: відповідає за формування вимог клієнта та пріоритети завдань. представляє інтереси замовника.
    - Scrum Master: відповідає за впровадження Scrum та допомагає команді вирішувати будь-які проблеми.
    - Розробники: команда, що відповідає за виконання задач та досягнення цілей проекту.
2. Збори:

- Sprint Planning: обговорення та планування завдань на наступний спрінт.
- Daily Standup: короткі зустрічі кожен день для узагальнення виконаної роботи та планування наступних кроків.
- Sprint Review: перегляд виконаної роботи та збирання відгуків.
- Retrospectives: перевірка процесів, вирішення існуючих проблем та визначення сфер, які потрібно вдосконалити

3. Артефакти:
    - Product Backlog: список всіх вимог до проекту.
    - Sprint Backlog: список завдань, запланованих на певний спрінт.
    - Інкремент: готовий продукт після завершення кожного спрінту.

Власники продукту (Product owner) Scrum розуміють вимоги клієнтів і бізнесу, а потім створюють і керують невиконаними продуктами на основі цих вимог. Оскільки гнучкі команди за своєю конструкцією є гнучкими та чуйними, власник продукту несе відповідальність за те, щоб вони приносили найбільшу цінність. Бізнес представлений власником продукту, який розповідає розробці, що важливо забезпечити. Довіра між цими двома ролями має вирішальне значення.

Власник продукту повинен не тільки розуміти клієнта, але й мати бачення цінності, яку команда scrum надає клієнту. Власник продукту також збалансовує потреби інших зацікавлених сторін в організації.

Отже, власник продукту повинен взяти всі ці дані та визначити пріоритети роботи. Це, мабуть, їхній найважливіший обов’язок, тому що суперечливі пріоритети та нечіткі напрямки не лише зменшать ефективність команди, але й можуть порушити важливі довірчі відносини, які компанія має з командою розробників.

Agile команди створені для перевірки та адаптації. Це означає, що зміна пріоритетів може призвести до значних змін у структурі команди, результатах

роботи, а також кінцевому результаті. Тому важливо, щоб scrum-команди були успішними та щоб лише одна особа встановлювала пріоритети. Ця особа є власником продукту.

Scrum Guide визначає обов'язки власників продукту як:

- Управління беклогом scrum. Це не означає, що вони єдині, хто розміщує в беклогі елементи нового продукту. Але зрештою вони відповідальні за відставання, яке команда розробників намагається вирішити. Це означає, що власник продукту повинен знати про все, що є в беклозі, а інші люди, які додають елементи до беклогу продукту, повинні переконатися, що вони спілкуються з власником продукту. Ми навіть пропонуємо безкоштовний шаблон scrum, який спрощує налагодження та роботу вашого беклогу.
- Управління релізами. Спринт — це не цикл релізу, а цикл планування. Це означає, що scrum-команди можуть досягти результатів у будь-який час. В ідеалі вони мали б часто доставляти дані протягом спринту, дозволяючи аналізу спринту перевірити реальне використання клієнтами та відгуки. Однак безперервна доставка не завжди можлива, тому потрібні інші моделі випуску. Для власника продукту важливо знати, коли речі можна і потрібно випускати.
- Управління зацікавленими сторонами. До будь-якого продукту буде залучено багато зацікавлених сторін, починаючи від користувачів, клієнтів, керівництва та керівництва організації. Власнику продукту доведеться працювати з усіма цими людьми, щоб ефективно переконатися, що команда розробників приносить користь. Це може означати великий обсяг управління зацікавленими сторонами та спілкування.

Роль власника продукту в Scrum є винятково важливою, оскільки він має безперервний вплив на визначення та реалізацію цільового стану продукту. Він є мостом між командою, клієнтами, користувачами та іншими стейкхолдерами, що забезпечує узгодженість вимог та виконання роботи.

Скрам-майстер — це роль, яка відповідає за те, щоб склеїти все разом і переконатися, що скрам виконується добре. На практиці це означає, що вони допомагають власнику продукту визначати цінність, команді розробників забезпечувати цінність, а команді scrum ставати кращими. Scrum master — це лідер-слуга, який не лише описує підтримуючий стиль лідерства, але описує те, що вони роблять на щоденній основі.

Вони служать власнику продукту, допомагаючи йому краще розуміти та передавати цінності, керувати невиконаними документами, допомагати їм планувати роботу з командою та розподіляти цю роботу, щоб забезпечити найефективніше навчання. Обслуговуючи команду розробників, скрам-майстер допомагає їм самоорганізуватися, зосередитися на результатах, досягти «зробленого приросту» та керувати блокувальниками. Scrum master також служить організації в цілому, допомагаючи їм зрозуміти, що таке scrum, і створити середовище, яке підтримує scrum.

Скрам-майстер фокусується на:

- Прозорість. Для ефективної перевірки та адаптації важливо, щоб потрібні люди бачили, що відбувається. Але насправді це набагато складніше, ніж здається. Завдання scrum-майстра полягає в тому, щоб команда scrum працювала прозоро. Приклади включають створення карт історії та оновлення сторінок Confluence ретроспективними ідеями.
- Емпіризм. Основна для scrum і agile підходів ідея про те, що найкращий спосіб планування — це виконувати роботу та вчитися на цьому. Емпіричний процес непростий і вимагає, щоб scrum-майстер навчав команду scrum щодо розподілу роботи, опису чітких результатів і перегляду цих результатів.
- Самоорганізація. Сказати команді розробників, що вони можуть самоорганізуватися, означає, що команда самоорганізується. Насправді самоорганізація приходить з часом і потребує допомоги та підтримки. Скрам-майстер заохочуватиме членів команди вийти за межі своєї зони комфорту та спробувати різні речі та

використовувати такі практики, як «покер делегування», щоб викрити та кинути виклик попередньо визначеним уявленням про межі ролей та обов’язки.

- Цінності. Scrum визначає 5 цінностей: сміливість, цілеспрямованість, відданість, повага та відкритість не тому, що їх приємно мати, а тому, що вони створюють середовище фізіологічної безпеки та довіри. Це середовище необхідне для процвітання спритності. Дотримання цінностей є обов’язком кожного члена команди scrum, але scrum master бере активну роль у заохоченні та нагадуванні всім про важливість цих цінностей.

Scrum master обслуговує власника продукту в плануванні спринту та огляді спринту, гарантуючи, що цінність чітко описана та встановлено напрямок. Вони обслуговують команду розробників у щоденних сутичках, гарантуючи, що робота відбувається та що блокувальники видаляються. Вони також беруть на себе відповідальність за блокери, які не в змозі команди вирішити. Scrum-майстер гарантує, що кожна можливість покращення стає прозорою для команди scrum, а ретроспектива має чіткий набір результатів, які можна виконати.

Scrum Master не є традиційним "менеджером". Він більше фасилітатор і коуч для команди, що постійно прагне до навчання, зростання та покращення. Ключовим є те, що Scrum Master створює умови, за яких команда може самостійно рухатися до своїх цілей.

Scrum дозволяє командам швидко реагувати на зміни вимог та вдосконалювати продукт. Використання Scrum сприяє покращенню комунікації в команді, підвищенню продуктивності та результативності проекту. Вона є популярною не лише в сфері розробки програмного забезпечення, але й в багатьох інших галузях, де використання Agile-принципів відображається на покращенні ефективності управління.

Ще однією важливою відмінністю Scrum від інших методологій є використання справ (дослівно "Scrum meeting") – коротких щоденних зустрічей, які тривають 15 хвилин. Їх ціль - усім членам команди повідомити про те, що

вони зробили з моменту останнього збору, над чим працюватимуть наступного дня, і які є поточні проблеми або перешкоди.

Ці та багато інших елементів Scrum спрямовані на підтримку гнучкості, стійкості до змін і ефективного спілкування між членами команди.

Scrum охоплює невизначеність і творчість. Він створює структуру навколо процесу навчання, дозволяючи командам оцінювати як те, що вони створили, так і, що не менш важливо, як вони це створили. Фреймворк Scrum використовує те, як команди насправді працюють, і дає їм інструменти для самоорганізації та швидкого підвищення як швидкості, так і якості роботи. У своїй основі Scrum базується на простій ідеї: щоразу, коли ви починаєте проект, чому б не регулярно перевіряти його, перевіряти, чи те, що ви робите, рухається в правильному напрямку, і чи це насправді те, чого хочуть люди? І запитайте, чи є способи покращити те, як ви робите те, що ви робите, якісь способи робити це краще та швидше, і що може заважати вам це робити. [6]

Переваги методології Scrum:

- Гнучкість: спроможність швидко реагувати на зміни вимог та наголошення на результативності та прозорості.
- Висока взаємодія замовника: Scrum активно залучає замовника у процес розробки, що сприяє відповідності результату очікуванням замовника.
- Швидке впровадження нового функціоналу: кожен спрінт завершується готовим інкрементом, що дозволяє оперативно впроваджувати новий функціонал.
- Ефективне управління ризиками: часті ітерації дозволяють швидко виявити проблеми та внести корективи.
- Зосередженість на важливих завданнях: сконцентрованість на обмеженій кількості завдань під час кожного спрінту сприяє уникненню розпорошення уваги та зосередженості на важливих завданнях.

Недоліки методології Scrum:

- Потребує залучення всієї команди: команда повинна бути добре згуртованою та залученою в процес розробки для успішного впровадження Scrum.
- Залежність від дисципліни команди: scrum передбачає виконання всіх ролей та обов'язків, тому вимагає дисциплінованості від усіх учасників.
- Вимагає чітких вимог: недостатня чіткість в вимогах може призвести до невизначеності під час планування та виконання спрінтів.
- Впроваджується не завжди успішно: впровадження Scrum може потребувати часу та зусиль, особливо для команд, які працювали за традиційними методологіями управління проектами.

Незважаючи на ці недоліки, методологія Scrum залишається однією з найпопулярніших та вдалих Agile-методологій для управління проектами, особливо в сфері розробки програмного забезпечення.

2.4 Методологія Kanban

Методологія Kanban - це підхід до управління робочими процесами, що базується на візуалізації завдань та обмеженні робочого навантаження. Основною ідеєю Kanban є репрезентація потоку роботи на дошці, де кожне завдання представлене карткою та проходить через різні колонки, що відображають його статус.

У Kanban немає фіксованих ітерацій (таких, як спрінти в Scrum), і завдання можуть бути розподілені на команду в будь-який момент. Наприклад, коли команда вільна, вони можуть взяти наступне завдання з беклогу. Це дозволяє команді бути більш гнучкою й ефективно управляти робочим процесом.

Основні етапи робочого процесу, як правило, включають у себе колонки для завдань у розробці, перевірці, тестуванні та завершенні. Кожне завдання пересувається через ці колонки, що візуалізує його прогрес та поточний статус.

Крім того, завданням можуть бути надані кольорові мітки або відображені іншим чином для візуального виділення важливих або додаткових ознак.

Методологія Kanban спрямована на оптимізацію робочих процесів, уникнення перевантаження роботою та ефективне управління робочим потоком. Kanban допомагає командам виокремлювати та вирішувати проблеми в робочих процесах, що забезпечує покращення продуктивності та якості виконання завдань.

Переваги методології Kanban:

- Гнучкість: Kanban дозволяє гнучко вирішувати, коли та як робити зміни в поточних процесах.
- Оптимізація робочого потоку: Kanban допомагає зменшити час, який споживають завдання на перехід з одного етапу в робочому процесі на інший.
- Відсутність перевантаження: завдяки обмеженню кількості робочих завдань на кожному етапі можливо уникнути переливання та залучити команду до більш ефективної роботи.
- Візуалізація робочого процесу: Kanban дошка надає чітку картину поточного статусу завдань, що полегшує врахування загального стану процесу.

Недоліки методології Kanban:

- Можлива зміна пріоритетів: без чіткого управління пріоритетами завдань можливе змішування важливих та менш важливих завдань, що може призвести до невизначеності.
- Відсутність фіксованих ітерацій: якщо не належно керувати процесом виконання завдань, можливе збільшення часу вирішення одних завдань та зменшення інших, що може вплинути на успішність реалізації проекту.
- Важкості в горизонтальних проєктах: для складних проектів, які потребують великої кількості різноманітних завдань, Kanban може бути менш ефективним у порівнянні з іншими методологіями.

Незважаючи на ці недоліки, методологія Kanban показує високу ефективність в оптимізації робочих процесів та контролі за виконанням завдань.

2.5 Методологія XP

Екстремальне програмування (Extreme Programming, XP) - це методологія розробки програмного забезпечення, яка ставить у центр гнучкість та високу якість продукту. XP сприяє ефективному управлінню змінами під час розробки та активному залученню клієнта до процесу створення програмних продуктів.

Групи XP виконують практично всі дії з розробки програмного забезпечення одночасно. Аналіз, проектування, кодування, тестування та навіть розгортання відбуваються дуже часто. Це багато, що потрібно зробити одночасно. XP робить це, працюючи в ітераціях: тижневі кроки роботи. Кожного тижня команда трохи планує випуск, трохи розробляє дизайн, трохи кодує, трохи тестує і так далі. Вони працюють над історіями: дуже маленькими функціями або частинами функцій, які мають споживчу цінність. Щотижня команда публікує від чотирьох до десяти історій. Протягом тижня вони працюють над усіма етапами розробки кожної історії. Наприкінці тижня вони розгортають своє програмне забезпечення для внутрішньої перевірки. [7]

Основні принципи і практики XP включають:

- Парне програмування (Pair Programming): два розробники спільно працюють над тим же завданням, обговорюючи та перевіряючи один одного.
- Тестування перед написанням коду (Test-Driven Development, TDD): розробники спочатку пишуть автоматичні тести, а потім реалізують код, що дозволяє виявляти та виправляти помилки на ранніх стадіях.
- Інтеграція та постіна поставка (Continuous Integration & Continuous Delivery): інтеграція коду здійснюється автоматично та регулярно, а результати проекту готові для поставки в будь-який момент.

- Проміжні випуски (Small Releases): програмне забезпечення розробляється і випускається невеликими інкрементами, що дозволяє швидко вносити зміни та оцінювати їх вплив.
- Проста архітектура (Simple Design): розробники ставлять перед собою завдання максимально спростити архітектуру програмного забезпечення.
- Включення клієнта (Customer Involvement): залучення замовника та користувачів для отримання постійного зворотнього зв'язку та правильного напрямку розвитку продукту.
- Визначеність стандартів (Coding Standards): набір правил та стандартів для писання коду, що сприяє покращенню якості та зрозумілості програмного забезпечення.

Ключовою метою XP є створення високоякісного програмного забезпечення, що відповідає вимогам користувачів та здатне швидко адаптуватися до змін. Вона активно застосовується в сфері розробки програмного забезпечення та дозволяє командам досягати високих результатів у процесі розробки програмних продуктів.

Команди XP можуть складатися з одного досвідченого програміста та одного менеджера з продукту, але повний XP може бути надмірним для такої маленької команди. Найменша команда, яку можна використовувати із повним XP, складається з п’яти осіб: чотирьох програмістів (один виконує роль тренера) та одного менеджера продукту (який також виконує функції керівника проекту, експерта з домену та тестувальника). Команда такого розміру може виявити, що менеджер із продуктів перевантажений; якщо так, програмістам доведеться втрутитися. Додавання експерта або тестувальника домену допоможе. [7]

Переваги методології Extreme Programming (XP):

- Якість продукту: XP спрямована на створення високоякісного програмного забезпечення завдяки використанню методів тестування, парного програмування та зобов'язання до простої архітектури.

- Реагування на зміни: методологія XP дозволяє швидко та ефективно реагувати на зміни в вимогах користувачів та ринкових умов.
- Активне залучення замовника: XP передбачає постійне залучення замовника для забезпечення відповідності продукту вимогам та очікуванням.
- Більша відповідальність: кожен учасник команди в XP бере на себе більшу відповідальність за результат та якість продукту.

Недоліки методології Extreme Programming (XP):

- Постійний натиск: парне програмування, тестування перед написанням коду та інші практики XP можуть нести значний ментальний навантаження для команди.
- Вимоги до часу: тестування перед написанням коду та інші стандарти якості можуть займати більше часу, що може призвести до збільшення часу витрат на кожну задачу.
- Важкість впровадження: для підприємств, які використовують традиційні методи розробки, перехід до XP може потребувати зміни в організаційній культурі та практиках.

Методологія XP показує себе надзвичайно ефективно в умовах, де важливо високоякісний та гнучкий продукт. Однак, вона може потребувати від команди значного зусиль та відповідальності для успішного впровадження.

2.6 Методологія Lean

Методологія Lean базується на концепції створення вартості для клієнта шляхом визначення, усунення та мінімізації зайвих витрат у робочих процесах. Це досягається за допомогою оптимізації робочих процесів, встановлення стандартів ефективності та покращення якості продукції шляхом врахування вимог клієнта та уникнення марнотратства. Методологія Lean розглядається як система, що прагне завжди поліпшуватися, виявляючи та корінно вирішуючи проблеми, що виникають у виробничому середовищі, щоб досягти найкращих результатів.

Основні принципи Lean включають в себе визначення вартості з орієнтацією на клієнта, оптимізацію робочих процесів шляхом усунення зайвих витрат, підвищення продуктивності та якості шляхом створення стандартів та проведення систематичних покращень. Lean спонукає до постійного виявлення та усунення всіх видів зайвих витрат, труднощів і непослідовностей у виробничих і ділових процесах.

До ключових принципів відноситься:

- керуйтеся попитом клієнтів – не робіть нічого, доки клієнт не вимагатиме цього.
- збільште потік – як тільки щось розпочато, закінчіть це.
- зрештою, якщо застосувати наведене вище правило, хтось хоче це зараз, тож навіщо це тримати?
- визначайте та видаляйте відходи – навчіться визначати відходи в усіх їх формах: матеріал, час і ресурси. Потім зробіть усі необхідні кроки для його усунення.
- оголосити війну варіаціям – мінливість вбиває. Це створює атмосферу невизначеності, що спричиняє багато марнотратства, згаданого вище. Систематично виявляйте та усувайте першопричину, а не лише симптоми.
- організуйте своїх людей навколо результатів, які ви хочете – створіть організаційну структуру навколо доставки ваших продуктів або послуг клієнтам. Переконайтеся, що існує чітка індивідуальна або командна відповідальність за наскрізну доставку.
- озбройте своїх співробітників на всіх рівнях навичками, щоб успішно виконувати свої ролі.
- чіткі та прості заходи та засоби контролю – економічне середовище є швидким, і все відбувається дуже швидко, тому вам потрібно бути в курсі ситуації та мати ранні індикатори попередження про критичні моменти бізнесу.
- намалюйте чітку та переконливу картину того, як виглядає майбутнє у вашій «ідеальній державі» – а потім розкажіть усім!

Цей останній пункт є абсолютно критичним. Без чіткого розуміння того, куди ви йдете, ваша організація буде плутатися і знаходити власні напрямки, які, ймовірно, не будуть узгоджуватися з цілями бізнесу і, звичайно, не узгоджуватимуться одне з одним. Моя улюблена цитата якогось усміхненого кота з відомої дитячої книжки ідеально підсумовує це: «Якщо ти не знаєш, куди йдеш, будь-яка дорога приведе тебе туди». Якщо ви починаєте програму змін без чіткий пункт призначення, коли ви дійдете «туди», ймовірно, це буде не те місце, де ви дійсно хотіли бути, і це займе втричі більше часу, ніж ви очікували. Недобре, тому докладніше про це пізніше.

Є багато інших принципів і правил, які детально описують конкретні теми, але для мене ці є кардинальними, і вони настільки фундаментальні, що варто витратити трохи часу, щоб зрозуміти їх більш детально. [8]

Ця методологія розглядається як культурна трансформація, яка стимулює постійні зміни, покращення та вдосконалення в усіх аспектах підприємства, незалежно від його розміру чи галузі. Lean активно застосовується в різних галузях, від виробництва до послуг, допомагаючи підприємствам покращити ефективність, зменшити затрати та вдосконалити якість продукції.

Переваги методології Lean:

- Зменшення витрат: Lean спрямована на усунення зайвих витрат, що дозволяє зменшити витрати виробництва та ділових процесів.
- Покращення якості: методологія Lean допомагає забезпечити стабільну якість продукції та послуг завдяки усуненню зайвих етапів та стандартизації процесів.
- Збільшення продуктивності: зменшення часу та зусиль, спрямованих на виконання зайвих завдань, призводить до підвищення продуктивності та ефективності.

Недоліки методології Lean:

- Вимагає культурної зміни: впровадження Lean значить культурну трансформацію підприємства, що може бути складним та вимагати великих зусиль.

- Складнощі впровадження: Lean може вимагати значної часу та зусиль для успішного впровадження та дотримання стандартів.
- Суперечності: іноді стандартизація та усунення зайвого може призвести до конфліктів між підприємством та його споживачами або постачальниками.

Мислення методологією Lean – це певною мірою те, чим є певні люди вроджена здатність бачити іншими очима. Однак більшість людей можна навчити принаймні замислюватися про те, що їх оточує, і сказати собі: «Повинен бути простіший спосіб зробити це».

Це, у поєднанні з хорошим знанням деяких основних інструментів і технік, забезпечить вам довгий шлях у подорожі. Однак одна чи дві особи в бізнесі, які мають «правильні речі», не можуть цього зробити.

Успіх вимагає хорошого лідерства, критичної маси людей, які певною мірою його досягають, і пристрасті з боку організації в цілому. [8]

Насамперед, вивчення Agile методологій, таких як Scrum, Kanban, Extreme Programming (XP) та Lean, дозволяє краще розуміти принципи гнучкого управління проектами та розробки програмного забезпечення. Кожна з цих методологій має свої унікальні переваги та особливості, які можуть бути використані в різних контекстах та сферах діяльності.

Scrum стимулює ефективну комунікацію та регулярні ітерації, Kanban спрямований на візуалізацію робочих процесів та управління потоком роботи, XP покликаний забезпечити високу якість, а Lean спрямований на усунення витрат та покращення продуктивності.

Agile підходи можуть бути надзвичайно корисними для організацій, які прагнуть досягнути гнучкості, ефективності та високої якості у своїй діяльності. Вони стимулюють постійне вдосконалення, сприяють взаємодії замовника, підвищують продуктивність та сприяють більш успішному управлінню проектами. Крім того, кожна методологія може бути адаптована для конкретних потреб та особливостей кожного проекту або організації.

Серед різних Agile методологій Scrum та Kanban вважаються одними з найефективніших управлінських практик для команд розробки програмного забезпечення та проєктного менеджменту.

Scrum є популярною методологією, яка базується на інкрементальному плануванні, регулярних статусних зустрічах та підсумковому перегляді виконаної роботи. Його структурований підхід до управління проєктами та акцент на ітераціях дозволяють командам ефективно керувати завданнями та реагувати на зміни вимог. Scrum сприяє підвищенню продуктивності та ефективності команди завдяки встановленому процесу та структурі методології.

Kanban, з іншого боку, наголошує на візуалізації робочих процесів та управлінні потоком роботи. Він допомагає командам відстежувати та керувати завданнями, уникати перевантаження та полегшує управління завданнями у реальному часі.

Обидві методології створюють можливість для ефективного розподілу та виконання завдань, а також сприяють постійному покращенню процесів розробки. Таким чином, виокремлення Scrum та Kanban з-поміж інших Agile методологій обґрунтоване їхньою ефективністю та вдалим використанням у сучасному управлінні проєктами в IT-компаніях.

3 ІНСТРУМЕНТИ ДЛЯ ВПРОВАДЖЕННЯ МЕТОДОЛОГІЙ

Існує кілька популярних платформ, які дозволяють командам впроваджувати та керувати Agile методологіями. Деякі з найбільш відомих платформ включають:

- Jira: створена для планування, виконання та відстеження програмних проектів шляхом використання Agile методологій, зокрема Scrum та Kanban.
- Trello: управління завданнями та проєктами з використанням дошок, списків та карток, що може бути налаштоване для впровадження Agile методологій.
- Asana: платформа для управління проєктами, завданнями та комунікацією, що надає засоби для реалізації Agile практик.
- Microsoft Azure DevOps: інтегрована набір інструментів для спільної розробки програмного забезпечення, яка підтримує Agile управління проектами.
- Targetprocess: платформа для управління проєктами та командами, що надає інструменти для впровадження Agile методологій та практик.

Ці платформи допомагають командам ефективно впроваджувати та керувати проектами за допомогою Agile методологій, надаючи засоби для планування, відстежування та виконання завдань у відповідності з вибраною методологією.

Розглянемо лише основні та найбільш популярні платформи.

Jira - це популярна платформа управління проектами, яка розроблена компанією Atlassian і широко використовується для планування, відстеження та керування різноманітними видами проєктів за допомогою Agile методологій. Jira надає широкий спектр можливостей, які дозволяють командам розробників, менеджерам проєктів та стейкхолдерам ефективно керувати робочим процесом, ефективно комунікувати та спільно працювати над реалізацією проєктів.

Основні функції Jira включають:

- Створення завдань: користувачі можуть створювати, призначати та відстежувати завдання для подальшого виконання.
- Спрощене планування: Jira надає засоби для планування та призначення завдань, встановлення пріоритетів та визначення термінів виконання.
- Управління ресурсами: платформа дозволяє керувати ресурсами, встановлювати завдання та визначати роль кожного члена команди.
- Підтримка Agile методологій: Jira надає підтримку для Scrum, Kanban та інших Agile практик, що сприяє організації процесів розробки.
- Формування звітів: платформа дозволяє створювати звіти про прогрес, витрати, якість та ефективність проєктів.
- Інтеграція з іншими інструментами: Jira інтегрується з багатьма іншими інструментами для розробки, тестування та спільної роботи.

Загалом, Jira є потужним інструментом для управління проектами, який надає широкі можливості для впровадження та керування Agile методологіями та підтримує команди в розробці програмного забезпечення, управлінні проектами та багатьма іншими сферами діяльності.

Trello - це веб-орієнтована платформа для спільної роботи, яка дозволяє створювати та керувати дошками, списками та картками. Перевагою Trello є його простота та зручність у використанні, що робить його привабливим вибором для осіб та команд, які шукають засіб для організації завдань та проєктів. Основні компоненти Trello включають:

- Дошки: користувач може створювати різні дошки для організації проєктів, завдань чи команд. Кожну дошку можна налаштовувати та редагувати на рівні користувача.
- Списки: кожна дошка містить окремі списки, які можуть відображати етапи або категорії завдань. Картки можна переміщати між списками для позначення прогресу чи зміни статусу завдання.

- Картки: картки представляють конкретні завдання, проєкти або ідеї й містять у них деталі, завдання, крайні строки, коментарі та відомості.
- Команди: Trello дозволяє користувачам створювати команди та спільно працювати над дошками та картками у межах командних проєктів.

Trello також надає велику кількість розширень та інтеграцій з іншими платформами, що дозволяє розширити його функціональні можливості. Загалом, Trello залишається популярним інструментом для управління завданнями та проєктами завдяки своїй простоті, зручності та можливостям налаштування.

Microsoft Azure DevOps, раніше відомий під назвою Visual Studio Team Services, є інтегрованою платформою для управління життєвим циклом розробки програмного забезпечення і DevOps-процесами. Ця платформа має широкий спектр інструментів, які допомагають надійно керувати проєктами, використовуючи принципи Agile, для підвищення продуктивності та якості програмного забезпечення. Основні компоненти Microsoft Azure DevOps включають:

- Azure Repos: інструмент контролю версій та керування кодом, який дозволяє зберігати, керувати та розгортати код.
- Azure Boards: використовується для планування, відстеження та керування завданнями та робочими потоками, включаючи підтримку Agile методологій, таких як Scrum та Kanban.
- Azure Pipelines: надає інструменти для автоматизованої збирання, тестування та розгортання програмного забезпечення, а також надає конфігурацію DevOps.
- Azure Test Plans: дозволяє створювати, керувати та виконувати тестові роботи для забезпечення якості програмного забезпечення.
- Azure Artifacts: надає можливості для керування пакетами та спільного використання коду через різноманітні репозиторії пакетів.

DevOps платформа Microsoft Azure DevOps, завдяки своїм інструментам, сприяє спільному використанню знань, автоматизації процесів та збільшенню продуктивності, що дозволяє створювати, тестувати та розгортати програмне забезпечення з більшою ефективністю й швидкістю.

Оцінюючи платформи Jira, Trello та Microsoft Azure DevOps з точки зору управління та керування проєктами і впровадження Agile методологій, Jira видається найбільш привабливою вибором.

Jira має вражаючий набір функцій, що дозволяє планувати, відстежувати, керувати та виконувати завдання, встановлювати пріоритети та використовувати Agile методології, такі як Scrum та Kanban. Вона надає ключові засоби для вирішення складних завдань управління проєктами, а також можливості для управління портфелями, звітності та інтеграції з іншими інструментами.

Trello, з іншого боку, є ефективним інструментом спільної роботи, але він має обмежені можливості порівняно з Jira. Він може бути корисним для невеликих команд та простих проєктів, але для складніших завдань, можуть бути необхідні додаткові функції.

Microsoft Azure DevOps, хоча й є потужним інструментом для спільної розробки програмного забезпечення та DevOps-процесів, зазвичай використовується для великих проектів і орієнтований на розробників програмного забезпечення та DevOps інженерів.

Отже, враховуючи можливості для керування проєктами та впровадження Agile методологій, Jira видається найкращим варіантом для багатьох компаній.

4 СУЧАСНІ ПРАКТИКИ УПРАВЛІННЯ ПРОЄКТАМИ В IT-КОМПАНІЯХ НА ПЛАТФОРМІ JIRA

4.1 Користувацькі історії

Користувацька історія (User Story) - це найменша одиниця роботи в гнучкій системі. Це кінцева мета, а не функція, виражена з точки зору користувача програмного забезпечення.

Користувацька історія є неформальним загальним поясненням функції програмного забезпечення, написаним з точки зору кінцевого користувача чи клієнта. [9]

Мета користувацької історії полягає в тому, щоб сформулювати, як частка роботи поверне певну вартість для клієнта. Зазначимо, що «клієнтами» не обов'язково повинні бути зовнішні кінцеві користувачі в традиційному сенсі, вони також можуть бути внутрішніми клієнтами або колегами вашої організації, в яких відомостям вашої команди.

Користувацькі історії - це кілька речень простою мовою, які видаляють очікуваний результат. Вони не деталізуються. Вимоги додаються пізніше після узгодження командою.

Історії гарно вписуються в гнучкі рамки, такі як scrum та kanban. У scrum, користувацькі історії додаються до спринтів та виконуються протягом тривалості спринту. Команди kanban притягують користувацькі історії до свого беклогу та виконують через свій робочий процес. Саме ця робота з користувацькими історіями допомагає scrum-командам покращувати оцінки та планування спринтів, що призводить до більш точного прогнозування та більшої гнучкості. Завдяки історіям, команди kanban вивчають, як управляти роботою в процесі (WIP) та можуть додатково вдосконалювати свої робочі процеси.

Користувацькі історії також є будівельними блоками більших гнучких рамок, таких як епіки та ініціативи. Епіки - це великі роботи, розгорнуті в набір історій, а кілька епіків складають ініціативу. Ці більші структури гарантують,

що повсякденна робота розробницької команди (над історіями) сприяє організаційним цілям, закладеним в епопеях та ініціативах.

Оскільки історії користувачів не містять технічного жаргону, вони зрозумілі як розробникам, так і команді клієнтів.

Кожна історія користувача представляє окрему частину функціональності; тобто те, що користувач, швидше за все, зробить в одному налаштуванні. Це робить історії користувачів доречними як інструмент планування. Ви можете оцінити цінність перенесення історій між випусками набагато краще, ніж ви можете оцінити вплив виключення одного чи кількох тверджень «Система повинна…».

Ітеративний процес — це процес, який прогресує шляхом послідовного вдосконалення. Команда розробників робить перший крок до системи, знаючи, що вона неповна або слабка в деяких (можливо, багатьох) областях. Потім вони послідовно вдосконалюють ці ділянки, поки продукт не стане задовільним. З кожною ітерацією програмне забезпечення вдосконалюється шляхом додавання більшої кількості деталей. Історії добре підходять для ітеративної розробки, оскільки також можна повторювати історії. Для функції, яку ви забажаєте з часом, але яка зараз неважлива, ви можете спочатку написати велику історію (епік). Коли ви будете готові додати цю історію в систему, ви можете вдосконалити її, розірвавши епік і замінивши меншими історіями, з якими буде легше працювати. [10]

4.2 Оцінювання (Estimation)

Оцінка важка. Для розробників програмного забезпечення це один із найскладніших — якщо не найважчий — аспектів роботи. Він повинен враховувати низку факторів, які допомагають власникам продуктів приймати рішення, що впливають на всю команду — і на бізнес.

Сторі поінти (Story points) – це одиниця вимірювання для вираження загального розміру історії користувача, функції чи іншої роботи. Коли ми оцінюємо бали історії, ми призначаємо значення балів кожному елементу.

Необроблене значення, яке ми призначаємо, не має значення. Важливі відносні значення. Історія, якій присвоєно двійку, має бути вдвічі більшою, ніж історія, якій присвоєно одиницю. Це також має бути дві третини історії, яка оцінюється як три сюжетні точки.

Кількість пунктів історії, пов'язаних з історією, представляє загальний розмір історії. Немає встановленої формули для визначення розміру історії. Швидше, оцінка точки історії – це об'єднання кількості зусиль, витрачених на розробку функції, складності її розробки, ризику, властивого цьому, тощо.

Є два поширених способи почати роботу. Перший підхід — вибрати історію, яка, на вашу думку, буде однією з найменших історій, з якою ви працюватимете, і сказати, що ця історія оцінюється в 1 бал. Другий підхід полягає в тому, щоб вибрати історію, яка здається дещо середнього розміру, і дати їй номер десь посередині діапазону, який ви очікуєте використовувати. Після того, як досить довільно призначено значення балів для першої історії, кожна додаткова історія оцінюється шляхом порівняння з першою історією або будь-якими іншими, які були оцінені. [11]

У розробці власнику продукту доручається визначити пріоритетність беклогу — упорядкованого списку робіт, який містить короткий опис усіх бажаних функцій і виправлень для продукту. Власники продукту вловлюють вимоги бізнесу, але не завжди розуміють деталі впровадження. Таким чином, хороша оцінка може дати власнику продукту нове уявлення про рівень зусиль для кожного робочого елемента, що потім повертається до його оцінки відносного пріоритету кожного елемента.

Коли команда інженерів починає процес оцінювання, зазвичай виникають запитання щодо вимог та історій користувачів. І це добре: ці запитання допомагають усій команді краще зрозуміти роботу. Спеціально для власників продуктів розбиття робочих елементів на деталізовані частини та оцінки за допомогою сторі поінтів допомагає їм визначити пріоритетність усіх областей роботи. І як тільки вони отримають оцінки від команди розробників, власник продукту нерідко змінює порядок елементів у беклозі.

Оцінювання — це командний вид спорту. Ключовим є залучення всіх (розробників, дизайнерів, тестувальників та ін.) до команди. Кожен член команди по-різному бачить продукт і роботу, необхідну для створення історії користувача. Наприклад, якщо керівництво продукту хоче зробити щось, що здається простим, розробники та тестувальники можуть мати іншу думку, зважаючи на попередній досвід.

Подібним чином зміни в дизайні вимагають не лише внеску команди дизайнерів, але й розробників та тестувальників. Залишення частини ширшої команди продукту поза процесом оцінки створює нижчі оцінки, знижує моральний дух, оскільки ключові учасники не відчувають себе включеними, і погіршується якість програмного забезпечення.

На жаль, сторі поінтами часто зловживають. Сторі поінти працюють неправильно, коли їх використовують для оцінки людей, призначають детальні часові рамки та ресурси, а також коли їх помилково сприймають як показник продуктивності. Натомість команди повинні використовувати сторі поінти, щоб зрозуміти масштаб роботи та пріоритетність роботи. [12]

4.3 Епіки

Епік (Epic) - це осяжна робота, яку можна розбити на кілька менших історій, іноді їх називають "завданнями" в Jira. Епіки часто охоплюють кілька команд, декілька проектів і навіть можуть відстежуватися на кількох дошках.

Епіки майже завжди випускаються протягом набору спринтів. При поступовому отриманні командою більше відомостей щодо епіку через розробку та зворотний зв'язок від клієнта, користувацькі історії будуть додаватися та зніматися за необхідності. Це головне в епіках: обсяг гнучкий, заснований на отриманому відгуку від клієнта та робочому ритмі команди. [13]

Епіки не є абсолютною основою гнучкої програми, але вони є практичними драйверами для більшості гнучких команд і команд DevOps. Розуміння того, де вони вписуються в здорову гнучку програму, створює контекст для вашої роботи, а розбиття їх на історії створює імпульс.

Епіки повинні бути конкретними та вимірюваними, щоб менеджери могли відстежувати їх і переконатися, що вони сприяють досягненню більших стратегічних цілей. Вони допомагають створити ієрархічну структуру в роботі та сприяють розстановці пріоритетів, щоб цінність доставлялася з оптимальною якістю та найшвидшим способом.

Елементи епіку в Agile складаються з назви, що відображає бізнес-мету, і менші користувацькі історії або завдання, які представляють окремі робочі елементи, які потрібно виконати. Історії користувачів можна поширювати між кількома командами, відділами та ієрархіями. Їх виконання матиме прямий вплив на епік, що, у свою чергу, позначається на стратегічних планах. У великих організаціях ряд епіків сприяє досягненню більших стратегічних цілей.

4.4 Дорожні карти

Дорожня карта (Roadmap) продукту визначає, як розвивається продукт чи рішення з часом. Дорожня карта в гнучкій розробці надає важливий контекст, який надає командам можливість досягати як інкрементальних, так і проектних цілей. Дорожні карти складаються з ініціатив, які представляють собою великі області функціоналу, і включають часові рамки, що вказують, коли функція буде доступна. По мірі роботи та отриманих командою знань приймається, що дорожня карта зміниться, щоб відобразити цю нову інформацію - можливо, дискретно або значно. Мета полягає у тому, щоб зосередити дорожню карту на поточних умовах, що впливають на проект та довгострокові цілі, щоб ефективно співпрацювати з зацікавленими сторонами та реагувати на конкурентне середовище. [14]

Наведено просту дорожню карту (див. рис. 4.3.1) для команди продукту, з ініціативами у квадратних блоках та часовими рамками, вказаними відмітками майлстоунів червоного кольору.

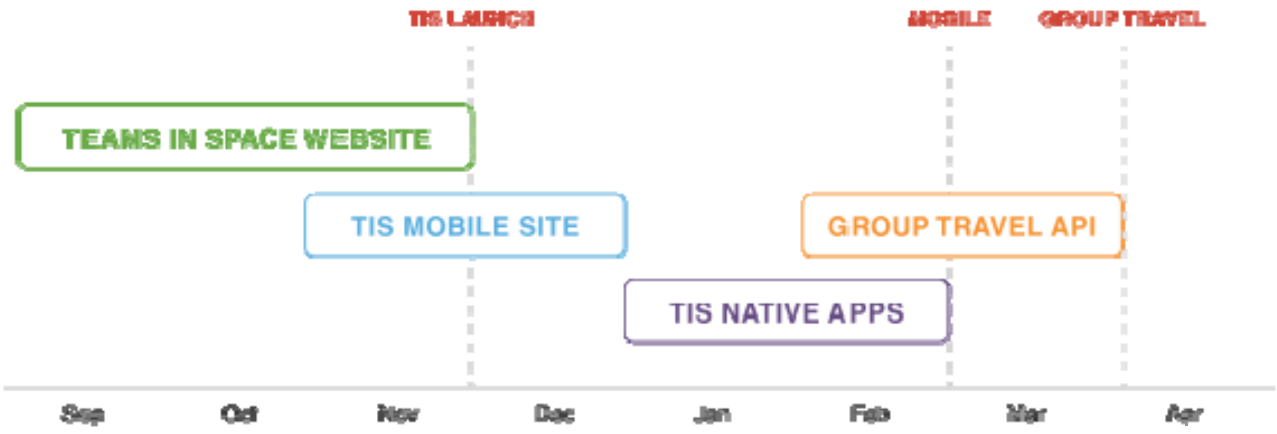

Рисунок 4.4.1 – Дорожня карта (Roadmap)

4.5 Вимоги

Кожна ініціатива в дорожній карті розкривається в набір вимог. Гнучкі вимоги - це легкі описи потрібного функціоналу, а не 100-сторінкові документи, що характерні для традиційних проектів. Вони змінюються з часом та базуються на загальному розумінні командою потреб клієнта та бажаного продукту. Гнучкі вимоги залишаються легкими, поки всі члени команди не розвивають спільне розуміння через безперервні розмови та співпрацю. Тільки тоді, коли приступають до втілення, вони конкретизуються повними деталями.

4.6 Беклог

Беклог (Backlog) встановлює пріоритети для гнучкої програми. Команда включає всі робочі елементи в беклог: нові функції, помилки, вдосконалення, технічні чи архітектурні завдання тощо. Власник продукту визначає пріоритетність роботи над беклогом для команди інженерів. Потім команда розробників використовує пріоритетні резерви як єдине джерело істини щодо того, яку роботу потрібно виконати.

Використовуючи Scrum, ми завжди виконуємо найціннішу роботу в першу чергу. Власник продукту, за допомогою решти команди Scrum та зацікавлених сторін, остаточно відповідає за визначення та керування послідовністю цієї роботи та передачу її у формі пріоритезованого (або

впорядкованого) списку, відомого як backlog продукту. Під час розробки нового продукту елементи резервування продукту спочатку є функціями, необхідними для задоволення бачення власника продукту. Для поточної розробки продукту резервна версія продукту також може містити нові функції, зміни в існуючих функціях, дефекти, які потребують ремонту, технічні вдосконалення тощо. [15]

Дорожня карта та вимоги команди є основою для беклогу продукту. Ініціативи дорожньої карти розбиваються на кілька епіків, і кожен епік матиме кілька вимог та історій користувачів.

Щоб правильно визначити пріоритети в беклозі треба враховувати:

- Клієнтський пріоритет
- Терміновість отримання зворотного зв'язку
- Відносна складність впровадження
- Симбіотичні відносини між робочими елементами (наприклад, B легше, якщо ми спочатку виконуємо A)

Хоча перед власником продукту поставлено завдання визначити пріоритетність невиконаних завдань, це не робиться на порожньому місці. Ефективні власники продукту шукають відгуків і відгуків від клієнтів, дизайнерів і команди розробників, щоб оптимізувати робоче навантаження та доставку продукту.

Після створення беклогу продукту важливо регулярно підтримувати його, щоб не відставати від програми. Власники продукту повинні переглядати відставання перед кожною нарадою з планування ітерації, щоб переконатися, що пріоритети правильні та відгуки з останньої ітерації включені. Регулярний перегляд невиконаних завдань часто називають «усуненням невиконаних завдань» у agile колах (деякі використовують термін уточнення невиконаних завдань).

Коли беклог стає більшим, власникам продуктів потрібно згрупувати відставання на короткострокові та довгострокові елементи. Короткострокові елементи мають бути повністю конкретизовані, перш ніж вони будуть позначені як такі. Це означає, що були складені повні історії користувачів,

налагоджена співпраця з дизайном і розробкою, а також зроблені оцінки розробки. Довготермінові елементи можуть залишатися дещо невизначеними, хоча було б гарною ідеєю отримати приблизну оцінку від команди розробників, щоб допомогти визначити їх пріоритетність. Ключове слово тут — «приблизне»: оцінки зміняться, коли команда повністю зрозуміє та почне працювати над цими довгостроковими пунктами.

Беклог служить зв'язком між власником продукту та командою розробників. Власник продукту може в будь-який час змінити пріоритети роботи в беклозі на основі відгуків клієнтів, уточнення оцінок і нових вимог. Однак після завершення роботи зведіть до мінімуму зміни, оскільки вони заважають команді розробників і впливають на фокус, потік і моральний дух.

Також інснують антишаблони на які треба звертати увагу.

- власник продукту визначає пріоритетність беклогу на початку проекту, але не коригує його, коли надходить відгук від розробників і зацікавлених сторін.
- команда обмежує елементи в беклозі тими, які спрямовані на клієнта.
- беклог зберігається як документ, який зберігається локально та нечасто надається до нього доступ, що не дозволяє зацікавленим сторонам отримувати оновлення.

Кмітливі власники продукту ретельно доглядають за беклогом продукту своєї програми, роблячи його надійним і доступним для спільного використання планом робочих елементів проекту.

Зацікавлені сторони оскаржуватимуть пріоритети, і це добре. Стимулювання дискусій навколо того, що важливо, синхронізує пріоритети кожного. Ці обговорення сприяють формуванню культури групового визначення пріоритетів, гарантуючи, що всі мають однакове мислення щодо програми.

Беклог продукту також служить основою для планування ітерацій. Усі робочі елементи мають бути включені до беклогу: історії користувачів, помилки, зміни дизайну, технічна заборгованість, запити клієнтів, дії з

ретроспективи тощо. Це гарантує, що робочі елементи кожного будуть включені до загального обговорення для кожної ітерації. Потім члени команди можуть досягти компромісів із власником продукту перед початком ітерації, маючи повне знання всього, що потрібно зробити.

4.7 Спринт

Scrum структурує розробку продукту в циклах роботи, які називаються спринтами, ітерації роботи, які зазвичай тривають 1–4 тижні.

Спринти мають фіксовану тривалість і закінчуються в певну дату, незалежно від того, завершено роботу чи ні; вони ніколи не подовжуються. [16]

Спринти є основою методології scrum і agile, і правильне спринтування допоможе вашій гнучкій команді створювати краще програмне забезпечення з меншими головними болями.

«За допомогою scrum продукт створюється за допомогою серії ітерацій, які називаються спринтами, які розбивають великі складні проекти на невеликі частини», — сказала Меган Кук, керівник відділу продуктів Jira в Atlassian. [17]

Багато хто асоціює scrum-спринти з agile розробкою програмного забезпечення настільки, що scrum і agile часто вважаються одним і тим же. Вони не є. Agile — це набір принципів, а scrum — це основа для того, щоб зробити все.

Багато подібностей між цінностями agile та процесами scrum призводять до справедливої асоціації. Спринти допомагають командам дотримуватися agile принципу «частої доставки робочого програмного забезпечення», а також реалізовувати цінність гнучкого «реагування на зміни замість дотримання плану». Цінності scrum прозорості, перевірки та адаптації доповнюють гнучкість і є центральними в концепції спринтів.

Переконайтеся, що команда ставить і розуміє мету спринту та те, як буде вимірюватися успіх. Це ключ до того, щоб усі були узгоджені та рухалися вперед до спільної мети.

Переконайтеся, що у вас є доглянутий беклог з вашими пріоритетами та залежностями в порядку. Це може бути великою проблемою, яка може зірвати процес, якщо ним не керувати належним чином.

Переконайтеся, що ви добре розумієте швидкість і що вона відображає такі речі, як відпустка та зустрічі команди.

Використовуйте нараду з планування спринту, щоб конкретизувати інтимні деталі роботи, яку потрібно виконати. Заохочуйте членів команди намітити завдання для всіх історій, помилок і завдань, які надходять у спринт.

Залиште роботу, де ви не зможете виконати залежні завдання, як-от роботу іншої команди, дизайни та інше.

Нарешті, коли рішення чи план прийнято, переконайтеся, що хтось зафіксував цю інформацію у вашому інструменті керування проектами. Таким чином, і рішення, і обґрунтування легко побачити для всіх пізніше.

Не вводьте занадто багато історій, не переоцінюйте швидкість і не включайте завдання, які неможливо виконати в спринті. Не треба налаштовувати себе або свою команду на невдачу.

Не треба забувати про якість або технічний борг. Обов’язково виділіть час на контроль якості та роботу, не пов’язану з функціями, як-от помилки та інженерний стан.

Не дозволяйте команді мати розпливчасте уявлення про те, що відбувається в спринті. Закріпіть це й не зосереджуйтесь на швидкості руху настільки, щоб забути переконатися, що всі рухаються в одному напрямку.

Крім того, не беріться за великий обсяг невідомої роботи або роботи з високим ризиком. Розділіть історії, які є великими або мають високу невизначеність, і не бійтеся залишити частину цієї роботи для наступного спринту.

Якщо почуєте занепокоєння від команди, будь то швидкість, робота з низьким рівнем точності або робота, яка, на їхню думку, є більшою, ніж вони оцінили, не ігноруйте це. Вирішіть проблему та за потреби виконайте повторне калібрування.

4.8 Планування спринта

Планування спринту – це подія, яка починає спринт. Метою планування спринту є визначення того, що може бути досягнуто під час спринту та як ця робота буде досягнута. Планування спринту здійснюється у співпраці з усією командою scrum. [18]

Спринт — це певний період часу, протягом якого виконується вся робота. Однак перш ніж ви зможете почати дію, вам потрібно налаштувати спринт. Вам потрібно вирішити, скільки часу буде тривати, мета спринту та місце старту. Сеанс планування спринту розпочинає спринт із визначення порядку денного та фокусу. Якщо все зроблено правильно, це також створює середовище, де команда мотивована, кидається викликом і може бути успішною. Погані плани спринту можуть зірвати команду, поставивши нереалістичні очікування.

- “Що” – власник продукту описує мету (або ціль) спринту та те, які елементи відставання сприяють досягненню цієї мети. Команда scrum вирішує, що можна зробити в майбутньому спринті та що вони будуть робити під час спринту, щоб це відбулося.
- “Як” – команда розробників планує роботу, необхідну для досягнення мети спринту. Зрештою, кінцевий план спринту — це переговори між командою розробників і власником продукту на основі цінності та зусиль.
- “Хто” – Ви не можете планувати спринт без власника продукту чи команди розробників. Власник продукту визначає ціль на основі цінності, яку він прагне. Команда розробників має зрозуміти, як вони можуть або не можуть досягти цієї мети. Якщо будь-який з них відсутній у цій події, це робить планування спринту майже неможливим.

Чудовою відправною точкою для плану спринту є беклог, оскільки він надає список «речей», які потенційно можуть бути частиною поточного

спринту. Команда також повинна розглядати поточну роботу, виконану в спринті, і мати на увазі потенціал.

Найважливішим результатом зустрічі з планування спринту є те, що команда може описати мету спринту та те, як вона почне працювати над досягненням цієї мети. Це видно в резерві спринту.

Для успішного планування спринту потрібна певна дисципліна. Власник продукту повинен бути підготовлений, об'єднавши уроки з попереднього аналізу спринту, відгуки зацікавлених сторін і бачення продукту, щоб вони підготували сцену для спринту. Для прозорості, резервна версія продукту має бути актуальною та уточненою, щоб забезпечити ясність. Уточнення беклогів є необов'язковою подією в scrum, тому що деякі беклоги не потребують цього. Однак для більшості команд краще зібрати команду разом, щоб переглянути та уточнити беклог перед плануванням спринту.

Планування спринту має обмежуватися не більше двох годин на кожен тиждень спринту. Так, наприклад, нарада з планування спринту для двотижневого спринту триватиме не більше чотирьох годин. Це називається «timeboxing», або встановлення максимального часу, який команда потребує для виконання завдання, у даному випадку це планування спринту. Scrum master несе відповідальність за те, щоб зустріч відбулася, таймбокс зрозумілий. Якщо команда задоволена до завершення таймбоксу, подія закінчилася. Timebox – це максимальний дозволений час; немає мінімального допустимого часу.

Під час планування спринту легко «в'язнути» в роботі, зосереджуючись на тому, яке завдання має бути першим, хто його має виконувати та скільки часу це займе. Для складної роботи рівень інформації, яку ви знаєте на початку, може бути низьким, і більша частина її ґрунтується на припущеннях. Scrum — це емпіричний процес, що означає, що ви не можете планувати заздалегідь, а радше навчаєтеся на практиці, а потім повертаєте цю інформацію назад у процес.

Ціль спринту описує мету спринту на високому рівні, але пункти відставання також можна писати з урахуванням результату. Історії

користувачів — чудовий спосіб описати роботу з точки зору клієнта. Історії користувачів переорієнтовують дефекти, проблеми та покращення на результат, якого прагне клієнт, а не на виявлену проблему.

Планування спринту вимагає певного рівня оцінки. Команда повинна визначити, що можна або не можна робити в спринті: оцінене зусилля проти можливостей. Оцінку часто плутають із зобов'язаннями. Оцінки за своєю природою є прогнозами, які базуються на наявних знаннях. Такі методи, як сторі поінти або визначення розміру футболки, додають цінності процесу, надаючи команді інший погляд на проблему. Однак вони не є магічними інструментами, які можуть дізнатися правду, коли її немає. Чим більше невідомих, тим менш імовірно, що оцінка буде правильною.

Хороша оцінка вимагає середовища, заснованого на довірі, де інформація надається вільно, а припущення обговорюються з метою навчання та вдосконалення. Якщо після завершення роботи оцінки використовуватимуться в негативний, конфронтаційний спосіб, тоді ймовірно, що майбутні оцінки будуть або набагато більшими, щоб гарантувати, що вони більше ніколи не будуть помилковими, або час, витрачений на їх створення, буде значно довшим, оскільки команда буде виконувати другу роботу здогадується, що хвилюється про наслідки помилки.

Легко настільки загрузнути в деталях планування спринту, що ви забудете, що головна мета планування спринту полягає в тому, щоб побудувати план «достатньо» для наступного спринту. Цей план повинен зосередити команду на цінних результатах. Хороший план спринту мотивує кожного, визначаючи результат і чіткий план успіху. Але будьте обережні, плануючи занадто заздалегідь. Замість того, щоб будувати найповніший план спринту «враховується кожна хвилина спринту», зосередьтеся на меті та створіть достатній запас спринту, щоб почати. Далі переконайтеся, що беклог продукту впорядкований, щоб дозволити команді розпочати роботу, якщо вони досягли цілі спринту раніше.

Scrum — це структура процесу, спрямована на вирішення складних проблем. Складні проблеми вимагають емпіричного процесу (навчання на

практиці). Емпіричні процеси дуже важко спланувати, тож не обманюйте себе – ви не можете побудувати ідеальний план. Натомість зосередьтеся на результатах і починайте. Це не повинно бути важко, навіть якщо проблема, яку ви вирішуєте.

4.9 Огляд спринта

Огляд спринта – це спільна зустріч, яка зазвичай проводиться в кінці кожного спринту. Це коли команда проходить через робочі елементи, які вони виконали під час спринту або ітерації. Спринт-перевірка гарантує, що ключові зацікавлені сторони в курсі подій, і це дає їм змогу надати відгук. Наприкінці кожного перегляду спринту власник продукту або керівник групи використовує цей відгук, щоб створити необхідні робочі елементи в резерві продукту. [19]

Власник продукту відкриває огляд спринта, що розпочинається з представлення доробки продукту. Доробка продукту — це об'єднання усіх завершених пунктів беклогу продукту.

Потім команда демонструє роботу, яку вони зробили. Вони показують замовнику та всім зацікавленим сторонам, що вони розробили, і вказують, які пункти беклогу продукту було виконано.

Також проводиться детальна дискусія про виконану роботу, де всі зацікавлені сторони мають можливість дати відгук та коментувати результати. Ідея полягає в тому, щоб обговорити, чи були досягнуті цілі спринту, і як це покращило результат продукту.

Це є цінним моментом, оскільки дозволяє отримати оперативний зворотний зв'язок від клієнтів і розробників, що може вплинути на наступні пункти беклогу продукту та бути корисними під час планування наступного спринту.

На завершення огляду спринту власник продукту має огляд беклогу продукту. Він приймає рішення про те, що потрібно зробити далі – які пункти беклогу продукту варто включити в наступний спринт.

Тож, з точки зору команди, огляд спринту – це можливість представити свою роботу, отримати відгук, почути нові ідеї та оцінити ефективність своєї роботи. Замовник і стейкхолдери отримують можливість впливати на продукт та процес його розробки, переглядати роботу і виконувати корективи на ранньому етапі розробки.

4.10 Ретроспектива

Ретроспектива Agile — це зустріч команди, яка має на меті детально перевірити ваші процеси, вирішити існуючі проблеми та визначити сфери, які потрібно вдосконалити. Метою ретроспективної зустрічі є безперервне покращення продуктивності команди, якості створеної цінності та її надання шляхом вивчення того, що вийшло добре, що не вдалось, і які зміни можна внести для покращення.

Ретроспективи Agile можуть проводитися в кінці спринту або ітерації для команд, які використовують Scrum, або через регулярні проміжки часу, якщо ви використовуєте Kanban. Зараз ретроспективна техніка широко використовується як ефективний інструмент, що використовується різними командами, які цінують постійне вдосконалення, і не тільки для розробки програмного забезпечення.

Ретроспектива — це час для членів команди, щоб обміркувати свою роботу, визначити сфери, які потрібно вдосконалити, і поставити цілі для наступної ітерації. Зустріч має фасилітувати один із членів команди, який може вести її конструктивно. [20]

Ретроспективи — одна з практик Agile, яка допомагає командам обмірковувати свою роботу та вчитися, що є ключовим принципом способів роботи Agile. Ретроспективи сприяють прозорості та відкритому спілкуванню між членами команди Agile, що може позитивно вплинути на співпрацю та організаційну культуру.

Основна мета ретроспектив — знайти способи постійно вдосконалювати робочий процес команди, роблячи його більш ефективним, продуктивним і приємним. Це забезпечує основу для членів команди, щоб оцінити, що вони зробили добре і що їм потрібно вдосконалити.

Ретроспективи мають багато переваг: вони можуть допомогти командам стати більш продуктивними, покращити комунікацію, обмінюватися знаннями та сприяти командній та міжкомандній співпраці. Це перелік переваг проведення ретроспектив.

Ретроспективи створюють середовище для обдумування робочих процесів, визначення областей, які потрібно вдосконалити, і внесення необхідних змін для постійного вдосконалення.

Ретроспективи забезпечують платформу для членів команди, щоб нести відповідальність за свої дії, і допомагають переконатися, що всі усвідомлюють свої обов’язки та працюють над тими самими цілями.

Ретроспектива може допомогти виявити та усунути перешкоди, які заважають продуктивності, забезпечуючи більшу ефективність команди.

У ретроспективі заохочується обмін, що сприяє створенню середовища для співпраці, де кожен має можливість зробити свій внесок.

Ось інструкція з п’яти кроків щодо того, як можна провести ретроспективу Agile.

1. Підготовка до ретроспективи

Ретроспектива має проходити у невимушеній обстановці з достатньою кількістю місць, щоб усі могли взяти участь. Фасилітатор повинен представити мету ретроспективи та бажані результати та створити безпечний простір, де будь-хто може поділитися своїми думками та почуттями, не боячись засудження чи критики.

2. Збір даних

Метою цього кроку є переконатися, що всі погоджуються щодо тем, які обговорюватимуться під час зустрічі. Команда повинна скласти список із трьох

найважливіших речей, які вони хочуть покращити, а потім за допомогою даних поглянути на те, як вони зараз працюють у цих сферах.

3. Генерування інсайдів

Узгодивши дані з командою, наступним кроком є генерація інсайдів. Під час цієї фази можна поставити кілька простих запитань, наприклад, які закономірності є очевидними в даних, що було несподіваним у даних, чи дізналися ви щось із даних і чи дізналися ви щось, переглядаючи дані чого не знали раніше.

Грунтуючись на цьому розумінні, команда повинна розробити план вирішення цих проблем і створити графік реалізації своїх рішень.

4. Узгодження наступних кроків

Прийміть рішення щодо наступних кроків для покращення процесу. Це може бути щось настільки просте, як зміна часу зустрічі, або таке складне, як повна зміна процесу. Розташуйте пріоритети цих активних елементів на основі їхнього впливу та здійсненності.

5. Закриття Ретроспективи

Закриття зустрічі включає підведення підсумків обговорення, визначення наступних кроків для покращення та відзначення того, що пройшло добре. Заплануйте подальшу перевірку прогресу.

Найважливіша річ, яку ви повинні враховувати під час проведення Agile Retrospective, це залучення всіх з усіх рівнів управління, а також зацікавлених сторін. Ідея, яка лежить в основі цього, полягає в тому, що кожен має свій голос і точку зору, які можуть бути цінними для вдосконалення.

Тривалість ретроспективи Agile залежить від тривалості ітерацій у вашому процесі. В ідеалі на обмірковування, групування та обговорення найважливіших проблем потрібно не більше години. Слідкуйте за часом для кожної фази, встановивши таймер. Команда повинна знати, коли перейти до інших питань, встановивши чіткі вказівки.

Успішна гнучка ретроспектива вимагає відкритого, чесного та конструктивного мислення. Це також вимагає людей, які готові вчитися на власних помилках і бути гнучкими у своєму підході. Це пояснюється тим, що ретроспективи — це не просто вправа з формування команди, а спосіб виявлення та вирішення проблем, які виникають у команді.

Бажаним результатом гнучкої ретроспективної зустрічі є визначення того, що пройшло добре, а що ні, і як покращити процес. Ця зустріч має стати часом для команди, щоб поміркувати над своєю роботою та визначити сфери, які потрібно покращити. Таким чином команда може вчитися як на своїх успіхах, так і на невдачах.

Частота ретроспективи Agile залежить від потреб команди, а також від рівня зрілості команди. Періодичність проведення ретроспектив залежить від методу чи рамок керівництва роботою. Однак зазвичай вони проводяться раз на два тижні або раз на місяць.

5 МЕТОДИ ОПТИМІЗАЦІЇ ПРОГРЕСУ КОМАНДИ

5.1 OKRs

OKR — це популярна стратегія управління, яка визначає цілі та відстежує результати. Це допомагає створити узгодженість і взаємодію навколо вимірюваних цілей. Запропонований у 80-х роках Ендрю Гроувом, відомим як «батько OKR», підхід до встановлення вимірюваних цілей справді отримав популярність на порозі нового століття, коли Google прийняв OKR. Фреймворк OKR, який отримав похвалу за його ефективність у виконанні сміливих місій, зосереджуючи увагу та обізнаність кожного на одній сторінці — щодо найважливіших тем — тепер використовується такими технологічними гігантами, як LinkedIn, Twitter і Microsoft, серед багатьох інших.

Ключем до успішного впровадження структури OKR є наявність належних засобів для вимірювання, відстеження та оцінки результатів поставлених цілей.

Зрозуміти значення OKR передбачає спочатку розуміння його основних компонентів – «цілі» та «ключові результати».

Ціль – це короткий і привабливий опис вашої мети. Це повинно мотивувати вашу команду на досягнення цього. Цілі представляють угоду про те, які цілі компанії є першочерговими. [21]

OKR - це удосконалення широко використаної практики управління за допомогою цілей (Management by Objectives, MBO). Відмінність у тому, що OKR - це процес налаштований на співпрацю, а не топ-даун бюрократичний процес. Пітер Друкер, який перший використовував MBO, пропонував менеджеру переглядати цілі організації, а потім встановлювати цілі робітника. У відміну, OKR ґрунтуються на тому, що команди всередині організації беруть високорівневі цілі та вдосконалюють їх для кожної конкретної області. Якщо командам потрібно співпрацювати з іншими партнерами організації для досягнення високорівневих цілей, ці команди можуть співпрацювати та досягати OKR разом, щоб забезпечити належну відповідність. Це зміна у

менталітеті, де питання змінюється з «чи ми були зайняті виконанням завдань?» на «чи ми змінили ситуацію для процвітання нашої організації?».

Цілком корисно думати що "процес OKR" обмежується лише "ціллю" та "ключовим результатом". Статичні цілі, які не регулярно переглядаються та вдосконалюються залежно від зміни обстановки, швидко стають неактуальними й позбавленими значення. Поєднання чітких цілей з невеликим набором конкретних, вимірюваних результатів та регулярний процес перегляду прогресу в напрямку цих показників - ось що робить OKR дійсно корисним.

Так, цілі ключових результатів (OKRs) є ефективним методом оптимізації прогресу команди. OKRs надають чітку річну чи квартальну стратегію та визначають конкретні цілі, які команда може виконати для досягнення стратегічних цілей.

Цілі ключових результатів допомагають командам:

- Сфокусувати увагу на виконанні найважливіших завдань.
- Визначити та виміряти критичні показники успіху.
- Спрямувати усіх членів команди на досягнення загальної мети.

OKRs також сприяють врегулюванню членів команди на спільні цілі та забезпечують взаєморозуміння щодо пріоритетів та вимог бізнесу. Це може посилити співпрацю та впровадження дієвих стратегій для досягнення бажаного прогресу команди.

Цифри вказують на 60% - 90% невдач, коли йдеться про виконання стратегії. Розрив між баченням компанії та її досягненням вимагає комплементарного підходу, щоб спрямувати команди та окремих людей на досягнення однієї мети.

Гнучкий підхід до управління проектами представляє механізм і методи, які команда використовує для спільної роботи. Водночас OKR представляють наші бізнес-цілі та визначають напрямок нашого бізнесу. Поєднання філософії OKR і гнучкого управління проектами може допомогти узгодити роботу з місією компанії, сприяти залученню співробітників і підвищити продуктивність.

Прийняття гнучкого підходу до управління роботою є чудовою основою для впровадження значущих процесів, практик і принципів у нашу роботу. Запровадження ітерації, співпраці, проактивності та змін на рівні відділу та окремому рівні означає, що нам вдалося прийняти гнучкість мислення. Незважаючи на те, що Agile успішно налагоджує наші робочі процеси, наші цілі також повинні розділятися та прийматися всіма.

Нам потрібен спосіб довести наші цілі до відома окремих членів команди, щоб досягти повного узгодження на кожному рівні організації. Структура OKR забезпечує простий, але дуже ефективний спосіб зробити це – прозорість у всій компанії. Спільні головні цілі гарантують, що люди усвідомлюють цінність своєї роботи для бізнесу, краще співпрацюють і відчувають більшу відповідальність за свою роботу.

Великою перевагою та життєво важливою частиною структури OKR є регулярність їх оцінювання за допомогою частих каденцій або церемоній Agile. Хоча організації можуть налаштовувати їх відповідно до власних потреб, звичайною практикою є щорічна оцінка високих цілей і щокварталу групові OKR. Цей підхід дозволяє організаціям і командам більш адекватно реагувати на зміни, що відбуваються на шляху, і впроваджувати вдосконалення в наступній ітерації.

У той час як один із основних принципів Agile-менеджменту проектів вимагає частішої доставки робочого програмного забезпечення, внесення змін і пристосування до ефективності, OKR роблять те саме для цілей. OKR справді можуть процвітати в гнучкому середовищі, що дозволяє швидше встановлювати цілі, оперативно оцінювати їх і бути відкритими для вдосконалення.

Крім того, налаштування OKR є компасом, який дозволяє нам відстежувати, чи ми рухаємось у правильному напрямку. Вони інформують нас про стан нашого прогресу та вимірюють результати нашого бізнесу.

OKR допомагають гнучким командам відстежувати прогрес і вносити необхідні зміни на цьому шляху, а також звужувати фокус і командні зусилля

для досягнення конкретної мети. Усунення потенційного розширення масштабу наших цілей має важливе значення для слідування наміченим шляхом компанії.

Відсутність узгодженості між бізнесом і командами є однією з головних причин кожної трансформації Agile, як вказує останній звіт про стан Agile. Інфраструктура OKR має на меті вирішити цю проблему завдяки покращеній видимості та прозорості.

Застосування гнучкого способу управління проектами та масштабування гнучкості в усій компанії також вирішує схожі завдання: узгодження з цілями компанії, створення гнучких команд, які адаптуються до змін і зміщення пріоритетів, а також досягнення синхронізації між стратегією та виконанням.

Гнучке мислення вимагає простоти та досягнення сталого темпу роботи. При застосуванні на стратегічному рівні такий підхід до виконання роботи сприяє досягненню орієнтованих на дії та конкретних цілей компанії.

З одного боку, організаційна гнучкість випливає з повної прозорості стратегічних цілей, щоб вони могли стати більш зрозумілими та легко доступними для всіх зацікавлених сторін. З іншого боку, постійна співпраця між клієнтами та внутрішніми командами під час визначення вашого бізнес-бачення дає змогу реагувати на зміни спокійно та без стресу.

Agile вимагає, щоб ви уникали попереднього планування, щоб виділити більше часу на дії, що додають цінність. Застосування цілей і ключових результатів на стратегічному рівні та їх регулярний перегляд сприяє їх своєчасній синхронізації з ринковими тенденціями, що розвиваються, або зміною бізнес-пріоритетів.

Поєднання двох підходів під час встановлення ваших високорівневих цілей створює чітке розуміння поточного пріоритету та дозволяє зосередитися на створенні фактичної цінності для клієнтів.

В основі Agile лежить принцип приємних змін під час виконання проекту. Сприйнятливість до змін дає вам можливість адаптуватися та задовольняти останні вимоги.

Оскільки ринкові умови та вимоги клієнтів завжди можуть змінюватися, надзвичайно важливо створити середовище, де різка зміна пріоритетів не внесе

хаос у робочий процес. Наскільки синхронна комунікація є життєво важливою для Agile-команд, синхронізація ваших стратегічних цілей та їх виконання має постійно пульсувати.

Встановлення регулярних циклів зворотного зв'язку для обмірковування та оцінки ваших стратегічних OKR створить ще один рівень гнучкості на рівні управління вашою організацією. Це також допомагає легко визначити можливості покращення.

Представлення цілей і ключових результатів (OKR) впливає як на ефективність бізнесу, так і на поширення нової організаційної культури.

Встановлення високих цілей надсилає чітке повідомлення кожному члену команди про прагнення та напрямок компанії. Це створює прозорість і служить основою для конструктивного обговорення результатів бізнесу. Обізнаність усіх команд щодо OKR компанії дозволяє проактивно мислити та залучатися до виконання та результатів.

За винятком узгодження зі стратегічними планами організації, OKR на рівні окремої команди дозволяють зосередитися на короткострокових цілях. Це здійсненне планування з цільовим циклом 1-2 місяці дає змогу оцінити продуктивність, визначити віхи та покращити наступний цикл ітерації.

Заохочуючи автономію та прозорість у всій компанії, OKR створюють середовище залучення та відповідальності. Залучення співробітників не тільки впливає на продуктивність, але й задає тон довготривалій зміні мислення.

Ключем до правильного налаштування OKR є глибоке розуміння їхнього призначення. Хоча хороший набір OKR суворо залежить від сфери діяльності, серед інших галузевих факторів, написати ефективні OKR можна, відповівши на кілька простих запитань.

Мета цілей полягає в тому, щоб повідомити про пріоритети компанії. Вони повинні представляти бачення компанії. Отже, ваша мета повинна відповідати на запитання «Що». Крім того, переконайтеся, що ваші цілі є значущими, чіткими та надихаючими.

Ключові результати представляють засоби для оцінки та відстеження виконання вашої цілі. Таким чином, ваші ключові результати повинні

відповідати на запитання «Як». Вони мають бути конкретними та реалістичними, і ви повинні мати можливість їх виміряти.

Рішення Jira Align від Atlassian допомагає поєднати вашу бізнес-стратегію з технічною реалізацією. Там, де OKR відстежують результати виконання, Jira Align надає чітке уявлення про цілі, ключові результати, які ви відстежуєте, і роботу над досягненням цілей. Функції OKR, вбудовані в Jira Align, допомагають вашій команді згуртуватися, залишатися налагодженими та миттєво поєднувати роботу з наданням цінностей. [22]

6.2 Звітність та метрики Scrum

Проект Scrum контролюється за допомогою частої перевірки проекту з подальшою необхідною адаптацією. Частина цієї інформації передається віч-на-віч. Наприклад, Daily Scrum відкритий для всіх; учасники можуть швидко відчути тон, ставлення та прогрес спринту. Оглядова нарада Sprint надає щомісячну інформацію про те, чи створює проект цінні функції, а також про якість і можливості цих функцій. Інша інформація повідомляється письмово. Наприклад, Backlog продукту детально описує вимоги проекту та перераховує їх у порядку пріоритету. Кожен може оцінити Backlog продукту, оскільки він зберігається у загальнодоступній папці, а його місцезнаходження повідомляється.

На додаток до динамічної інформації, що надається письмовою та візуальною інформацією, наприкінці кожного спринту генеруються офіційні звіти. Ці звіти надають статичний знімок прогресу проекту. Звіти використовуються, щоб усі, хто цікавиться проектом, були в курсі його прогресу. Уся ця інформація, як динамічна, так і статична, вважається частиною звітності проекту Scrum. [23]

Метрики Scrum — це конкретні точки даних, які команди Scrum відстежують і використовують для підвищення ефективності та результативності. Коли метрики Scrum визначені, зрозумілі та впроваджені,

вони можуть стати інформацією, яка допоможе скерувати та покращити гнучкість команди. [24]

Scrum-команди використовують метрики для інформування про прийняття рішень і підвищення ефективності планування та виконання. Їх також можна використовувати для встановлення базової лінії щодо статус-кво та встановлення цільових цілей і планів покращення. При цьому немає жодного галузевого стандарту, щоб порівняти всіх із сьогоднішніми. Це тому, що порівняння точок даних за відсутності контексту схоже на порівняння яблук і устриць. Кожна команда унікальна – вони можуть бути різними за розміром, використовуваними технологіями, видом роботи тощо.

Кожна команда має узгодити набір показників для відстеження та визначити, як їх використовувати. Це не індивідуальні зусилля і не те, що керівництво чи керівництво може визначити та забезпечити виконання від імені команд.

Немає точного показника, який скаже вам, чи ваші гнучкі команди працюють так добре, як вони можуть. Підвищення ефективності стає можливим завдяки регулярному врахуванню того, що ви дізналися про продуктивність своєї команди, у її роботі. Збір і аналіз даних у формі показників є об’єктивним способом дізнатися більше про вашу команду та способом вимірювання будь-яких коригувань, які ви вирішите внести в поведінку вашої команди.

Показники можна використовувати для вимірювання всього, що ви вважаєте доречним, і це може бути потужним інструментом, якщо використовувати його для покращення спілкування та співпраці. Ці показники фактично стають ключовими показниками ефективності (KPI), які допомагають визначити, що важливо для вашої команди та вашого бізнесу. Використовуючи KPI та тенденції даних, щоб показати, як певні точки даних впливають на поведінку та прогрес, ви можете налаштувати поведінку своєї команди та спостерігати, як внесені вами зміни впливають на важливі для неї дані. [25]

В екосистемі Jira існують влаштовані звіти для перегляду метрик. Для кожної окремої дошки є можливість дізнатися метрики.

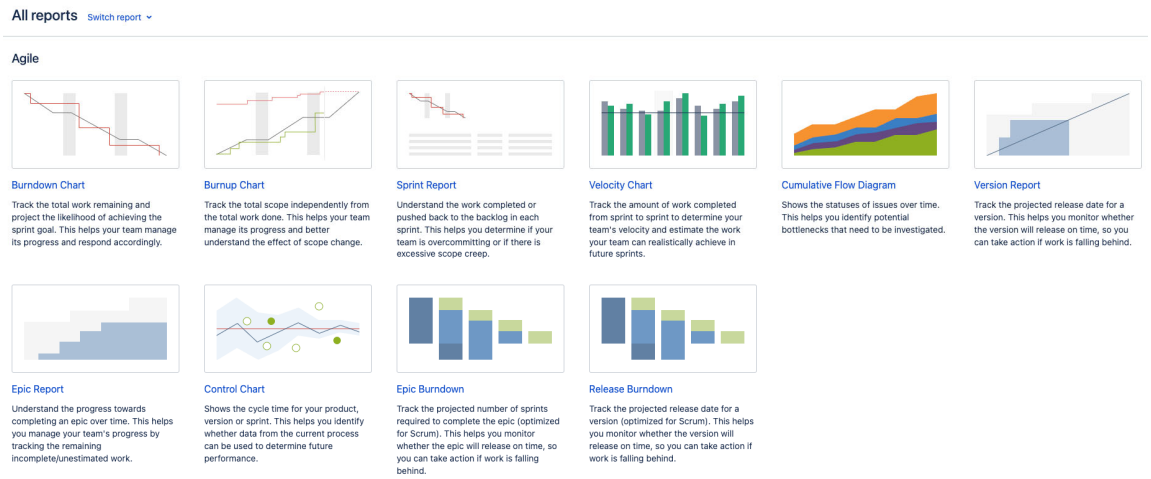

Рисунок 6.2.1 – Список доступних звітів

Рисунок 6.2.2 – Приклад звіту Velocity Chart

Але ці звіти не надають повну інформацію, вони показують лиш часткові метрики. Для цього в Jira теж є рішення – Jira Marketplace.

Ринок додатків Jira надає безліч додаткової цінності для користувачів, шляхом додавання нових функцій та можливостей до основної платформи. Він включає в себе різноманітні додатки для покращення управління завданнями, розширення можливостей звітів, автоматизації процесів та багато іншого.

Особливо важливим є те, що Jira Marketplace надає засоби для створення різноманітних звітів, що дозволяє користувачам аналізувати дані та моніторити прогрес проектів. Інструменти для репортінгу надають можливість створення власних звітів, вибору необхідних метрик та візуалізацію різних типів даних (таких як графіки, діаграми, таблиці тощо).

Отже, завдяки Jira marketplace користувачі мають доступ до широкого спектра рішень для збільшення продуктивності, поліпшення аналітичних

можливостей та взагалі розширення функціоналу платформи для досягнення кращих результатів у управлінні проєктами та роботою з командами.

Одним з яскравих прикладів є Cross-team Velocity Chart від української продуктової компанії Broken Build. Найбільшою перевагою перед влаштованим звітом є можливість об’єднувати декілька дошок та дивитись глобальну картину прогресу всієї компанії, а не окремо по дошкам. Багато додаткових метрик дають можливість знайти слабкі сторони та звернути на них увагу, покращивши прогрес при цьому.

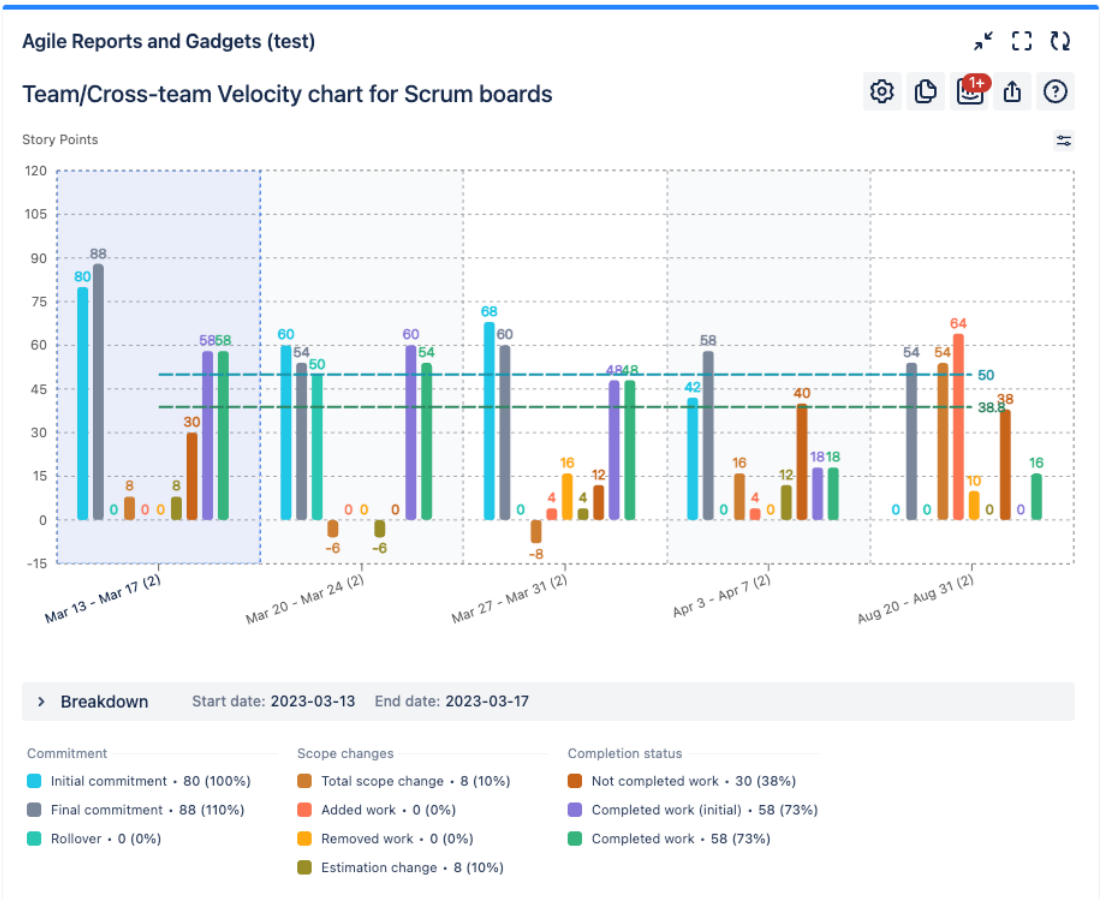

Рисунок 6.2.3 – Приклад звіту Velocity Chart від продукту з Jira Marketplace

Однією з найважливіших метрик є метрика Work In Progress (WIP). Вона показує скільки роботи ще в процесі. Цієї метрики взагалі немає на звітах, які надає Jira. Звісно, його можна вирахувати порівнявши кількість роботи яку потрібно було виконати та кількість роботи яку виконали. Але в таких спосіб не можна порівняти ці значення на великих об'ємах даних.

Щоб робочий елемент не враховувався як незавершений, має бути конкретна точка відправлення від процесу. Відправлення можна визначити як

доставку фактичному кінцевому користувачеві або доставку до іншої команди або процесу. Наприклад, якщо є команда розробників

відповідальний за власне розгортання у виробництві, тоді ця команда може вважати елемент таким, що залишився лише після того, як було здійснено розгортання у виробництві. Або інша команда, яка не відповідає за розгортання, може вважати предмет відправленим лише після того, як його обґрунтовано було передано наступній операційній групі, яка потім займатиметься розгортанням. Знову ж таки, визначення точки відправлення вірно незалежно від того, чи працюєте ви з системою тяги чи штовхання.

Підводячи підсумок, для цілей поточного визначення ваша команда повинна визначити конкретну точку, коли вона вважає, що робота досягла процесу, і вона повинна визначити конкретну точку, де робота вийшла з процесу. Визначення цих двох меж системи є вирішальною відправною точкою в передбачуваному проектуванні процесу. Після того, як ви приймете ці рішення, усі робочі елементи між цими двома точками вважатимуться незавершеною роботою: усі окремі одиниці цінності для споживача, які увійшли до даного процесу, але не вийшли з нього.

Якщо визначення WIP є складною частиною, то її вимірювання є легкою частиною. Щоб обчислити WIP, ви просто підраховуєте дискретну кількість робочих елементів у межах вашого процесу, як визначено вище. [26]

Але не достатньо просто створити звіт. Кожен звіт та кожну метрику треба вміти читати, розуміти та робити правильні висновки.

Візьмемо за приклад звіт за першу половину спринта (див. рис. 6.2.4) і зробимо висновки щодо метрик. Для наочності обрані відсоткові значення для відображення звіту, а не абсолютні, щоб не робити акцент на числах. На даному етапі важливіше бачити відсоткові значення.

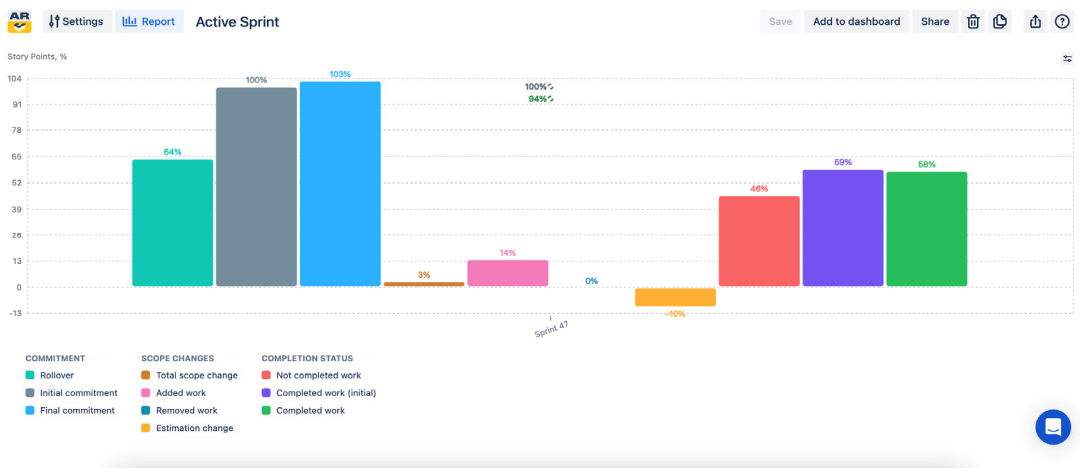

Рис 6.2.4 – Приклад звіту за першу половину спринта

На рисунку бачимо, що зелений стовпчик (виконана робота) більший за червоний (не виконана робота). Так як це середина спринта робимо висновок, що команда рухається в правильному напрямку. Аномалій не спостерігається, до кінця спринта уся робота повинна бути виконана. Такий висновок має право на існування, але він не відображає всієї суті того, що можна тут побачити і які висновки зробити.

Тепер зробимо більш детальний аналіз звіту щоб виявити проблемні зони команди.

Ключові показники:

1. Rollover: 64%

• Це означає, що 64% роботи з попереднього спринту було перенесено до поточного спринту. Це відносно високий показник і свідчить про проблеми із виконанням запланованої роботи в попередніх спринтах.

2. Initial Commitment: 100%

• Це базова лінія для спринту. Команда взяла на себе певну роботу на початку спринту.

3.Final Commitment: 103%

• Розмах спринту збільшився на 3% під час спринту. Це не значне збільшення, але вказує на певне розширення масштабів.

4. Total Scope Change:

• Загальна зміна обсягу: 3%

• Додаткова робота: 14%

• Видалена робота: 0%

• Зміна оцінки: -10%

• Команда додала на 14% більше роботи та переоцінила деякі завдання, що призвело до чистої зміни обсягу на 3%.

5. Статус завершення:

• Not Completed Work: 46%

• Completed Work (initial): 59%

• Completed Work (final): 58%

Аналіз продуктивності:

Позитивні аспекти:

• Високий відсоток виконання початкових зобов'язань: команда виконала 59% спочатку взятих завдань, тобто більше половини.

• Виконання додаткової роботи: незважаючи на збільшення додаткової роботи на 14%, команді вдалося виконати 58% остаточного зобов'язання.

Негативні аспекти:

• Висока частота повторень: 64% є досить значним і вказує на те, що команді важко завершити заплановану роботу протягом спринту.

• Високий відсоток невиконаних робіт: оскільки 46% роботи не виконано, майже половина запланованої роботи залишається незавершеною.

• Розповсюдження масштабу: під час спринту кількість робіт збільшилася на 14%, що може призвести до нездатності команди виконати всі завдання.

Загальна оцінка:

Наразі команда стикається з труднощами та не працює оптимально.

Рекомендації:

1. Вирішення проблеми, пов'язаної з перенесенням завдань в наступний спринт:

• дослідити, чому такий високий відсоток роботи переноситься. Визначте блокувальники і усуньте їх, щоб покращити здатність команди виконувати завдання під час спринту.

2. Покращення планування та оцінки:

• підвищити точність планування та оцінки, щоб краще відповідати можливостям команди. Переконатися, що відставання в спринті є реалістичним і досяжним.

3. Контроль зміни обсягу роботи:

• впровадити суворіші засоби контролю, щоб керувати розповзанням обсягу роботи. Будь-які доповнення до беклогу слід ретельно розглядати та затверджувати на основі їх пріоритетності та впливу.

4. Зосередження на першочергових завданнях:

• розставити завдання за пріоритетністю, щоб найважливіша робота була виконана першою. Це допоможе отримати найбільшу цінність і зменшити частоту ролловерів.

5. Регулярні огляди:

• регулярно переглядати прогрес і коригувати, якщо необхідно. Огляди в середині спринту можуть допомогти виявити проблеми на ранній стадії та внести корекції на курс.

Впроваджуючи ці рекомендації, команда може покращити свою продуктивність, зменшити частоту перекидань і досягти вищого показника завершення в майбутніх спринтах.

На цих двох прикладах наочно показано як по-різному можна сприймати один і той самий звіт.

ВИСНОВКИ

На основі обраних методів оптимізації, було виявлено, що впровадження цілей ключових результатів (OKRs) та використання метрик Scrum сприяють покращенню продуктивності та результативності управління проєктами в IT-компанії.

Використання цілей ключових результатів (OKRs) надає команді засоби для систематичного визначення, вимірювання та досягнення стратегічних цілей. Цілі узгоджені та легко відслідковуються, надаючи команді чіткий фокус на результат та впевненість в плануванні та виконанні завдань.

Метрики Scrum використовуються для вимірювання продуктивності команди та керування процесом розробки. Вони дозволяють оперативно виявляти проблемні зони та приймати вчасні корективи для поліпшення ефективності.

Окрім того, ресурси, які надає Jira marketplace, сприяють досягненню кращих результатів. Спеціальні інструменти для звітності надають можливість створення різноманітних звітів, аналізу даних та моніторингу прогресу проєктів.

Застосування OKRs, метрик Scrum та інших інструментів, які надає Jira, сприяє покращенню результативності та прозорості у діяльності команди, а отже, може бути ефективним варіантом оптимізації управління проєктами в IT-компанії.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Dima, A. M., Maassen, M. A. From Waterfall to Agile software: Development models in the IT sector, 2006 to 2018. Impacts on company management. Journal of International Studies, 11(2), 315-326. – 11 с.

2. Agile vs. waterfall project management. URL: https://www.atlassian.com/agile/project-management/project-management-intro. (18.02.2024)

3. Agile Project Management - What is it and how to get started?. URL: https://www.atlassian.com/agile/project-management. (18.02.2024)

4. Robert C. Martin. Clean Agile. Back to Basics. – Pearson Education, Inc., 2020. – 191 с.

5. Agile Manifesto. URL: https://www.atlassian.com/agile/manifesto. (12.02.2024)

6. Jeff Sutherland. Scrum: The Art of Doing Twice the Work in Half the Time. – Penguin Random House Company, New York, 2014. – 204 с.

7. James Shore, Shane Warden. The Art of Agile Development.– Sebastopol: O'Reilly Media, Inc., 2008.– 413 с.

8. John A. A. Earley. The Lean Book of Lean. – The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, United Kingdom, 2016. – 271 с.

9. User stories with examples and a template. URL: https://www.atlassian.com/agile/project-management/user-stories. (03.03.2024)

10. Mike Cohn. User Stories Applied for Agile Software Development. – Pearson Education, Inc. Rights and Contracts Department, Boston, 2009. – 291 с.

11. Mike Cohn. Agile estimating and planning. – Pearson Education, Inc. Rights and Contracts Department, Boston, 2006. – 320 с.

12. John C. Goodpasture. Project Management the Agile Way. – J. Ross Publishing, Inc., 300 S. Pine Island Rd., Suite 305, Plantation, 2016. – 393 с.

13. Agile epics: definition, examples, and templates. URL: https://www.atlassian.com/agile/project-management/epics. (05.03.2024)

14. Mark C. Layton, Steven J Ostermiller, Dean J. Kynaston. Scrum for dummies. – John Wiley & Sons, Hoboken, 2023. – 411 с.

15. Kenneth S. Rubin. Essential Scrum. – Pearson Education, Inc., Permissions Department, New Jersey, 2013. – 498 с.

16. Jeff Sutherland. Scrum Handbook. – Scrum Training Institute, Somerville, 2015. – 66 с.

17. Scrum sprints. URL: https://www.atlassian.com/agile/scrum/sprints. (07.03.2024)

18. Sprint planning. URL: https://www.atlassian.com/agile/scrum/sprint-planning. (11.03.2024)

19. What is Sprint review. URL: https://www.zoho.com/sprints/sprint-reviews.html#:~:text=A%20sprint%20review%20is%20a,during%20the%20sprint%20or%20iteration. (17.03.2024)

20. What Is an Agile Retrospective and How It Boosts Team Performance. URL: https://businessmap.io/agile/project-management/agile-retrospective. (18.03.2024)

21. How Can Agile and OKRs Work Together? URL: https://businessmap.io/agile/agile-at-scale/okrs-in-agile. (10.04.2024)

22. OKRs: the ultimate guide to objectives and key results. URL: https://www.atlassian.com/agile/agile-at-scale/okr. (10.04.2024)

23. Ken Schwaber. Agile Project Management with Scrum. – Microsoft Press A Division of Microsoft Corporation One Microsoft Way Redmond, Washington, 2004. – 175 с.

24. Scrum metrics. URL: https://www.atlassian.com/agile/scrum/scrum-metrics. (22.04.2024)

25. Christopher W. H. Davis. Agile Metrics in Action. – Special Sales Department Manning Publications Co., Shelter Island, 2015. – 270 с.

26. Daniel S. Vacanti. Actionable Agile Metrics for Predictability. – Serenity Press Pty Ltd., Perth, Australia, 2016. – 292 с.