РОЗДІЛ 5. МЕТОДОЛОГІЯ WATERFALL ТА AGILE. SCRUM ТА KANBAN

5.1. Waterfall та AGILE. Scrum та Kanban

Waterfall чи AGILE?

Найбільш поширені підходи в розробці software:

- **методика Waterfall;**
- **методика AGILE;**
- **метод Scrum;**
- **метод Kanban.**

Waterfall (водоспад) - методика управління проєктами, яка має на увазі **послідовний перехід з одного етапу на інший** без пропусків і повернень на попередні стадії. **AGILE** – методика управління різноманітними проєктами, побудована **на системі ідей і принципів «гнучкого» управління проєктами**, на основі яких розроблені популярні методи **Scrum, Kanban** і інші. Ключовий принцип - розробка через короткі ітерації (цикли), в кінці кожного з яких замовник (користувач) отримує робочий код або продукт.

Довідка.

Варіант 1. США

- *Эджл, Amer. | ‘ædʒl | или | ‘ædʒəl |.*
- *З наголосом на 1-й склад.*

Варіант 2. Британський.

- *АджАйл, Brit. | ‘adʒʌɪl | или | ‘ædʒaɪl |*
- *Наголос на 1-й склад, але звучить, ніби на обоє склади.*
- *В україномовному просторі біль часто вживано 2-й варіант — аджАйл.*

5.2. Підхід до опису проєктів Waterfall

Каскадна модель (англ. Waterfall model, іноді перекладають як модель «Водоспад») (рис. 14.1) - модель процесу розробки програмного забезпечення, в якій процес розробки виглядає як потік, що послідовно проходить фази:

1. Визначення вимог.
2. Проєктування.
3. Конструювання (також «реалізація» або «кодування»).
4. Втілення.

5. Тестування та налагодження (також «верифікація»).
6. Інсталяція.
7. Підтримка.

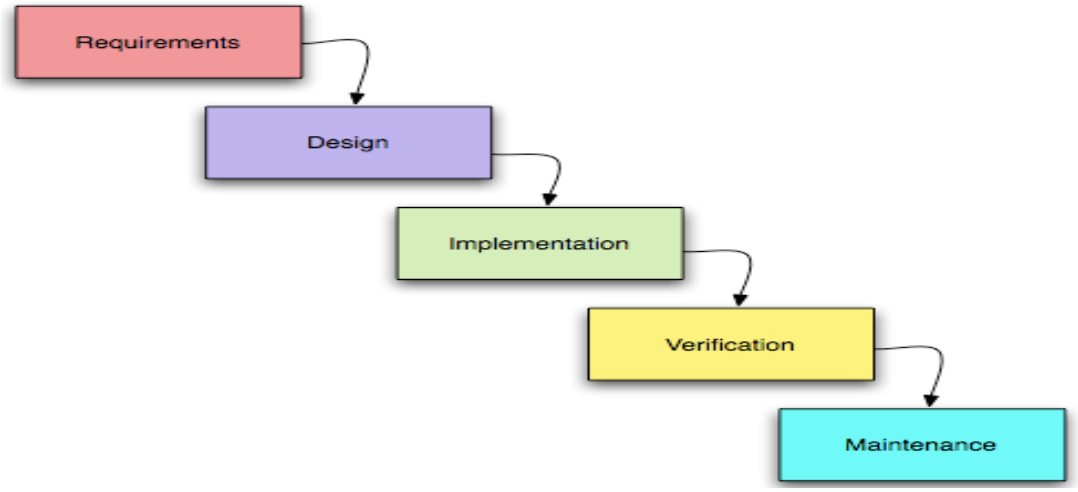

Рис. 14.1 Waterfall

Waterfall model засновано на статті У. Ройса опублікованій в 1970 році, в якій вказано можливість доопрацювання до ітеративної моделі розробки.

Каскадна модель має на увазі, що перехід від однієї фази розробки до іншої відбувається тільки після повного і успішного завершення попередньої фази (рис. 14.2), і що переходів назад або вперед або перекриття фаз - не відбувається.

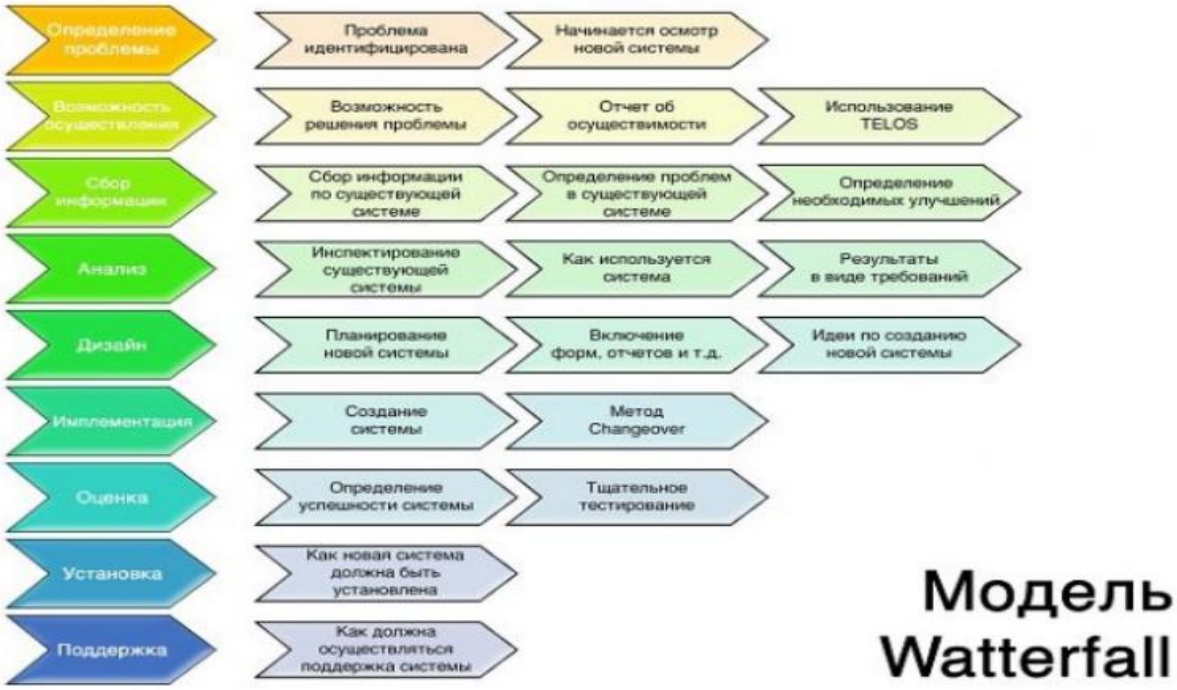

Рис. 14.2 Модель Waterfall

“-” (недоліки):

- недостатня гнучкість;
- як самоціль формальне управління проєктом на шкоду термінам, вартості та якості.

“+” (переваги):

- В управлінні великими проєктами формалізація є великою цінністю, так що може суттєво знизити ризики проєкту та досягти більшої його прозорості.

В PMBOK 3-ї версії формально була закріплена лише методика «каскадної моделі».

Починаючи з PMBOK 4-ої версії крім «каскадної моделі» залучені і альтернативні – гнучкі ітеративні методи.

“+” Waterfall

- **Вартість і терміни** виконання **зрозумілі** ще до початку робіт. Тому замовник точно знатиме, коли проєкт завершиться і який бюджет потрібно витратити.
- Інтуїтивно **зрозуміла структура роботи**, як для досвідчених фахівців, так і для новачків.
- Детально **структурований план робіт** і продумана документація.
- Завдяки зручній звітності **легко відстежити витрачений час**, можливі ризики і використовувані ресурси в процесі роботи над проєктом.
- **Завдання, які ставляться перед командою ясні** і не змінюються протягом усього проєкту.
- **Якість проєкту займає першочергове місце**, а витрачений час і бюджет відходять на другий план.

“-” Waterfall

- **Вимоги до проєкту** закріплюються на початку і **не можуть змінюватися до закінчення робіт**. Цей факт позбавляє проєкт гнучкості.
- **Витрачається великий обсяг грошових коштів, часу і ресурсів**.
- **Неможливість внесення змін** у процесі розробки.
- **Замовник побачить готовий проєкт тільки після його релізу**, при необхідності змін можуть знадобитися додаткові кошти і час.
- **Взаємодія** між етапами розробки **повністю відсутня**.
- При використанні каскадної моделі **продукт тестується після його випуску**. Тому в більшості випадків проблеми виявляються тільки на етапі тестування.

Waterfall підійде, коли:

- Проєктні вимоги ретельно продумані і незмінні. У замовника є ***чітко сформульована концепція продукту***.
- ***Технології та інструменти відомі*** заздалегідь.
- Розроблюваний продукт ***складний і вимагає великих витрат***.
- ***Пріоритетом є якість продукту.*** Часові та грошові витрати мають другорядне значення.
- ***Замовник не планує брати участь в проєкті***. Він бачить тільки готовий продукт. Проєкт повністю розробляється на аутсорсинг.
- Клієнту ***важливо знати точні терміни виконання всіх робіт*** над проєктом. Виконавець повністю несе відповідальність за зрив термінів і незапланований збільшення бюджету.
- ***Реалізація аналогічного проєкту, з яким вже стикалися розробники.***

5.3. AGILE – Гнучкі методи

Гнучка розробка ПЗ (англ. **Agile (гнучка) software development**), **agile-методи** - узагальнюючий термін для цілого ряду підходів і практик, заснованих на цінностях Маніфесту гнучкої розробки програмного забезпечення і 12 принципах, що лежать в його основі.

Маніфест гнучкої розробки програмного забезпечення (англ. Agile Manifesto) - основний документ, що містить опис цінностей і принципів гнучкої розробки програмного забезпечення, розроблений в лютому 2001 року на зустрічі 17 незалежних практиків декількох методів програмування, які іменують себе **«Agile Alliance»**.

Текст маніфесту доступний на більш ніж 50 мовах (в т. ч. На українській), і включає в себе 4 цінності, 12 принципів.

4-и цінності в AGILE

1. **Люди і взаємодія** важливіше процесів та інструментів.
2. **Працюючий продукт** важливіше вичерпної документації.
3. **Співпраця з клієнтом** важливіше узгодження умов контракту.
4. **Готовність до змін** важливіше проходження попереднім планом.

Таким чином, не заперечуючи важливості того, що справа, ми все-таки більше цінуємо те, що зліва.

12 принципів AGILE

1. **Найвищим пріоритетом є задоволення потреб клієнта**, завдяки регулярному і ранньому постачанню коштовного програмного забезпечення.

2. **Зміна вимог вітається**, навіть **на пізніх** стадіях розробки.

3. **Працюючий продукт слід випускати якомога частіше**, з періодичністю від кількох тижнів до кількох місяців.

4. Протягом всього проєкту **розробники і замовники повинні щодня працювати разом**.

5. Над проєктом повинні працювати **мотивовані професіонали**. Щоб робота була зроблена, створіть умови, забезпечте підтримку і повністю довіриться їм.

6. **Безпосереднє спілкування** є найбільш практичним і ефективним способом обміну інформацією як з самою командою, так і всередині команди.

7. **Працюючий продукт** - основний показник прогресу.

8. Інвестори, розробники і користувачі повинні мати можливість **підтримувати постійний ритм** нескінченно.

9. Постійна **увага до технічної досконалості** і якості проєктування підвищує гнучкість проєкту.

10. **Простота** - мистецтво мінімізації зайвого клопоту - вкрай необхідна.

11. Найкращі вимоги, архітектурні та технічні рішення народжуються у **самостійно організованих команд**.

12. Команда повинна систематично аналізувати можливі способи поліпшення ефективності і відповідно **коригувати стиль своєї роботи**.

“+” AGILE

- **Внесення необхідних змін** і впровадження нового функціоналу може відбуватися **незалежно від циклу розробки продукту**, що значно підвищує конкурентні переваги готового проєкту.
- Проєкт **складається з коротких і зрозумілих циклів**, після закінчення яких клієнт отримує робочий продукт.
- **Гнучкий процес коригувань** в будь-який ітерації дозволяє знизити виробничі ризики.
- - Досить **швидкий реліз пробної версії** для подальших коригувань і тестування.
- **Високий ступінь залученості всіх членів команди** і постійна взаємодія з замовником. Він завжди в курсі, на якій стадії знаходиться проєкт.
- **Показником ефективності є робочий продукт**, що вимагає високого професіоналізму від виконавців і грамотної організації робочого процесу.

“-” AGILE

- Розрахувати **кінцеві витрати практично неможливо** - вимоги можуть постійно змінюватися в залежності від особливостей проєкту. Складність полягає в тому, що вони можуть суперечити вже існуючій структурі.
- **Agile вимагає великої залученості в процес** і повного занурення в нього, що буває складно, особливо для молодих підрядників.
- **Можливість частого внесення правок** може обернутися ризиком **в нескінченному вдосконаленні проєкту**. Тут також можлива і зворотна сторона - зниження якості продукту.

Коли застосовують AGILE (рис. 14.3)***:***

- Коли **перелік вимог достатньо не визначений**, а зміни повинні вноситися максимально швидко.
- У проєкті працює **досвідчена команда з високим рівнем професіоналізму**.
- **Замовник бере активну участь** в розробці протягом усього проєкту.
- Клієнту важливо вносити зміни максимально оперативно на будь-якому етапі роботи.
- Якщо необхідно **швидко і в короткі терміни створити робочу версію продукту**.
- **Новий проєкт є стартапом.**
- **Ніша**, для якої розробляється продукт, **схильна до постійних змін**.

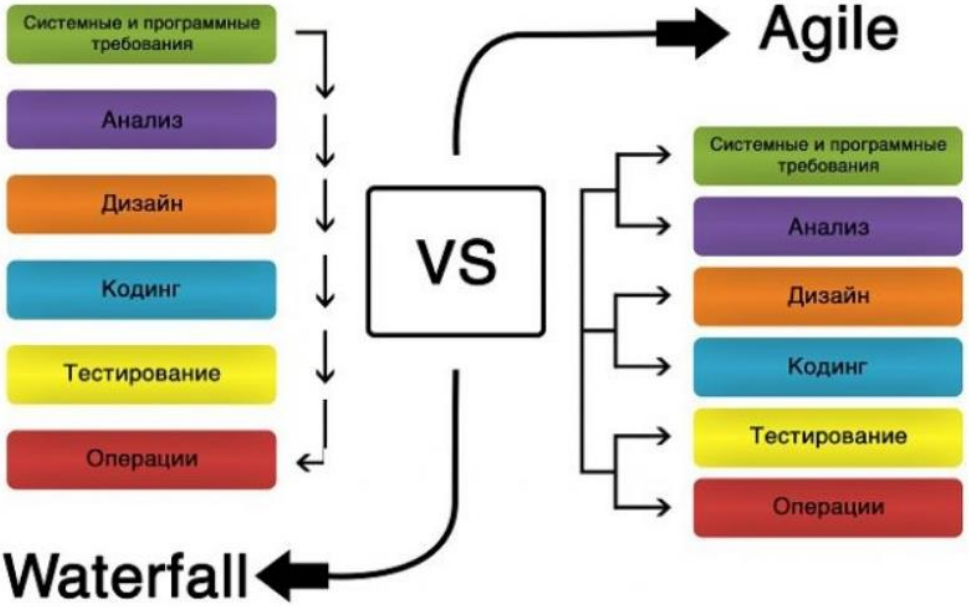

Рис. 14.3 AGILE проти Waterfall

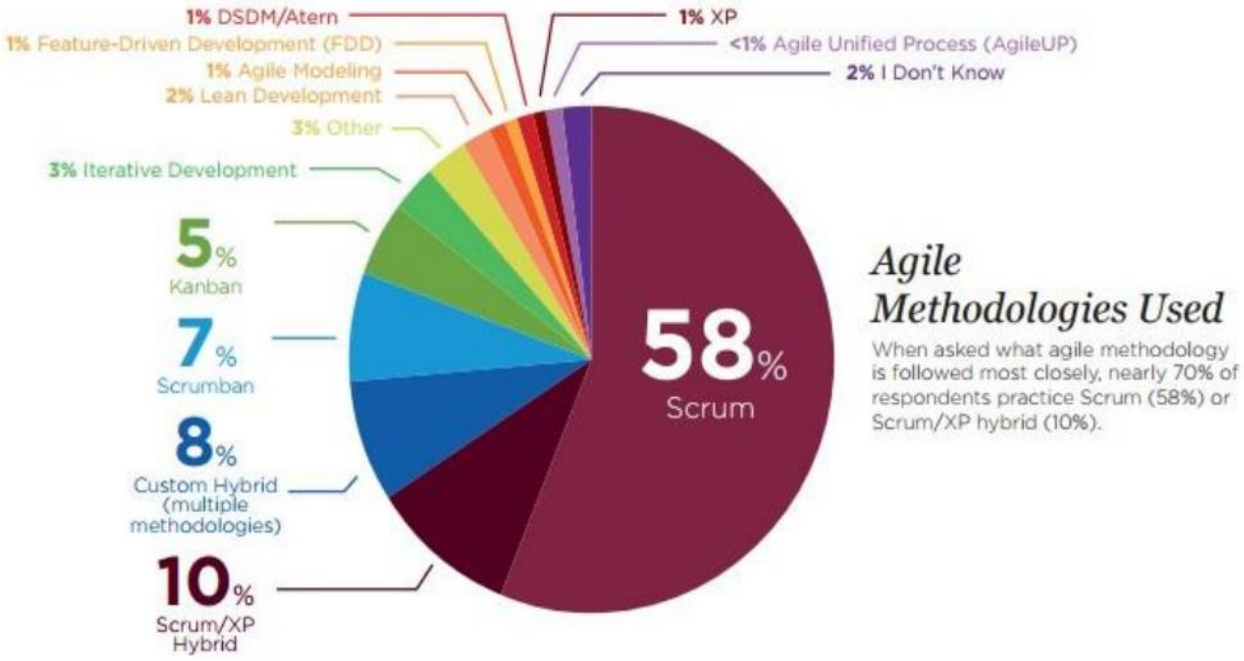

Рис. 14.4 Використання методології AGILE

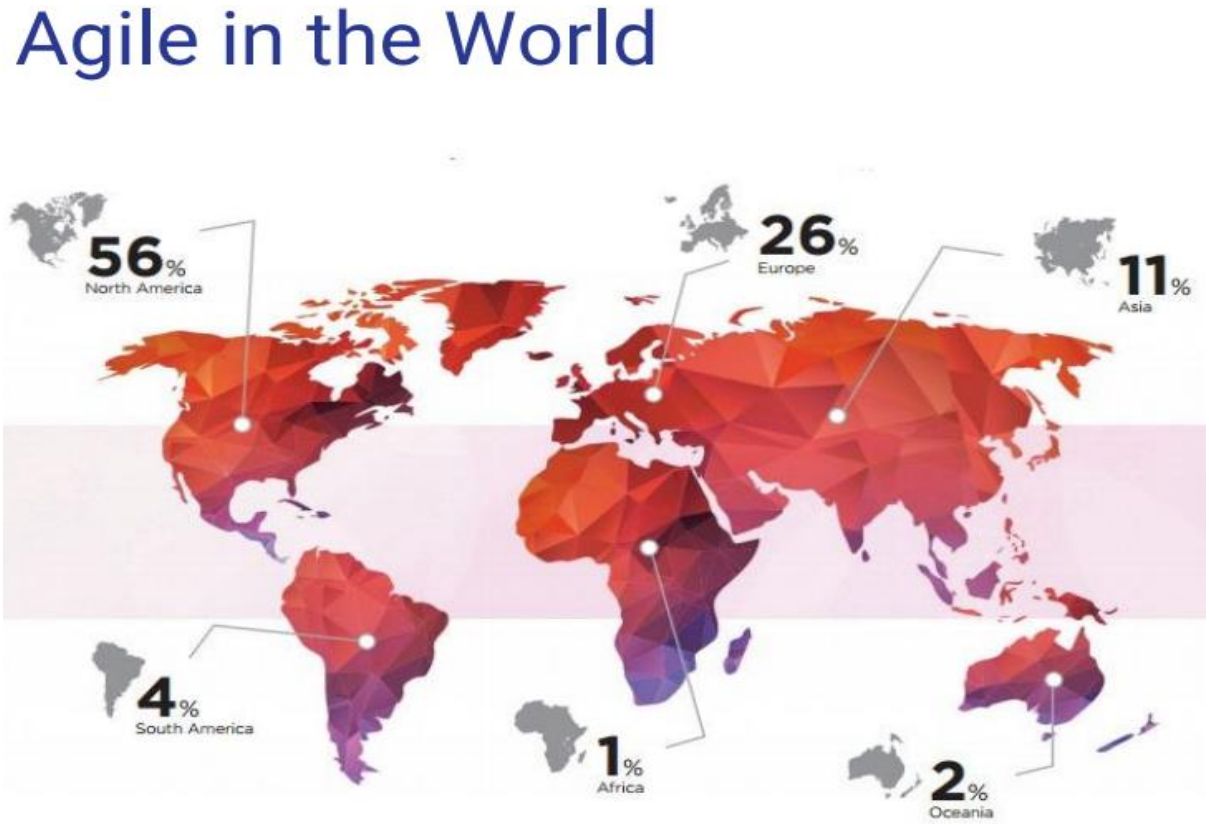

Рис. 14.5 Розповсюдження AGILE в світі

5.4. Порівняння Scrum та Kanban

Завдання команд **scrum** (з англ. – «сутичка») - **поставити працююче ПЗ по частинам** (складовим, компонентам) **за ряд проміжків часу, які називаються спринтами** (в англ. «sprint»).

Команди scrum прагнуть створювати **цикли навчання для швидкого збору і обліку відгуків клієнтів**. **Scrum-команди** використовують особливі ролі (таблиця 14.1), створюють спеціальні артефакти і проводять регулярні збори, щоб **робота йшла в потрібному руслі**. Найкраще методика scrum написана **в керівництві по scrum**.

Суть **kanban** (від яп. 看板 «рекламний щит, вивіска») - **в візуалізації роботи, обмеження обсягу незавершеною роботи і досягнення максимальної ефективності** (або швидкості).

Kanban-команди прагнуть максимально скоротити час, який витрачається на виконання проєкту (або призначеної для користувача історії) від початку до кінця. Для цього вони **використовують дошку kanban** і безперервно **вдосконалюють свій робочий процес**.

Таблиця 14.1 Порівняння Scrum та Kanban

	Scrum	**Kanban**
Графік	Регулярні спринти з фіксованою протяжністю (наприклад, 2 тижні)	Неперервний процес
Підходи до релізу	В кінці кожного спринту	Не перервна поставка
Ролі	Власник продукту, scrum-майстер, команда розробників	Обов’язкових ролей нема
Ключові показники	Швидкість	Час виконання, час циклу, обсяг завершеної роботи (WIP)
Відношення до змін	Під час спринту команди не повинні вносити зміни	Зміни можуть відбутися у будь-який момент

14.5. Універсальна схема Scrum

Управління продуктом в Scrum складається з декількох універсальних для кожного контексту кроків (рис. 14.6).

1. Обирається **«Власник продукту»** - людина, яка стає сполучною ланкою **між ринком (замовником продукту або кінцевим споживачем) і командою**

виконавців. Ця людина відповідає за збільшення цінності продукту і зі старту бачить загальний задум.

2. **Збирається команда виконавців**, компетентність якої повинна поєднуватися з **умінням злагоджено працювати**.

3. Визначається **Scrum-майстер** (рис. 14.7). Тут майстер - це **адміністратор**, стежить за ходом роботи в команді, але не командувач, а допомагає в навчанні, що забезпечує планове проведення зборів і т. ін.

4. **Створюється список вимог до мети і продукту** з розстановкою пунктів за пріоритетом. Цей список змінюється по ходу розвитку проєкту.

5. **Учасники команди оцінюють кожен пункт списку**, вирішуючи скільки **часових і матеріальних ресурсів** буде **потрібно** для реалізації завдання.

6. **Власник продукту, майстер і учасники команди проводять збори, де в спільному обговоренні планується спринт - короткий (щонайбільше місяць** в практиці розробників ПЗ) етап, на який планується рішення певної частини завдань. У деяких контекстах такий етап називають ітерацією. Передбачається, що протягом кожного спринту-ітерації команда напрацьовує певну кількість балів, яку в наступному спринті бажано збільшити, щоб продемонструвати зростання продуктивності.

7. Для інформування всіх учасників процесу **створюється інформаційна дошка** (рис. 14.8), що заповнюється стікерами, з поділом на те, що потрібно зробити, що знаходиться в роботі, і що зроблено. У міру виконання завдань, стікери переміщаються з однієї колонки в іншу.

8. **Короткі загальні збори проводяться щодня**. На них озвучуються перешкоди на шляху, визначається вже зроблене для користі проєкту і заплановане.

9. **Кожен спринт закінчується детальним оглядом результатів роботи** і, що не менш важливо, обговоренням характеристик процесу в минулому спринті. Якщо можна щось поліпшити, то обговорюються спрямовані на оптимізацію **нововведення, які будуть впроваджені в наступному спринті**.

Бэклог (або баклог) - це журнал роботи, що залишилась, яку необхідно виконати команді

Рис. 14.6 Схема процесу Scrum

Складові елементи Scrum. Scrum дошка.

Ролі	*Артефакти*	*Процеси*
• Власник продукту (Product Owner) • Скрам-майстер (Scrum Master) • Команда (Team)	• Беклог продукту • Беклог спринту • Інкремент продукту	• Планування спринту • Огляд Спринту • Щоденний Скрам

Рис. 14.7 Ролі, артефакти та процеси в Scrum

Рис. 14.8 Приклад Scrum-дошки

Scrum був розроблений для проєктів, в яких необхідні **«швидкі перемоги» в поєднанні з толерантністю до змін**. Крім того, цей метод підходить для ситуацій, коли **не всі члени команди мають достатній досвід в тій сфері, в якій реалізується проєкт** - постійні комунікації між членами командами дозволяють брак досвіду або кваліфікації одних співробітників за рахунок інформації і допомоги від колег.

*Онлайн телеканал **Netflix** є відмінним прикладом швидких поставок результатів. Сайт ресурсу оновлюється кожні два тижні завдяки **Scrum**, який не просто дозволяє працювати з високою швидкістю, але й акумулює призначений для користувача досвід і дає можливість виявити найголовніше для клієнтів.*

*В ході кожної ітерації, розробники додають і тестують нові функції сайту і прибирають ті, якими не користувалися клієнти. За словами команди **Netflix**, основна перевага **Scrum** в тому, що **він дозволяє «швидко помилятися»**. Замість того, щоб довго і з великими витратами готувати великий реліз, поставки раз в два тижні по **Scrum** мають невеликий розмір. Їх легко відстежувати і, якщо щось йде не так, швидко виправляти.*

Scrum дуже вимогливий **до команди проєкту**. Вона повинна бути невеликою (5-9 чоловік) і крос-функціональної - тобто члени команди повинні володіти **більш ніж однієї компетенцією, необхідної для реалізації проєкту**. Наприклад розробник ПЗ повинен володіти знаннями в тестуванні і бізнес-аналітиці. Робиться це для того, **щоб частина команди не «простоювала»** на різних етапах проєкту, а також для того, щоб співробітники могли допомагати і підміняти один одного.

Крім того, члени команди повинні **бути «командними гравцями»**, активно брати на себе відповідальність і вміти самоорганізуватися. Підібрати таку зрілу команду дуже непросто!

Scrum підходить не для всіх команд і організацій ще й тому, що пропонований процес може не підійти для розробки конкретного продукту - **наприклад промислової програмно технічної системи**.

Scrum може допомогти коли:

- Концепція змінюється по ходу роботи.
- Потрібно запустити основу проєкту з мінімальним бюджетом і термінами.
- Нервовий замовник - часто показуємо роботу.

Scrum не може допомогти якщо:

- Терміни/бюджет недостатні в принципі (не можуть допомогти і інші методики).
- Команда не мотивована / недосвідчена.
- Клієнт хоче все, на вчора і за безкоштовно.

Результати.

Для клієнта:

- Отримання найважливіших, з точки зору бізнесу, цінностей в найкоротші терміни.

Для команди:

- Ефективність.
- Творчість.
- Задоволення.

14.6. Можливості Kanban

Методологія kanban - це система постановки завдань, при якій всі етапи проєкту **візуалізують на спеціальній дошці**. **Скільки етапів - стільки і стовпців в kanban-дошці** (рис. 14.9 – 14.11). Члени команди можуть бачити поточний стан завдання на будь-який момент часу. Це передбачає повну прозорість роботи.

Мета kanban - зробити проєкт **наочним, відстежити готовність робіт** і проконтролювати **навантаження фахівців**.

Для спрощення контролю робочий процес візуалізують на дошці, поділеної на колонки. Кожна колонка - це поточний стан робіт. Безпосередньо **завдання відображають в kanban-картках** - там можна прочитати їх опис, рівень важливості та додаткову інформацію. **Коли завдання завершує певний етап, картку з її описом переносять у відповідну колонку**. Поглянувши на дошку, можна відразу зрозуміти, яка ситуація з проєктом.

Надо сделать	В работе	Выполнено
Задача 4	Задача 3	Задача 1
Задача 5		Задача 2
Задача 6		

Рис. 14.9 Kanban-дошка. Приклад 1

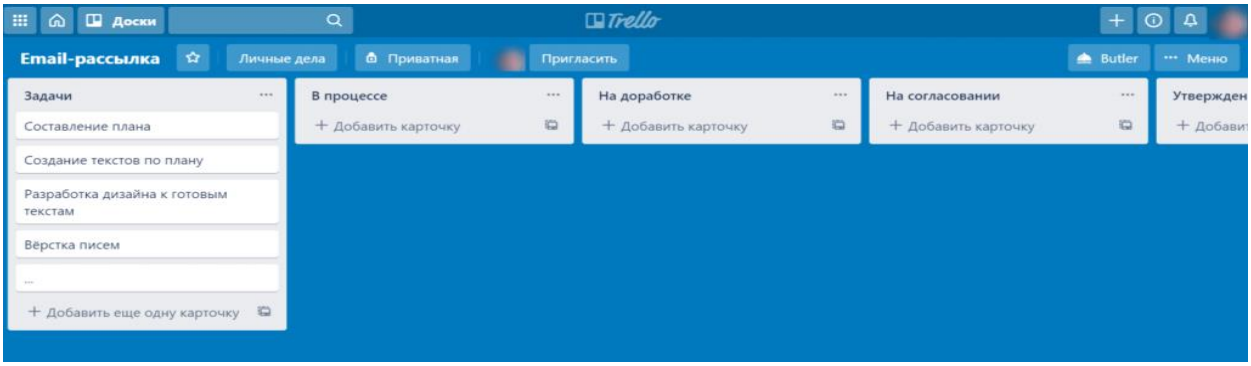

Рис. 14.10 Kanban-дошка. Приклад 2

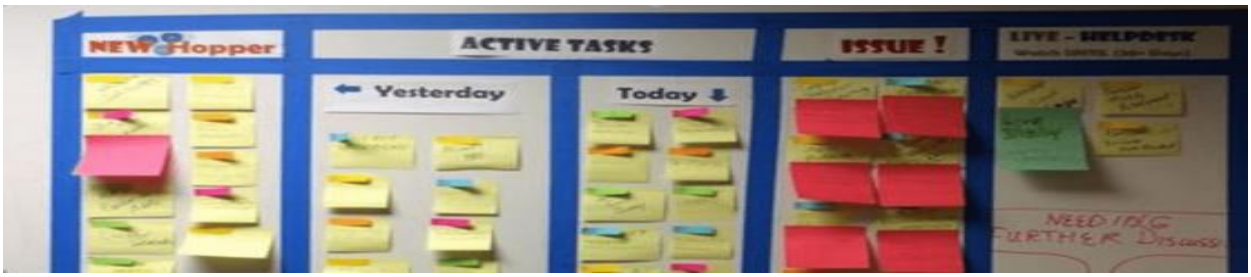

Рис. 14.11 Kanban-дошка. Приклад 3

Чотири базових принципи kanban

1. Відштовхуйтеся від того, що у вас є зараз.

Методологія не закликає **одномоментно** змінювати структуру компанії і ролі співробітників. Навпаки, потрібно **впроваджувати зміни у вже існуючу систему**.

2. Прагнути до поетапних, постійних і еволюційних змін.

Іншими словами - **йти до великої мети маленькими кроками**. На перший погляд може здатися, що глобальні зміни принесуть більше користі і прибутку. Але потрібно пам'ятати, що також вони принесуть величезні ризики.

Поступовий рух до мети - більш гнучкий і безпечний підхід.

3. Поважайте потокові процеси і ролі.

Потрібно зберегти те, що працює добре. Це стосується і відносин, і посад, і процесів. Зв'язки з людьми допоможуть отримати підтримку змін, а налагоджені процеси - удосконалити нестабільні.

4. Підтримувати лідерство на всіх рівнях

Прагнути бути лідерами і пропонувати зміни повинні співробітники на всіх рівнях, а не тільки менеджмент.

Принципи kanban

Візуальне відображення завдань. Всі завдання повинні бути представлені **у вигляді карток і відображені на дошці**. Дуже важливо оновлювати статус завдань. Наприклад, якщо розробники підготували код і передали в тестування, то картка із завданням повинна перейти до відповідного стовпця. Таким чином, будь-який учасник команди в будь-який момент часу може подивитися на якому етапі знаходиться завдання.

Обмеження по стовпцях WIP (work in progress або роботу, що виконується одночасно) на кожному етапі виробництва (**принцип один виконавець – одна задача!**). Щоб система рано чи пізно не "захлинулася" від потоку завдань, необхідно встановлювати обмеження. Наприклад, на kanban-дошці в стовпці Analysis (аналітика) у нас **працюють 2 людини і вони можуть обробляти не більше 2 задач**, немає сенсу навантажувати їх більше, так як наступні етапи системи будуть простоювати. Обмеження по стовпцях підбираються дослідним шляхом.

Фокус на невиконаних завданнях. Дивлячись на дошку із завданнями в першу чергу приділяйте увагу тим завданням, які "**зависають**" в тому чи іншому стовпці. Якщо у вас якийсь із етапів займає найбільше часу, то спробуйте перерозподілити ресурси або ж додати людей, якщо є така можливість.

Постійне поліпшення. Як тільки ви **врівноважити навантаження в системі**, вам буде простіше спостерігати за всім процесом в цілому. Виміряйте час циклу (скільки завдання висить в окремому стовпці, а скільки від моменту потрапляння в To do, до релізу Done). Міняйте навантаження в системі і скорочуйте час на проходження всіх стадій.

Приділяйте увагу дрібницям. Наприклад, якщо код, який пишуть розробники періодично не проходить тестування і повертається на доопрацювання, то можливо, є варіанти поліпшити якість розробки, щоб в тест потрапляв більш якісний продукт?

Переваги Kanban:

- Помилки видно одразу.
- Гнучкість планування.
- Швидка ідентифікація проблемних місць.
- Прозорість (всі бачать хто чим зайнятий).

Недоліки Kanban:

- Якщо в команді більше п'яти осіб, ефективність методології знижується.
- Довгострокове планування неможливе.

14.7. Порівняння Kanban та Scrum. Висновки

Хоч і **Kanban, і Scrum** відносяться до **AGILE**, між ними є кілька серйозних відмінностей.

По-перше, довжина ітерацій. Якщо в **Scrum заздалегідь планується спринт** (зазвичай тижневий або двотижневий), який не змінюється, то в **Kanban нові завдання можна додавати коли завгодно**. Основна мета **Kanban** - закрити завдання, а **Scrum** - закрити спринт.

По-друге, в **Scrum** завдання **оцінюють в Story points або в годинах**, щоб прорахувати, де буде команда після спринту. У **Kanban** розраховують **тільки середній час на виконання одного завдання**.

По-третє, **Scrum** більше підходить **для старту проєкту**, тому що дозволяє точніше визначити терміни і тісно взаємодіяти з командою. **Kanban** ж більш популярний **для підтримки вже готового продукту, його розвитку та модернізації** - в ньому менше комунікації з командою, а завдання приходять в непередбачуваною послідовності.

Питання до розділу 5

1. Найбільш поширені підходи в розробці software?
2. Що становить методика «водоспад». Її фази?
3. Плюси та недоліки методики «водоспад»?
4. Методика гнучких методів. Її принципи та цінності?
5. Плюси та недоліки методики гнучких методів?

6. Порівняльні відмінності методики «водоспад» та методики гнучких методів?
7. Схема та складові методики Scrum?
8. Сильні та слабкі сторони методики Scrum?
9. Методика Kanban. Схема та складові?
10. 4 базові принципи Kanban?
11. Сильні та слабкі сторони Kanban?
12. Надайте порівняльну характеристику методам Scrum та Kanban?