**Перший рівень**

Склади порівняльну таблицю найбільш поширених методологій

| № | Назва методології | Сильні сторони | Слабкі сторони |
| --- | --- | --- | --- |
| 1. | Waterfall | Замовнику легко контролювати роботу над проектом - замовник знає коли запускається кожний етап (маючи просту і зрозумілу модель легше спілкуватись з замовником та звітувати про результати) | Для того, щоб перейти на наступний етап, потрібно повністю завершити попередній (мінус, бо виникають витрати по часу) |
|  |  | Вимоги виписуються максимально детально і чітко (з такими вимогами легше працювати всій команді проекту) | Якщо потрібно внести зміни у вимоги, потрібно повертатись до першої стадії |
|  |  | Вартість проекту визначається чітко і відразу (часто це велика перевага для замовника) | Замовник бачать готовий продукт тільки в кінці і тільки тоді може дати зворотній зв’язок |
|  |  |  | Якщо потрібно внести зміни - збільшення затрат (бо потрібно додатково проходити всі попередні етапи розробки) |
|  |  | Повне документування кожного етапу (легше передати проект іншій команді чи ввести в проект нову людину, бо все детально розписано) | Тестування на передостанньому етапі розробки (погано, бо чим швидше виявлена помилка, тим нижча її вартість (фінансові і часові ресурси на її виправлення) |
|  |  | Список вимог переходить на етап дизайну тільки коли клієнт його погоджує та підписує (все чітко і зрозуміло) | Розробники пишуть багато технічної документації, що затримує процес роботи (більші витрати часу). Чим більше документації на проекті, тим частіше зміни треба вносити і довше їх погоджувати. |
|  |  |  |  |
| 2 | V-модель | Раннє тестування, контроль процесу на кожному етапі (великий плюс, оскільки зменшується вартість помилки) | Не є гнучкою, немає можливості динамічно вносити зміни. |
|  |  | Замовнику легко контролювати роботу над проектом - замовник знає коли запускається кожний етап (маючи просту і зрозумілу модель легше спілкуватись з замовником та звітувати про результати) | Довготривала |
|  |  | Одночасно виписуються вимоги та описується процес тестування системи на кожному етапі. | Якщо в архітектурі була зроблена помилка, то повернутись і виправити її буде дорого, як і в каскадній моделі. |
|  |  | Кількість помилок в архітектурі зводиться до мінімуму. | Потрібно додатково багато тестувальників, що вплине на вартість проекту |
|  |  | Корисна тоді, коли важлива надійність і ціна помилки дуже висока (наприклад, при роботі над подушками безпеки для автомобілів) |  |
|  |  |  |  |
| 3 | Ітераційна | Створення робочого прототипу (MVP) на ранніх стадіях розробки після чого відбувається доопрацювання продукту. Дозволяє бачити якусь картинку вже з самого початку. Ітерації займають невеликий проміжок часу від 2 до 4 тижнів. Відповідно кожного разу можна отримати від розробника фідбек, а також показати що зробили/над чим працювали. | Не завжди всі вимоги відомі до початку роботи, що може ускладнити процес роботи над проектом. |
|  |  | Дуже гнучка модель, готова до змін вимог на будь-якому етапі розробки (для проектів де це важливо, це беззаперечний плюс) |  |
|  |  | Корисна, коли замовник не має повного розуміння яким має бути продукт в кінцевому етапі і не може відразу прописати чітке технічне завдання. |  |
|  |  | Тестуючи кожну ітерацію легше виявити дефекти, бо масштаб менший. |  |
|  |  | Зручна для великих проектів, коли невідомі вимоги, але відомі ключові/деякі вимоги |  |
|  |  | Зручна для проектів, коли ми точно знаємо, що вимоги будуть мінятись в процесі розробки |  |
|  |  |  |  |
| 4 | Інкрементарна | Гнучка модель, готова до змін вимог на будь-якому етапі розробки (для проектів де це важливо, це беззаперечний плюс) | Робота над продуктом поділена на частини, фінальний продукт в кінці. Не створюється MVP. |
|  |  | Зручна для проектів, коли ми точно знаємо, що вимоги будуть мінятись в процесі розробки | Кожна команда програмістів працює над своєю частиною роботи і може реалізувати інтерфейс продукту по-різному |
|  |  | Тестуючи кожен інкремент легше виявити дефекти, бо масштаб менший. | Кожен наступний етап розпочинається після повного завершення попереднього (затрати по часу) |
|  |  | Зручна для великих проектів, коли не відомі вимоги, але відомі ключові/деякі вимоги |  |
|  |  |  |  |
|  |  |  |  |
| 4 | Спіральна | Багато уваги приділяється оцінці ризиків (важливо для деяких сфер) | Розробка триває довго і коштує дорого (постійний аналіз ризиків) |
|  |  | Замовник може побачити MVP вже на ранніх етапах розробки | Є ризик застряти на початковому етапі і нескінченно удосконалювати першу версію продукту, не переходячи до наступних. |
|  |  | Зміни можуть бути внесені на пізніших етапах розробки | Немає зворотнього зв’язку між етапами |
|  |  | Строгий контроль над документацією, як результат постійного аналізу ризиків | Не підходить для невеликих проектів |
|  |  |  | Велика кількість проміжних етапів розробки, а відповідно велика кількість документації. |
|  |  |  | На початку не відома дата завершення роботи над проектом (може бути мінусом для замовника) |

**Другий рівень**

**2. Напиши розгорнуті відповіді (0,5 - 1 сторінки тексту) на такі два питання:**

* **На твою думку, чому з’явився Agile-маніфест?**
* **Які проблеми він мав вирішити і чи це вдалося?**

До моменту підписання Agile-маніфесту вже використовувались гнучкі методики управління проектами (Scrum, Crystal Clear та ін.). Ці методики розвивались на противагу важким методиками розробки програмного забезпечення, яскравим прикладом яких був каскадний метод.

Проблема важких методик полягала в тому, що компанії були надміру зосереджені на плануванні та документуванні своїх етапів розробки програмного забезпечення. Була мета знайти заміну старим методологіям та процесам управління проектами, які показали свою непрацездатність у сучасних проектах.

Agile-маніфест підписали представники гнучких методик управління проектами. Автори розуміли, що роблять щось спільне, а тому їм потрібно знайти спільну мову між цими методиками. Їм потрібно було виписати базові принципи, які є спільними для всіх та об’єднати зусилля.

**Третій рівень**

**2. Ти – засновник/ця стартапу і плануєш випустити на ринок мобільний застосунок для обміну світлинами котиків. Яку методологію ти обереш для процесу розробки і чому? Відповідь текстово обґрунтуй.**

Використаю ітераційну модель:

1. Попрошу зробити MVP з мінімальним функціоналом: можливість зареєструватись, завантажити фото з мобільного телефону, опція надсилання іншим учасникам, які мають такий застосунок.
2. Аналізуємо відгуки та думки користувачів, а також зворотній зв’язок замовника.
3. Переходимо до наступної ітерації та додаємо цікаві фічі на основі аналізу відгуків користувачів, а також фідбеку від замовника - можливість стежити за акаунтом певного користувача і бачити світлини його котиків в першу чергу; можливість коментувати під світлинами; можливість робити надписи на світлинах, ставити емодзі та інтерактивні елементи.
4. Переходимо до наступної ітерації та додаємо цікаві фічі на основі аналізу відгуків користувачів, а також фідбеку від замовника : можливість стрімити, можливість продавати фото котиків через застосунок; проведення конкурсів “Напухнастіший котик”, “Найкумедніший котик” і тд.
5. Переходимо до наступної ітерації - показ реклами в мобільному застосунку; функція визначення уподобань для таргету реклами.

Використала саме ітераційну модель, бо була загальна ідея, але складно було прописати точні вимоги до неї. Крім того, важко було визначити чи є аудиторія в такого продукту, чи буде користуватись вона популярністю і в якому вигляді. MVP допоміг отримати відповіді на ці запитання та зрозуміти чи варто вкладати в цю ідею гроші. Поступове вдосконалення MVP та аналіз зворотнього зв’язку від замовника та користувачів допомагає визначати напрямок руху, які фічі можуть бути цікавими для ЦА. А також прибирати зайве.