Beet Seed — відпрацюй навички на базовому рівні.

Склади порівняльну таблицю найбільш поширених методологій:

| № | Назва методології | **Сильні сторони** | **Слабкі сторони** | **Для якої галузі є доцільною** |
| --- | --- | --- | --- | --- |
| 1 | **Waterfall** | Чіткість структури: допомагає слідкувати кожному етапу (аналіз, дизайн, розробка, тестування, впровадження) які йдуть суворо один за одним.  Простота в управлінні: допомагає легко прогнозувати строки, бюджет та обсяг роботи.  Добре підходить для проєктів з фіксованими вимогами, які зрозумілі з самого початку.  Повна документація, що важливо для сертифікацій та перевірок. | Відсутність гнучкості: важко реагувати на зміни, якщо вони виникають у процесі.  Проблеми, виявлені пізно, можуть спричинити значні затримки або переписування коду.  Користувач бачить продукт лише наприкінці, що в результаті може не відповідати його очікуванням.  Період тестування настає досить пізно і унеможливлює усунення критичних помилок.  Мало зворотного зв’язку протягом проєкту. | В медичній галузі — наприклад, ПЗ для МРТ-сканерів або електронні медичні картки, де важливими є стабільність та відповідність нормам.  В авіації та космічних сферах — системи навігації, контролю польотів, де помилки можуть бути дуже критичними.  В банківській сфері — старіші системи, де вимоги чітко регламентовані та важлива кібербезпека. |
| 2.. | **V-model** | Кожен етап розробки має чітку відповідність етапу тестування, що допомагає дотримуватись плану розробки.  Можливість виявити помилки раніше, оскільки перевірка закладена в саму структуру моделі.  Більш контрольована якість продукту, яка важлива особливо для критичних систем.  Повернення до попередніх етапів простіше, ніж у Waterfall. Процес розробки сповільнюється, але не блокується повністю. | Висока вартість проєкту через велику кількість перевірок і документації.  Мала гнучкість до змін на пізніх етапах - може негативно вплинути на задоволення користувача, особливо якщо потреби змінювались в процесі розробки або були прогалини у вимогах.  Користувач бачить результат лише після завершення всіх етапів, тому складніше врахувати зворотний зв’язок у процесі. | Розробка вбудованих систем, апаратного забезпечення (hardware), де критичність помилок надзвичайно висока.  Автомобільна, авіаційна, медична галузі — всюди, де критична безпека та надійність. |
| 3. | **Incremental** | Програмне забезпечення створюється поетапно, завдяки чому перші версії продукту можна випустити досить швидко.  Розробка розділена на кілька логічних інкрементів, що дозволяє швидко реагувати на зміни на наступних етапах, покращувати продукт з кожним інкрементом та враховувати зворотний зв’язок клієнта після кожного завершеного етапу.  Найважливіші функції впроваджуються першими, що дозволяє сконцентруватись на пріоритетах. | Потрібне ретельне планування всіх інкрементів ще на початку, щоб уникнути конфліктів між модулями.  Кожен інкремент має бути стабільним і завершеним – вимоги до нього “заморожуються” після початку розробки.  Може бути складно інтегрувати всі інкременти в єдину систему, особливо без сильної архітектури. Це важливо, бо клієнт на кожному етапі бачить окрему частину, але повноцінне відчуття “продукту як єдиного цілого” з’являється лише під кінець — і якщо інтеграція виконана неякісно, зростає ризик розчарування. | Добре підходить для мобільних застосунків та онлайн-сервісів, які часто оновлюються і розвиваються поступово.  Це допомагає контролювати кожен процес розробки, швидко реагувати на зворотний зв’язок і поступово формувати якісний та стабільний фундамент продукт.  Підходить компаніям, де продукт розвивається поетапно, але зберігається потреба в контролі над вимогами. |
| 4. | **Spiral** | Підходить для великих і складних проєктів, де є ризики та нечіткі вимоги - функціонал зростає поступово.  Кожна фаза має постановку цілей, оцінку ризиків, проєктування, реалізацію та валідацію, що робить процес контрольованим і передбачуваним.  Регулярний контакт із клієнтом на кожному витку дозволяє вчасно виявляти й виправляти проблеми, отримувати фідбек і вносити зміни.  Можливість поступового нарощення функціоналу з глибшим аналізом на кожному етапі. | Недешева та часозатратна, тому не підходить для малих проєктів чи стартапів.  Вимагає високої кваліфікації команди, особливо у сфері оцінки ризиків.  Порушення плану (графіку чи бюджету) може призвести до втрати ефективності або навіть зриву проєкту.  Суворе дотримання етапів може уповільнити гнучкість при швидких змінах. | У проєктах з високим ступенем невизначеності: великі ІТ-системи, банківські рішення, авіація, оборонна промисловість, енергетика.  Коли ризики мають ключове значення і потрібно мати контроль на кожному кроці. |
| 5. | **SCRUM** | Гнучкий підхід до розробки - дозволяє швидко адаптуватися до змін і потреб користувача.  Високий рівень взаємодії всередині команди: підтримують темп і забезпечують прозорість.  Зосередження на цінності: команда фокусується на створенні робочого функціоналу, а не на надмірній документації.  Клієнт може бачити прогрес і впливати на продукт на кожному етапі. Зростає вдоволення фінальним продуктом. | Потреба в досвідченому менеджменті - без чіткої ролі й лідерства команда може втратити фокус.  Погано визначений або невчасно оновлений беклог може призвести до хаотичного планування спринтів.  Підходить не всім командам — вимагає самоорганізації, відповідальності, високої комунікації. | Ідеально підходить для стартапів, мобільних застосунків, інноваційних проєктів, де є постійні зміни, важливо швидко тестувати гіпотези, потрібен ранній вихід на ринок із базовою версією, команда невелика, але згуртована. |
| 6. | **KANBAN** | Немає жорстких часових рамок — завдання виконуються у своєму темпі, що зменшує стрес у команді та перевантаження.  Візуальна дошка забезпечує повну прозорість та наочність процесу, що полегшує контроль за навантаженням і пріоритетами.  Добре підходить для підтримки та безперервної розробки, де завдання з’являються поступово. | Відсутність фіксованих спринтів або дедлайнів може призвести до затягування завдань, якщо команда не достатньо самоорганізована.  Може виникати хаос у пріоритетах, якщо не встановити обмеження на кількість завдань у роботі. | Проєкти з безперервною розробкою, де завдання надходять нерівномірно - частіше це продуктові компанії, які не мають чітких дедлайнів.  Мобільні застосунки, коли вже є основний реліз але треба робити часті невеликі оновлення. |

Beet Sprout — детальніше заглибся в практику.

На твою думку, чому з’явився Agile-маніфест?

Які проблеми він мав вирішити і чи це вдалося?  
  
  
На мою думку, Agile-маніфест виник у відповідь на стрімкий розвиток технологій і зростаючу складність програмного забезпечення, а також на труднощі, які виникали при використанні традиційних послідовних методів розробки (наприклад, Waterfall). Розробники часто витрачали надмірний час на планування, документацію і формальні процеси, через що випуск продуктів затягувався, а компанії втрачали конкурентоспроможність.

Які проблеми мав вирішити Agile-маніфест і чи це вдалося?

Маніфест був спрямований на зменшення бюрократії, збалансування швидкості і якості розробки, а також активне залучення клієнта і користувачів до процесу створення продукту. Гнучкий підхід Agile дозволяє краще адаптуватися до змін вимог і швидко реагувати на зворотний зв’язок.

Завдяки Agile, процес розробки став більш ітеративним і гнучким, що дійсно сприяє швидшому випуску продуктів, які краще відповідають потребам ринку. Ключовою стала людська комунікація і співпраця всередині команди та з клієнтом, що підвищило задоволеність замовника та кінцевого користувача. Таким чином, Agile успішно вирішує більшість проблем традиційних підходів і залишається актуальним стандартом розробки.

**З особистого досвіду хочу провести порівняння послідовного методу з гнучким:**  
  
Я тривалий час працювала в державному освітньому закладі, де основою педагогіки була бюрократія — чітке і послідовне написання планів, суворе дотримання вимог документації. Більшість робочого часу йшла на оформлення документів, а не на розробку, впровадження та адаптацію моїх занять. Це типовий приклад послідовної (Waterfall) моделі, де на реальні зміни та результати часто не залишається достатньо ресурсів.

Згодом я перейшла працювати у приватний освітній заклад, де одразу відчула гнучкість і сучасний підхід. Хоча я також готувала плани і методичні матеріали, ця документація була лише базою, вхідними даними для роботи. Уся увага була зосереджена на розвитку, зростанні і адаптації до потреб користувачів — у моєму випадку учнів. Саме такий підхід відповідає принципам Agile, коли гнучкість і швидке реагування на зміни важливіші за суворе дотримання початкових планів.

Mighty Beet — різнобічно опануй тематику уроку.

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.   
  
  
Я обрала б методологію Agile, зокрема Scrum framework, тому що вона є гнучкою і орієнтованою на швидке реагування на зміни. Для стартапу, який тільки виходить на ринок, дуже важливо швидко тестувати гіпотези та отримувати зворотний зв’язок від користувачів. Scrum дозволяє регулярно випускати робочі версії продукту, що дає змогу якомога раніше презентувати його цільовій аудиторії.

Крім того, Scrum підтримує тісну співпрацю між членами команди, що особливо цінно на етапі становлення стартапу. Завдяки спринтам та постійному аналізу результатів команда може оперативно вносити зміни у функціонал, фокусуючись на реальних потребах користувачів. Це сприяє швидшій адаптації продукту до ринку та підвищенню його конкурентоспроможності.

Постійна комунікація з клієнтом збільшує ймовірність випуску затребуваного продукту вчасно.

Також ретроспектива наприкінці кожного спринту дозволяє виявити сильні й слабкі сторони команди, що буде корисним для подальшого розвитку стартапу.