Контрольна Робота

Контрольна робота з “Управління проектами та розподіленими командами”

студента другого курсу МП-21 Захаров Дмитра

Варіант 1

Завдання 1. Що таке проект. Складові проекту, життєвий цикл проекту (загальні принципи)

Під терміном “проект” в контексті нашого курсу (тобто IT галузі) зазвичай розуміють деякий процес розробки програмного забезпечення або цифрового продукту для подальшого використання людьми. Хоча, насправді, під цим терміном мова може йти про проект довільного характеру: це може бути соціальний, політичний, науковий проект і всю теорію, що ми застосовуємо у контексті IT сфери, може бути застосована для цих цілей також.

Проект можуть робити по доволі широкому спектру причин: для створення бібліотек, фреймворків, автоматизації ручних процесів, фінансових додатків (маркетплейсів, crowdfunding платформ), допоміжних сервісів тощо.

Якщо трохи більше загиблюватись у специфіку, то під проектом можемо розуміти процес розробки деякою командою фахівців з деяким виділенним бюджетом за деякий естімейт часу. Тобто, ми оперуємо наступними поняттями:

- **Бюджет**
- **Час**
- **Професійність та розмір команди**
- **Якість проекту**

Про ці компоненти та як вони розподіляються в залежності від конкретної ситуації мова буде йти пізніше в інших запитаннях.

Отже, як і будь-який процес, в якому ми використовуємо різні ресурси, треба вміти їми користуватися та мати *методологію*. Тобто, якщо говорити більш технічно, порівнюючі з деякими термінами програмування, мати деяку парадигму або архітектуру, згідно якої йде розподілення часу та послідовності

роботи розробників, комунікація з клієнтом та між командою, бюджетом, розміром команди та багатьма іншими факторами. Якщо такої парадигми не має, то складно вклатися одночасно у бюджет, час та мати на виході ціль, яку потребує клієнт, тому розробка такої “архітектури” є питанням ключовим.

В будь-якій методиці побудування структури процесу розробки можна виділити такі компоненти:

- Розбиття всього процесу на життєві фази.
- Розбиття цих життєвих фаз на етапи та окремі завдання.
- Організаційна структура розробників продукту.
- Розподіл відповідальності.

Зазвичай узгодження методології з клієнтом, її розробка — це функція менеджера проекту. Він також відповідальний за те, щоб усі фази проекту та окремі завдання були виконані вчасно, редакції флоу розробки у випадку якихось проблем, розуміння взаємодії між членами команди (наприклад, хто кого блочить, хто може в цей час зробити або доробити якісь інші компоненти тощо).

Будь-який проект починається з фази аналізу вимог, в якому формалізуються вихідні данні (тобто який продукт повинен вийти у кінці, який дедлайн та чи потрібен *MVP*, наприклад) та вимоги. Далі зазвичай йде етап планування — вибираються методи вирішення завдання (як в плані складу команди, так і специфіки технічної реалізації), який час потрібен та деякі критерії, на які команда буде орієнтуватися під час розробки. Далі йде процес розробки. Але звичайно в реальності процес не буває настільки лінійним: зазвичай, цей процес є ітеративним, і процес розробки циклічно повертається до стадії аналізу, де оновлюються вимоги, естімейт часу та інших ресурсів, знову йде розробка тощо. Отже, процес ніби йде по спіралі.

В загальному виді схема життєвого циклу складається з 4 фаз та була запропонована у схемі американського інституту управління проектами:

1. **Концепція** — збирання початкових данних та аналіз стану проекту на деякій ітерації.
2. **Розробка** — підготовка до реалізації проекту. Назначаються члени команди, проводяться архітектурні коли, створюються календарні плани, формулюються потреби в ресурсах тощо.

3. **Реалізація** — імплементація запланованого у минулій фазі. Зі сторони менеджменту оцінюється темп, робляться правки у координації між членами команди.
4. **Завершення** — вирішення конфліктів, аналіз результатів, можливо проведення ретроспективи тощо. Також можливо створення сапорт команди для підтримання проекту, якщо основний набір функціоналу був реалізований.

Завдання 2. Управління вартістю проекту. Фінансове обгрунтування IT проекту.

Як ми вже згадували у завданні 1, всього виділяють 4 обмеження на ресурси у проджект-менеджменті:

- Вартість (*Cost*)
- Обсяг (*Scope*)
- Часові обмеження (*Time*)
- Якість (*Quality*)

І всі ці компоненти не є незалежними один від одного. Наприклад, нехай в нас на розробці деякого невеликого сервісу є 5 розробників: дизайнер, 2 фронтенд розробника, 1 бекенд розробник та девопс. Якщо в деякий момент розробки ми вирішили прибрати фронтенд розробника на інший проект, то нам знадобиться більше часу для тієї самої якості, але менший бюджет. Якщо ми захочемо краще протестувати проект щоб уникнути багів, тобто збільшити якість продукту, нам потрібно більше часу. Якщо хочемо зробити проект швидше, додамо 1 бекенд розробника. Час зменшиться, а бюджет збільшиться. Дуже спрощено це співвідношення можемо записати так:

$$\mathrm{Cost} + \mathrm{Scope} + \mathrm{Time} = \mathrm{Quality}$$

До чого ми навели це правило? Тому що воно безпосередньо описує правило, згідно якого ми управляємо бюджетом проекту.

Зазвичай у більшості проектів найбільша частина бюджету складається з витрат на ресурси: по-перше, це звісно людські ресурси, а далі це може бути інтеграція з іншими сервісами (умовно під’єднання зовнішньої *API* або якихось сервісов по типу *AWS*) або обладнання тощо.

І аналіз бюджету — це дуже складний аналітичний процес. Наприклад, можна проаналізувати найбільш затратні частини проекту і переконатися, що ми використовуємо оптимальні тарифи. Можемо замінити ресурси на більш дешеві (однак ми втратимо якість, згідно формулі вище). Можна зменишити *Scope.* Або знайти більш дешеву альтернативу до ресурсів, але знадобиться час для аналізу, хоч це і не змінить якості продукту.

Також варто відмітити, що є різні методики аналізу витрат на проекті. В цьому завданні опишу *ABC аналіз.*

ABC (Activity based costing) аналіз — це підхід, у якому ми спочатку визначаємо вартість окремих операцій, що потрібно для розробки продукту, а потім для кожної з цих операцій підбираємо схему розподілу, за допомогою якої обчислюється вартість послуги.

В цій концепції маємо наступні елементи:

- *Операція* — дія, що виконується на деякому етапі розробки.
- *Атрибут* — мітка, присвоювана операції в *ABC системі.*
- *Ресурс* — елемент, необхідний для виконання операції.
- *Драйвер ресурсу* — кількісний зв’язок між ресурсом та операцією.
- *Вартість операції* — назва каже сама за себе.
- *Об’єкт витрат* — продукт, для створення якого ми робимо операцію.

І зазвичай розподіл витрат при такому підході (він називається операційно-орієнтований) передбачає наступні етапи:

- Розподіл витрат підприємства на ресурси (тобто по суті на тих/те, хто/що виконує операції) з формуванням драйверів ресурсів.
- Перенесення вартості ресурсів на операцію з формуванням драйверів ресурсів.
- “Типізація” операцій на основні та обслуговуючі
- Перенесення вартості основних операцій на об’єкти витрат

Тобто маємо, що по суті до собівартісті продукту під час всього процесу розробки послідовно додаються собівартість здійснюваних операцій. Цей метод дає доволі гнучкий контроль бюджету, бо дає дуже зручну і інформативну структуру розподілу.

Отже, маючі цю методику, ми можемо зазделегідь спланувати та оцінити доцільність інвестицій. Для такої оцінки є окремий термін: оцінка окупності інвестицій — ROI (про це далі пізніше).

Тепер перейдемо до *фінансового обгрунтування IT проекту.* Зазвичай, проекти роблять для отримання грошей (навіть якийсь безкоштовний софт може збільшити аудиторію, яка потенційно буде купувати платні продукти цього ж розробника). Тому оскільки розробка не є безкоштовною, потрібно оцінити, а чи буде потенційний прибуток більший за собівартість?

І насправді процес не такий простий, як здається: тобто немає такого, що ми в один момент витратили A грошей і одразу за це отримали B грошей. Для характеристики прибутку і витрат також треба враховувати час і тому доцільно використовувати термін *грошовий потік*.

Оцінка проекту та його грошові потоки залежить від 2 факторів:

- Розподіл грошових потоків у часі (тобто, наприклад, як швидко він зможе принести гроші. Одне діло отримати 1 мільйон доларів за 2 дні, а зовсім інше за рік)
- Ризикованість проекту — варто порівнювати ризик вкладання грошей у проект з альтернативними можливостями вкладання цих грошей.

Варто описати, що ж значить, що проект є окупним. Для цього варто зрозуміти термін “точка беззбитковості”. По суті — це момент часу після вже деякого релізу проекту (наприклад, після *MVP*), коли прибуток дорівнює собівартості на розробку цієї версії релізу. Тобто коли ми досягли цієї точки, то йдемо вже тільки у плюс.

І по суті інвестор зацікавлений у 2 речах: щоб ця точка беззбитковості була якнайближчою після реліза та щоб після цієї точки був максимальний прибуток.

Чисельно період окупності можна описати наступним чином:

$$\mathrm{Account\ payback\ period} = \frac{\mathrm{Investment\ amount}}{\mathrm{Annual\ income}}$$

Але ця формула не враховує ризики при здійсненні проекту.

Тому придумали показник NPV — *Net Present Value*, що враховує очікуваний майбутній дохід за вирахуванням його первісної вартості. Формула для розрахунку має вид

$$\mathrm{NPV} = \sum_{i=1}^{n} \frac{\mathrm{CF}_i}{(1+r)^i}$$

Де n — це кількість років, протягом яких ми рахуємо показник NPV, на iому році маємо платіж CF_i (який може бути як позитивним, так і негативним), а r — ставка дисконтування, що дорівнює необхідної прибутковості інвестицій.

Варто ще згадати показники IRR (Internal Return of Rate) — ставка дисконтування така, що $\mathrm{NPV}\Big|_{r=\mathrm{IRR}} = 0$ та ROI (Return on investments) — індикатор ефективності, співвідношення прибутку або збитку від реалізації стосовно витрат, що розраховується за формулою

$$\mathrm{ROI} = \frac{\mathrm{Current\ value\ of\ Investment}}{\mathrm{Cost\ of\ Investment}} - 1$$

Якщо $\mathrm{ROI} < 0$, то проект збитковий.

Завдання 3. Інкрементна модель.

Інкрементна модель — це модель, при якій повні вимоги по проекту поділяються на різні складові. Тобто маємо якийсь набір циклів розробки, і разом вони становлять життєвий цикл “мульти-водоспад”.

В цьому методі ми розбиваємо увесь процес на дрібні модулі, що відносно легко створюються. Причому кожен цей модуль проходить усі головні фази: визначення вимог, проектування, реалізація, тестування. Ця методика передбачає, що ми спочатку випускаємо базовий функціонал, а далі зверху до нього вбудовуємо нові функції, які теж повністю ретельно проробляються. Важливо відрізняти такую модель від ітеративної, де ми спочатку робимо усе у вигляді прототипу/начерку, але з кожною ітерацією лише покращуємо та додаємо деталей до базового функціоналу).

Наприклад, ми вирішили створити відеогру-платформер. На першій ітерації ми можемо створити базову механіку руху персонажу та першу локацію з усіма монстрами. Цей етап ми тестуємо, повністю полірує і викатуємо умовно на реліз. Далі на наступній ітерації ми додаємо ще одну локацію з монстрами та може новою механікою. Тестуємо, викатуємо на реліз. І так повторюємо, додаючі нові локації з кожною ітерацією. Якщо ж ми б використовували ітеративну модель, то ми б зробили усі локації одразу з базовою механікою героя, але наприклад усі малюнки, анімації були недороблені та накидані “на

швидку руку”. А далі на наступній ітерації домалювали б монстрів, на наступній додали анімацій і т.д.

Інкрементну модель варто використовувати коли вимоги чітко зазначені та зрозумілі, або/та якщо потрібно раннє виведення або/та є кілька ризикових фіч і цілей.

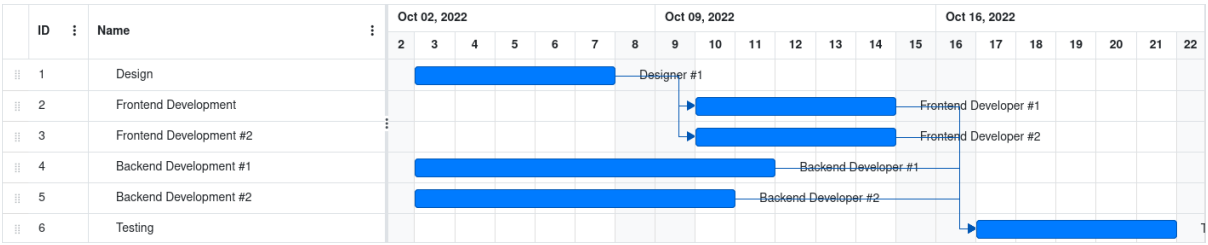
Головними плюсами такої моделі є:

- Клієнт може давати відгук щодо кожної версії продукту
- Є змога переглянути ризики, які пов’язані з витратами і дотриманням графіка

З мінусів можна виділити те, що структура системи має бути чітко визачена з самого початку і термін здачі усього проекту цілiком може бути доволі складно контролювати, бо є похибка на кожній окремій ітерації.

Завдання 4. Скласти розклад команди

З кількості годин можемо одразу побачити, що завдання доволі мале за своїм розміром, бо максимальний естімейт з усіх 4 членів команди — це приблизно 12.5 робочих днів, тобто приблизно 2.5 тижні, тому немає сенсу придумувати якусь дуже складну структуру. Наприклад, нехай розклад буде таким:



Нехай 2 бекенд розробники зроблять увесь бекенд за $100 \cdot 0.5 = 50$ годин — це приблизно $6 - 7$ днів. Паралельно їм дизайнер зробить свою роботу за 40 годин, тобто за робочий тиждень. Одразу після нього підключаємо 2 фронтенд розробника, що зроблять роботу та $80 \cdot 0.5 = 40$ годин, тобто теж за тиждень. К 3 тижню бекенд та фронтенд будуть готовими, тому підключаємо тестера, який свою роботу закінчить за 40 годин, тобто теж за робочий тиждень. Ітого маємо, що усю роботу ми можемо закінчити за 3 робочі тижня, тобто $3 \cdot 5 \cdot 8 = 120$ годин.

Насправді на практиці (правда, працював я на доволі малій кількості проектів як бекенд розробник) тестерів можуть підключити вже на етапі закінчення бекенд розробки для перевірки АПІ, що може сильно зекономити час на вияснення де

виникає проблема під час інтеграції: на фронтенді чи бекенді. Тому якщо врахувати, що бекенд може виділити спочатку 2 дні на тест бекенду (на 100 годин бекенду йому більше часу не знадобиться, зазвичай це доволі швидко перевіряється), а далі вже залишок часу на фронтенд (тобто 3 дні), то можна ще додатково зекономити 2 дні, тобто можемо все закінчити за 102 години. Хоча звісно краще розраховувати все десь з коефіцієнтом $\times 1.5$, якщо це ще не зроблено на естімейтах девелоперів, бо щось точно піде не так (теж з'ясовано з практики).