**Тема. Робота з часом та бюджетом проекту**

1. **Параметри проєктного трикутнику**
2. **Визначення бюджету витрат та критичний шлях проекту**

Щоб здати успішний проєкт, менеджер повинен стежити за терміном, бюджетом та скоупом (обсягом) робіт. Кожен параметр впливає на два інші, тому проджекти вигадали метафору проєктного трикутника. З ним легше відстежувати зміни у показниках проєкту.

**Проєктний трикутник**— це фігура із трьома вершинами: час, бюджет та скоуп. Він ілюструє баланс ресурсів у проєкті. Якщо змінити одну вершину, то зміняться дві інші. Тому задачі проджекту стежити за тим, щоб усі три параметри були в порядку. Якщо з ними все гаразд, то і з проєктом теж.

**

*Ось приклад*, коли команда вирішує заощадити на бюджеті. Команда хоче зменшити витрати, і для цього скорочує час на розробку фіч. Тому QA проводить не всі тести та залишає деякі баги невиправленими. У продакшені їх помічають користувачі, і ставлять низькі оцінки додатку в App Store. Скоротивши вершину «час», команда зменшила бюджет та скоуп завдань. Але з цими параметрами знизилася якість.

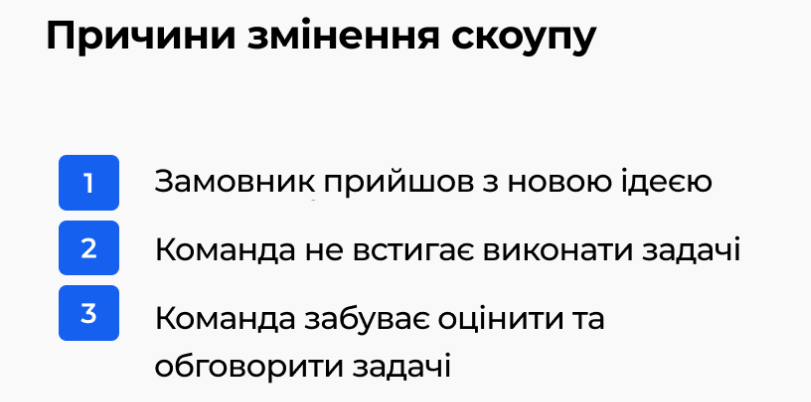
**Розглянемо кожну складову.**

**1. Скоуп**— це задачі у твоєму проєкті, які потрібно виконати за контрактом. Це результат роботи дизайнерів, розробників та тестувальників. Задач може бути багато, і всі вони відрізняються за структурою та складністю. Їх обсяг та кількість можуть змінюватися з часом. Якісь фічі зникають з беклогу через непотрібність, інші з'являються за бажанням замовника. *Тому важливо стежити, як у твоєму багтрекері рухаються задачі.* Перевіряй, що зроблено, що у процесі, а на що у вас не вистачає часу. Так ти розумітимеш, скільки завдань із запланованих за контрактом ви виконали.

Не забувай стежити за скоупом щодня — через стендап, багтрекер та звіти.

Іноді скоуп проєкту змінюється, тому що команда не встигає закінчити задачі й вирішує від них відмовитись, Наприклад, якщо проєкт необхідно терміново поставити користувачам або показати інвесторам. Тоді скоуп скорочується.

Буває й так, що замовник має нові ідеї. Також на грумінгу чи процесі розробники можуть бути неправильно інтерпретувати задачі. Тоді ваш скоуп теж змінюється.



**Причина 1. Замовник прийшов із новою ідеєю.**Уяви, що після демо замовник вирішив ускладнити продукт і пропонує додати нову фічу. Ви погоджуєтесь. Тоді загальний скоуп проєкту змінюється, бо додається нова функціональність.

*Довідково.* Демо-зустрічі проводяться на кінці кожної ітерації, щоб продемонструвати результати роботи за цей період.

Демо-зустріч у проекті може включати презентацію нових функцій, які були розроблені, вирішення проблем, які виникли в попередній ітерації, або демонстрацію тестових варіантів продукту для отримання відгуків від користувачів. Метою демо-зустрічі в проекті є забезпечення зворотного зв'язку та вдосконалення продукту на основі цього зворотного зв'язку. У демо-зустрічі можуть брати участь розробники, тестувальники, керівники проекту та інші зацікавлені сторони, залежно від конкретних потреб проекту.

**Причина 2. Команда не встигає виконати задачі та вирішує від них відмовитися.**Ви оцінюєте зусилля та кажете, що впровадите нову ідею за місяць. Але за два тижні розробники розуміють, що за такі терміни не встигають зробити роботу на хорошому рівні.

Щоб встигнути вчасно, *ви можете «викинути» зі скоупу задачі — наприклад, тестування.* Звісно, ​​якщо замовник погодиться.

*Інший варіант скорочення термінів* — знизити якість. З одного боку, відмовившись від хорошого тестування, продукт програє в якості. Загалом, так і є — якщо багів багато, то він просто не буде нормально працювати, та користувачі підуть. З іншого, не всі помилки у беклозі потрібно виправляти. Баги, як і фічі, розв'язують задачі користувачів. Вони мають цінність, яку потрібно оцінити та пріоритизувати. Уяви, що в онлайн-магазині зламалася кнопка оплати. Цей баг треба лагодити якнайшвидше, інакше бізнес втрачає гроші.

**Причина 3. Команда забуває оцінити та обговорити частину завдань.**Буває й так, що ви просто забули щось оцінити чи погано обговорили задачі на грумінгу. Тоді втрачені деталі почнуть спливати вже у процесі розробки. І якщо без них фіча дійсно не працюватиме, то їх доведеться включати в скоуп.

*Наприклад*, зараз твоя команда розробляє фічу «Додавання відео з галереї». Ви завели кілька завдань та успішно їх закриваєте. Але уяви, якщо всередині спринту ви зрозуміли, що забули про «Permission» — доступ Filmy до галереї на телефоні. Без нього користувач просто не зможе отримати доступ до своїх файлів, а значить, не вибере відео та не перейде на екран редагування. Тому ви включаєте задачі вже в ході розробки, і їхня загальна кількість збільшується.

Змінювати скоуп в процесі розробки завжди боляче. Щоб цього уникнути, бізнес-аналітики та проджекти витрачають багато часу на докладний опис та планування завдань.

Не забувай стежити за задачами в багтрекері, щоб тримати всі зміни у скоупі та затримки під контролем. *Наприклад*, у Jira відстежувати скоуп можна за допомогою поля fix version.

*Довідково.*

Версії Jira Software відображають стан проекту на певний момент часу. Вони допомагають організувати роботу за допомогою створення контрольних точок, яких потрібно досягти. Ви можете пов'язати завдання проекту з конкретною версією та організувати спринти для виконання роботи у рамках цієї версії.

Коли команда розробників виправляє проблему, вони звичайно вказують в полі "Fix version" ту версію програмного продукту, у якій було виправлено проблему. Це поле допомагає керівництву проекту та користувачам бачити, в якій версії програмного продукту проблема була виправлена, і забезпечує більш точну інформацію про те, які вірсії програмного продукту є стабільними. "Fix version" також може використовуватися для планування релізів продукту, коли команда вказує, в якій версії програмного продукту планується додати певні функції або виправити проблеми. Це поле допомагає команді планувати роботу та орієнтуватися в тому, які проблеми потрібно виправляти першими, а які можуть залишатися на наступні версії.

Завжди відмічай задачі у багтрекері **мітками (на картці)**. Заведи окремий тег для завдань, які були додані в процесі спринту. Так легше контролювати зміни у скоупі.

Тепер ти знаєш, що на скоуп впливають як замовник, так і сама команда. Пам'ятай, що якщо не встигаєте до дедлайну, від деяких завдань можна відмовитись (а точніше, перенести їх на потім): наприклад, від юніт тестів або низки нефункціональних вимог.

**Якість продукту – в центрі трикутника**

Розглянемо основні компоненти що забезпечують якість продукту.

*Юніт тести*—це тестування окремих модулів чи компонентів у програмі. Їх пишуть розробники на етапі написання коду. Ізолюють його частину та перевіряють, чи все працює і чи не зламає новий код старі компоненти системи. Якщо не писати юніт-тести, то кількість багів при подальшому тестуванні та у продакшені збільшиться. Коли настане час чинити несправність, у коді буде складно розібратися, адже його ніхто не перевіряв. Тому якщо є ресурси, то юніт тести краще не опускати. Так, у перспективі ви заощадите загальний час та гроші, адже без тестів додавати нові фічі стане дорожче.

Але трапляються випадки, коли від них таки можна відмовитися. Це невеликі сайти, прототипи та інші проєкти, які не будуть розвиватися далі. А також терміни, що підганяють, і жорсткі дедлайни — наприклад, коли проєкт потрібно показати інвесторам.

***Нефункціональні вимоги***теж впливають на якість продукту. Пам'ятай, що відмова від юніт тестів та нефункціональних вимог веде до технічного боргу. Якщо ви щось не встигаєте, задачі не зникають, а накопичуються і перетворюються на недороблену функціональність та баги. Найпоширеніший такий наслідок — **техборг.** *Наприклад*, він виникає, якщо замість запланованих 10 днів команда намагається закінчити фічу за 6. З одного боку, фіча здана раніше. З іншого — через швидкі рішення в коді та архітектурі почнуть накопичуватися проблеми. Адже робота, запланована на 4 дні, що залишилися, нікуди не пропадає. Ці задачі вирушають у техборг.

Якийсь час з техборгом можна нічого не робити, але в певний момент поганий код і тимчасові рішення дадуть про себе знати. Працездатність додатка погіршиться настільки, що продукт може зламатися. *Наприклад*, можна втратити дані користувачів і відновлювати їх кілька днів. Або виявити вразливість, що дозволяє вкрасти конфіденційні дані. Все це проблеми, наслідки яких бізнес може не пережити. А ще з технічним боргом складніше запроваджувати нові фічі. Адже старі проблеми все ще нікуди не діваються. Код стає заплутанішим, і писати новий поверх такого «монстра» стає все важче.

*Приклад,* Уяви, що ви вирішили випустити бота, який допомагає школярам старших класів писати мотиваційні листи для вступу до університету. Ви знаєте, що продукт повинен розв'язати проблему, але поки що не впевнені, як. Ви також бачите, що на ринку поки що немає зручних рішень. Щоб перевірити продуктову гіпотезу, необхідно спробувати кілька варіантів роботів: з різними діалогами, прикладами листів, ToV (тон і гучність голосу) і форматами спілкування. Все це необхідно тестувати на реальних користувачах. Тому ваше завдання — швидше випустити MVP, а потім уже вдосконалювати додаток. Інакше виявиться, що ви витратили купу часу на бездоганний дизайн, швидкість завантаження та легкий у навігації UI, а бот у такій формі нікому не потрібний.

1. **Наступна складова трикутника.**

**Бюджет**— це кошти, виділені на роботу команди. У нього закладають час розробників та будь-які витрати, пов'язані з проєктом: закупівля пристроїв, ліцензій, витрати на серверне оточення.

Витрати включають такі базові статті (Див. Додаток А):

Зарплату та премії працівникам

Витрати та утримання та амортизацію обладнання

Оренда офісу, Інтернет

Ліцензії (якщо їх вартість не компенсується клієнтом)

Тімбілдинги (може включати в себе різні види діяльності, такі як тренінги, ігри, спільні відвідування, спортивні заходи тощо)

Відрядження (якщо їх вартість не компенсується клієнтом)

Інші витрати

Додаток А **Приклад розрахунку витрат**

Таблиця 1

Розрахунок балансу робочого часу одного працівника

|  |  |  |
| --- | --- | --- |
| Найменування показника | Одиниця  виміру | Величина показника |
| 1. Кількість робочих днів | дні |  |
| 2.Неявки на роботу, у т.ч.:   1. основні і додаткові відпустки 2. по хворобі 3. по навчанню 4. виконання громадських і державних обов’язків 5. інші (з дозволу адміністрації) | -//-  -//-  -//-  -//-  -//-  -//- |  |
| 3. Кількість явок на роботу | -//- |  |
| 4. Середня тривалість робочої зміни | год. |  |
| 5. Дійсний фонд робочого часу | -//- |  |
| 6. Трудомісткість задач | тис. н-год. |  |
| 7. Плановий коефіцієнт виконання норм | % |  |
| 8. Чисельність основних робітників | чол. |  |

Таблиця 2

Розрахунок фонду заробітної плати працівників

|  |  |
| --- | --- |
| Найменування статей фонду оплати праці | Сума, грн. |
| 1. Нормована заробітна плата (по тарифу) |  |
| 2. Премія |  |
| 3. Нормована заробітна плата з премією |  |
| 4. Доплата за роботу у нічний час |  |
| 5. Доплати за сумісництво |  |
| 6. Оплата пільгових годин |  |
| 7. Оплата тарифних відпусток |  |
| 8. Оплата відпусток по навчанню |  |
| 9. Доплата за виконання державних і громадських обов’язків |  |
| 10. Інші доплати |  |
| 11. Разом доплати і надбавки |  |
| 12. Доплати і надбавки разом у % до нормованої заробітної плати з премією |  |

Таблиця 3

Штатний розклад керівників, спеціалістів, МОП та іншого персоналу

|  |  |  |  |
| --- | --- | --- | --- |
| Найменування  посади | Чисельність, чол. | Оклад середній, грн. | ФЗП, грн. |
| 1.Керівники: |  |  |  |
| Разом |  |  |  |
| 2.Спеціалісти: |  |  |  |
| Разом |  |  |  |
| 3.МОП: |  |  |  |
| 4.Інші: |  |  |  |
| Разом |  |  |  |

Таблиця 4

Кошторис витрат на утримання і експлуатацію обладнання

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Найменування  статей витрат | Разом план  на рік, тис. грн. | у тому числі, тис.грн. | | | | | | |
| Матеріали  допоміжні | Енергія | Заробітна  плата | Відрахування  на соціальні  заходи | Амортизація | Послуги   інших цехів | Інші |
| 1. Амортизація обладнання |  |  |  |  |  |  |  |  |
| 2.Поточний ремонт і експлуатація обладнання |  |  |  |  |  |  |  |  |
| 3. Внутрішні перевезення вантажів |  |  |  |  |  |  |  |  |
| 4. Вартість МШІ (інструментів) |  |  |  |  |  |  |  |  |
| 5. Інші витрати |  |  |  |  |  |  |  |  |
| Разом |  |  |  |  |  |  |  |  |

Таблиця 4.1

Розрахунок вартості обладнання та встановленої   
потужності двигунів

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Найменування обладнання | К-ть, шт. | Встановлена потужність, кВт | Середня  ціна од. обл., грн. | Сумарна потужність, кВт | Вартість обладнання, тис. грн. |
|  |  |  |  |  |  |
| Разом |  |  |  |  |  |

Таблиця 5

Кошторис загальновиробничих витрат

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Найменування статей витрат | Разом  план на рік,  тис. грн. | у тому числі, тис. грн. | | | | | |
| енергія | зарплата | відрахування | амортизація | послуги  ін. цехів | інші |
| 1. Утримання апарату управління |  |  |  |  |  |  |  |
| 2. Утримання іншого персоналу |  |  |  |  |  |  |  |
| 3. Утримання будівель і споруд |  |  |  |  |  |  |  |
| 4. Поточний ремонт будівель і споруд |  |  |  |  |  |  |  |
| 5. Вартість МШІ |  |  |  |  |  |  |  |
| 6.Раціоналізація і винахідництво |  |  |  |  |  |  |  |
| 7. Охорона праці |  |  |  |  |  |  |  |
| 8. Інші |  |  |  |  |  |  |  |
| Разом |  |  |  |  |  |  |  |

Таблиця 6

Зведений розрахунок витрат

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Найменування  статей витрат | *Разом план на рік,*  тис. грн. | у тому числі по елементах, тис.грн. | | | | | | | |
| основні  матеріали | допоміжні матеріали | енергія | зарплата | відрах. на соц.заходи | амортизація | послуги  інших цехів | інші витрати |
| 1. Матеріали |  |  |  |  |  |  |  |  |  |
| 2. НЗП  з премією |  |  |  |  |  |  |  |  |  |
| 3. Доплати і надбавки |  |  |  |  |  |  |  |  |  |
| 4. Відрах. на соціальні заходи |  |  |  |  |  |  |  |  |  |
| 5. Вартість спецінструменту та оснастки |  |  |  |  |  |  |  |  |  |
| 6. Витрати на ремонт і експлуатацію обладнання |  |  |  |  |  |  |  |  |  |
| 7. Загальновиробничі витрати |  |  |  |  |  |  |  |  |  |
| 8. Разом |  |  |  |  |  |  |  |  |  |

**В цілому, КРІ проекту** (ключові результативні індикатори) є метрикою, яка допомагає вимірювати результативність та ефективність виконання проекту. КРІ повинні бути пов'язані з цілями проекту та допомагати оцінювати досягнення цих цілей. Для проекту з фокусом на рентабельності, ключовими результативними індикаторами (КРІ) можуть бути:

*Чистий прибуток*: вимірюється як різниця між загальними витратами проекту та його доходами. Це може бути одним з головних КРІ для проекту, що фокусується на досягненні прибутку.

*Маржинальний дохід*: вимірюється як різниця між доходами та витратами на виробництво чи надання послуги. Цей показник допомагає оцінити, наскільки вигідним є проект.

*Рентабельність проекту*: вимірюється як співвідношення чистого прибутку до загальних витрат. Цей показник допомагає оцінити, наскільки ефективним є проект в контексті його вартості.

*Кількість клієнтів або продажів*: вимірюється як кількість клієнтів або продажів, що були зроблені в рамках проекту. Цей показник допомагає визначити, наскільки успішним був проект у залученні клієнтів та збільшенні обсягу продажів.

*Відсоток повторних продажів*: вимірюється як частка клієнтів, які здійснили повторні покупки або використали послуги проекту знову. Цей показник допомагає оцінити ефективність проекту у збереженні клієнтів та збільшенні лояльності.

КРІ для project margin залежить від:

* Фінансових цілей компанії
* Типу контракту
* Важливості проекту для команди
* Ризиків проекту
* Об’ємів проекту

Формула для розрахунку чистого прибутку:

Net Profit Margin = (Revenue - Cost) / Revenue



Маржа - це прибуток, який отримує компанія від продажу своїх товарів або послуг, після відрахування витрат на їх виробництво, рекламу, зарплати працівників та інших витрат.

Revenue - це сукупний дохід, який компанія отримує від продажу своїх товарів або послуг.

Cost - це сума грошей, яку компанія витрачає на виробництво товарів або надання послуг.

***Отже, я***кщо від рівня revenue відняти витрати (cost) і результат поділити на revenue, то отримаємо показник, який називається чистий прибуток (net profit margin) або рентабельність продукції.

*Види маржі:*

Labor margin – у витрати включені лише зарплата

Cross margin – у витрати включені не тільки прямі витрати проекту, але й загальні витрати компанії (зарплата адміністративного персоналу, оренда офісу, податки та інші операційні витрати)

*Пошук балансу при створенні бюджету проекту:*

* Зменшення скоупу проекту, створення MVP
* Перегляд технічного рішення
* Оптимізація структури команди
* Зменшення додаткових витрат в проекті

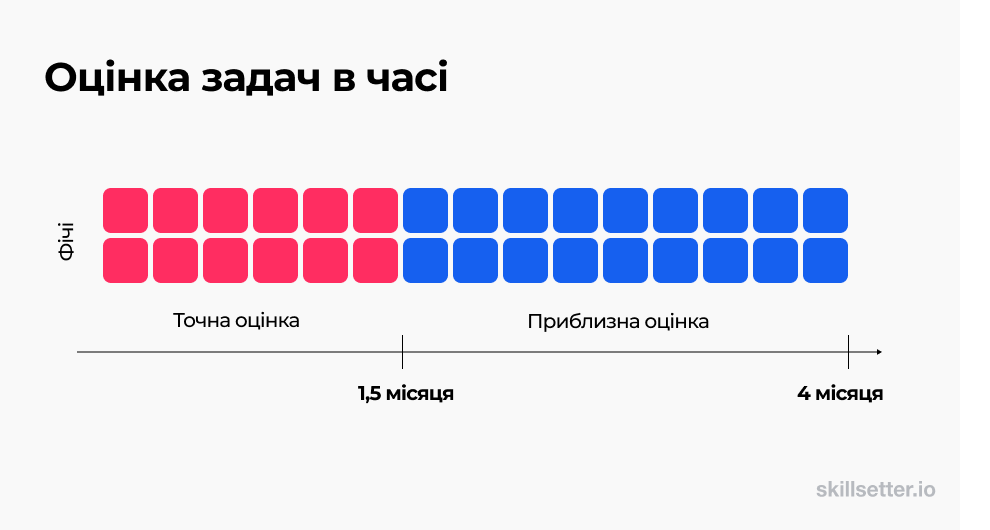
Таблиця

Планування підтримки маржинальності після початку проекту

|  |  |
| --- | --- |
| Шляхи збільшення дохідної частини | Шляхи зменшення витратної частини |
| Збільшення рейтів (оплати за годину) для клієнтів з певною періодичністю (цей пункт краще закладати в контракт з самого початку) | Оптимізація піраміди сеньйорів в проекті (баланс senior – middle – junior) |
| Платні овертайми (додатковий час роботи) | Зменшення рівня ризиків в проекті |
| Збільшення команди для прискорення розробки | Робота з фрілансерами: найм фрілансерів може бути дешевшим за найм повноцінних співробітників або залучення зовнішніх компаній. |

1. **Третя складова трикутника. Час.**

Ти вже вмієш керувати скоупом та знаєш, як розрахувати вартість. Але тепер Олівія просить назвати час готовності першої версії Filmy. Потрібно зробити прогноз по роботі, що залишилася, і зрозуміти, за скільки спринтів ви встигнете її підготувати.



Якщо команді потрібна приблизна оцінка термінів, можна використовувати Burndown. З його допомогою менеджер відстежує Velocity проєкту і кількість спринтів, що залишилися. У Jira такий графік складається автоматично.

Burndown chart - це графік, який відображає кількість залишкової роботи (backlog) у проекті по днях чи ітераціях. В Jira, Burndown chart є одним із основних засобів для відстеження прогресу виконання проекту.

Кожного дня/ітерації, Burndown chart показує, скільки роботи має бути виконано та скільки роботи вже було завершено. Якщо лінія залишкової роботи спускається швидко, це означає, що команда успішно виконує роботу відповідно до плану. Якщо лінія залишкової роботи піднімається, це означає, що команда не встигає виконувати заплановану роботу.

Burndown показує, скільки сторі поінтів команда виконувала в кожному спринті, і як це число змінювалося від спринту до спринту.

На основі динаміки Velocity проджект може прогнозувати наступні ітерації. Кожен стовпець — це один спринт. Velocity є ключовим елементом для планування роботи команди в майбутньому та для прогнозування того, скільки часу знадобиться для виконання конкретних задач. Velocity розраховується шляхом підрахунку кількості завершених задач (story points) протягом одного спринту. Чим більше story points команда може завершити за спринт, тим вищий її velocity..

Однак, варто зазначити, що Velocity не є статичною метрикою і може змінюватись з часом. Він залежить від рівня складності задач, які виконує команда, і від того, чи вона здатна ефективно співпрацювати та вирішувати проблеми. Тому, для отримання точних результатів, Velocity повинен розраховуватись на основі декількох спринтів, а не лише одного.

Burnup chart - це графік, що відображає виконання роботи в проекті на основі двох осей: вертикальна ось показує обсяг роботи, яка повинна бути виконана, а горизонтальна ось показує час. Він дозволяє відстежувати, наскільки ефективно команда виконує роботу та який прогрес вона робить відносно запланованої роботи.

Графік Burnup показує дві лінії: лінію "виконана робота" та лінію "запланована робота". Лінія "виконана робота" відображає кількість завершеної роботи протягом часу, тоді як лінія "запланована робота" показує, скільки роботи повинно бути виконано за даний період. Якщо лінія "виконана робота" зростає швидше, ніж лінія "запланована робота", це означає, що команда виконує роботу швидше, ніж планувалось. Якщо лінія "виконана робота" спадає нижче лінії "запланована робота", це означає, що команда відстає від графіка виконання роботи.

Лінія "обсяг робіт" також може бути відображена на графіку Burnup. Ця лінія показує загальний обсяг робіт, який потрібно виконати протягом проекту. Лінія "обсяг робіт" може допомогти команді зрозуміти, скільки роботи залишилося виконати та який прогрес їх виконання. За допомогою графіка Burnup з лінією "обсяг робіт" команда може більш точно прогнозувати час виконання проекту та зрозуміти, чи їх поточна швидкість роботи є достатньою, щоб завершити проект вчасно.

Burnup chart дозволяє команді зрозуміти, чи вони знаходяться на правильному шляху щодо виконання роботи, чи потрібні якісь корективи в плануванні, та як їх прогрес відповідає запланованій роботі в часі. Він є важливим інструментом для відстеження прогресу проекту та управління ризиками.

Отже, головний принцип роботи з бюджетом, термінами та скоупом — це не брати на себе та на команду більше, ніж ви можете зробити. Тому коли замовник надсилає 30 фіч на пів року розробки, краще запропонувати йому почати з 10 найважливіших і запланувати роботу на перші 2 місяці. Адже 10 завдань відстежити простіше, ніж 30.

Чим більше завдань, тим більше невизначеності, ризику і менше точність в оцінках. Ще 20 фіч можна оцінити лише приблизно, щоб у замовника склалося уявлення про те, скільки загалом займе розробка. Якщо він погодиться, команда працюватиме у комфортному режимі.

Але бувають ситуації, коли клієнт наполягає на нереальних дедлайнах і не йде на переговори. Іноді команда з інших причин не встигає зробити роботу вчасно. Для таких випадків є такі варіанти дій.



Розберімо ці принципи на прикладі.

Завдання. Уяви себе проджектом. Ти робиш фічу «додавання тексту на фотографію» для соцмережі типу Instagram. На розробку виділено два тижні, але замовник наполягає на скороченні терміну до семи днів. Переговори про додаткові бюджети закінчуються безрезультатно, а отже, потрібно думати, як скорочувати час розробки. Спочатку твої вимоги виглядають так:

**Фіча:**Додати можливість писати текст на зображенні.

**Задачі:**

1. Зробити текст жирним

2. Зробити текст курсивним

3. Вибрати один із 20 шрифтів.

4. Вибрати колір тексту

5. Вибрати фон тексту

6. Перемістити текст на екрані pinch жестом

7. Видалити текст

*8. Додати одну з 5 анімацій*

*9. Додати тінь тексту*

*10. Додати прозорість.*

Ти вирішуєш спростити фічу. Останні три пункти звучать не критично. Без анімації, тіні та прозорості фіча працюватиме справно. Якщо їх прибрати, то ти не порушиш цілісність функціональності. Текст все ще можна додавати, змінювати та видаляти. Ти телефонуєщ замовнику і пояснюєш, що потрібно скорочувати скоуп завдань, щоб встигнути до дати. Замовник погоджується, і команда заощаджує три дні розробки. Завдяки скороченню скоупу вони тепер укладаються на тиждень та два дні.

Але цього не достатньо. Щоб стиснути термін до тижня, ти запитує тімліда, які роботи можна розпаралелити. Тімлід пропонує перерозподілити роботу в команді та віддати все, що пов'язано зі стилями, першому розробнику. А другий візьме на себе підключення додаткових шрифтів та логіку додавання та видалення тексту.

## Завдання

Як гадаєш, що ще можна попросити зробити розробників, щоб робота йшла швидше?

###### Відповідь

Розробники можуть прискоритися та писати код низької якості. У майбутньому його буде складніше змінювати, підтримувати та масштабувати. Але якщо бізнесу справді потрібен результат за тиждень, а важлива функціональність не працює — її треба доробити будь-якими способами.

Отже, ти просиш розробників **знизити якість коду та відмовитися від юніт тестів, написавши їх пізніше.** Внаслідок паралельної роботи та спрощеної розробки команда заощаджує ще два дні. В підсумку вони досягають нових термінів і залишається тиждень. Розробку більше утискати не варто, інакше фіча працювати не буде.

До нових умов ти закладав на **тестування** цілий тиждень. Замовник сказав, що фіча йому потрібна для презентації інвесторам. Це означає, що в продакшн вона не піде, і справжніх багів ніхто не побачить. Пам'ятаючи це, ти просиш тестувальників прогнати основний сценарій тестів на стабільному пристрої. Ця задача потребуватиме лише один день. В результаті тестів вони знаходять два баги та виправляють їх ще за день.

Ти заздалегідь домовляється з командою, що до релізу, можливо, доведеться працювати понаднормово і запитуєш про можливості фахівців. Швидше за все, буде залучена не вся команда, а один розробник та один QA. Якщо вони готові на додаткову роботу, ти готуєшся до **овертаймів**.

Таким чином, через тиждень розробки та двох днів тестування замовник отримує фічу з описаним сценарієм стабільної роботи.

На практиці краще не допускати випадків, коли команда регулярно переробляє і намагається виконати всі ідеї замовника в нереальні дедлайни. У нормальному режимі роботи вершини проєктного трикутника доведеться міняти за власним бажанням і лише за гострою потребою.

**2. Визначення бюджету витрат та критичний шлях проекту**

**Бюджет витрат**

***В нашому проекті,*** у Fixed price договорах у тебе фіксований бюджет, а в Time and Material робота оцінюється частинами, з оплатою за фактом виконання. За витратами слід стежити, щоб не виходити за рамки зазначеного у договорі бюджету та вести звітність для замовника. Для цього необхідно орієнтуватися на заплановані години й рахувати витрачені. Без цього ти легко загубишся у витратах.

Щоб було легше відстежувати різницю між оцінкою та фактом робіт, див. Додаток Таблиця, лист. 2 «Бюджет витрат Спринт 1».

*Рекомендації.*

Щоб порахувати вартість роботи кожного фахівця, помнож його ставку на кількість виконаних годин. Наприклад, за тімліда замовник платить $45 за годину. За останній спринт він відпрацював 80 годин. Отже, на його задачі було витрачено 3600 доларів від загального бюджету.

Щоб твої розрахунки були точними, домовся з командою, що кожен день усі учасники будуть вносити відпрацьовані години у багтрекер. Збирай дані по годинах своєї команди та заповнюй таблицю.

Не забувай не лише рахувати витрачений бюджет, а й порівнювати його з вашою початковою оцінкою. Так ти зрозумієш, наскільки команда до неї потрапляє.

В таблиці Додатку, лист. 1 «Velocity» можна провести розрахунок кількості спринтів, які приблизно будуть включені до проекту. ***Нагадаємо***, що ви завершили фічу «Додавання відео» і зробили першу частину фічі «Додавання тексту на відео». В таблиці позначено як *виконано*. Щоб дізнатися Velocity, тобі потрібно порахувати всі сторі поінти, виконані за перший спринт. Май на увазі, що задачі, які зроблені не до кінця, не рахуються. Наприклад, у першому спринті Кері не встиг доробити задачі «експорт», тому її не рахуємо.

## Метод критичного шляху

Якщо потрібна більш точна оцінка термінів проєкту, можна скористатися**методом критичного шляху. (таблиця додатку, лист. «Критичний шлях»)**

Критичний шлях — це послідовність завдань, від яких залежать терміни початку та закінчення всього проєкту. Тільки з їх завершенням проєкт можна вважати виконаним. Задачі на критичному шляху називаються критичними діями. Якщо вони відкладаються, зміщується дата фінішу всього проєкту. Тому критичний шлях визначає загальну тривалість роботи.

Щоб розрахувати, потрібно максимально розпаралелити всі задачі проєкту. Залишай ті задачі, які потрібно виконувати послідовно. В IT це майже завжди розробка. Дизайн, тестування та збір вимог займають менше часу, і їх можна робити паралельно з іншими задачами.



На слайді видно, що паралельно із розробкою першої фічі дизайнер малює UI для другої. А поки розробники роблять другу фічу, команда QA тестує першу. Ні дизайн другої фічі, ні тестування першої в критичний шлях не увійдуть, тому що вони коротші за розробку і виконуватимуться паралельно. Але зверни увагу на послідовність дизайну та розробки для першої фічі та розробки та тестування — для третьої. Нескладно здогадатися, що не можна розпочати розробку першої фічі без дизайну та завершити проєкт без тестування останньої.

Тому, щоб дізнатися про критичний шлях, тобі потрібно порахувати тривалість розробки (включаючи стабілізацію), дизайну та тестування. Додатково заклади час на підготовку, постачання та ризики. Для їх розрахунку можна скористатися коефіцієнтами, про які ми говорили в попередньому розділі.

Таким чином, критичний шлях можна розрахувати за такою формулою, якщо у твоїй команді один розробник:

**(First feature's UI + Entire development + Communications, risks and delivery + Last feature's QA)/8h + Sprint 0**

Якщо розробника два, то розробку і час на комунікацію треба поділити на два. Тоді формула буде такою:

**(First feature's UI + (Entire development + Communications, risks and delivery)/2 + Last feature's QA)/8h + Sprint 0**

А якщо розробника три як у тебе, то коефіцієнт для розробки та комунікації буде 2.7. Їм потрібен час на синхронізацію та передачу знань, тому цифра не 3, а трохи менша.

**(First feature's UI + (Entire development + Communications, risks and delivery)/2.7 + Last feature's QA)/8h + Sprint 0**

Те саме справедливо і для більшої кількості інженерів. Якщо їх 4, то закладай десь 3.6, якщо 5, то 4.5 і так далі. Не забувай, що вся робота відбувається паралельно.

Щоб дізнатися, скільки робочих днів займе створення проєкту, потрібно застосувати формулу:

**(First feature's UI (дизайн інтерфейсу першої функції)+ (Entire development + Communications, risks and delivery)/2.7 + Last feature's QA (контроль якості останньої функції)/8h + Sprint 0**

4+(153.25+41)/2.7+45/8+3=4+71.94+5.625+3=84.565≈ 85

Значення Entire development та Communications, risks and delivery у людино-днях таблиця автоматично розділила на 2.7. Цей коефіцієнт застосовується, тому що у вашій команді 3 розробники.

*Отже,* ми розглянути роботу термінами проєкту двома способами. Перший — Velocity. З ним легко рахувати спринти, що залишилися. Другий метод — це критичний шлях. Він показує точну оцінку та враховує зовнішні залежності, наприклад, коли ви працюєте з іншою командою.