**Тема 6. Оцінка та розподіл задач, планування спринта**

1. **Оцінка та розподіл задач**
2. **Планування спринта**

**1. Оцінка та розподіл задач**

У процесі виконання проекту ми повинні володіти інформацією чи все у нас йде по плану. Для цього ми маємо дати відповідь на питання: Які критерії оцінки? З чим порівнювати? Де узяти той еталон на який орієнтуватися? Яким чином ідентифікувати проблеми і оперативно зреагувати, ще до того як вони стали невиправними? Як зрозуміти, що саме слід покращувати у себе на проекті?

Немає кращого способу аніж обрати відповідний тестовий показник (test metric), визначити його середнє значення та відстежувати прогрес згідно цього показника чи набору показників протягом наступних тижнів, місяців, років.

Отже, якщо коротко підсумувати, **метрики** потрібні, щоб ефективно управляти проектом:

Інформування.

Передбачати різноманітні моменти.

Розуміти що саме потрібно вдосконалювати

Для оптимізації процесів.

Діагностувати проблеми, локалізувати, виправляти і перевіряти їх.

Приймати правильні рішення.

Метрики важливі на усьому циклі розробки програмного забезпечення (SDLC). Проте потрібно аналізувати аспект ефективності застосування кожної метрики з точки зору, на якому етапі розробки проекту від отриманої інформації ми отримаємо максимальний результат.

**Метрика** — іншими словами це показник поточного стану досягнення цілі.

Ефективність і якість роботи команди в роботі над виконанням задач визначається переважно **за годинами, швидкістю виконання та сторі поінтами.**

Володіючи таким набором показників можна порівнювати команду з нею ж самою у різні моменти часу або з іншими командами.

Інша метрика – **реальний час роботи команди**. Її призначення збільшити точність планування, а по-друге, відстежувати і керувати ефективністю роботи команди.

**Метрика додаткових витрат часу** це витрати часу скажімо на мітинги, налаштування середовища, пусте очікування якогось функціоналу який є ще неготовий до загального часу.

Точність оцінки часу по видам, типам робіт =

Оціночний час роботи/Фактичний час роботи

***Для застосування цієї метрики*** використовується відповідний коефіцієнт додаткових витрат час, який буде у нас підстраховочний, коригувальний до наступних наших оцінок, щоб вони були точнішими при плануванні.

В аутсорс-проєктах за договором **Time and Material фічі оцінюються у годинах**. Це потрібно, щоб менеджер міг порахувати час на кожну задачу та скласти інвойс для замовника про особисте навантаження учасників команди.

**Довідково.** З годинами легше керувати завантаженням. Уяви таку ситуацію. У вас є договір Fixed price з виділеним бюджетом. Команда оцінила якусь задачу у 10 годин, а витратила лише 8. Дві години залишилися невикористаними. Ти можеш витратити їх на інші задачі або взагалі не використовувати й принести додатковий прибуток компанії — адже замовник уже виділив бюджет на 10 годин. Зі сторі поінтами такі нюанси не вирахуєш.

Години потрібні, щоб точніше оцінити час на розробку. Адже в сторі поінтах не оціниш кожну підзадачу у фічі. А от у годинах можна. Сторі поінти виставляє вся команда, а про години потрібно запитувати тільки того, хто виконуватиме підзадачі. У СП оцінку дають і розробники, і дизайнери, і тестувальники це такий загальний орієнтир для всіх. А в годинах може оцінити роботи лише виконавець.

## Оцінка по сторі поінтах

Сторі поінт — це метод відносної оцінки в методології Agile. З ним команда оцінює фічі щодо умовної шкали — сторі поінта.

Методика визначення stоry point проекту полягає у наступному:

* за основу приймається найпростіший елемент backlog, який становить еталонною одиницею;
* всі інші елементи backlog порівнюються з визначеною еталонною одиницею.

Одиниця виміру stоry point включає: трудомісткість робіт; складність робіт та можливі ризики при реалізації елемента backlog.

При оцінці stоry point також обирають шкалу:

* степінь 22: 1, 2, 4, 8, 16, 32, 64, 128.;
* ряд Фібоначі (Ряд Фібоначі - це послідовність чисел, де кожне наступне число утворюється шляхом додавання двох попередніх. Таким чином, перші два числа в ряду завжди будуть 0 і 1, а кожне наступне число буде дорівнювати сумі двох попередніх чисел.): 0,1 2, 3 ,5 ,8, 13, 21 і т.д.;
* модифікований ряд Фібоначі: 0, 0,5, 1, 2, 3, 5, 8, 13, 20, 40, 100;
* шкала bucket estimation XS, S, M, L, XL.

Наприклад, backlog в 2 stоry point означає в 2 рази більше витрат, ніж в тому, що має 1 stоry point. У кожної задачі є таке поле.

Stоry point використовуються для допомоги команді вирішити обсяг задач, що може бути завершено в спринті.



Техніки для оцінки stоry point:

1. **Покер планування:**
* кожен учасник команди розробки повинен мати свій набір карт зі змістом шкали;
* визначається еталонна stоry point та присвоюється значення 1;
* кожен член команди оцінює розмір кожної stоry point без обговорення з іншими, порівнюючи з еталоном або іншими stоry point, раніше оціненими;
* обирає карту з оцінкою і відкладає в сторону;
* в кінці оцінки всіх stоry point усіма учасниками команди карти відкриваються та обирається середня оцінка за умови, що оцінки не мають значного відхилення (3…5…8);
* якщо оцінки мають великі відхилення, то визначається їх min та max значення та обґрунтовується учасником його оцінка по відношенню до min та max значення (1 min 5, 5 …20 max).

Такий вид оцінки підходить для оцінки нових stоry point в backlog в Jira. Недолік – займає багато часу на визначення оцінки.



 2. Метод Bulk estimation (насипна оцінка):

* на картках (стікерах) виписуються шкали, наприклад Фібоначі;
* команда розподіляє backlog по цим карткам в стовпчик, тобто насипом на шкалу;
* при відхиленнях в оцінках по певним елементам можна застосувати покер планування.



Інші види:





**Довідково**. Статистично, чим більша оцінка задачі на початку (до прикладу 5, 8 або 13) - тим більше фактичного часу вона займе у підсумку. Чим більша оцінка, тим більше невизначеності та неточності. У свою чергу це привело до командного розуміння, що будь-яку задачу з оцінкою понад 2 SP треба розбивати на менші - і тоді точність оцінки фактично збільшується від 20 до 50%.

Завдання. Для Filmy потрібно оцінити підзадачі з відтворення відео з фічі **«Обрізка відео» у годинах**.

Щоб дізнатися години, ти звернешся до розробників. Якщо програміст, якому дістанеться задача, оцінив роботу в 12 год, то бери це значення. До цієї роботи також можна долучити тімліда.

Спочатку оцініть скільки задача займе за часом, і напиши оцінку в годинах. А потім попроси всю команду оцінити цю задачу у сторі поінтах.

**Отже визначаємо роботу в сторі поінтах.**

На початку проєкту команда обирає еталонний сторі поінт. Ви знаходите «ідеальну» задачу — ту, за якою можна виміряти вагу інших. Як правило, вона буде найменшою або найлегшою в беклозі. Її вага має бути зрозумілою всій команді. Оцінки іншим задачам ви виставите, орієнтуючись на цей зразок.

Спочатку вибираємо еталонний сторі поінт. Отже, командою вирішено, що є всім зрозуміла фіча «Публікація в соцмережах», однак є кілька соцмереж. Ваше рішення

**Дробити фічі на менші юзер сторі.** Розбивай фічу на юзер сторі по 1-4 дні розробки, залежно від кількості розробників. Наприклад, на вашому першому грумінгу ви оцінювали фічу «Публікація в соцмережі». У такому формулюванні задача звучить абстрактно. Програміст не зрозуміє, який контент і в які соцмережі публікувати.

Якщо поставити задачу як «**Зробити публікацію в соцмережі**», то команда скаже, що на її виконання піде тиждень або більше. Ця оцінка буде приблизною та нереалістичною. Але фічу можна поділити, і тоді невеликі задачі оцінювати буде набагато легше. Ви так і вчинили, розбивши весь процес публікації на юзер сторі:

 «Публікація в YouTube»,

«Публікація в Instagram»,

«Публікація в TikTok»

 «Публікація в Vimeo».

Пропонується взяти фічу «Публікація відео на YouTube» - 1 сторі поінт. Має бути зрозумілою усім.

Тепер оцінімо інші фічі, виходячи з нашого еталонного сторі поінта.

У вашому скоупі є дві різні фічі ( див. таблицю Додаток):

**1. вибір розміру кадру.** Перша фіча відносно нескладна — вибрати та застосувати розмір кадру. Її можна оцінити всього в 3 сторі поінта.

**2. Picture in Picture.** А ось технологія Picture in Picture несе значні ризики. У цій фічі набагато більше підзадач, і ніхто з твоєї команди з ними раніше не стикався. Її оцінять вже у 13 сторі поінтів.

Сторі поінти використовують не просто, щоб оцінити складність завдань. Вони допомагають відстежувати їхнє виконання. Розберімося, як це робити.

Перед стартом кожного спринту команда планує виконати певну кількість сторі поінтів. Цю початкову оцінку ви потім порівняєте з тим, що встигли зробити за фактом.

Кількість сторі поінтів, які команда встигає виконати за спринт, це Velocity. Ця метрика показує швидкість команди та допомагає планувати завантаження на майбутні спринти.

**Швидкість роботи (velocity) команди** –  розраховується як відношення реалізованих story points (або вимог, або user stories) за кілька, наприклад, 4-5 ітерацій (Sprint) до кількості обраних ітерацій.

Призначення метрики виявити можливості роботи команди для подальшого планування обсягу робіт і аналізу трендів її розвитку. Також метрика дозволяє стежити за швидкістю роботи , спостерігати за тим, які внутрішні процеси чи зовнішні впливи на команду можуть на цю швидкість вплинути.

Розрахуємо Velocity = Кількість story points за N ітерацій/N

Для цього проджекти використовують [Velocity chart](https://support.atlassian.com/jira-software-cloud/docs/view-and-understand-the-velocity-chart/). У Jira він виглядає так.



Сірі стовпці — це кількість сторі поінтів, які команда планує на кожен спринт. Зелені стовпці — це завершені сторі поінти. В ідеалі обидва стовпці у спринті мають бути однакової висоти.

Velocity команди завжди буде змінюватися від спринту до спринту. Згодом показник стабілізується та буде приблизно однаковим кожну ітерацію.

Цю метрику можна використати ще й для **прогнозування**. Просто розрахуй середнє значення Velocity кількох останніх спринтів, коли у команди вже накопичилася значуща статистика продуктивності. Отримане число буде орієнтиром наступного спринту.

В таблиці Додатку до проекту бачимо перелік функцій додатку та визначені командою сторі поінти. Планується, що за результатами робота команди за перший спринт буде виконано 8 сторі поінт.

Важливо прагнути виконувати всі сторі поінти, заплановані на спринт. Так, проджект бачить, наскільки початкова оцінка відрізняється від реальних можливостей команди.

Але в СП зручно рахувати не всім, тому проджекти часто поєднують їх з оцінкою в годинах.

Сторі поінти корисні, щоб стежити за прогресом у тривалій перспективі. А години потрібні, щоб вести звітність для замовника та уточнювати час на виконання завдань з кожним учасником команди окремо.



Отже, підся грумінгу з командою було визначено такі параметри плану по годинах на виконання робіт (таблиця Додатку до проекту).

В таблиці уже внесені задачі на найближчі спринти. Вона формується на основі описаних юзер сторі. Кожна фіча – це окрема юзер сторі. Її зазвичай складають менеджер та тімлід, а на зустрічі команда вносить коригування.

У годинах було оцінено час на розробку, дизайн та тести. Решту роботи у проєкті можна розрахувати за такими коефіцієнтами.



Якщо поставлено завдання. Розрахувати час для функції "Рисунку в рисунку" (на основі даних додатку), то як це зробити. У Вас нема досвіду і Ви звертаєтесь до наставника.

Наставник:

Ці цифри можна закладати на початку проєкту, коли не можна точно визначити, скільки часу займе та чи інша робота. Май на увазі, що цифри зразкові й змінюватимуться від проєкту до проєкту.

У майбутньому ти орієнтуватимешся не на них, а на специфіку свого проєкту. Наприклад, якщо у вас багато коду і ви не проводите [юніт тести](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true), то коефіцієнт **стабілізації** буде вищим.

*Довідково.* Стабілізація системи - це процес контролю та підтримки сталості або збереження певного стану системи, коли вона піддається зовнішнім впливам або змінам внутрішніх параметрів.

У неперевіреній системі з безліччю залежностей у коді буде більше багів. Тому команді потрібно брати додатковий час на налагодження програми.

А чим більше в системі багів, тим довше команда QA шукатиме помилки, тобто займатиметься **тестуванням**. Без нього в кінцевий продукт можуть проникнути серйозні помилки. Це критичний недогляд, якщо в системі багато користувачів, що платять.

**Час тестування** залежить від кількості пристроїв, на яких ви перевіряєте програму. Додаток для відеодзвінків можна запустити лише на одній з версій iOS і переконатися, що багів немає, картинка працює і звук йде. А можна піти далі й залучити три різні пристрої. Швидше за все, знайдуться помилки: десь гальмує відео, а десь запізнюється звук.

У формулі вище на роботу **менеджера** закладається 0.15 від часу на розробку, але це число може змінюватись в залежності від специфіки проєкту. Згодом цей коефіцієнт змінюватиметься. Ти визначиш, що на задачі тобі потрібно більше, або, навпаки, менше часу. Орієнтуйся на 0.15, але підлаштовуй його під себе.

**Комунікація** залежить від кількості людей у ​​команді. Чим їх більше, тим довше ви будете домовлятися та обговорюватимете задачі.

**Наступний норматив. Постачання готового коду**. Коефіцієнт [постачання](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true) залежить від частоти релізів. Наприклад, Amazon та Netflix оновлюються [по тисячі разів на день](https://techbeacon.com/devops/10-companies-killing-it-devops). У цих проєктах працює багато команд. Кожна відповідає за свій шматочок сервісу, пише його, тестує й одразу постачає користувачеві. У невеликих компаніях постачання можуть відбуватися щомісяця і навіть рідше.

**Ризики** залежать від складності проєкту. Якщо команда розробляє інтернет-магазин, можна сміливо брати готові рішення на [WordPress](https://wordpress.com/) або [Magento](https://magento.com/). Інтернет-магазинів багато: всі проблеми, які можна було вирішити, вже кимось вирішені.

А ось якщо ти робиш сервіс для керування безпілотним автомобілем, то зіткнешся з великою кількістю запитань. Це нова сфера, там немає усталених рішень. Вони ще не підтверджені часом та чужим досвідом. Відповідно, і ризиків у такому проєкті буде більше.

Отже, ми визначилися з оцінками у годинах (план) та в сторі поінтах. Але, все може змінитися,. Вже наприкінці першого спринту зрозуміємо, наскільки реалістично оцінюємо задачі.

**2. Планування спринту**,

**Планування спринту** — це зустріч, з якої починається будь-яка ітерація. На ній ви обговорюєте, що увійде в беклог майбутніх двох тижнів, хто і за яку задачу візьметься і який має бути результат.

Спочатку грумінг, а потім планування зрозумілих для всіх завдань.

Отже, командою визначено, що до першої ітерації включити:

1. «Вибір відео з галереї»: візьмемо всі задачі з додавання відео.
2. «Додавання тексту»: можливість додавати, переміщати та видаляти написане.
3. Також можна взяти кілька завдань з експорт: спробуємо найбільший процес експорту.

Уже першому демо зможемо показати щось цікаве.

Завдання. Порахуй скільки це годин і сторі поінтів.

Для виконання цього завдання ти звертаєшся до наставника.

Наствник

**1. Розбивай юзер сторі на підзадачі.** Ці підзадачі повинні укладатися в 4-12 години розробки. Якщо звернемося до прикладу зі збереженням відео в хмарне сховище з твого скоупа, то недостатньо просто сказати програмісту: «Розроби хмарне сховище». Адже у хмарі є свій інтерфейс, зберігаються дані користувачів, є механіка завантаження відео та логіка доступу до нього з різних девайсів. Там є бекенд, фронтенд, і все це потрібно тестувати.

Завжди розбивай фічу на підзадачі за типами робіт: UI, дослідження, юніт тести, UI-тести тощо. На кожну таку задачу відводь максимум півтора дня. Якщо не вдається дробити на такі періоди, повертайся до пункту «**Дробити фічі на менші юзер сторі»**. Тоді твоя команда матиме детальну картину того, що потрібно зробити. Будуть добре проглядатись етапи роботи і в яку точку треба прийти. А якщо ні, то на грумінгу ви забудете про тести, дизайн і дасте нереалістичну оцінку. А вже в процесі розробки витратите більше часу на додаткове уточнення задач та переписування коду.

**2. Роби прототипи та дослідження для складних фіч.** Якщо програмісти не розуміють, як створити комплексну фічу, можна спочатку зробити прототип. Замість вести розробку від і до, команда створить спрощену версію фічі, протестує її і зрозуміє, куди рухатися далі.

Ще можна закласти додатковий час вивчення фічі. Наприклад, якщо у вашій команді ніхто не робив Picture in Picture. досвідчений розробник, відразу порадив взяти 8 годин на її вивчення, тому що багато чого в цій фічі може піти не так.

**3. Закладай середню розбіжність у майбутні оцінки.** Як ми вже говорили, розбіжність у 20% із початковою оцінкою вважається нормою.

Іноді причина розбіжності факту з прогнозами у загальних процесах. Наприклад, менеджер погано описує вимоги та робить недостатньо докладну декомпозицію задачі. Або фіча нова та здається незрозумілою команді, як це сталося з Picture in Picture. Ці проблеми стосуються всіх, тому винось їхнє обговорення на ретро.

Отже:

До першого спринту увійшли **11 завдань**, і всі успішно розподілені між розробниками. (Додаток).

Командою було визначено такий порядок завдань:



Наставник:

Таке співвідношення робіт вважається середнім по індустрії — завжди орієнтуйся саме на свою ситуацію.

*У перших ітераціях* краще сфокусуватись на поточних задачах.

*На другий та третій спринт* спринт віддай більше часу під нові фічі, доопрацювання старих (+45% часу) та Research and Development (+10%).

Зі стабілізацією, рефакторингом та багфіксом починай щільніше працювати приблизно *з п'ятого спринту*, коли в тебе з'явиться перший техборг.

При розподілі завдань можна скористатися такими пропорціями.



**45%** — нові фічі та доробки за старими фічами. До цього пункту входить функціональна розробка, стабілізація та багфікс.

**15%** — Багфікс: старі помилки програми, які не увійшли до попереднього пункту. Помилки у нових фічах [рекомендується](https://t.me/pm_god/88) закладати у попередні 45% та писати [DoD (Definition of Done)](https://www.notion.so/aquarelleyep/DOD-a3546748dda64fc3b17728ba5dfa22b1)[[1]](#footnote-1), щоб їх уникати

**15%** — [рефакторинг](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true): видалення та зміна старого коду, борги по тестах, [CI (Continuous Integration)](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true)[[2]](#footnote-2).

**10%** — дослідження нових фіч та технологій — Research and Development (R&D). Сюди входить перевірка гіпотез, прототипи фіч, [proof of concept](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true) (перевірка концепції та демонстрація) для клієнтів, проба нових бібліотек та фреймворків. На відміну від попереднього пункту, ці роботи спрямовані не на наявний, а на майбутній код.

**15%** — запас на випадок непередбачених завдань.

Обов'язково зважай на зовнішні фактори, щоб не зрушувати терміни проєкту. Продумай, що ви з командою робитимете, якщо хтось захворів або якщо більшу частину спринту зайняли національні свята. Ти не зможеш поставити одразу всю команду на одну фічу — навантаження потрібно розподілити.



Розподіл завдань у спринті:

***Довідково.*** *У спринті 2 тижні, тобто усі 80 годин на 1 людину. До першого спринту увійшли 11 завдань та 8 сторі поінт.*

*У кожного розробника від 20 до 40 годин навантаження. І це тільки розробка. Але після написання коду буде ще й етап тестування, а отже, і фікс багів. Також у спринті є стендапи, ретро, ​​демо, планінг та грумінг. На них теж йде час. Тому зазвичай «чистої» роботи планують приблизно на 50% часу.*

Завдання. Визначи бюджет часу на 1 спринт 2 тижні, якщо у команді 3 розробника по повну ставку і 1 тестувальний на 1,25 ставки.

З першого разу навряд чи вдасться розподілити задачі ідеально, і це нормально. Зазвичай проджект займається цим не один, а разом із тімлідом. Менеджер краще знає пріоритети у продукті та залежності у проєкті. А тімлід розуміє, на що здатні програмісти у його команді.

Припустимо, всі пропорції спринту враховані, а задачі акуратно розподілені між розробниками. Але навіть при таких вхідних готуйся, що ви можете не встигнути виконати всі заплановані сторі поінти.

Нічого страшного. В індустрії вважається, що похибка 30% при попаданні в початкову оцінку — це нормально. Особливо для команд, які тільки-но почали працювати над проєктом.

Тож у новому проєкті закладай більше часу на ризики. Поступово, через 3-6 місяців, ви освоїтеся і можна буде встановити орієнтир похибки в оцінці — 10-15%.

Тепер, коли ти розумієшся на експертизі своєї команди, тобі буде легше розподіляти навантаження і вибудовувати очікування від роботи. Не забувай, що планувати спринт потрібно з тімлідом, який підкаже, як роздати задачі.

## Резюмуємо

* Декомпозицію фічі потрібно робити так, щоб команді не доводилося витрачати додатковий час на уточнення вимог або розбивати задачі на дрібніші. З цим допоможе тімлід.
* Оцінка завдань проводиться в сторі поінтах та годинах. Сторі поінти допомагають відстежувати прогрес від спринту до спринту. Години потрібні, щоб складати звітність для замовника та домовлятися з кожним спеціалістом про навантаження окремо.
* Velocity — це кількість сторі поінтів, які команда виконує за один спринт. Ця метрика допомагає прогнозувати завантаження на наступні спринти, а також слідкувати за прогресом у роботі.
* Грумінг — це зустріч із командою, на якій ви оцінюєте задачі для найближчих спринтів.
* Планування спринту — це вибір завдань на найближчу ітерацію та призначення на них фахівців.
* Грумінг та планування спринту краще проводити різними зустрічами. Це потрібно, щоб на грумінгу ви чітко визначилися із задачами, а на плануванні спринту лише розподіляли їх між учасниками.
* В аутсорсі досвідчені проджекти оцінюють проєкт ще до його продажу клієнту. Ця процедура проводиться разом з менеджером з продажу і включає оцінку термінів і бюджетів проєкту.
1. DoD (Definition of Done) — це критерії завершення задачі. Вони визначають, що потрібно зробити, щоб задачу можно було вважати виконаною. [↑](#footnote-ref-1)
2. **Неперервна інтеграція** ([англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *Continuous Integration*) — практика [розробки програмного забезпечення](https://uk.wikipedia.org/wiki/%D0%A0%D0%BE%D0%B7%D1%80%D0%BE%D0%B1%D0%BA%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F), яка полягає у виконанні частих автоматизованих складань проєкту для якнайшвидшого виявлення та вирішення інтеграційних проблем. [↑](#footnote-ref-2)