**Тема 10. Якість, тестування і випуск продукту**

1. **Планування та оцінка якості продукту**
2. **Тестування**
3. **Захід Демо**
4. **Перший реліз**
5. **Планування та оцінка якості продукту**

**З чого складається якість проекту?**

За [чек-листом](https://docs.google.com/document/d/1XM1TzqXHiR5mrFbnbtR8-BOBqjimWBX6xSW6PJJEwtk/edit?usp=sharing), що наведено нижче ти легко визначиш якість у своїх проєктах — нових чи поточних. Ти, як прожект, обговорюй кожен із пунктів із командою, щоб знаходити проблеми та покращувати процеси.

Чек-лист - це список пунктів або завдань, які потрібно виконати або перевірити в певному процесі, проекті, події або діяльності.

**Чек-лист на визначення якості продукту**

1. **Покриття проекту юніт та авто тестами**

Відсоток покриття має становити від 60 до 90%.

**2. Часті релізи.** Основна мета частого випуску нових версій полягає в тому, щоб забезпечити швидкий цикл зворотного зв'язку з користувачами, швидко реагувати на змінні вимоги ринку, вносити виправлення помилок та вдосконалення в програмне забезпечення та забезпечувати більш гнучкий та швидкий процес розробки.

*Орієнтир якого потрібно прагнути — кілька разів на день. Рідше за один раз на місяць — потенційна проблема.*

**Визначаємо частоту релізів**. Май на увазі, що реліз не завжди означає, що користувачі побачать зміни. Можна залити код, але зробити так, що він не буде доступним. Частота релізів дуже залежить від проєкту. В ідеалі постачання проводять кілька разів на день, хтось — раз на тиждень. Але якщо рідше одного разу на місяць — то це вже потенційна проблема.

Май на увазі, що у Filmy немає частих постачань. Ваш перший реліз станеться, як тільки ви розробите всю основну функціональність.

## Завдання. Як думаєш, чому релізи краще проводити якомога частіше?

###### Відповідь автора

Чим рідше команда випускає код у продакшн, тим більше його накопичується. А **що більше коду — то вища ймовірність, що він зламається при релізі.** Уяви, що ви розробляєте месенджер, і за останній місяць встигли додати туди голосовий чат та можливість переказувати гроші. Ви робите реліз і в саппорт (служба підтримки) починають сипатися баги. Ці дві нові фічі пошкодили старий функціонал. У результаті інженери не змогли швидко полагодити помилки, і реліз довелося відкласти. Команда витратила час і засмутила своїх користувачів. Якби ви впроваджували переказ грошей та голосовий чат шматочками або хоча б окремо, то, швидше за все, нічого не зламали б. А якби щось пішло не так, то на виправлення пішло б менше часу — ви ж випустили не так багато коду і розумієте, яка саме частина призвела до нової помилки.

Отже, чим менше коду ми випускаємо, тим менше нових залежностей утворюється, і тим менше багів виявляється у продакшені. А якщо їх таки виявили користувачі, то команда швидко все перегляне та виправить. Адже знайти помилку в маленькому шматочку легше, ніж у цілому простирадлі коду. І навпаки, якщо ми випустили відразу кілька фіч, складно уявити, як система поводитиметься у продакшені. Можливо, вона взагалі впаде.

**3. Швидке постачання нового коду.** Основна ідея "швидкого постачання нового коду" полягає в тому, що розробники регулярно вносять невеликі зміни в код та випускають їх в продакшн (в робоче середовище або до кінцевих користувачів) якомога швидше, зазвичай за допомогою автоматизованих процесів розгортання (deployment) та тестування (testing). Це відмінюється від традиційного підходу, коли великі зміни в коді можуть відкладатися на довгий час, наприклад, до випуску нової версії програмного забезпечення.

*Твій орієнтир - 5-10 хвилин на постачання. 60 хвилин – найвища границя.*

**4. Малий відсоток білдів, що впали.** "Малий відсоток білдів, що впали" відноситься до відсоткового співвідношення між загальною кількістю білдів (побудованих версій) програмного забезпечення та кількістю білдів, які завершилися неуспішно або "впали", тобто не пройшли випробування, тестування або автоматичну перевірку якості.

*10% - це норма. Якщо більше, то потенційна проблема.*

Наставник.

2. **Стеж за** **відсотком** [білдів](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true)**, що падають**. 10% — це оптимальний орієнтир, але якщо ваша цифра вища, то це привід розбиратися і щось міняти. Чим частіше падають білди, тим повільніше нові фічі та баги виявляються у тестувальників, і тим довше команда закриває задачі. Адже замість роботи над фічами програмісти витрачатимуть час на налагодження [системи збірки](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true) (за допомогою програмного забезпечення, яке автоматично збирає версію програми (білд) із вихідного коду, тобто перетворює його на продукт, яким можна користуватися або тестувати).

**5. Мінімальна кількість інцидентів на продакшені та звернень на підтримку**

*Чим рідше пишуть засмучені користувачі, тим краще йдуть справи у бізнесу.*

**3. Стеж за кількістю** [**інцидентів**](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true)**,** тобто багів у продакшені, які приели систему до неробочого стану або порушив її функціонування  **у продакшені**.Це баги, через які додаток або працює не так, або перестає працювати зовсім. І, звичайно, кількість таких випадків — важлива метрика якості.

## Завдання. Як думаєш, чому кількість інцидентів важливо враховувати при оцінці якості продукту?

Якщо багів багато і вони помітні, тобто перетворюються на інциденти, почнуть писати засмучені користувачі. *Наприклад*, ми випустили банківську програму, а там не можна переказувати гроші на іншу картку. Це основна фіча. Якщо вона не буде працювати, користувачам більше нема чого з нами залишатися. Посиплеться купа скарг, а це вже багато говорить про нашу якість.

**6. Високе значення Uptime.** Uptime (час безвідмовної роботи) відноситься до періоду, протягом якого система, сервіс або програмне забезпечення функціонує без перерви, тобто є доступним та працездатним для користувачів або клієнтів. Uptime вимірюється відсотками та використовується для оцінки надійності технічних систем.

*Орієнтуйся на те, що програма має бути доступна від 99% часу на місяць.*

**4. Стеж за метрикою** [Uptime](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true) (**ц**е метрика, що показує відсоток часу, коли система доступна. Вона протилежна даунтайму часу, коли продукт не працює). Це кількість часу, коли програма доступна. Можливо, тобі знайома ситуація, коли хочеш сплатити за покупку в інтернет-магазині, заходиш на сайт, але бачиш екран із повідомленням, що система обслуговується.Це означає, що стався [аутедж](https://docs.google.com/document/d/137fhxEOWa_tTR8z5SJpWSjFcWYCPx3w4UE7dgJFo_nc/edit?usp=sharing) — тобто система впала. Часті аутеджі загрожують бізнесу втратою користувачів та виторгу. Тому компанії орієнтуються на 99% та більше аптайму на місяць. Здається, що це багато, але якщо перевести в години, то 1% аутеджу — це понад 7 годин на місяць. Мінус цілий робочий день!

**7. Мінімальна кількість багів у беклогу та невеликий розмір техборгу**

*Чим менше у проекті помилок, тим легше його розвивати та підтримувати.*

**5. Стеж за** **кількістю багів у беклозі та розміром техборгу.** Але так вийшло, що немає універсальної формули, щоби виміряти цю метрику. Справді буває, що в проєкті 200 багів, але жоден з них не помітний користувачам і не впливає на роботу продукту. Ви можете залишити їх у беклозі й спокійно працювати далі. А може й навпаки. Уяви, що в музичному сервісі не завантажується вікно оплати на 40% всіх доступних девайсів. Програма втрачає майже половину користувачів, що платять. Начебто баг один, але так сильно б'є по бізнесу.

**Питання. Як ти думаєш для чого нам потрібно класифікувати баги?**

**Довідково. Класифікація багів.**

*За ступенем критичності бувають баги:*

* блокуючі (blocker), що роблять неможливою подальшу роботу з додатком. Наприклад, функціонал сайту доступний тільки зареєстрованим користувачам, а лист для підтвердження реєстрації на e-mail не приходить;
* важливі (major), через які система не функціонує належним чином. Наприклад, завантажений аватар не зберігається на сервері;
* нормальні (normal), найбільш рідко зустрічаються (наприклад, непрацююча кнопка «Запам'ятати мене»);
* незначні (minor) або невеликі баги, наприклад, помилки верстки на якійсь сторінці в адмінці тощо. Такі баги виправляються не по одному, а після того, як буде зібрано кілька десятків або тисяч таких помилок.

*За пріоритетністю виділяють баги:*

* fix in release – виправити в новій версії продукту. Як правило, це баги, виявлені при тестуванні нового функціоналу системи;
* must fix – терміново виправити. Часто це блокуючі баги, які усувають до виходу нової версії в спеціальному сервіс паку;
* fix if time – виправити, якщо дозволяє час;
* never fix – ніколи не виправляти, наприклад, баги знайдені у продукті, який вже не підтримується.

***Технічний борг*** *також відомий як борг проектування або борг код.* **Загальними причинами технічного боргу є (комбінація):**

* Тиск бізнесу, коли бізнес розглядає запуск продукту якомога раніше, до того, як усі необхідні зміни зроблено, та нагромаджує технічний борг з цих незавершених змін.
* Брак розуміння або процесів, коли бізнес є сліпим до концепції технічного боргу та приймає рішення без врахування можливих наслідків.
* Брак створення малозалежних компонентів, коли функції не є [модульними](https://uk.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F), через що проект є недостатньо гнучким для адаптації до змін бізнесових потреб.
* Брак набору тестів, що заохочує швидке та ризиковане латання [багів](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%B3).
* Брак документації, коли код створюють без необхідної супровідної документації. Час на створення такої документації є боргом, який необхідно буде погасити.
* Брак співпраці, коли знання не поширюють в межах організації і ефективність бізнесу страждає, або коли недостатньо курують молодших розробників.
* Паралельне програмування водночас на двох або більше гілках може призвести до накопичення технічного боргу через зусилля, які колись необхідно буде потратити на злиття змін у єдину кодову базу. Що більше змін відбувається в ізоляції, тим більшим в результаті є борг.
* Відтермінування рефакторингу — В той час як проектні вимоги розвиваються, може стати очевидним, що місцями код став громіздким та мусить бути порефактореним задля відповідності до майбутніх вимог. Чим довше цей рефакторинг відкладають на потім, і що більше коду пишуть в рамках поточного вигляду, тим більше назбирується боргу, який необхідно буде погасити до завершення рефакторингу.
* Брак відповідності до стандартів, коли ігнорують стандартні індустрійні особливості, фреймворки, технології. Одного разу настане час інтеграції зі стандартами, і зробити це раніше буде дешевше (подібно до «відтермінування рефакторингу»).
* Брак знань, коли розробник просто не знає, як писати елегантний код.
* Брак володіння, коли програмні напрацювання, [замовлені ззовні](https://uk.wikipedia.org/wiki/%D0%90%D1%83%D1%82%D1%81%D0%BE%D1%80%D1%81%D0%B8%D0%BD%D0%B3), доводиться [рефакторити](https://uk.wikipedia.org/wiki/%D0%A0%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D0%BE%D1%80%D0%B8%D0%BD%D0%B3) чи переписувати на місці власними інженерами.
* Погане технічне керівництво з недостатньо продуманими командами, що поступають вниз ланцюгом команд, радше підвищує, а не знижує, технічний борг.
* Зміни специфікації в останній момент; мають потенціал до проникнення протягом всього проекту, без достатнього часу чи бюджету на впровадження в документацію та перевірку.
* Скоуп Допінг (з англ. Scope Doping), неконтрольоване зменшення обсягу робіт чи їх відкладання проектними менеджерами або технічними працівниками. Явище трохи відмінне від Тиску бізнесу в тому, що, хоч ТБ може бути причиною, СД також спричинить тиск бізнесу внаслідок неконтрольованого обсягу робіт, який було відтерміновано.

**8. Ваша документація у порядку**

Повинні бути коментарі до коду, опис API та детально описана логіка роботи фіч (специфікація), яка може бути відображена у вигляді плану, діаграми потоку даних, блок-схеми, псевдокоду, коду програми або інших форматів, залежно від рівня абстракції та потреб проекту.

Довідково. Опис API (Application Programming Interface) відноситься до документації, яка надає вичерпну інформацію про доступні функції, методи, параметри, формати даних та інші деталі взаємодії між різними програмними компонентами чи сервісами. API описує правила, за якими різні програми можуть взаємодіяти між собою, дозволяє одному програмному забезпеченню використовувати функції іншого, або забезпечує можливість взаємодії зовнішнім розробникам з додатків сторонніх розробників.

**9. Команда проводить код ревью**. Код ревью (code review) - це процес огляду і аналізу програмного коду розробленого розробником, з метою перевірки його якості, ефективності, безпеки, згідно внутрішніх стандартів розробки, кращих практик програмування та вимог проекту. Код ревью може проводитися колегами розробника, які мають відповідні знання та досвід у відповідній технології або проекті.

*Деякі баги навіть не з'являться, якщо програмісти перевіряють роботу один одного, перш ніж її випустити.*

**6. Проводь код ревʼю (**це перевірка коду на помилки та стиль. Код рев’ю проводиться усередині команди. Рев'ювер залишає коментарі, в яких пропонує зробити зміни у коді. Якщо помилок немає, код можна передавати до тестування**)** Це дуже корисна практика. Деякі баги навіть не з'являться, якщо розробники ретельно перевірятимуть код один одного, перш ніж його випускати. Шукають баги, лазівки у безпеці, помилки у стилістиці та оформленні. Якщо правок багато, то код переписується. А якщо коментарів немає, можна передавати тестувальникам.

**10. Команда проводить перформанс тестування та аудит безпеки**

Довідково. Перформанс тестування (performance testing) - це вид тестування програмного забезпечення, який спрямований на оцінку продуктивності, ефективності та стійкості додатка або системи в умовах реального навантаження.

Перформанс тестування може включати різні види тестів, такі як:

*Навантажувальне тестування* (load testing) - оцінка реакції системи на велике навантаження, для виявлення максимальної продуктивності системи та виявлення можливих проблем з відповідністю системи запитам в режимі реального часу.

*Стрес-тестування* (stress testing) - випробування системи на межах її можливостей, з метою виявлення недоліків, таких як витоки ресурсів, падіння системи, втрата даних, та інших проблем, що можуть виникнути при великому навантаженні.

*Продуктивність* в реальних умовах (endurance testing) - оцінка стабільності та продуктивності системи при тривалому використанні в реальних умовах, з метою виявлення можливих проблем, пов'язаних з накопиченням ресурсів, втратою продуктивності з часом та ін.

*Конфігураційне* тестування (configuration testing) - випробування системи при різних конфігураціях, з метою виявлення можливих проблем, пов'язаних з різними налаштуваннями системи, апаратними або програмними параметрами, операційними системами, мережевими налаштуваннями та іншими факторами конфігурації.

*Оптимізаційне* тестування (optimization testing) - виявлення можливостей для оптимізації продукту, таких як швидкодія, ефективність використання ресурсів, оптимальне використання алгоритмів та структур даних, з метою покращення продуктивності та ефективності системи.

*Масштабованість* тестування (scalability testing) - оцінка здатності системи розширюватись та пристосовуватись до збільшення обсягу даних, користувачів, транзакцій або навантаження в майбутньому, з метою виявлення можливих обмежень та проблем масштабованості.

**7. Стеж за нефункціональними вимогами.** QA проводять [тестування продуктивності та тестування безпеки](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true). Для звичайного проєкту такі тести варто робити раз на один або два місяці. А для систем, де важливі перформанс та безпека, їх взагалі краще включити до DoD. Переконайтеся, що команда дійсно користується DoR та DoD, і що ви перевіряєте за цими критеріями всі ваші фічі.

*Для проектів з високими вимогами до перформансу та безпеки ці тести мають бути включені до DoD. Для інших проектів їх можна проводити раз на 1-2 місяці.*

**Команда користується DoR та DoD**

Ви перевіряєте всі фічі за критеріями готовності до розробки та зробленості.

**DoD та DoR.** Definition of Done і Definition of Ready

Щоб не трапилося так, що ви спроєктували, розробили, протестували фічу, але на демо все одно чогось не вистачило, команда має працювати за критеріями готовності.

Кожен спеціаліст має своє розуміння готовності фічі.

Розробник вважає, що «готово» це коли написаний код.

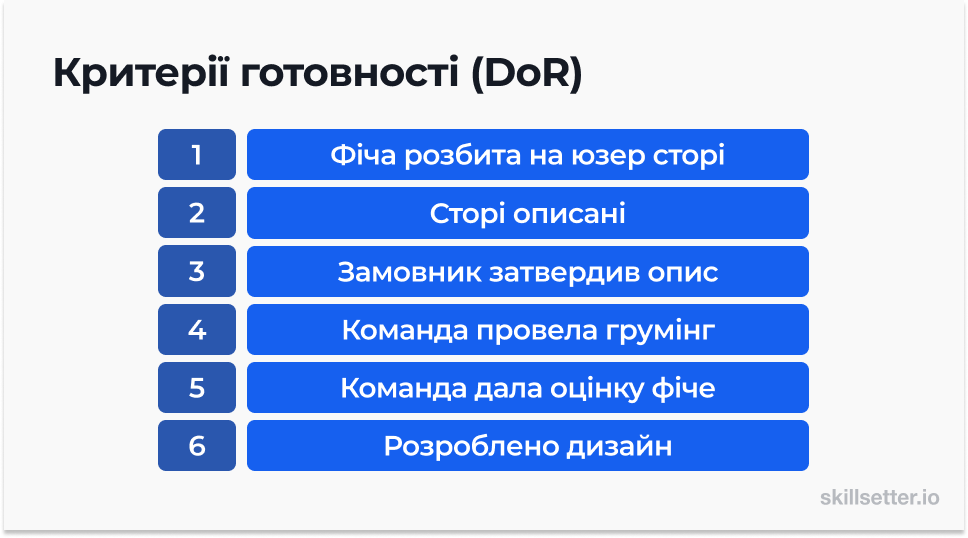
Тестувальник закінчує роботу, як тільки фіча протестована.

А для замовника готовність — це коли продукт можна надавати користувачам.

**Definition of Done і Definition of Ready**

**# DoR**

***DoR (Definition of Ready)*** — це критерій готовності задачі до розробки. Він визначає, що потрібно зробити, щоб програмісти почали писати код. Тобто можна вносити в стовчик «до розробки». Пункти DoR змінюються від проєкту до проєкту, але базові критерії виглядають так:



Продект!

Розробляй DoR та затверджуй критерії з командою. Так ви позбавите себе багатьох непорозумінь і проблем у розробці.

DoR розміщуй в Confluence. У майбутніх проєктах ти звірятимеш по них усі фічі до етапу їх розробки.

**# DoD**

DoD (Definition of Done) - це критерії завершеності завдання. Вони визначають, що потрібно зробити, щоб завдання можна було вважати виконаним. DoD допомагає команді зрозуміти, коли картку в багтрекері можна перенести зі стовпчика "У процесі" до стовпчика "Готово".

Критерії готовності задачі потрібні, щоб команда мала точне уявлення про те, що означає завершена задача. Це допоможе розробникам та команді QA здавати повністю дороблені фічі.

Тепер визначимось, що має входити до DoD по розділам роботи:

* 1. **Розробка:**
* Код написано, створено [фіча бранч](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true). Це окрема гілка в репозиторії, де розробник пише код для фічі. Як тільки код готовий і перевірений, фіча-бранч вливається в іншу гілку
* Код пройшов валідацію у [аналізаторі коду](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true). Це програма, що перевіряє код на працездатність.
* [Код рев’ю](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true) пройдено. Це перевірка коду на помилки та стиль. Код рев’ю проводиться усередині команди
* Юніт тести написані та працюють
* [Інтеграційні тести](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true) написані та працюють, тобто це тести, що перевіряють чи торкнеться додавання нової фічі старого функціоналу
* [Авто тести](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true) написані та працюють
  1. **Тестування:**
* [Тест-кейси](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true) написані. Тест-кейси це набір дій, які потрібно зробити QA-інженеру, щоб перевірити працездатність фічі
* Тестування проведено на узгодженому списку пристроїв
* Якщо у постачанні будуть баги, їх [наявність](https://www.bugsnag.com/blog/not-all-bugs-are-worth-fixing) погоджено із замовником
* Білд віддано на приймальне тестування замовнику. В цьому контексті білд - це вихідний результат процесу збірки вихідного коду, що може бути використаний для тестування, розгортання, випуску програмного продукту або його компонентів.
  1. **Постачання (реліз):**
* Постачання узгоджено та заплановано на певну дату
* [Гілка з фічею вмержена у майстер](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true). Мерджити означає процес злиття різних гілок розробки в одну. Так, наприклад, розробник може писати код для фічі в окремій гілці, а потім влити в головну гілку. Тоді фіча додасться до решти функціонала.
* Проведено [деплой](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true). Деплой це процес розгортання або випуску програмного продукту або його компонентів в реальне виробниче середовище, щоб вони стали доступними для використання кінцевим користувачам
* Проведено [смоук тест](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true)[[1]](#footnote-1) (тобто це перевірка системи на стабільність), багів не виявлено (= реліз не потрібно скасовувати) .
* Користувачам надіслано [реліз ноутси](https://www.notion.so/aquarelleyep/Release-notes-6ec08c6b200f49858f074007328c0056). "Реліз ноутси" (англ. "release notes") - це документ або запис, який містить інформацію про випуск нової версії програмного продукту, зазвичай відповідає наступному запитанню: "Що нового?"

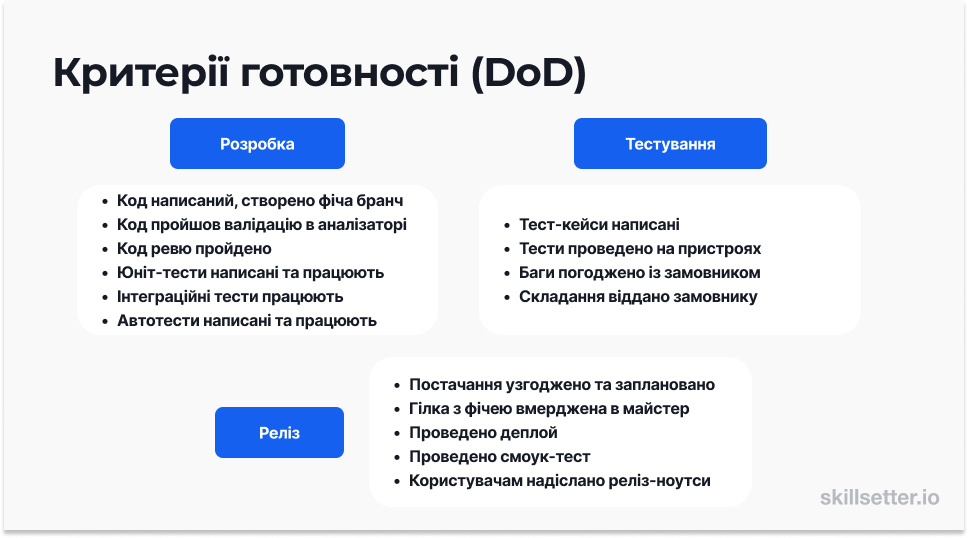


Рис. Критерії готовності DoD (Definition of Done)

Отже, пункти вище — це лише один із багатьох можливих прикладів. Для кожного проєкту DoD буде власним. Він визначається технологіями, методологією розробки, постачання та умовами контракту.

Наприклад, якщо ти працюєш за методологією Waterfall, то в DoD буде більше пунктів про затвердження вимог із замовником, а реліз буде лише один — наприкінці проєкту. У проєктах за договором Time and Material ставлення до вимог м'якше, а релізи можуть проводитись раз на тиждень.

Розглянемо деякі методики тестування:

**TDD, BDD, ATDD** - це методики тестування програмного забезпечення, які базуються на різних підходах до тестування та спрямовані на покращення якості програмного забезпечення.

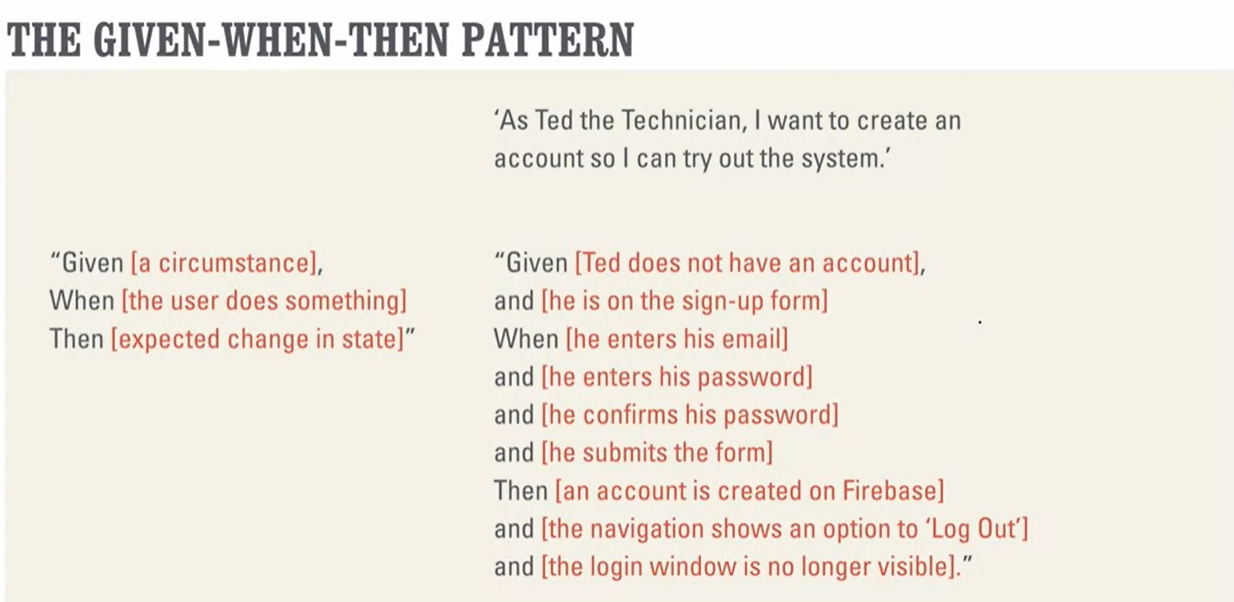
TDD (Test-Driven Development) - це методика розробки програмного забезпечення, в якій тестування є основним елементом процесу розробки. Розробник спочатку створює тестові сценарії, які описують очікувану поведінку програмного забезпечення, а потім пише код, щоб ці тестові сценарії пройшли успішно. Цей процес може допомогти виявити помилки в коді раніше, сприяє покращенню дизайну програми та полегшує підтримку програми.

BDD (Behavior-Driven Development) - це методика розробки програмного забезпечення, в якій основний фокус ставиться на опис поведінки програми з точки зору бізнес-процесу. BDD використовує формат "Given-When-Then" для опису сценаріїв, де "Given" описує початковий стан системи, "When" - дію, яка спричинює зміну, та "Then" - очікуваний результат. BDD допомагає покращити співпрацю між розробниками, тестувальниками та бізнес-аналітиками, а також забезпечує більш точне відображення бізнес-процесів у програмному забезпеченні.

ATDD (Acceptance Test-Driven Development) - це методика, яка базується на BDD та використовує автоматизовані тестові сценарії для перевірки відповідності програмного забезпечення вимогам клієнта або користувача. ATDD допомагає забезпечити, що програма відповідає вимогам користувача, а також полегшує підтримку програми.

ATDD методика:

<https://coursera.org/share/8c8327e0f556a11ca1d3f3547a2bcdb5>



«З огляду на (обставина).

Коли (користувач щось робить)

Потім (очікувана зміна стану)"

«Як технік Тед, я хочу створити обліковий запис, щоб випробувати систему»

«**Враховуючи або дано** (Тед не має облікового запису),

і (він знаходиться у формі реєстрації)

**Коли** (він вводить свою електронну адресу)

і (він вводить свій пароль)

і (він підтверджує свій пароль)

і (він подає форму)

**Потім** [створюється обліковий запис у Firebase]

і [навігація показує опцію «Вийти»]

і (вікно входу більше не відображається).

1. **Тестування**

Одна зі складових якісного проєкту — це тестування. Якщо його не проводити, у системі буде багато багів, а вони, своєю чергою, погіршують роботу продукту.

**Що входить в обов’язки QA.** Основні обов'язки QA (Quality Assurance, або контроль якості) можуть варіюватися залежно від компанії, проекту, продукту та ролі QA-спеціаліста. Однак, основні завдання QA можуть включати наступне:

1. Розробка та виконання тест-планів: Розробка стратегій тестування, планування, створення та виконання тест-кейсів, тест-сценаріїв, тест-датасетів та інших тестових документів для перевірки відповідності вимогам, функціональності та якості програмного продукту.
2. Виявлення та документування дефектів: Виявлення, реєстрація та документування дефектів, помилок, аномалій та інших невідповідностей в роботі програмного продукту, включаючи опис проблеми, кроки для відтворення, приклади даних та іншу відповідну інформацію.
3. Виконання різноманітних видів тестування: Виконання різноманітних видів тестування, таких як функціональне тестування, регресійне тестування, випадкове тестування, навантажувальне тестування, безпекове тестування, тестування на різних платформах, браузерах та пристроях, тестування відповідності вимогам та інше.
4. Взаємодія з розробниками та іншими стейкхолдерами: Взаємодія з розробниками, бізнес-аналітиками, менеджерами проектів та іншими учасниками процесу розробки програмного продукту для вирішення виявлених проблем, встановлення причин дефектів та розуміння вимог та функціональності продукту.
5. Валідація вимог та документації: Перевірка відповідності вимог та документації продукту, перевірка наявності та актуальності документації, виявлення можливих невідповідностей та внесення рекомендацій щодо вдосконалення.
6. Тестування різних платформ та пристроїв: Тестування програмного продукту на різних платформах, таких як робочі станції, сервери, мобільні пристрої, планшети, різні операційні системи, браузери та інші варіанти використання, з метою впевненості в його коректній роботі на різних конфігураціях.
7. Аналіз ризиків: Виявлення та аналіз потенційних ризиків та небезпек, пов'язаних з роботою програмного продукту, включаючи виявлення можливих вразливостей, помилок безпеки, ризиків для даних користувачів, ризиків дотримання вимог та стандартів, та надання рекомендацій щодо їхнього врегулювання.
8. Взаємодія зі замовниками: Взаємодія зі замовниками, замовниками вимог, менеджерами проекту та іншими стейкхолдерами з метою вирішення питань, пов'язаних з вимогами, документацією, вимогами до якості та відповідності продукту встановленим критеріям якості.
9. Відстеження та звітність: Відстеження результатів тестування, складання звітів про виявлені дефекти, відмічення виконання тест-планів, відстеження відмов, аналіз результатів тестування та підготовка звітів для розробників, керівництва та інших зацікавлених сторін.
10. Вдосконалення процесів тестування: Участь в вдосконаленні процесів тестування, впровадженні нових методів, інструментів та підходів до тестування з метою покращення ефективності та якості процесу тестування.
11. Технічна підтримка: Надання технічної підтримки розробникам та іншим членам команди в процесі виявлення та виправлення дефектів, аналізу та відлагодження проблем, пов'язаних з якістю програмного продукту.
12. Тестування безпеки: Виявлення потенційних загроз безпеці продукту, тестування заходів безпеки, виявлення можливих вразливостей та атак на продукт, аналіз результатів та надання рекомендацій щодо вдосконалення безпеки.
13. Взаємодія з командою розробників: Співпраця з розробниками, комунікація щодо виявлених дефектів, тестових вимог та інших питань, пов'язаних з якістю продукту.
14. Технічне тестування: Виконання технічного тестування, такого як тестування функціональності, продуктивності, завантаження, масштабованості, сумісності, настроюваності та інших аспектів, залежно від вимог продукту.
15. Автоматизація тестування: Розробка, виконання та підтримка автоматизованих тестів, використання відповідних інструментів та технологій для автоматизації процесу тестування.

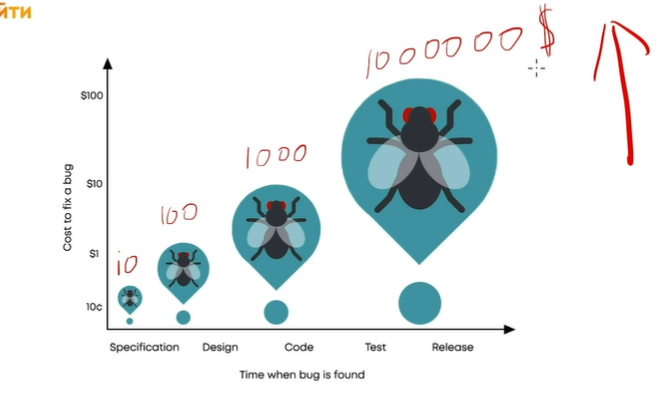


Рис. Вартість тестування для компанії на етапі створення продукту. Вже на етапі планування QA має здійснювати налаштування процесу тестування)

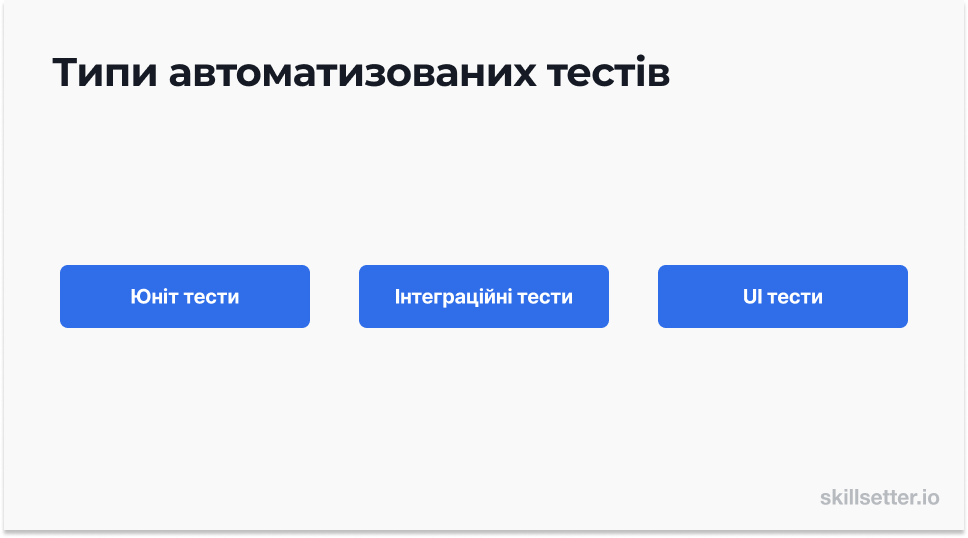
## Види тестування

## Автоматизовані тести

## Види автоматизованого тестування:

## <https://drive.google.com/file/d/1VvG64TYZpd5kexp7O0y_SkeHU9NQ_Nkx/view?usp=sharing>

Це тип тестів, який **виконується автоматично**. Простіше кажучи, це код, який перевіряє інший код. Серед автоматизованих тестів можна виділити три основні: юніт тести, інтеграційні тести та UI тести.



Розглянемо кожен докладніше.

**Юніт тести** — це код, який перевіряє окремі модулі (юніти) системи. Його пишуть програмісти на етапі розробки, інколи навіть до написання самого коду. *Наприклад,* у Filmy є код, який додає текст на відео. Один з юніт тестів для цієї фічі змінюватиме шрифт і перевірятиме, що він дійсно є на відео.

Якщо юніт тести пишуться ще до коду, це називають TDD (Test Driven Development) підхід.

**Інтеграційні тести** перевіряють взаємодію з іншими компонентами та системами. Вони дуже схожі на юніт тести й відрізняються тільки тим, що перевіряють не свою власну систему, а залежність від зовнішньої.

Цей тип тестування включає в себе тестування інтерфейсів між компонентами, обмін даними та взаємодію між системами. Інтеграція може бути між різними модулями програмного забезпечення або між різними системами. Компоненти програмного забезпечення можуть бути різного типу, наприклад:

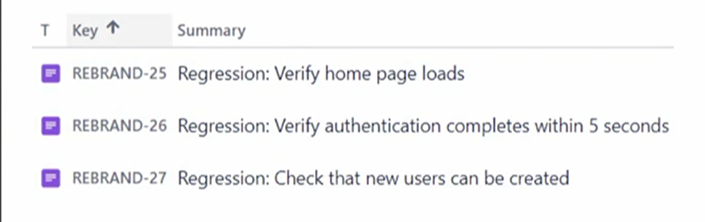
* Модулі або класи програмного забезпечення, які виконують певні функції або операції.
* Бібліотеки програмного забезпечення, які містять набір функцій або методів, які можуть бути використані в різних частинах програми.
* Сервіси програмного забезпечення, які забезпечують функціональність, доступну для використання в різних частинах програми або в різних програмах.
* Бази даних, які містять дані, які використовуються в програмному забезпеченні.

Інтеграційне тестування може бути виконане на різних рівнях, таких як:

Компонентне тестування - перевірка, як окремі компоненти програмного забезпечення працюють разом.

Системне тестування - перевірка, як система як цілісність працює, коли всі компоненти були інтегровані.

Інтерфейсне тестування - перевірка, як інтерфейси між компонентами працюють.



*Перевірте завантаження домашньої сторінки*

*перевірка автентифікації завершується протягом 5 секунд*

*перевірте, чи можна створювати нових користувачів*

Рис. Приклад задач, які можна протестування за допомогою автоматизації.

**UI тести** [емулюють](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true) роботу пристрою. [Емуляція](https://www.techopedia.com/definition/4787/emulation) — це сукупність логічних та технічних засобів та ресурсів, спрямованих на повну імітацію технічного пристрою на іншому ПЗ. Вони перевіряють, наскільки добре працює UI та чи зможе користувач взаємодіяти із системою. Вони також перевіряють зв'язок декількох фіч між собою. Так, роблячи щось в одній фічі, ти можеш побачити, що відбувається в іншій.

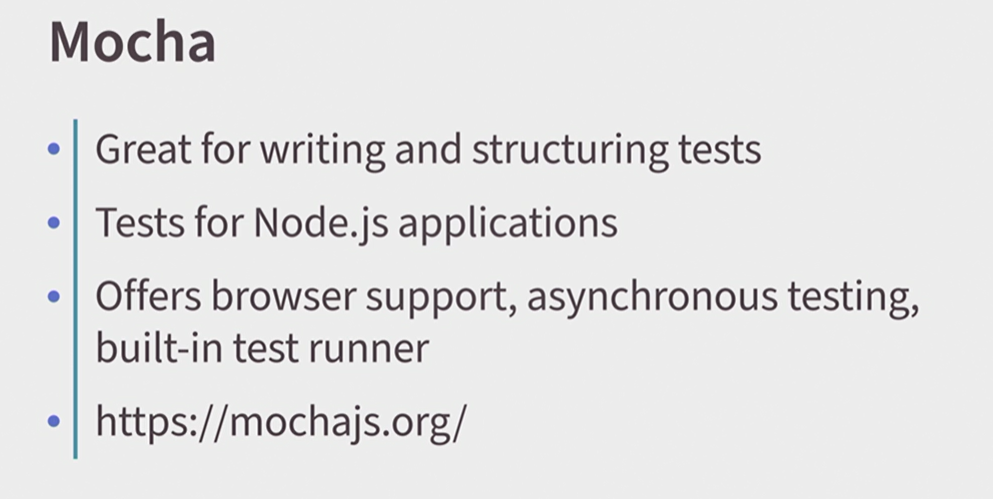
Приклад, наш проект. Уяви, що ви вирішили перевірити додавання тексту на відео в Filmy з пристрою IPhone 7. Для цього ви запустите симулятор смартфона, а на ньому ваш відеоредактор. UI тест перевірить потрібний сценарій користувача: від появи іконки додавання тексту на екрані до збереження відео. Тест буде натискати на всі кнопки на своєму шляху, а ви переконаєтеся, що вони працюють як потрібно.

З трьох типів автоматизованих тестів складається **піраміда тестування**.



Як бачите з рисунку, юніт тестів проводиться найбільше. Їх швидше писати, легше підтримувати й вони менш вартісні. А ще, на відміну від двох інших типів, юніт тести найшвидше виконуються.

Відомі фреймоврки для автоматизації тестування Unit, наприклад, JUnit для java або PyUnit для Pyton та інші:



Чудово підходить для написання та структурування тестів

Тести для програм Node.js

Пропонує підтримку браузера, асинхронне тестування,

вбудований засіб запуску тестів



Можна використовувати з будь-яким варіантом JavaScript

Не вимагає браузера або DOM

Чистий і зрозумілий синтаксис

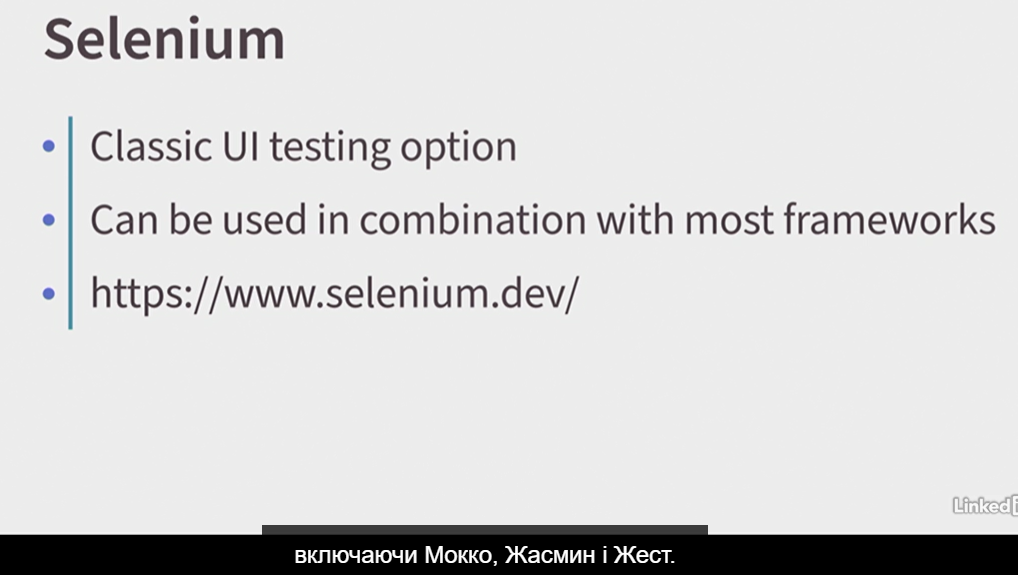
Має програму тестування, яку можна налаштувати



Створено Facebook для тестування React

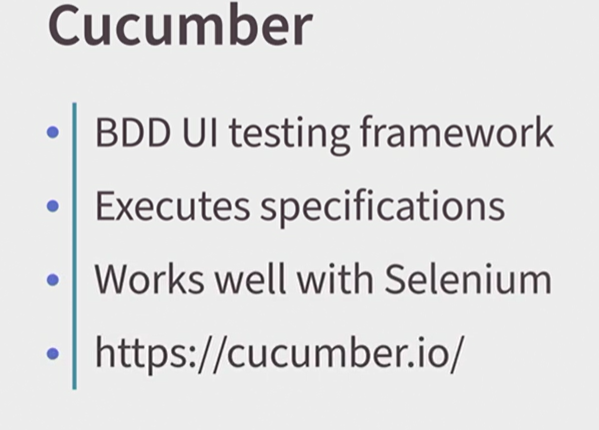
Має вбудовану програму запуску тестів, бібліотеку (мокінгу[[2]](#footnote-2)) імітації та звіт про покриття коду

Для тестування інтерфейсу користувача:



Класичний варіант тестування Ul

Можна використовувати в поєднанні з більшістю фреймворків



BDD Ul testing framework. Суть BDD Ul (Behavior-Driven Development з використанням мови Ul) тестування полягає в створенні та виконанні тестів на основі специфікацій або сценаріїв, які описують поведінку системи. Ці специфікації виражаються за допомогою мови, яку можуть розуміти всі учасники проекту, включаючи бізнес-аналітиків, тестувальників та розробників, і зазвичай використовують мову Ul (наприклад, Gherkin).

Виконує специфікації (або сценарії)

Добре працює з Selenium



Фреймворк тестування Ul

Швидко, легко та надійно

Працює в будь-якому браузері

[**Google Optimize**](https://optimize.google.com/)**– сервіс для тестування інтерфейсу сайту.**

Тут можна проводити А/В та мультиканальні тестування, збирати та аналізувати дані про поведінку користувачів на сайті.

**Довідково**. Суть *A/B-тестування в ІТ-проектах* полягає в порівнянні двох альтернативних варіантів (контрольної групи А та експериментальної групи В) з метою визначення, який варіант є більш ефективним за певними метриками. Коли користувачі взаємодіють з проектом (наприклад, веб-сайтом, мобільним додатком, електронною комерцією), їм випадковим чином призначається один з варіантів (контрольний або експериментальний) і збираються дані про їх поведінку та результати.

Головна ідея полягає в тому, що дві групи отримують різні варіанти проекту, а різниця в їхніх результатах відображає ефективність різних варіантів. Завдяки збору даних можна визначити, який варіант краще відповідає меті тестування, наприклад, забезпечує вищу конверсію, більше продажів, менше відмов, більше взаємодії тощо.

*Мультиканальне тестування (*також відоме як мульти-армі тестування) - це розширена версія A/B-тестування, в якій порівнюється більше двох варіантів (армів) проекту одночасно. В мультиканальному тестуванні можуть брати участь три або більше варіантів, дозволяючи одночасно порівнювати більше альтернатив та визначати оптимальний варіант на основі результатів тестування.

Інструмент повністю безкоштовний.

Як [налаштувати A/B тест за допомогою Google Optimize - покроковий мануал](https://netpeak.net/ru/blog/kak-nastroit-a-b-test-s-pomoshch-yu-google-optimize-poshagovyi-manual/) .

[**ASO Giraffe**](https://asogiraffe.com/)**– сервіс для A/B-тестування програм.**

ASO Giraffe платний, але тріал включає нескінченну кількість тестів. Під час реєстрації можна вибрати кросплатформне тестування, або «single-platform».

Є безкоштовний двотижневий тріал.

Докладніше про A/B-тести можна дізнаєтесь у [статті](https://skillsetter.io/blog/AB-test-questions) . Усередині шаблон для підготовки експерименту.

[**Evanmiller**](https://www.evanmiller.org/)**— сервіс для проведения A/B-тестів**

Тут можна розрахувати, скільки людей буде потрібно для дослідження, або перевірити достовірність гіпотези за допомогою критерію Хі-квадрат.

*Довідково.* Критерій Хі-квадрат використовує спостережені значення в таблиці спряженості (спостережувану частотну таблицю) та порівнює їх з очікуваними значеннями, які випливають з нульової гіпотези - гіпотези про відсутність залежності між змінними. Критерій Хі-квадрат розраховує спеціальну статистику Хі-квадрат, яка має розподіл Хі-квадрат, і використовується для розрахунку p-значення - ймовірності отримати такий або більший рівень спостереженої залежності між змінними, якщо нульова гіпотеза справді вірна.

Обмеження інструмент повністю безкоштовний.

[**UsabilityHub**](https://usabilityhub.com/product/five-second-tests)**- сервіс для UX-досліджень.**

Тут можна проводити опитування, п'ятисекундні випробування, випробування першого кліка.

Обмеження: тести не більше двох хвилин.

З чого почати навчання: [гайди від UsabilityHub](https://usabilityhub.com/guides) .

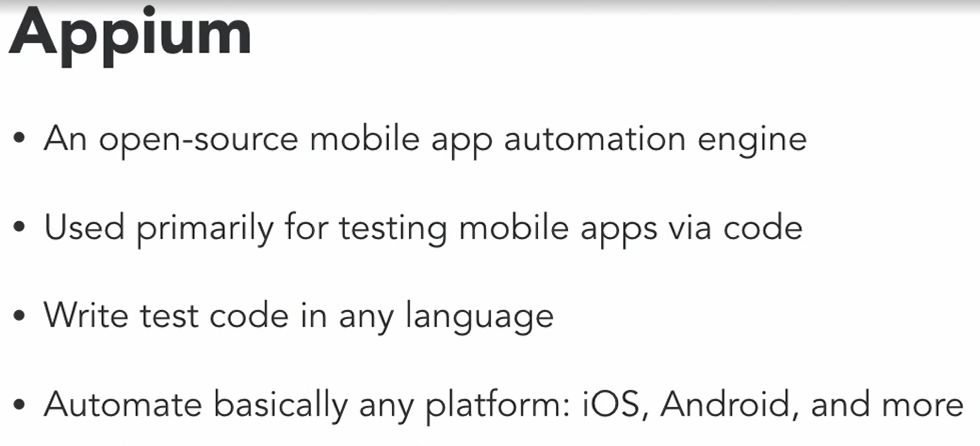
[**Lookback**](https://lookback.io/)**— сервіс для віддаленого тестування UX.**

Сервіс дозволяє тестувати сайт, програму або прототип віддалено. Учасник транслює екран у режимі реального часу під час дзвінка.

Обмеження: є безкоштовний 14-денний тріал, далі від $99 на місяць.

З чого розпочати навчання: [демо-відео від Lookback](https://www.notion.so/Lookback-Shorts-587e67e29db64ade84cf86a4c732c9f6).

Для мобільних додатків



Механізм автоматизації мобільних додатків із відкритим кодом

Використовується переважно для тестування мобільних програм за допомогою коду

Напишіть тестовий код будь-якою мовою

Автоматизуйте практично будь-яку платформу: iOS, Android тощо

## Наступний вид тестування:

## Ручні тести. Ці тести виконуються вручну, тому так і називаються. Розгляньмо їх основні типи.

**Функціональне тестування** — це основний вид тестування на проєкті. Щоб його провести, тестувальник дивиться вимоги, перевіряє за ними фічу та шукає баги у тому, як вона працює.

**Регресійне тестування** — це тестування старих, вже готових фіч. QA проводять його, щоб переконатися, що нова фіча не зламала чинний функціонал. Щоб нові фічі не ламали те, що є, регресійне тестування потрібно проводити частіше. *Наприклад,* розробники месенджера типу Telegram можуть зробити фічу «Позначити повідомлення як прочитане». А QA відразу перевірять ситуацію із сусідніми фічами: «Лічильник нових повідомлень» та «Подвійна галочка навпроти повідомлення співрозмовника в чаті».

Але тестувальники не завжди встигають зробити регресію разом із функціональним тестуванням. Його основний обсяг виконується прямо перед релізом, а це багато роботи. Тому, чим більше розробники проводять автоматизованих тестів, поки пишуть код, то менше зрештою займає регрес. І тим скоріше команда випускає релізи.

*Приклад тест кейсу для ручного тестування*

Переконайтеся, що логотипи головного веб-сайту відповідають новому бренду

Кроки

1. Очистіть кеш

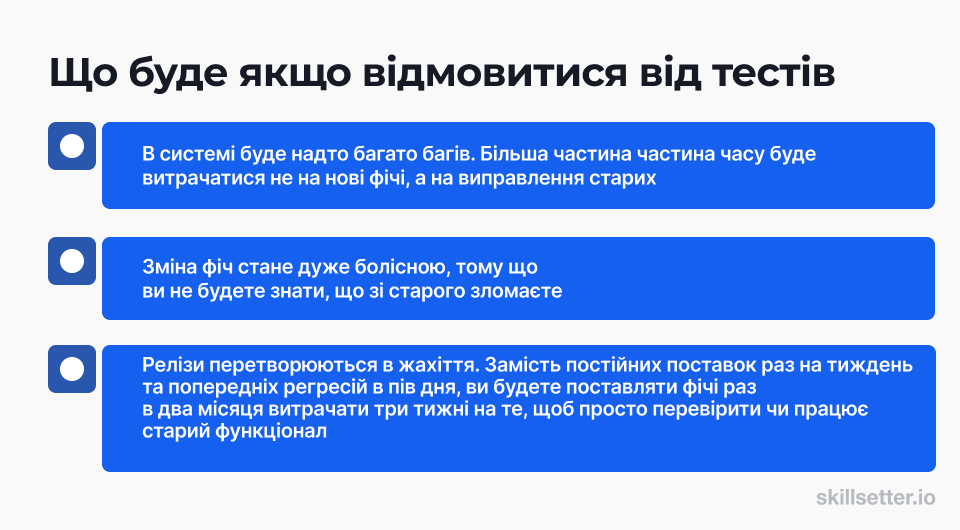
2. Перейдіть за посиланням і переконайтеся, що логотип оновлено новим значком і стилями

3. Перейдіть за посиланням і переконайтеся, що логотип оновлено новою піктограмою та стилем

Очікуваний результат

Усі основні файли логотипів відповідають новому бренду

Наставник: Намагайся вкладати ресурси в автоматизовані тести. Звичайно, на їх написання та підтримку йде 10-15% бюджету і не менше часу. Але, як і усунення техборгу, автоматизовані тести — це інвестиція у розвиток проєкту. І виходить, що насправді тести заощаджують час, тому що регресія не розростається, і зрештою ви витрачаєте менше. Поясни це замовнику та заручися його підтримкою у виділенні бюджету на автоматизовані тести. Якщо їх не писати, то скоро ви зіткнетеся з тим, що:



В деяких проєктах потрібно проводити ще й **тестування безпеки[[3]](#footnote-3) та продуктивності[[4]](#footnote-4)**.

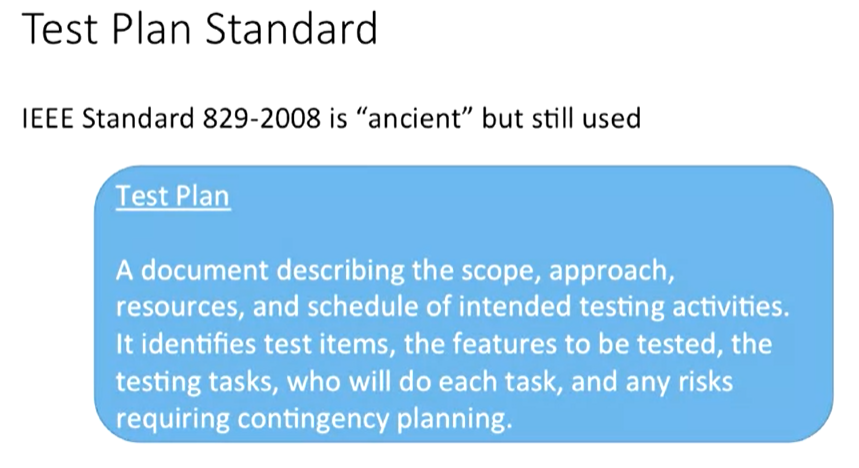
Вони використовуються не скрізь і не так часто, як тести, які ми розглянули вище.

Але це не всі можливі тести. Насправді їх дуже багато. І набір тестів відрізняється від проєкту до проєкту.

Рекомендація. Подивися [цей матеріал](https://www.softwaretestinghelp.com/types-of-software-testing/) на Software Testing Help — там є список із понад 40 тестів, які команди регулярно використовують.

Відео. Інструменти для розробки,тестування та постійного постачання коду: <https://drive.google.com/file/d/1Wj6pqTo45DXyeqLd7bWxKTUp6ae5PxEb/view?usp=sharing>

Відео. Інструменти для моніторингу стійкості, продуктивності та безпеки системи: <https://drive.google.com/file/d/1npg7n14e2q4JdSq3RoFtR1fblNdnXJAL/view?usp=sharing>



Документ, що описує обсяг, підхід, ресурси та розклад запланованих заходів тестування. Він визначає тестові елементи, функції, які потрібно перевірити, тестові завдання, хто виконуватиме кожне завдання та будь-які ризики потрібно планування на випадок надзвичайних ситуацій.

**Компоненти плану тестування**

Підхід/стратегія тестування

Область застосування

Зміст

Ресурси/Тестове середовище

Критерії входу та виходу

Матриця вимог (для відстеження)

Що НЕ тестується

Тестові випадки та сценарії [окремий документ(и)]

## 3. Захід Демо

Демо — це зустріч довжиною 30-90 хвилин, де команда показує результат роботи за ітерацію. Його проводять або в останній день спринту, або перші пару днів нового. Туди запрошують замовника, стейкхолдерів та всю команду.

*Спільні демо потрібні, щоб у команд складалася об'єктивна картина про те, що відбувається в компанії*. Так вони дивляться проєкти інших, черпають інсайти з чужих виступів та покращують свої процеси.

Проджект і команда розуміють, що їх напрацювання дивитиметься не лише замовник, а й уся компанія. Тому вони приділяють більше часу підготовці: ретельніше репетирують виступ і намагаються завершити фічу до демо.

*CEO[[5]](#footnote-5) може використовувати демо як можливість відстежувати роботу команд*, але це об'єктивний підхід. Для розуміння процесів CEO має встановлювати [OKR](https://www.whatmatters.com/faqs/okr-meaning-definition-example/) («OKR» означає «цілі та ключові результати») і слідкувати за ключовими метриками.

*Демо спрямоване на презентацію результатів*, а кожна команда має свій. І всі вони різні. На загальних демо не так багато часу на обговорення проблем. Ця подія націлена на презентацію результатів. Вже після демо команди можуть обмінятися враженнями та обговорити виконану роботу. Це потрібно для того, щоб розуміти рівень інших команд і краще готуватися до демо.

А ще спільні *демо — це чудова мотивація*, щоб краще підготуватися. Адже ти розумієш, що маєш виступати перед кількома командами. Намагаєшся показати хороший рівень.

Наставник. На демо не обов'язково показувати кожну зроблену за спринт задачу. Не зупиняйся на юніт-тестах, дослідженнях та виправленні багів. Натомість сфокусуйся на тому, що буде цікаво стейкхолдерам і принесе користь їхньому бізнесу.

Зроблена робота зазвичай показується на тестовому [білді](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true) додатка. Ви з командою вже протестували його на баги та підготували до демо. На зустрічі ви також надішлете додаток Олівії, щоб вона протестувала його на своєму девайсі.

Наш проект:

**Що на першому демо має показати команда Олівії?**

На цьому демо команда плануємо показати Олівії, як додаються відео з галереї телефону, як можна друкувати, переміщувати та видаляти текст. Тож замовнику буде зрозуміло, на якому етапі знаходиться проєкт. Вона побачить результати роботи, за яку платить гроші.

**Отже, результати демо зустрічі:**

## представлені перші можливості FIlmy:

## вибір відео з галереї,

## додавання, переміщення та видалення тексту

**Виявлені наступні баги у проекті під час демо:**

1) виявилося, що додаток не завантажується на планшеті в Олівії. У цьому спринті, на жаль, не встигли добре протестувати всі iOS пристрої, але обов'язково протестуємо у наступній ітерації. Це перша версія, не для продакшена. У ній можуть бути баги.

*Тобі як менеджеру треба розібратися, чому так вийшло і як не робити цієї помилки в майбутньому.*

**Під час демо Олівія виявила нові вимоги та питання до додатку**:

1) З галереї не можна вибрати відео в 4K (це формат відео з роздільною здатністю 3840 на 2160 пікселів), тому що команда обробляла лише відео від 360p до 1080p. На думку Олівії 4K відео може займати більше місця порівняно з традиційними форматами відео, однак його використання може дозволити користувачам отримувати більш якісне та реалістичне зображення в своїх відео проектах.

Додати підтримку 4К можна, але це займе ще трохи часу, тому що цей формат підключається по-іншому. Часу на це командра не закладала, тож і не працює.

2) Питання від Оліві. Чи зможемо ми знімати відео прямо в додатку? Завантажувати готові файли у Filmy — це одна справа. Але якщо користувач зможе вести фільмування та монтувати ролики в одному місці, буде набагато зручніше. Це питання Олівії належить до майбутніх змін у додатку. Ідея з'явилася у замовника вже під час демо та точно вплине на початковий скоуп проєкту.

**Які дії повинен вчинити проджект після демо з приводу цієї фічі?**

***Рішення:***

Обговорити цю нову фічу на окремому мітингу.

Зрозуміти, як саме хоче Олія, щоб вона працювала та виглядала.

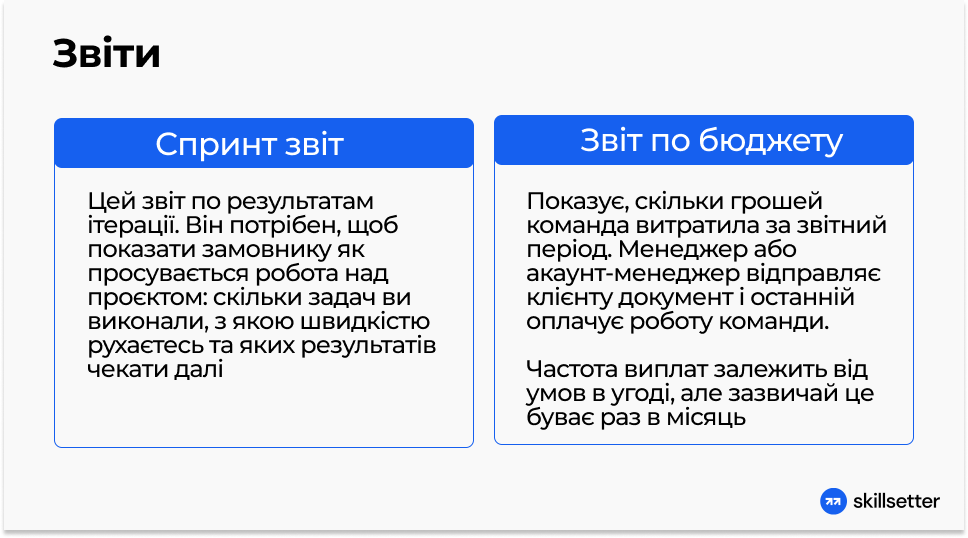
Оцінити задачу та зрозуміти, як це вплине на наш беклог.

Обговорити рішення з Олівією.

Наступні дії проджекта. Ти звертаєшся до наставника

Наставник.

Наприкінці спринту менеджеру потрібно не лише провести демо, розпланувати наступну ітерацію та переконатися, що всі етапи пройдені. Тобі потрібно надати замовнику звіти про роботу за два тижні. На завершення ітерації проджект готує для замовника спринт репорт, а наприкінці місяця — звіт по бюджету. Див. Додаток таблиця (лист Бюджет витрат\_Спринт 1). Просто використовуй таблицю і розрахуй всі ваші витрати за місяць. Туди потрібно буде вбити імена всіх учасників у команді, їх рейти та кількість відпрацьованих годин.



Графіки Sprint Burndown chart та Velocity chart ти вже знаєш. Їх можна скласти самостійно або прикріпити із Jira. Графіки Unit test та AQA (Automation Quality Assurance) показують кількість проведених тестів та їх перформанс. Ці дані можуть тобі дати тестувальники.

На даному етапі ти можеш дати замовнику інформацію про сторі поінти, velocity та задачі спринту. Враховуючи те, що у вас не вийшло в першій ітерації, можна спрогнозувати ризики. А ґрунтуючись на поточних задачах — запланувати таски на наступний спринт.

***Приклад*** Звіт по спринту 1

**Статус проекта:** все добре

**Спринт #:** 1

**Дата спринта:** 10/05/23 - 23/05/23

**Сторі поінти заплановані/завершені:** 8 / 8

**На Velocity спринта вплинули причини:**

Не дивлячи на те, що ми змогли виконати всі сторі поінти, в спринті були наступні перешкоди:

Даррен не встиг доробити задачу Custom UI. До нього долучився Кері, із-чого його задача була перенесена на наступний спринт.

Команда ще не встигла звикнути до режиму загальної роботи.

Команда раніше не працювала з відео редакторами. Багато задач були для розробників новими.

Недостатньо чітка оцінка задач на грумінгу.

Відсутніст DoD и DoR

**Velocity на наступний спринт: 8**

**Деталі спринта: задачі**

|  |  |
| --- | --- |
| **1. Додавання відео із галереї** |  |
| Загальний UI | закрито |
| Прототип — додавання із власної галереї | закрито |
| Кастомний UI галереї | закрито |
| Розширення | закрито |
| Вибирати відео з галереї | закрито |
| Фильтрувати тільки .mp4 и .mov | закрито |
| Сортувати відео в галереї: нові відео першими | закрито |
| **2. Додавання тексту. Частина 1** |  |
| Додавання тексту | закрито |
| Змінити позицію тексту за дією pinch | закрито |
| Видалити текст | закрито |

**Задачі на наступний спринт:**



|  |  |
| --- | --- |
| **6. Додавання тексту. Частина 2** | В роботу |
| Застосувати жирний шрифт, курсив та підкреслений |  |
| Застосувати один з вісьми шрифтів |  |
| Змінити колір тексту |  |
| Юніт тести |  |
| UI тести |  |
| **3. Експорт в різній якості** |  |
| UI |  |
| Процес експорту |  |
| Застосувати 360p |  |
| Застосувати 480p |  |
| Застосувати 720p |  |
| Юніт тести |  |
| UI тести |  |

**Також звіт має місти м**[**айлстоун (Milestone)**](https://www.productplan.com/glossary/roadmap-milestones/) — це подія, дата, дедлайн, до якої необхідно поставити певний обсяг роботи. Це може бути реліз, презентація продукту перед інвесторами, велика рекламна кампанія тощо.

Ризики в наступному спринті

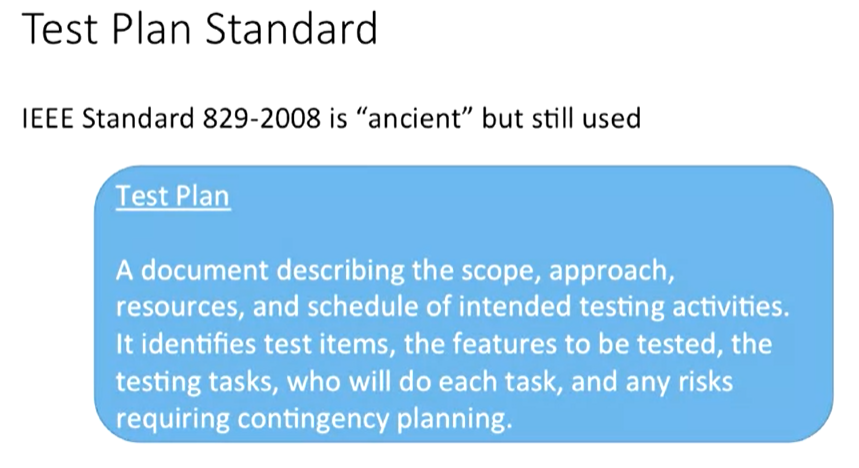
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Ризик | Опис | Ступінь | Як зменшити | відповідальний |
| Відповідність фічі DOR та DOR | Неправильно прописаны критерії готовності можуть привести до недоробок по фічам, втрати частини вимог | високий | Прописати DOR та DOR до початку спринта та затвердити з командою | Прожект-менеджер |
| Замовник не підтвердив опис юзер сторі для експорту | Якщо Олівія не встигне підтвердити опис вимог, фіча буде розроблена не так, як вона очікує. Знадобиться брати додатковий час на виправлення | високий | Ще раз написати Олівії та надати аргументи на користь збільшення бюджету та строків на виправлення помилок | Прожект-менеджер |
| Патрік бере лікарняний у зв’язку з операцією | Іншим розробникам потрібно буде взяти на себе його задачі | Середній | Поговорити з Кері та розподілити задачі так, щоб встигнути до наступної демо. Можливо відмовитись від якихось задач | Тімлід |

## 4. Перший реліз

Наш проект:

Пройшов час і команда повністю підготувалася до постачання першої версії продукту. Усі фічі Filmy готові та протестовані. Ну, принаймні ті, що були в **тест-плані** (див. нижче). Баги теж пофіксили, хоч це було й нелегко. Особливо із таймлайнами. У контексті редакторів відео та аудіо, таймлайн використовується для відображення та організації різних елементів проекту у відповідному порядку на осі часу. На таймлайні можуть бути розміщені відео- та аудіофрагменти, зображення, текстові об'єкти, ефекти та інші елементи проекту. Таймлайн дозволяє користувачам точно контролювати, коли та які елементи з'являються на екрані та як довго вони тривають.

*Довідково:*



Документ, що описує обсяг, підхід, ресурси та розклад запланованих заходів тестування. Він визначає тестові елементи, функції, які потрібно перевірити, тестові завдання, хто виконуватиме кожне завдання та будь-які ризики потрібно планування на випадок надзвичайних ситуацій.

**Компоненти плану тестування**

Підхід/стратегія тестування

Область застосування

Зміст

Ресурси/Тестове середовище

Критерії входу та виходу

Матриця вимог (для відстеження)

Що НЕ тестується

Тестові випадки та сценарії [окремий документ(и)]

Впоралися скоріше на півдня. Релізну гілку [вмереджили](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true) у майстер. А Джо склав deployment guide (усі вимоги та інструкція для деплою для розробників).

Наставник.

Оскільки гілка master завжди повинна містити працюючий код і ви завжди кладете в неї тільки готові фічі, кожен мерж - це практично невеликий реліз. При цьому обов'язково щоразу перезбирайте програму. Не факт, що його потрібно відразу випускати для всіх користувачів - занадто часті оновлення програми можуть дратувати їх. Але постарайтеся зібрати групу бета-тестерів (почніть з себе), для яких ви випускатимете програму після кожного мержа гілки в master. Таким чином ви прискорите отримання зворотного зв'язку, а чим швидше цикл ОС, тим швидше розвивається додаток і ваші навички. У цьому вся суть Agile.

Але для релізу недостатньо лише налаштувати процеси. Необхідно домовитися з командою, як саме відбуватиметься постачання. Для першого та кількох наступних релізів потрібно окремо зустрічатися з тімлідами команди розробки та QA. Потім усі звикають до процедури, і релізи можна не обговорювати. Хороша практика — описувати цей процес у Confluence .

У розмові з командою користуйся [цим](https://docs.google.com/document/d/164MP6ciZ010e5JRznDw9M5_ixGXqLYxpUjvt1P9JlV8/edit?usp=sharing) універсальним чек-листом. Там лежить усе, що треба зробити до релізу.

Що потрібно зробити до релізу?

Отримати **sign off** (це формальне ухвалення результатів проєкту клієнтом. Він підтверджує, що команда зробила саме те, що він очікував.) від замовника та погодити дату та час релізу

Переконайтеся, що команда доступна на час релізу, щоб підтримувати систему

Написати **реліз ноутси (**частина документації [програмного забезпечення](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) , в якій описуються зміни між випуском і попередньою версіями цього ПЗ)

Провести всі види тестів, зазначені у **тест-плані (**це загальна стратегія для тестування ПЗ **)**

Провести перформанс тести (продуктивності), якщо є фічі, що впливають на продуктивність

Виправити всі закладені баги або погодити із замовником, що вони залишаються в беклогу

Перевірити, що білд відповідає прийнятій **неймінгу конвенції (**це угода щодо найменування гілок. Суть конвенції — вкласти сенс в імена гілок та релізів, щоб зробити їх максимально інформативними для розробників)

Налаштувати бекап бази (резерву) та всіх важливих даних продакшену

Підготувати **deployment guide (**усі вимоги та інструкція для деплою для розробників) та дотримуватися його

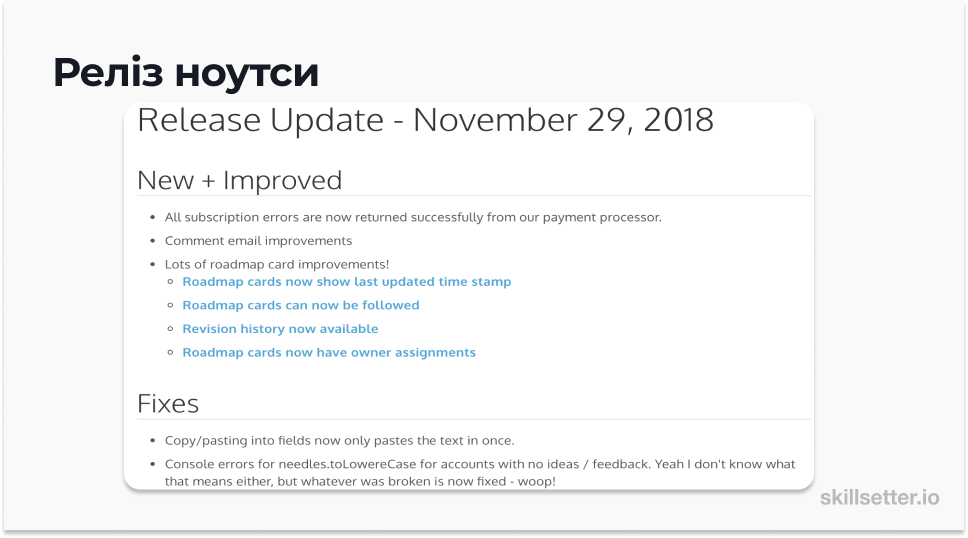
Запланувати **post-deployment скрипти (**це скрипти (алгоритми дій) для звернення до бази даних після кожного деплою. З їх допомогою можна створити базу даних, якщо її ще немає, або змінити її налаштування)

Опублікувати додаток у App Store.

**Вмерджіть (злити) релізну гілку** в master

Отже. Ви разом з командою пройшли всі етапи підготовки крім одного — тобі залишилося написати реліз ноутси. Спершу поговоримо про те, що це таке і навіщо вони потрібні.

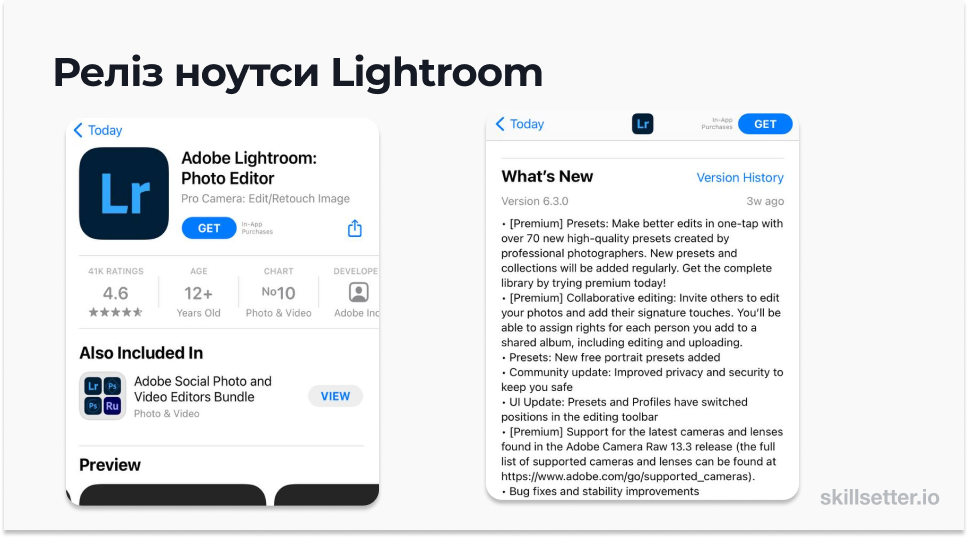
**Реліз ноутси** — це список усіх випущених фіч та змін у поточному постачанні. Виглядають вони приблизно так.



На слайді видно, які нові фічі увійшли до релізу, що було покращено та виправлено.

*Довідково.* Реліз ноутси бувають зовнішні та внутрішні.

**Зовнішні** пишуть для кінцевого користувача, щоб розповісти про важливі оновлення: що ви додали та як змінили продукт. Ви можете побачити їх в App Store або Play Market на сторінках додатків. Ось як вони виглядають у фоторедактора [Adobe Lightroom в App Store](https://apps.apple.com/us/app/adobe-lightroom-photo-editor/id878783582).



Як видно, у зовнішніх реліз ноутсах технічна документація не розписана докладно. Адже користувачам не важливо, як ви зарефакторили додавання ефектів на фото. А ось те, що тепер фіча працює швидше, дуже навіть цікавить. Їм цікаво, які нові можливості у роботі із програмою у них з'явилися.

Докладніше пишуться **внутрішні** реліз ноутси. Вони потрібні, щоб тримати в курсі змін команду, замовника та інших стейкхолдерів, якщо вони є. У внутрішніх реліз ноутсах вказані не тільки ключові зміни, але абсолютно всі [тікети](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true) (це завдання в багтрекері), які ви виправили за реліз. Дізнатися більше про внутрішні реліз ноутси можна в [цій статті](https://productmanagerhq.com/internal-release-notes/). Там ви знайдете і зручні способи їхнього опису.

Завдання. Напиши зовнішній реліз ноутс для Filmy: Version 1.0.

У першому релізі Filmy немає виправлень та оновлень, тому й у зовнішніх реліз ноутсах потрібно вказати лише набір фіч.

***Відповідь.***

- Вибирай відео (MP4 та AVI) зі своєї галереї та перетворюй їх на справжні музичні кліпи.

- Створюй нові проекти, зберігай на будь-якому етапі та продовжуй редагувати у будь-який момент

- Вибирай відповідний тобі формат картинки: 16:9 або 4:3

- З'єднуй кілька відео в одній доріжці. Створюй динамічні ролики тривалістю до 30 хвилин

- Регулюй довжину відео за допомогою функції обрізання

- Вибирай будь-який трек із величезної бази ITunes та додай до свого відео

- Пиши прямо на відео, використовуючи будь-який з 8 шрифтів та кольорів

- Використовуйте колірні фільтри, щоб надати кліпу потрібний настрій

- Управляй усіма доріжками на одному екрані: змінюй їх довжину та положення на таймлайні

- З'єднуй два відео в одній картинці, використовуючи функцію Picture in picture. Міняй відео місцями, використовуючи функції "top-down" або "right-left"

- Експортуй відео як 360p, 480p і 720p

- Поділися результатом у Instagram і TikTok - твої кліпи має побачити світ

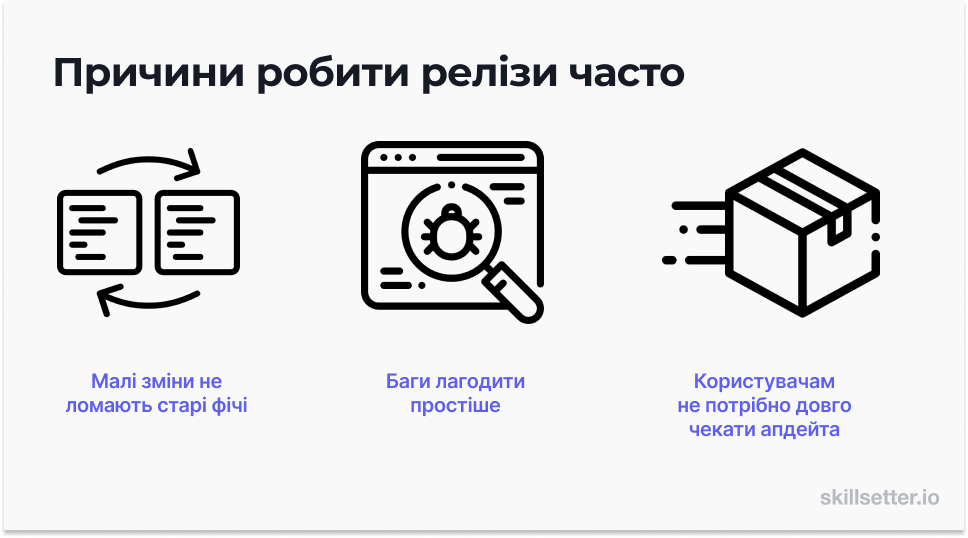
Отже. команда готує білд до випуску, замовник схвалює постачання і тепер Filmy доступний для завантаження в App Store. Всі фічі працюють справно, Олівія дякує команді, а Джон уже оголосив у своїх соцмережах про новий додаток для молодих виконавців.

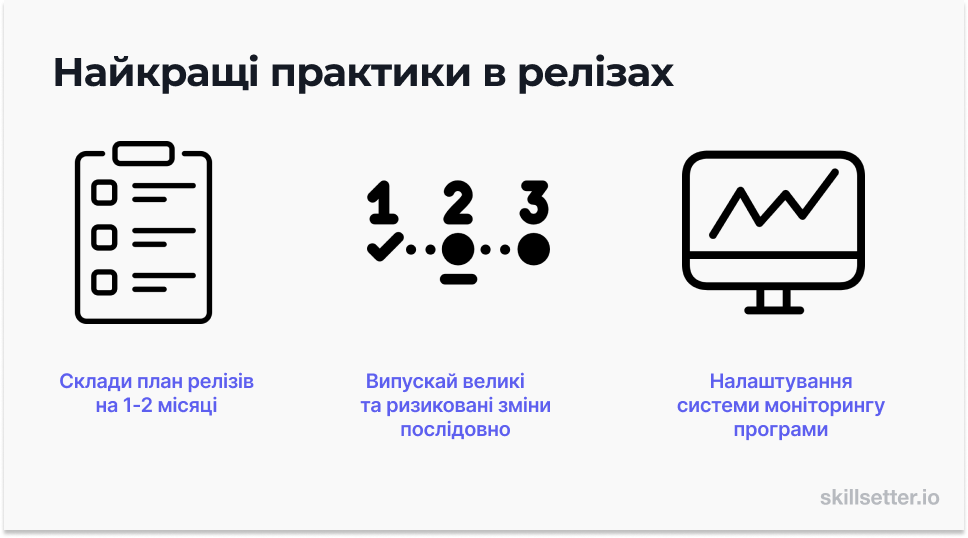
Ви як проджект проводите постачання вперше і з великою кількістю фіч. Багато може піти не так, навіть якщо ви все перевірили заздалегідь. До того ж у цей час Джон випускає новий сингл. Це означає, що до Filmy буде підвищена увага. На кону репутація замовника. Тому присутність при першому постачанні продукту є доречною. Адже, якщо буде важливий реліз, а команда не може залишитись і простежити, що все добре — це проблема менеджера. Значить, він не встежив за процесами та не доніс до розробників важливість постачання.

Але чим частіше проходять постачання, тим менше часу потрібно на їх супровід та підтримку. А це означає, що і присутність усієї команди буде лише витратою ресурсів. Уяви, що ви випускаєтеся щодня. Ніхто не захоче так часто затримуватись на роботі. Та це й не потрібно, адже зміни будуть невеликими.

Наставник.

**Випускайте релізи якнайчастіше.** Кожен проєкт має свій релізний цикл. Він і визначає, як часто виходять постачання. Що частіше, то краще. Ось головні причини чому:





Розглянемо ці практики.

Наставник.

1. По-перше, склади розклад релізів на 1-2 місяці наперед. Якщо твій план буде коротшим, то продукт буде складно продавати. Адже ви не продумали, що постачатимете користувачам. Докладні довгострокові плани не дуже надійні. Раптом за пару місяців проєкт зросте так, що ми випускатимемо щось нове двічі на тиждень. Тоді детальний план із щомісячними релізами на пів року вперед доведеться сильно міняти. Це додаткова робота.

2. Якщо вводиш велику та ризиковану зміну типу повного оновлення інтерфейсу, **викочуй код послідовно**. Спочатку спробуй апгрейд (процес поліпшення функцій або можливостей ) на невеликій частині користувачів. Якщо все працює і їм подобається, то розкочуй на інших. А якщо щось піде не так, то хоча б не всі постраждають.

Питання до проджекта. Додаток Filmy розкотили відразу на всіх, хоча у нас дуже багато фіч. Чи правильне є рішення?

Наставник.

Із досвіду. Минулого року команда розробляла платіжну систему, аналог Paypal. У нас була дуже важлива і чітка дисфункція (порушення або нестправність стану або системи). [Downtime](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true) (час, коли система недоступна. Протилежно до метрики uptime — коли система працює) має дорівнювати нулю, інакше наші партнери почнуть втрачати великі гроші. Відповідно, щоб сервіс не впав, ми постійно відстежували, що відбувається. Для цього **налаштували систему моніторингу**. З нею розробникам стало легше знаходити помилки у продакшені та швидко їх виправляти.

Питання до проджекта. Чи потрібно поговорити про це з тімлідом. Можливо, і в Filmy така система знадобиться?

Випадок Filmy відповідає усім цим вимогам. Ви проводите постачання вперше і з великою кількістю фіч. Багато може піти не так, навіть якщо ви все перевірили заздалегідь. До того ж у цей час Джон випускає новий сингл. Це означає, що до Filmy буде підвищена увага. На кону репутація замовника.

Але якщо буде важливий реліз, а команда не може залишитись і простежити, що все добре — це проблема менеджера. Значить, він не встежив за процесами та не доніс до розробників важливість постачання.

*Відповідь.*

Це додаткова робота та гроші.



## Що робити після першого релізу

Наш проект через тиждень

Користувачі побачили першу версію Filmy ще тиждень тому. Зараз у додатку понад тисячу людей, включаючи інфлюенсерів музичної індустрії.

Ви з командою готуєтеся робити нове постачання, до якого увійде хмарне сховище. Тепер користувачі зможуть зберігати свої проєкти у хмарі та редагувати їх із будь-якого пристрою. Додасться робота з сервером та даними користувачів.

Поки ви готувалися до нового релізу, вам написали перші користувачі. 15 людей поскаржилися, що програма вилітає при збереженні відео. Це велика проблема у продакшені. Якщо її не вирішити, люди почнуть залишати Filmy.

Сьогодні вранці написали ще троє людей, що у них вилітає система.

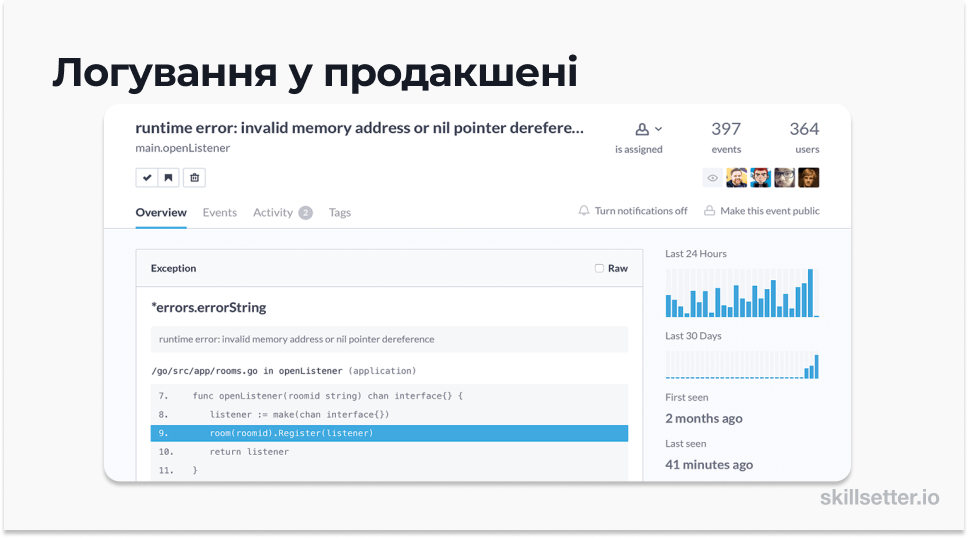
Помилка з'явилася не у всіх користувачів — лише на айпадах.

Публікація відео у соцмережі порушила роботу збереження відео на пристрій.

**Щоб розібратися, ти пишеш Кері якнайшвидше.**

**У тебе виникає питання.** Чи потрібно налаштувати **логування**, щоб легше з таким справлятися?

**А що таке логування?** Це така система, яка записує все, що відбувається у додатку. Ось приклад як вона виглядає.



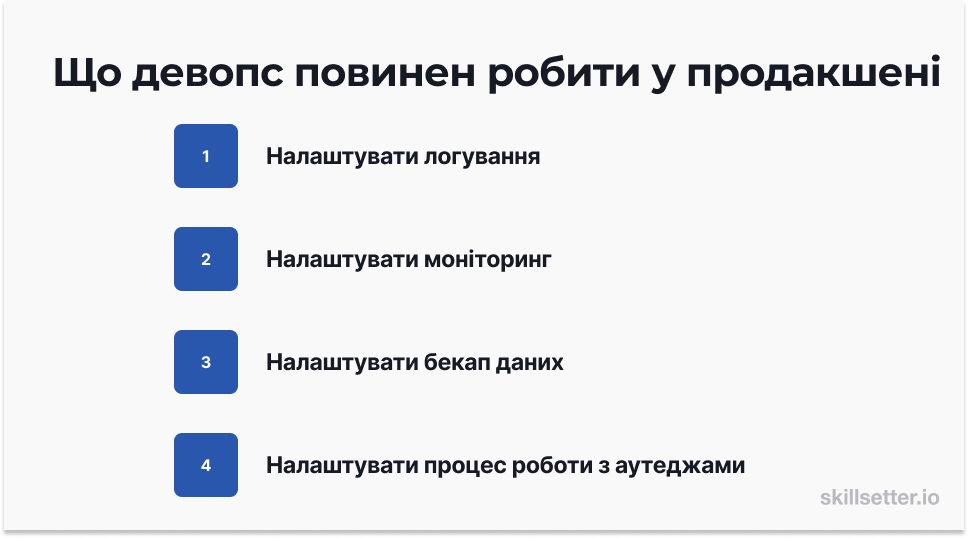
Отже, можна переглянути всі важливі дані про будь-якого користувача: ОС, пристрій, браузер, країну тощо. А ще відстежити послідовність запитів та відповідей, що призвели до бага. Наприклад, наш музикант з iPad Pro із Нью-Джерсі натиснув «Зберегти» та вилетів із Filmy. Це я припустив за відгуками.

**Кері відповідає**. Для мене користь логування очевидна. А з [логуванням](https://docs.google.com/document/d/12_49stcMoN_35crbAorBUryEXE4Fr4if/edit?usp=sharing&ouid=101698225119467330845&rtpof=true&sd=true) все буде, як на долоні. Ось зараз у мене такого інструменту немає. Тому, щоб знайти баг, доведеться довго копатися у коді. З логуванням я подивився б, що відбувалося з системою, знайшов би помилку і пофіксив. І взагалі, так можна не лише шукати, а й аналізувати баги. Через них знаходити вразливості у додатку, які скоро дадуть себе знати. У такому разі, логування нам справді знадобиться.

Ніхто не застрахований від критичних помилок у продакшені. Швидше за все, і Filmy теж рано чи пізно впаде. Питання вже в тому, як швидко команда зможе виправити ситуацію. Тому варто швидше обговорити це з Олівією, щоб домовитися про логування та бекап.

Отже. замовник погоджується на нові задачі, ти додаєш їх у скоуп, і команда починає налаштування. Тепер розробникам буде легше знаходити помилки в продакшені, а дані користувача будуть у безпеці.

Але логування та бекап — не єдині речі, від яких залежить робота додатка після релізу. Розгляньмо всі пункти, які варто враховувати, коли продукт вже випущений. За них відповідає девопс або бекендер в проєктах, де бека зовсім небагато.



**Налаштувати логування.** Повторимо, що це системи, які фіксують усе, що відбувається у додатку, і зберігають це до себе. Відкривши такий сервіс, розробник побачить, на якому моменті стався збій, і зможе швидше знайти та полагодити баг.

**Налаштувати моніторинг.** Цей тип сервісу показує, що відбувається з твоєю системою зараз. Уяви, що розробляєш Netflix із тисячею серверів. За роботою кожного потрібно постійно стежити, щоб у глядачів не переривався перегляд фільмів.

З моніторингом усі ці сервери зібрані на спеціальному дашборді. Кожен можна зайти й подивитися, що за процес там виконується, скільки пам'яті залишилося й інші деталі. Ось [так](https://grafana.wikimedia.org/d/000000273/mysql?orgId=1) (перейти за лінком) виглядає моніторинг у Вікіпедії. А [тут](https://grafana.com/blog/2019/05/16/worth-a-look-public-grafana-dashboards/) можна повивчати інші популярні сервіси.

У всіх моніторингах є надважлива фіча — надіслати повідомлення до будь-якого месенджера або навіть зателефонувати, якщо система зафіксувала серйозну помилку. Таке сповіщення налаштовується на людину, яка у твоїй команді відповідає за роботу сервісу — зазвичай, на девопса.

**Налаштувати бекап даних.** Як ти вже знаєш, дані користувачів необхідно зберігати. Особливо, якщо ви працюєте з користувачами, що платять. Якщо їхня інформація загубиться, то репутація бізнесу похитнеться. Щоб цього не сталося, девопси налаштовують автоматичне резервне копіювання даних, наприклад, раз на добу. Це і є бекап.

**Налаштувати процес роботи з аутеджами.** Аутедж — це серйозна проблема в системі, яка призводить до її неробочого стану. З постійними аутеджами користувачі втрачають інтерес до продукту, адже він стає часто недоступний.

Май на увазі, що час розробників обмежений, а сервер повинен працювати практично безперервно. Тому дізнайся, хто з команди відповідатиме за роботу серверів. Потрібно, щоб усі необхідні люди були доступні для раптового фіксування. А потім обговори час аптайму із замовником: на доступність та підтримку системи потрібен додатковий бюджет.

Якщо програма раптом впала, насамперед повідом представників бізнесу. Адже це їм пишуть незадоволені клієнти, яким треба щось відповідати. Але навіть коли всі «пожежі» загашені, робота з аутеджем не закінчується.

Після того як аутедж усунений, треба провести **міні-ретроспективу** інциденту. А замовник відчуватиме турботу і залученість, якщо ти **напишеш лист із висновками (post-mortem)** з того, що трапилося, і вашими діями, щоб проблема не повторилася. Також необов'язково перевіряти систему на інші вимоги. Адже інцидент був пов'язаний із сервером — він ніяк не торкається продуктивності програми, зрозумілості інтерфейсу та інших нефункціональних параметрів. І нарешті, про аутедж треба говорити не лише з тімлідом, а й з усією командою. Це досить великий інцидент, щоб винести його на загальне ретро.

Після того як ви з командою зрозуміли та виправили помилки, замовника потрібно заспокоїти. Напиши йому невеликий репорт про те, що трапилося, наприклад:

**Що трапилось**: Була здійснена DDoS атака на сайт. На вершині атаки робилось близько 8000 запитів.

**Наслідки**: Через перезавантаження сайт впав і не працював 40 хвилин.

**Рішення**: Відфільтрували та заблокували підозрілі IP-адреса.

**Висновки** (що ми зробимо, щоб проблема не повторилася): Налаштуємо [CloudFlare](https://www.cloudflare.com/), щоб швидше відбивати атаки середнього рівня. Більше про CloudFlare (**Інтегрована глобальна хмарна мережа**) можна дізнатися [тут](https://www.cloudflare.com/).

Як ти пам'ятаєш, після першого релізу Filmy у вас теж стався інцидент. Він не був пов'язаний з аутеджем, але торкнувся значної частини ваших користувачів.

## Завдання. Напиши листа замовнику про проблему на айпадах, з якою ви зіткнулися в продакшені. Щоб описати подробиці, повернися до діалогу з Кері вище. Використовуйте для листа такий шаблон.

**Що трапилося**:

**Збитки**:

**Рішення**:

**Висновки**:

###### Відповідь

**Що трапилося**: У користувачів з айпадами вилітала система при збереженні відео.

**Збитки**: Частина користувачів не могла зберегти свої проєкти. Скарги надійшли від 15 користувачів з айпадами.

**Рішення**: Розробники знайшли й полагодили баг у системі. Сусідня фіча — публікація відео в соцмережі — порушила роботу збереження відео на телефон. Від першого звернення до викочування фіксу пройшло 3 дні.

**Висновки** (що ми зробимо, щоб проблема не повторилася): Впровадили логування, щоб швидше знаходити подібні баги у продакшені та оперативніше їх усувати. Задачу додано до багтрекера:

Відправляючи замовнику подібні репорти ти не тільки тримаєш його в курсі, але й даєш зрозуміти, що тобі важлива його участь у процесі. Навіть якщо він не реагуватиме на ці листи, надсилай їх заради звітності. Потім, за можливих конфліктів і непорозуміння, їх можна використовувати як аргументи.

**Відбулося ще декілька релізів Filmy**

Ще через пару релізів база активних користувачів Filmy досягла 30 000, а Джон з Олівією вирішили розвивати проєкт далі. Вони не хочуть відпускати команду, тому продовжують контракт.

Ви з командою встигли спрацюватись, регулярно потрапляєте в оцінки та швидко реагуєте на зміни у продукті. Вам знадобився час, щоб прийти до цього, але результат — оцінка 4.8 на App Store, щасливі користувачі та замовник, готовий розвиватися з вами далі.

# Додаток.

# Тестування програмного забезпечення

Практичне заняття з Google Firebase по unit тестам:

<https://coursera.org/share/6ff14a7f4ee031e56d3c4ecd44c1770f>

<https://coursera.org/share/a43796c5328fd6d4b59a4f2196ac03e9>

<https://coursera.org/share/8224c93760fe1712b001fe376cacf326>

<https://coursera.org/share/3f17cb5a7127ef13fd3f95d85f8fe22a>

Практичне заняття з Google Firebase по інтеграційним тестам:

<https://coursera.org/share/a05a666b83f9e6f9180253751a7211eb>

Практичне заняття з Selenium по системним тестам:

<https://coursera.org/share/13e0be767f120de1ee177d9ff3577365>

<https://coursera.org/share/aa7a8b811a0a1990462cdb9bbccfc729>

## Зразок коду

**Зразок програми від HVAC in a Hurry на JSFiddle:** [**створення та керування користувачами за допомогою Google Firebase + засіб перевірки домену**](https://jsfiddle.net/acowan/tsk607z9/) **.** Це зразок програми, код якої наведено в тестових прикладах нижче.

**Приклад модульного тесту на JS Fiddle:** [**засіб перевірки домену модульного тестування (з Jasmine)**](https://jsfiddle.net/acowan/esd4hL7r/) **.** Це приклад модульного тесту, який ми розглянули на тижні 2.

**Практична задача модульного тестування на JS Fiddle** : [**Тест Jasmine Hello World, Running oaccess n jsFiddle**](https://jsfiddle.net/acowan/ya2dct10/) . Це простий приклад «Hello World», з яким ви можете пограти, якщо вам це цікаво.

**Приклад тесту інтеграції на JS Fiddle** : [**перевірка домену тестування інтеграції (з Jasmine)**](https://jsfiddle.net/acowan/jynd54h0/) .

**Зразок тестування системи для Selenium IDE** : [**HinH-Register-v5.side**](https://www.alexandercowan.com/wp-content/uploads/2018/10/HinH-Register-v5.side_.zip) . Це приклад тесту, який можна завантажити в Selenium IDE. **Кілька ключових зауважень щодо цього.**По-перше, якщо ви запустите його як є, він завершиться помилкою про повторюваного користувача, доки ви не оновите значення в команді, де ви вводите ідентифікатор користувача, на щось інше, ніж те, що там зараз – див. ціль «id=register\_email». По-друге, нова версія (3.4.2), здається, фактично не виконує команду Pause, тому я замінив її новою командою «чекати на присутність елемента». Ви помітите це на передостанньому кроці, де в демонстрації з’явилася команда «Пауза». Нарешті, просто пам’ятайте, що Selenium IDE і характер того, що вона робить, є трохи нестабільними! Якщо у вас виникли проблеми з цим наявним прикладом, спробуйте записати свій власний. Коли ви починаєте, не турбуйтеся про твердження тощо. Просто запустіть кілька команд, переконайтеся, що це працює, а потім додайте більше вмісту (включно з твердженнями чи іншими тестами) у міру просування.

**Отримання Selenium IDE** : нагадуємо, що це плагін браузера для Chrome або Firefox. Найкращий спосіб отримати це – просто загуглити «отримати Selenium IDE [Chrome]».

**Сторінка проекту та огляд Jasmine** : [**Jasmine (Jasmin, керований поведінкою)**](https://jasmine.github.io/)

1. це перевірка системи на стабільність [↑](#footnote-ref-1)
2. Мокінг (від англ. "mocking") в контексті програмування є підходом до тестування, коли створюються штучні замінники реальних об'єктів або компонентів системи, які ще не готові або важко доступні для тестування. Це дозволяє тестувати взаємодії між компонентами системи, незалежно від їх реальної реалізації. [↑](#footnote-ref-2)
3. [**Тестування безпеки**](https://www.guru99.com/what-is-security-testing.html) — це тип тестування програмного забезпечення, який виявляє вразливості, загрози та ризики у програмі. Він також запобігає атакам з боку зловмисників. Мета тестів безпеки — виявити всі можливі лазівки та слабкі місця в системі, які можуть призвести до втрати інформації. [↑](#footnote-ref-3)
4. [**Тестування продуктивності**](https://www.guru99.com/performance-testing.html) — це перевірка швидкості, часу відгуку, стабільності, надійності, масштабованості та використання ресурсів ПЗ за певного навантаження. [↑](#footnote-ref-4)
5. **CEO (Chief Executive Officer)** – це головний виконавчий директор, тобто «Перша особа» компанії, ТОП-менеджер, голова правління, генеральний директор, головний адміністратор чи інший керівник, відповідальний за управління всією організацією. [↑](#footnote-ref-5)