**Перевірка вимог**

Якщо бути точними, валідація та верифікація є двома різними видами діяльності в розробці програмного забезпечення. Верифікація визначає, чи результат певної стадії розробки відповідає вимогам (діяльність виконувалася правильно). Валідація оцінює, чи продукт задовольняє потреби клієнтів (результатом діяльності є правильний продукт).

Якщо перевести ці визначення на вимоги, то верифікація визначає, чи правильно були написані вимоги, тобто написані на високому рівні якості. Валідація вимог визначає, чи були написані правильні вимоги, такі, що відповідають бізнес-цілям. Ці два поняття тісно переплітаються. Для спрощення далі мова буде йти про перевірку (валідацію) вимог, але методи, які будуть описані, стосуються як і правильних вимог, так і вимог високої якості.

Валідація вимог дозволяє команді проекту створити правильне рішення, яке відповідає заявленим бізнес-цілям. Активності процесу перевірки вимог потрібні для визначення, що:

* Вимоги до програмного забезпечення точно описують передбачувані можливості та властивості системи, які задовольнять потреби різних зацікавлених сторін.
* Вимоги до програмного забезпечення виведені з бізнес-вимог, системних вимог, бізнес-правил та інших джерел.
* Вимоги є повними, їх можна практично реалізувати та перевірити.
* Усі вимоги необхідні, і весь їх набір є достатнім для досягнення бізнес-цілей.
* Усі представлення вимог узгоджені одне з одним.
* Вимоги забезпечують достатню основу для продовження проектування та розробки.

Валідація — це не один окремий етап, який виконується після виявлення та документування всіх вимог. Деякі діяльності валідації, такі як поступовий перегляд зростаючого набору вимог, проходять протягом ітераційних процесів виявлення, аналізу та специфікації. Інші види діяльності, такі як формальні інспекції, забезпечують остаточну оцінку якості перед проходженням базової лінії для набору вимог.

**Рецензування вимог**

Кожного разу, коли хтось, крім автора, перевіряє продукт на наявність проблем, відбувається рецензування (peer review). Рецензування — це потужний метод виявлення неоднозначних або неперевірених вимог, вимог, які визначені недостатньо чітко для початку проектування, та інших проблем. Різні види рецензування мають різні назви. Неофіційні перегляди корисні для інформування інших людей про продукт і збору неструктурованих відгуків. Однак вони не є систематичними, ретельними або послідовно виконаними. Підходи до неофіційного перегляду включають:

* перевірка "за столом" (peer deskcheck), коли ви просите одного колегу проаналізувати ваш продукт;
* колективна перевірка (passaround), коли ви запрошуєте декілька колег для паралельної перевірки продукту;
* наскрізний розбір (walkthrough), коли автор описує продукт, що створюється і просить його прокоментувати.

Неформальне рецензування добре підходить для виявлення явних помилок, протиріч і прогалин. Такі рецензії можуть допомогти виявити формулювання, що не відповідають характеристикам високоякісних вимог. Але рецензенту складно самотужки виявити всі неоднозначні вимоги. Він може прочитати вимогу і подумати, що розуміє її, і перейти до наступної. Прочитавши цю ж вимогу, інший рецензент може інтерпретувати її інакше, але також не побачити проблеми. Якщо ці два рецензенти не обговорять вимогу, неоднозначність може залишитись непоміченою і проявитися лише на пізніх етапах проекту.

Формальне рецензування є суворо регламентованим процесом. Після завершення формується звіт, в якому вказані матеріал, рецензенти та думка команди рецензентів про прийнятність продукту. Головний результат — сукупність усіх знайдених дефектів та порушених питань. Члени команди, яка виконує офіційну рецензію продукту, спільно відповідають за якість рецензії, хоча врешті-решт за якість продукту відповідають його творці. Форма офіційного рецензування, що найбільш добре себе зарекомендувала, це експертиза (inspection) — найдієвіший із доступних прийомів перевірки якості ПЗ.

**Процес експертизи**

Процес експертизи був розроблений Майклом Феганом в 1976 році. Експертиза була визнана рекомендованим прийомом у галузі розробки ПЗ. Цей метод підходить для перевірки будь-якого продукту, у тому числі документації вимог та дизайну, вихідного коду, документації тестування та планів проекту.

Експертиза – це чітко визначений багатоетапний процес. У ній бере участь невелика команда учасників, які ретельно перевіряють продукт на наявність дефектів та вивчають можливості покращення. Експертизи забезпечують рівень якості продукту, що передує підсумковому рівню.

Учасники експертизи повинні відображати чотири точки зору на продукт:

* Автор продукту та, можливо, колеги автора. Це думка бізнес-аналітика, який розробив документацію вимог.
* Люди, які є джерелом інформації для аспекту, який проходить перевірірку. У цій ролі можуть виступити представники користувачів чи автор попередньої специфікації. За відсутності специфікації вищого рівня, в експертизі повинні взяти участь представники клієнта, щоб підтвердити, що вимоги до специфікації вимог до ПЗ описані правильно і повно.
* Люди, які будуть виконувати роботу, пов’язану з аспектом, що перевіряється. Для експертизи специфікації вимог до ПЗ можна залучити розробника, тестувальника, менеджера проекту, а також розробника документації, тому що вони зможуть виявити проблеми різних типів. Тестувальник швидше за все виявить вимоги, що не піддаються перевірці, а розробник зможе визначити вимоги, які технічно нездійсненні.
* Люди, які відповідають за роботу продуктів, пов’язаних з аспектом, що перевіряється. Вони виявлять проблеми із вимогами до зовнішнього інтерфейсу. Вони також можуть виявити вимоги, зміна яких у специфікації, що перевіряється, вплине на інші системи (ефект хвилі).

**Контрольні списки дефектів**

Щоб допомогти експертам виявити типові помилки в продуктах, що перевіряються, варто розробити контрольний список дефектів для кожного типу документа про вимоги, що створюється у компанії. За допомогою таких контрольних списків ви привернете увагу перевіряючих до проблем, що часто виникають з вимогами. Контрольні списки є нагадуванням. Згодом люди засвоюють певний порядок і вже за звичкою шукають відповідні проблеми у кожному сеансі перевірки.

Повнота

* Чи включені до вимог усі відомі потреби клієнта чи системи?
* Чи відсутня у формулюванні вимоги якась необхідна інформація?
* Якщо так, чи вказано в документації, що вимогу треба доопрацювати?
* Чи визначено всі алгоритми, що відповідають функціональним вимогам?
* Чи визначено все зовнішнє обладнання, програмне забезпечення та інтерфейси взаємодії?
* Чи задокументована очікувана поведінка системи для всіх передбачуваних помилкових умов?
* Чи забезпечують вимоги адекватну основу для дизайну та тестування?
* Чи вказано пріоритет реалізації кожної вимоги?
* Чи є кожна вимога в рамках обсягу проекту, релізу чи ітерації?

Коректність

* Чи конфліктують якісь вимоги з іншими вимогами, чи повторюють їх?
* Чи написано кожну вимогу ясною, точною та недвозначною мовою?
* Чи можна перевірити кожну вимогу за допомогою тестування, демонстрації, перегляду чи аналізу?
* Чи всі перелічені повідомлення про помилки є унікальними та змістовними?
* Чи всі вимоги дійсно є вимогами, а не рішеннями чи обмеженнями?
* Чи можна реалізувати всі вимоги, не виходячи за межі встановлених обмежень?

Атрибути якості

* Чи всі цілі щодо продуктивності, безпеки та захисту сформульовані належним чином?
* Чи всі інші атрибути якості задокументовані та оцінені кількісно із зазначенням прийнятних компромісів?
* Чи були визначені всі критичні за часом функції та був заданий графік реалізації для них?
* Чи вирішено належним чином питання інтернаціоналізації та локалізації?
* Чи всі вимоги до якості можна виміряти?

Організація та можливість відстеження

* Чи всі вимоги організовані логічним та доступним чином?
* Чи коректні всі перехресні посилання на інші вимоги та документи?
* Чи написані всі вимоги на тому самому належному рівні деталізації?
* Чи кожна вимога ідентифікована унікально та коректно?
* Чи вказано для кожної функціональної вимоги посилання на джерело (наприклад, на системну вимогу чи бізнес-правило)?

Інші проблеми

* Чи не пропущено якісь варіанти використання чи потоки процесів?
* Чи не пропущені у варіантах використання будь-які альтернативні напрямки, виключення чи інша інформація?
* Чи визначені всі бізнес-правила?
* Чи не втрачені якісь наочні моделі, які необхідні для ясності та повноти?
* Чи не пропущено якісь специфікації звітів і чи вони повні?

**Рецензування вимог за правилами**

*Плануйте перевірку.*Запросіть різних рецензентів працювати над різними розділами документів. Починайте віддавати набори вимог на рецензування, коли готово, наприклад, 10% роботи, а не коли всі вимоги «готові». Виявлення серйозних дефектів якомога раніше та виявлення систематичних проблем у способі написання вимог – потужний метод запобігання дефектам, а не лише їх пошуку. Дайте рецензентам достатньо часу на рецензування як у сенсі реальних годин, так і календарного часу.

*Надайте контекст.* Повідомте рецензентам контекст документа і, можливо, проекту, якщо вони не працюють над тим самим проектом. Знайдіть рецензентів, які можуть надати корисний внесок завдяки своїм знанням.

*Визначте межі рецензування.* Повідомте рецензентам, який матеріал вивчати, на що звертати особливу увагу та які дефекти шукати. Запропонуйте використовувати контрольний список дефектів. Можна підвищити доступність та кваліфікацію команди рецензентів, попросивши різних перевіряльників перевірити різні розділи або використовувати різні частини контрольних списків.

*Обмежте повторне рецензування.* Рецензент може стомитися від перегляду того самого матеріалу і після третього циклу не помітити серйозні проблеми через «замиленість ока». Якщо потрібно, щоб рецензент переглянув якусь частину кілька разів, щоразу виділяйте зміни, щоб він міг зосередитися саме на них.

*Визначайте пріоритети областей, що рецензуються.* Визначте пріоритети тих частин вимог, які відрізняються високим ризиком або відносяться до функціональності, що рідко використовується. Також пошукайте області вимог, де знайдено небагато дефектів. Це може означати, що ці частини ще не пройшли рецензування, а не відсутність проблем.

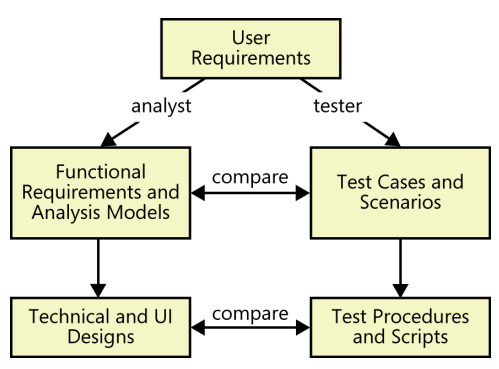
**Тестування вимог**

Тести, створені на основі функціональних або користувацьких вимог, допомагають учасникам проекту отримати уявлення про очікувану поведінку системи. Розробка тестів сама по собі допоможе виявити безліч проблем з вимогами, навіть до виконання цих тестів на готовому ПЗ. Будуть виявлені неясні та двозначні вимоги, тому що ви не зможете описати очікувану реакцію системи. Коли аналітики та клієнти разом створюють тести, вони приходять до загального розуміння того, як продукт працюватиме, і набувають впевненості в тому, що вимога сформульована правильно. Тестування - це потужний інструмент перевірки та затвердження вимог.

Концептуальні тести можна починати створювати, ґрунтуючись на вимогах користувача, вже на ранній стадії процесу розробки. Також тести можна застосовувати для оцінки функціональних вимог, моделей аналізу та прототипів. Тести повинні охоплювати основні потоки всіх варіантів використання, альтернативні потоки, а також винятки, визначені під час виявлення вимог та аналізу. Аналогічно, якщо ви визначили потоки бізнес-процесів, тести повинні охоплювати етапи бізнес-процесів та всі можливі шляхи прийняття рішень.

Ці концептуальні (чи абстрактні) варіанти тестування залежать від реалізації. Наприклад розглянемо варіант використання "Перегляд замовлення". Концептуальні тести можуть бути такими:

* Користувач вводить номер замовлення, яке хоче переглянути, це замовлення існує, користувач розмістив замовлення. Очікуваний результат: відображення деталей замовлення;
* Користувач вводить номер замовлення, яке хоче переглянути, це замовлення не існує. Очікуваний результат: з'являється повідомлення "На жаль, замовлення не знайдено";
* Користувач вводить номер замовлення, яке хоче переглянути, це замовлення існує, але воно розміщено не цим користувачем. Очікуваний результат: з'являється повідомлення «Нажаль, це не ваше замовлення».

В ідеалі бізнес-аналітик пише функціональні вимоги, а тестувальник - тести на основі одних і тих же матеріалів: вимог користувача. Неясності в вимогах користувача і різні інтерпретації виливаються в неузгодженість функціональних вимог, моделей та варіантів тестування. У міру того, як розробники перетворюють вимоги в інтерфейс і технічний дизайн, тестувальники можуть перетворити концептуальні тести в деталізовані процедури тестування для офіційного тестування системи.

В наступному прикладі продемонстровано зв’язок вимог та моделей їх візуалізації з першою стадією створення тестів для Системи відстеження хімікатів.

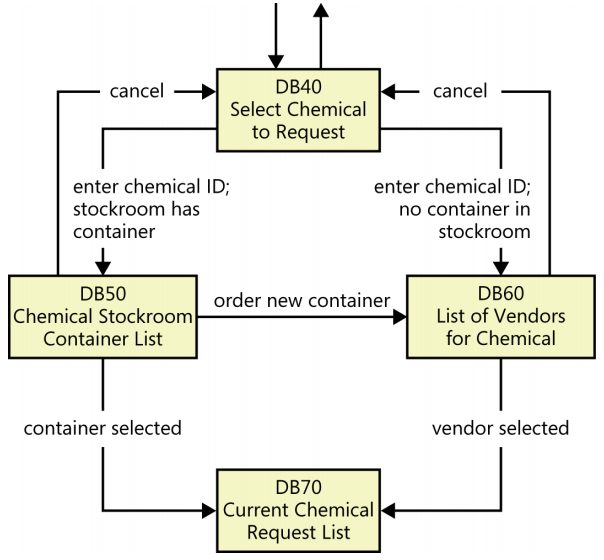
**Бізнес-вимоги.** Однією з основних бізнес-цілей є наступна: *Зменшити витрати на закупівлю хімікатів на 25% до кінця першого року.*

**Варіант використання**, який пов’язаний з цією бізнес-вимогою, називається «Замовлення хімікату». Він дозволяє користувачеві замовити контейнер з хімікатом, який вже є в наявності на складі хімікатів. Ось як звучить опис варіанта використання: *Співробітник, який розмістив замовлення на хімікат, вказує у замовленні необхідний хімікат, вводячи його назву або ідентифікатор або імпортуючи його структуру з відповідного графічного інструменту. Система виконує замовлення, пропонуючи контейнер із хімікатом зі складу або дозволяючи створити замовлення у постачальника.*

**Функціональна вимога.**

1. Якщо на складі є контейнери із хімікатами, на які надійшло замовлення, система надасть список доступних контейнерів.

2. Користувач або вибере один із зазначених контейнерів, або зробить запит на створення нового замовлення контейнера у постачальника.

**Карта діалогу.** На рисунку показано частину карти діалогових вікон для варіанта використання «Замовлення хімікату». Прямокутники представляють зображення інтерфейсу користувача (DB - dialog box), а стрілки показують можливі шляхи переходу від одного екрану до іншого. Ця карта діалогових вікон була створена на досить пізніх етапах розробки вимог, коли учасники проекту почали визначати конкретні екрани, меню, діалогові вікна та інші елементи карти, щоб можна було дати їм конкретні назви і почати думати про можливу архітектуру інтерфейсу користувача.

**Тест.** Оскільки для цього варіанта використання існує декілька можливих шляхів виконання, можна створити декілька тестів для обробки основного потоку, альтернативних потоків та винятків. Нижче наведено один тест, коли користувач представляє список контейнерів, доступних на складі.

*У діалоговому вікні DB40 введіть дійсний ідентифікатор хімікату; на складі є два контейнери з цим хімікатом. Відкриється діалогове вікно DB50, в якому показано ці два контейнери. Виберіть другий контейнер. DB50 закриється, і контейнер 2 буде додано в кінець списку поточних запитів на хімікати у діалоговому вікні DB70.*

А тепер найцікавіше – тестування вимог. Спочатку встановлюється відповідність між кожним варіантом тестування та функціональними вимогами. Треба переконатися, що за існуючого набору вимог усі тести «виконуються», а також, що кожній функціональній вимозі відповідає принаймні один варіант тестування. Тобто треба простежити шлях виконання кожного тесту на карті діалогу.

Відстежуючи шлях виконання кожного тесту, можна знайти некоректні чи пропущені вимоги, виправити помилки на карті діалогу і відшліфувати тести. Припустимо, що після такого “виконання” всіх тестів лінія від DB50 до DB60 на карті діалогу, позначена як “Замовити новий контейнер”, не має відповідного тесту. Можливі два пояснення:

* переміщення від DB50 до DB60 не допускається поведінкою системи. Бізнес-аналітик має прибрати цю лінію з карти діалогів. Якщо в специфікації є вимога, у якій зазначений цей перехід, цю вимогу також необхідно видалити;
* переміщення допускається поведінкою системи, але варіант тестування, який демонструє цю поведінку, відсутній.

Або навпаки, припустимо, що тест ґрунтується на власній інтерпретації варіанта використання тестувальником і дозволяє перейти безпосередньо від діалогового вікна DB40 до DB70. Однак на карті діалогу такої лінії немає, і тому за існуючих вимог тест “виконати” не можна. І знову ж таки, можливі дві інтерпретації, і необхідно визначити, яка з них є істиною:

* перехід від DB40 до DB70 не допускається поведінкою системи, отже тест неправильний;
* перехід від DB40 до DB70 допускається поведінкою системи, але на карті діалогу і, можливо, у специфікації до ПЗ немає вимоги, відповідної цьому тесту.

У цих прикладах аналітик та тестувальник об'єднали вимоги, моделі аналізу та тести для виявлення пропущених, помилкових чи непотрібних вимог задовго до написання будь-якого коду.

**Затвердження вимог із застосуванням критеріїв приймання**

Саме клієнти визначають, чи система задовольняє критеріям приймання (acceptance criteria). Критерій приймання - і, отже, приймальне тестування - є показником, чи задовольняє продукт задокументованим вимогам і чи годиться він для використання в запланованому середовищі. Делегування розробки тестів на прийнятність користувачам є ефективною стратегією розробки вимог.

**Критерії приймання**

Робота з клієнтами над розробкою критеріїв приймання дозволяє затвердити як вимоги, так і їхню реалізацію. Якщо клієнт неспроможний описати, як він оцінить, що система задовольняє конкретну вимогу, отже вимога сформульована недостатньо ясно. Критерії приймання визначають мінімальні умови, за яких систему можна вважати готовою до експлуатації.

Мислення в термінах критеріїв приймання означає зміну формулювання питання «Що вам потрібно робити за допомогою системи?» до «Як ви оцінюєте, що система задовольняє вашим потребам?». Заохочуйте користувачів при визначенні критеріїв приймання використовувати правило SMART, тобто критерії повинні бути конкретними (Specific), вимірними (Measurable), досяжними (Attainable), актуальними (Relevant) та обмеженими у часі (Time-sensitive). Критерії треба визначати так, щоб різні об'єктивні спостерігачі приходили до однакового висновку щодо того, чи виконуються ці критерії.

Визначення критеріїв приймання це не просто ствердження, що всі вимоги реалізовані або всі тести пройдені. Приймальні тести це лише частина критеріїв приймання. Критерії приймання можуть охоплювати такі аспекти, як:

* Визначена високопріоритетна функціональність, яка має бути в наявності та працювати належним чином і без якої продукт не може прийматися та використовуватися. (Іншу заплановану функціональність можна підготувати пізніше або можливості, які не дуже добре реалізовані, можна виправити без затримки початкового релізу.)
* Обов'язкові нефункціональні критерії або метрики якості, які повинні задовольнятися. (Деякі атрибути якості повинні бути хоча б мінімально реалізовані, хоча покращення юзабіліті, зовнішнього вигляду та продуктивності можна відкласти. Від продукту може вимагатися задоволення метрик якості, таких як певний мінімальний період роботи без відмови.)
* Всі інші наявні проблеми та дефекти. (Можна прийняти, що не повинно бути дефектів/помилок вище за певний ступінь серйозності, хоча незначні дефекти можуть бути.)
* Певні юридичні, нормативні чи контрактні умови. (Тільки при їх повному виконанні продукт може вважатися прийнятним.)
* Вимоги до підтримки перенесення та до інфраструктури або інші проектні вимоги, що не стосуються продукту. (Наприклад, до релізу продукту може бути потрібно забезпечити наявність навчальних матеріалів.)

Також буває корисним подумати про «критерії відмови», умови або результати оцінки, які можуть змусити стейкхолдерів не визнати систему готовою до релізу. Слід уникати конфліктуючих критеріїв приймання, наприклад, якщо виконання одного критерію не дозволяє задовольнити інший. Насправді виявлення суперечливих критеріїв приймання — один із способів виявлення суперечливих вимог.

У принципі, якщо всі критерії приймання з історії користувача виконуються, власник продукту приймає рішення про те, що користувацька історія реалізована. Тому клієнти повинні дуже точно формулювати важливі для них критерії приймання.

**Приймальні тести**

Приймальні тести становлять велику частину критеріїв приймання. Ключовим користувачам при прийнятті рішення про спосіб оцінки прийнятності системи варто взяти до уваги найбільш загальні та важливі варіанти використання, коли вони вирішують, як оцінювати прийнятність ПЗ. Фокус робиться на тестуванні основних потоків варіантів використання та їх виключеннях, а не на більш рідких альтернативних потоках.

Цінність написання приймальних тестів полягає в тому, що вони змушують користувачів думати, як система поводитиметься після реалізації. При написанні лише приймальних тестів проблема буде у тому, що вимоги існують лише у головах людей. Відсутність документування та порівняння альтернативних представлень вимог - користувацьких та функціональних вимог, моделей аналізу та тестів - може призвести до того, що ви помітите деякі помилки, невідповідності та прогалини.

Варто автоматизувати приймальне тестування. Це знадобиться, коли тести доведеться повторювати після внесення змін та додавання додаткової функціональності у наступних ітераціях чи релізах. Приймальні тести повинні орієнтуватися на нефункціональні вимоги та підтверджувати, що цілі по продуктивності досягаються на всіх платформах, система відповідає стандартам юзабіліті та всі відповідні вимоги користувача реалізовані.

Частина приймального тестування може виконуватись користувачами вручну. Тести користувацького приймального тестування повинні виконуватися після того, як набір функціональності вважається готовим до релізу. Це дозволяє користувачам спробувати та ознайомитися з працюючим ПЗ до його офіційного випуску. Клієнт або представник стейкхолдера повинен вибрати тести для приймального тестування, які охоплюють найбільш ризиковані частини системи. Приймальні тести повинні підтвердити, що рішення робить те, що повинно робити. Під час створення цих тестів треба використовувати близькі до реальності тестові дані. Інакше користувач, який виконує приймальне тестування, може помилково повідомити дефект просто тому, що звіт йому здається неправильним або він помітив неправильне обчислення через некоректність даних.

**Прототипування як спосіб зниження ризику**

"Дізнаюся, коли побачу" (IKIWISI - "I'll know it when I see it") - найстрашніша фраза для бізнес-аналітика. Відразу в уяві виникає картина: команда розробників втілює свої найкращі ідеї в розробці ПЗ, а користувачі потім кажуть: "Ні, не те, робіть все заново". Насправді, уявити майбутню програмну систему та ясно викласти вимоги до неї — нелегке завдання. Багатьом людям складно описати свої потреби, не маючи чогось конкретного перед собою, без зразка, та й критикувати набагато легше, ніж створювати.

Створення прототипів ПЗ робить вимоги реальнішими, наближає варіанти використання до реальності і закриває прогалини у розумінні вимог. Обговорення прототипів на ранніх стадіях процесу розробки допомагає стейкхолдерам дійти до загального розуміння вимог до системи, що зменшує ризик невдоволення замовників.

**Що таке прототипи і навіщо вони потрібні**

Прототип ПЗ - це часткове, можливе і попереднє втілення запропонованого нового продукту. Прототипи служать трьом основним цілям, які повинні бути зрозумілі з самого початку:

* **Уточнення, остаточне формулювання та затвердження вимог.** Прототип, що використовується як інструмент формулювання вимог, дозволяє досягти консенсусу, виявити помилки та оцінити точність і якість вимог. Оцінка прототипу користувачами вказує на помилки формулювання вимог, які можна виправити без великих витрат до створення реального продукту. Це особливо корисно у тих частинах системи, які не дуже зрозумілі або особливо ризиковані чи складні;
* **Створення альтернативного дизайну.** Як інструмент проектування, прототип дозволяє зацікавленим сторонам досліджувати різні методи взаємодії з користувачем, уявляти собі кінцевий продукт, оптимізувати зручність використання системи та оцінювати потенційні технічні підходи. Прототипи можуть продемонструвати здійсненність вимог на прикладі робочих дизайнів. Вони корисні для перевірки розуміння вимог розробником перед створенням фактичного рішення.
* **Створення підмножини, яка переросте в остаточний продукт.** Прототип, який використовується як інструмент конструювання, є функціональною реалізацією підмножини продукту, яка може бути розвинута до повноцінного продукту через послідовність невеликих циклів розробки. Це безпечний підхід, лише якщо прототип ретельно розроблений із задумом про можливий реліз із самого початку.

Основна мета створення прототипів – усунення неясностей на ранніх стадіях процесу розробки. І тому не потрібен прототип всього продукту. Треба зосередитися на високоризикових областях і відомих неясностях і вирішувати, для яких частин системи необхідний прототип і що ви сподіваєтеся з'ясувати, оцінюючи його. Прототип корисний для виявлення та усунення двозначних і неповних тверджень у вимогах. Користувачі, менеджери та інші зацікавлені особи проекту вважають, що прототип дає їм щось конкретне для розглядання, доки реальний продукт документується та розробляється. Для кожного прототипу, що створюється, обов'язково вказуйте (а також інформуйте стейкхолдерів), чому він створюється, що ви хочете дізнатися з його допомогою і що ви робитимете з прототипом після його оцінки.

**Макети та експериментальні зразки**

Коли люди говорять прототип ПЗ, вони зазвичай мають на увазі *макет* (mockup) передбачуваного інтерфейсу користувача. Його також називають *горизонтальним прототипом* (horizontal prototype). Такий прототип орієнтований на якусь частину інтерфейсу користувача і не зачіпає рівні архітектури або деталі функціональності. Такий прототип дозволяє користувачам досліджувати поведінку майбутньої системи у тих чи інших ситуаціях для уточнення вимог. Макет також дозволяє користувачам з'ясувати, чи зможуть вони за допомогою системи, що базується на прототипі, виконувати свою роботу.

*Експериментальний зразок* (proof of concept), також званий *вертикальним прототипом* (vertical prototype), втілює зріз функціональності програми від інтерфейсу користувача через усі рівні технічних сервісів. Експериментальний зразок діє як справжня система, оскільки торкається всіх рівнів її реалізації. Експериментальний зразок розробляється, коли є сумніви щодо здійсненності та стабільності передбачуваного підходу до архітектури системи чи коли треба оптимізувати алгоритми, оцінити запропоновану схему бази даних чи перевірити критично важливі часові вимоги.

**Одноразові та еволюційні прототипи**

Перш ніж створювати прототип, треба визначитися, чи буде він потрібен після оцінки користувачами або буде розвинений у частину продукту. *Одноразові прототипи* (throwaway prototype) створюються, щоб відповісти на питання, вирішити неясності та покращити вимоги до ПЗ. Якщо вирішено, що після виконання завдання робота з прототипом буде припинена, його варто конструювати якнайшвидше і дешевше.

Не варто знищувати макет, якщо існує можливість його повторного використання в майбутньому. Тим не менш, він не повинен стати частиною кінцевого продукту. Тому його можна називати *прототипом, що не випускається* (nonreleasable prototype).

*Каркас* (wireframe) — певна форма одноразового прототипу, яка використовується для дизайну нестандартних інтерфейсів користувача або веб-сайтів. Каркаси можна використовувати для кращого розуміння трьох характеристик сайту:

* концептуальних вимог;
* архітектури інформації чи дизайну навігації;
* детальний (high-resolution) дизайн сторінок.

Сторінки підготовлені в чорновому варіанті в процесі пошуку концептуальних вимог необов'язково повинні бути схожими на остаточні варіанти сторінок. Цей тип каркасу застосовується для роботи з користувачами, щоб зрозуміти, які дії вони можуть захотіти виконувати на екрані. Другий тип каркасу може взагалі не торкатися дизайну сторінок. Модель аналізу «карта діалогових вікон» - чудовий інструмент для вивчення та покрокового проходу навігації сторінками веб-сайту. Третій тип каркасу заглиблюється у деталі остаточного зовнішнього вигляду сторінок.

На відміну від одноразового прототипу, *еволюційний прототип* (evolutionary prototype) є міцним архітектурним «фундаментом» для поступового створення остаточного продукту — у міру прояснення вимог. Гнучка технологія (agile) - приклад використання еволюційного прототипу. Команди agile-розробки розробляють продукт у процесі серії ітерацій, використовуючи відгуки користувачів на ранніх ітераціях для коригування напрямів майбутніх циклів розробки. Це суть застосування еволюційних прототипів.

Таблиця 1 - Типові застосування прототипів ПЗ

|  | **Одноразовий** | **Еволюційний** |
| --- | --- | --- |
| **Макет** | Прояснення та уточнення користувацьких та функціональних вимог  Виявлення пропущеної функціональності  Дослідження можливих варіантів інтерфейсу користувача | Реалізація базових вимог користувача  Реалізація додаткових потреб користувача за пріоритетами  Реалізація та доопрацювання веб-сайтів  Адаптація системи до вимог бізнесу, які швидко змінюються |
|
|
|
| **Експериментальний зразок** | Демонстрація технічної здійсненності  Оцінка продуктивності  Отримання інформації для уточнення оцінок | Реалізація та нарощування основної багаторівневої функціональності та рівнів комунікації  Реалізація та оптимізація основних алгоритмів  Тестування та налаштування продуктивності |
|
|
|

На рис. 1 показано декілька способів комбінування різних видів прототипів. Наприклад, інформацію, отриману в результаті створення серії одноразових прототипів, можна використовувати для уточнення вимог, які потім реалізуються крок за кроком через послідовність еволюційних прототипів. Інший спосіб, показаний на рис. 1, це застосування одноразового макету для прояснення вимог перед розробкою остаточної версії дизайну інтерфейсу, в той час як одночасно за допомогою експериментального зразка перевіряється архітектура системи та базові алгоритми.

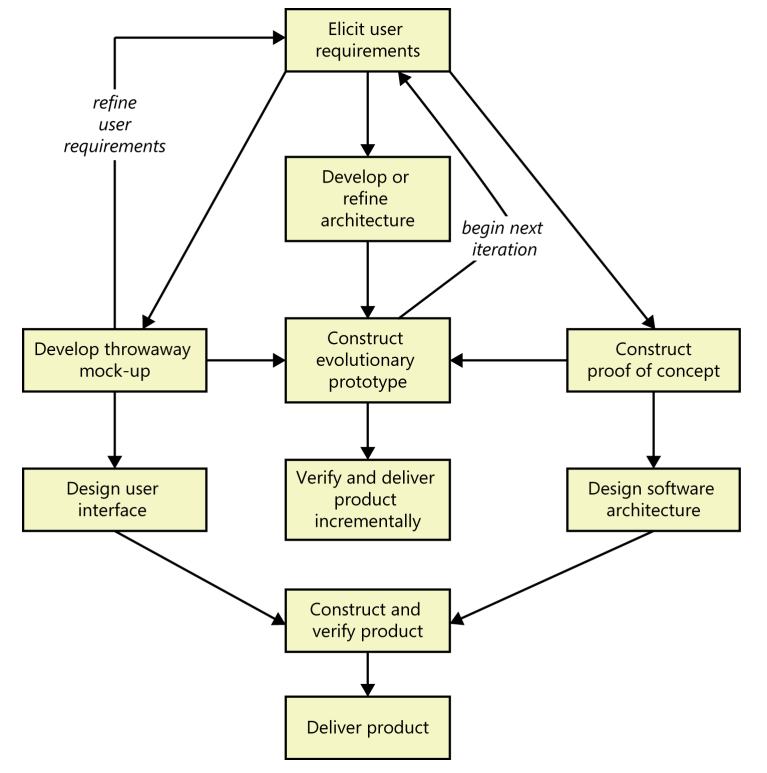


Рисунок 1 - Декілька можливих способів використання прототипів в процесі розробки ПЗ

**Паперові та електронні прототипи**

Не завжди для вирішення невизначеностей у вимогах потрібен прототип у вигляді коду, що виконується. *Паперовий прототип* (іноді його називають низькоякісним прототипом) - це дешевий, швидкий і низькотехнологічний спосіб з'ясувати, як може виглядати якийсь фрагмент системи. Паперові прототипи допомагають встановити, чи дійсно користувачі та розробники однаково розуміють вимоги. Вони дозволяють спробувати, практично не ризикуючи, зробити пробний крок у простір можливих рішень продукту до розробки виробничого коду продукту. Схожий метод, званий методом *розкадрування* (storyboard), показує пропонований інтерфейс користувача, без залучення користувачів до роботи з ним. Використовуються низькоякісні прототипи для вивчення функціональності та потоків, а високоякісні - для визначення точного зовнішнього вигляду та поведінки.

Для створення паперових прототипів потрібні інструменти не складніші, ніж папір, картки, наклейки та дошка. Дизайнер робить нариси екранів, як він їх представляє, не дбаючи про те, де точно розташовуватимуться елементи управління і як вони виглядатимуть. Користувачі охоче діляться своєю думкою про намальоване на папері/дошці, тоді як в деяких випадках вони не схильні критикувати комп'ютерний прототип, який їм полюбився, в який розробник, очевидно, вклав багато праці. Розробники також іноді противяться внесенню істотних змін до ретельно виконаного електронного прототипу.

Для створення *електронного одноразового прототипу* на вибір пропонується багато відповідних інструментів: від засобів малювання до серійних інструментів створення прототипів та графічних конструкторів інтерфейсу користувача. Існують також спеціалізовані інструменти для створення каркасів веб-сайтів. Такі інструменти дозволять з легкістю реалізовувати і оновлювати компоненти інтерфейсу користувача, незалежно від неефективності коду, що стоїть за ним. Звичайно ж, якщо ви створюєте еволюційний прототип, то маєте використовувати високоякісні засоби розробки із самого початку.

На ринку існує багато інструментів, що дозволяють моделювати застосунок до його побудови. Моделювання застосунків дозволяє швидко збирати макети сторінок, елементи управління інтерфейсу користувача, шляхів навігації та функціональності в щось, що дуже схоже на продукт, який передбачається створити. Можливість покрокового моделювання надає цінний механізм взаємодії з представниками користувачів над уточненням вимог та перегляду рішення.

**Робота з прототипами**

На рис. 2 показана одна з можливих послідовностей кроків розробки від варіантів використання до деталізованого дизайну користувача користувача через побудову одноразового прототипу.

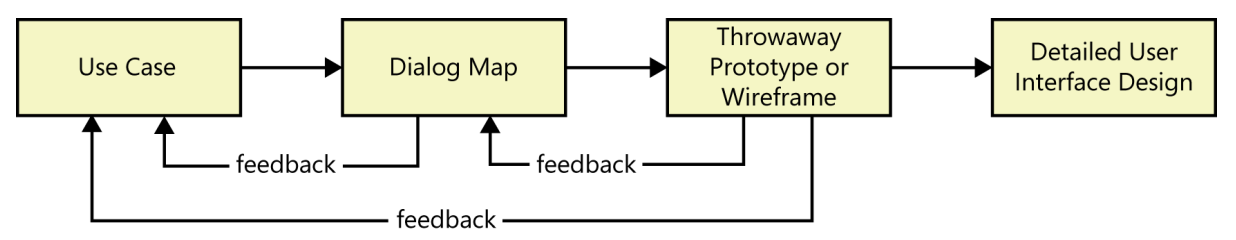


Рисунок 2 - Послідовність дій від варіантів використання до дизайну

Щоб наочно прослідити весь процес, подивимося на реальний приклад невеликого веб-сайту для просування книги мемуарів.

Таблиця 2 - Деякі варіанти використання для сайту письменника

| **Клас користувача** | **Варіанти використання** |
| --- | --- |
| Відвідувач | Отримати інформацію про книгу  Отримати інформацію про автора  Читати обрані розділи  Читати блог  Зв'язатися з автором |
| Клієнт | Замовити продукт  Завантажити електронну версію продукту  Звернутися за допомогою у вирішенні проблеми |
| Адміністратор | Керувати списком продуктів  Здійснити повернення коштів клієнту  Керувати списком повідомлень електронної пошти |

На рис. 3 показано частину карти діалогових вікон, що ілюструє концепцію архітектури сторінок. Кожен прямокутник представляє сторінку, яка братиме участь в операціях, описаних у варіантах використання. Стрілки є посиланнями, що забезпечують навігацію між сторінками.

Наступний етап - побудова одноразового прототипу чи каркасу обраних сторінок вироблення концепції візуального дизайну. Кожну сторінку можна накидати на папері від руки або створити модель за допомогою спеціального засобу візуального дизайну або створення прототипів.

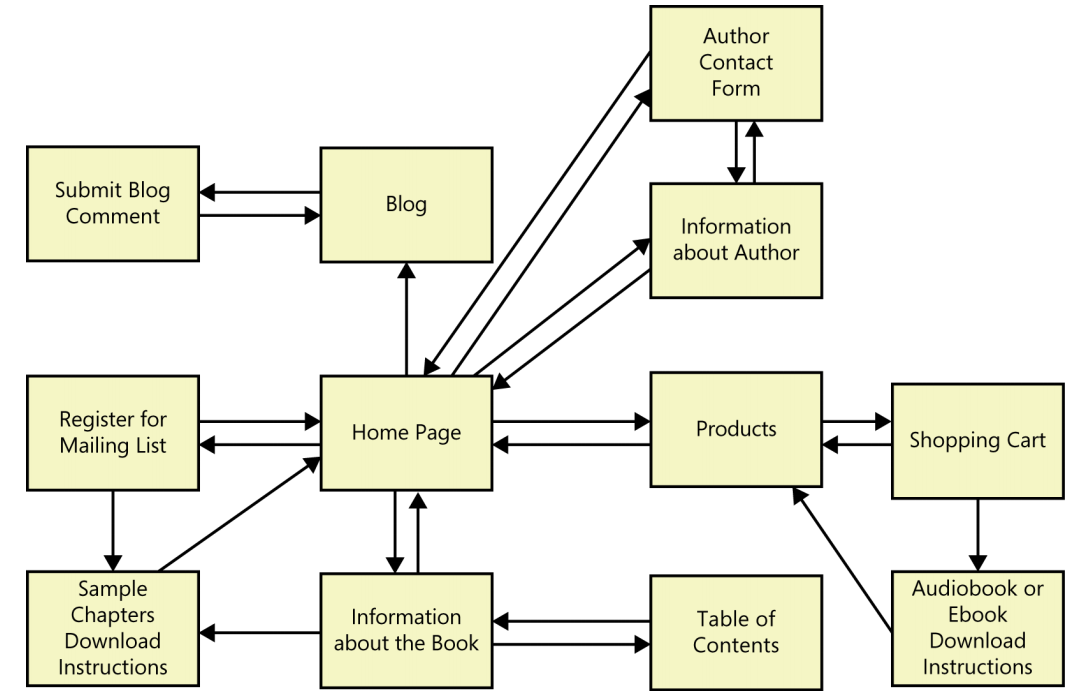


Рисунок 3 - Карта діалогів сайту

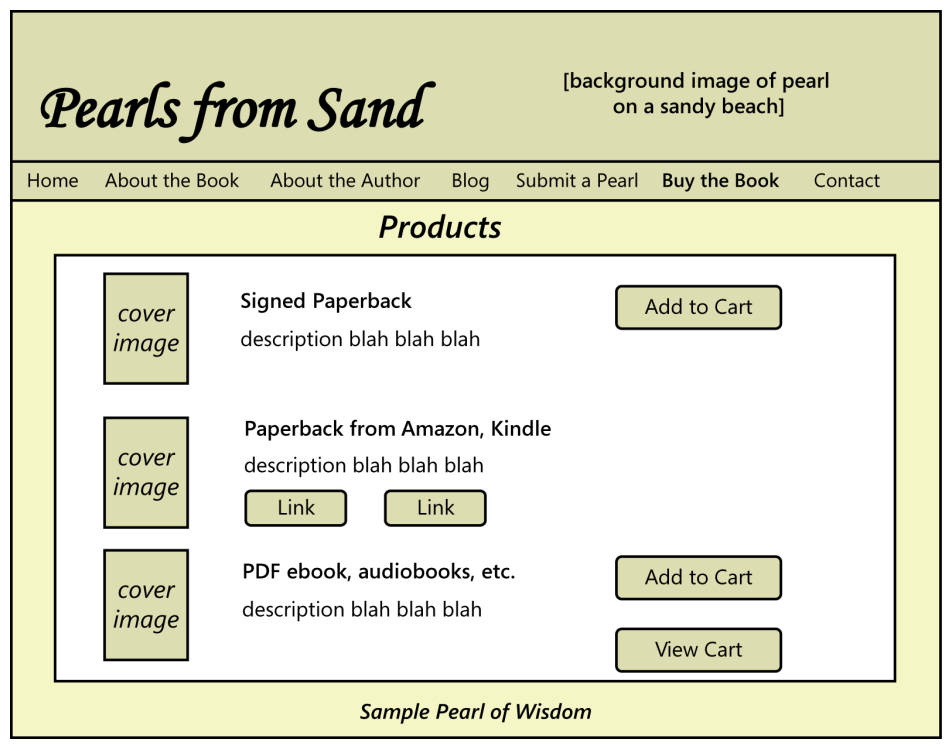


Рисунок 4 - Пример каркасу (wireframe) однієї сторінки сайту

Останній етап - створення детального дизайну інтерфейсу користувача. Такий поетапний підхід до дизайну інтерфейсу користувача дозволяє отримати більш якісні результати, ніж негайний перехід до детального дизайну сторінки без чіткого розуміння того, що можуть хотіти представники різних класів користувачів зробити при відвідуванні веб-сайту.

**Оцінка прототипу**

Оцінка прототипу пов'язана із тестуванням зручності використання. Ви дізнаєтесь більше, спостерігаючи за роботою користувачів з прототипом, ніж просто запитуючи їхню думку. Переконайтеся, що правильно підібрали групу користувачів для оцінки того чи іншого прототипу. Залучайте користувачів багатьох класів як досвідчених користувачів, так і новачків.

Для покращення оцінки горизонтальних прототипів створюйте сценарії, які проводять користувачів через послідовність дій, і ставте конкретні питання, щоб виявити потрібну інформацію. Сценарії оцінки складайте на підставі варіантів використання, історій користувача або функцій, які повинен представляти прототип. Сценарій попросить користувачів виконати певні завдання та проведе їх через неясні області прототипу. Наприкінці виконання кожного завдання і, можливо, у проміжних пунктах, сценарій задає пов'язані із завданнями питання, наприклад:

* Чи прототип реалізує всі необхідні функції так, як ви цього очікували?
* Чи не пропущено в прототипі будь-яку функціональність?
* Чи помітили ви будь-які можливі умови для виникнення помилки, які прототип не враховує?
* Чи немає у прототипі якихось непотрібних функцій?
* Наскільки логічною та повною вам здається навігація?
* Чи можна на вашу думку спростити якісь завдання, в яких занадто багато інтерактивних кроків?
* Чи ви знаходились у ситуації, коли не знали, що робити далі при роботі з системою?

Уникайте показувати користувачам «правильний» шлях реалізації тих чи інших функцій у прототипі та фіксуйте все, що дізнаєтесь у процесі оцінки прототипу. Макет дозволить уточнити специфікацію вимог до ПЗ. Для експериментального зразка задокументуйте процес та результати оцінки, у тому числі рішення, які ви прийняли про досліджені технічних процесів.

**Фактори успіху при використанні прототипів**

Використання прототипів ПЗ пропонує потужний набір методів, які можуть дозволити скоротити терміни розробки, задовольнити клієнтів та створити продукти високої якості. Щоб зробити прототипи ефективною частиною процесу складання вимог, варто притримуватися наступних рекомендацій:

* Додайте завдання створення прототипів в план свого проекту. Складіть графік розподілу витрат часу та коштів на розробку, оцінку та модифікацію прототипів.
* Сформулюйте мету кожного прототипу до початку його створення та поясніть, що станеться з ним у майбутньому: прототип буде відкинутий (або переміщений до архіву), а отримана з його допомогою інформація буде збережена, або він переросте в кінцеве рішення.
* Плануйте створення кількох прототипів, бо скоріш за все не вдасться створити те, що потрібно, з першої спроби.
* Одноразові прототипи створюйте якомога швидко та дешево. Вкладайте щонайменше зусиль у підготовку прототипів, які допоможуть відповісти на запитання або вирішити неясності у вимогах. Не доводьте до досконалості одноразові прототипи.
* Не вбудовуйте в одноразовий прототип детальну перевірку вхідних даних, методи безпечного програмування, обробку помилок або документацію коду.
* Не створюйте прототипи елементів, які не викликають питань, крім випадків дослідження альтернативних варіантів дизайну.
* Використовуйте правдоподібні дані у екранних формах та повідомленнях прототипу. Користувачі будуть відволікатися на нереалістичні дані та не зможуть сприйняти прототип як модель, що описує зовнішній вигляд та поведінку реальної системи.
* Не очікуйте, що прототип повністю замінить специфікацію вимог до ПЗ. Більшість прихованої функціональності лише мається на увазі в прототипі і має бути задокументована в специфікації вимог, щоб її вдалося реалізувати повно, точно і прозоро. Макети екранів не дають деталей визначення полів даних, критеріїв перевірки, взаємозв'язків між полями (таких, як елементи управління інтерфейсу користувача, що з'являються тільки якщо користувач робить певні дії з іншими елементами управління), обробки винятків, бізнес-правил та іншої важливої ​​інформації.