**Бізнес-правила**

Кожна організація працює згідно з широким набором політик, законів і галузевих стандартів. Такі галузі, як банківська справа, авіація та виробництво медичних приладів, мають відповідати низці нормативних актів, наказів тощо. Такі принципи контролю відомі під загальною назвою бізнес-правила або бізнес-логіка. Більшість бізнес-правил виникають поза контекстом будь-якої конкретної програмної системи. Бізнес-правила часто забезпечуються шляхом ручного впровадження політик і процедур. Однак у багатьох випадках програмне забезпечення також має забезпечити виконання цих правил.

Таблиця 1 - Як бізнес-правила можуть впливати на різні типи вимог до ПЗ

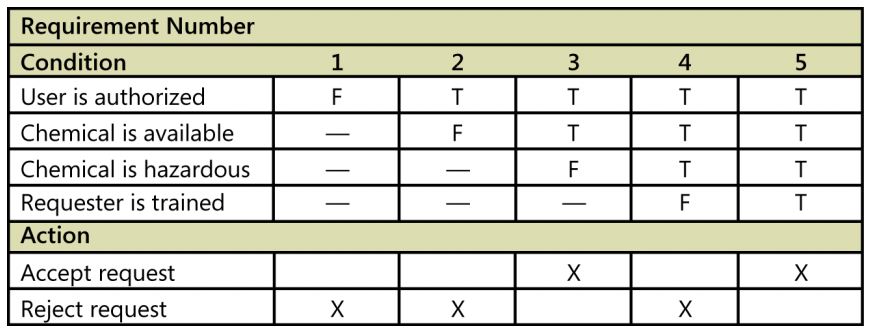
| Тип вимоги | Вплив бізнес-правил | Приклад |
| --- | --- | --- |
| Бізнес-вимога | З державних постанов можна вивести необхідні бізнес-цілі для проекту. | Система відстеження хімічних речовин має забезпечувати дотримання всіх державних нормативних актів про використання хімічних речовин та дотримання положень щодо звітності про утилізацію за останні п’ять місяців. |
| Користувацька | Політика конфіденційності визначає, хто з користувачів може, а хто не може виконувати певні завдання в системі. | Лише керівникам лабораторій дозволено створювати звіти про хімічний вплив для будь-кого, крім себе. |
| Функціональна | Політика компанії полягає в тому, що всі постачальники повинні бути зареєстровані та схвалені керівництвом, перш ніж буде оплачено рахунок. | Якщо рахунок-фактура отримано від незареєстрованого постачальника, Система поставок надішле йому електронною поштою форми для реєстрації. |
| Вимога до якості | Правила державних контролюючих установ можуть диктувати вимоги безпеки, які повинні виконуватись за допомогою функціоналу системи. | Система повинна вести записи про навчання з техніки безпеки, щоб переконатися, що користувачі пройшли відповідну підготовку, перш ніж вони зможуть подати запит на небезпечну хімічну речовину. |

Бізнес-правила іноді плутають з бізнес-процесами або бізнес-вимогами. Бізнес-вимогавказує на бажаний результат або високорівневу мету організації, яка розробляє або закуповує програмне рішення. Бізнес-вимоги служать обґрунтуванням для реалізації проекту. Бізнес-процесописує ряд дій, які перетворюють вхідні дані на вихідні для досягнення певного результату. Інформаційні системи часто автоматизують бізнес-процеси, що може призвести до підвищення ефективності та інших переваг завдяки виконанню заявлених бізнес-вимог. Бізнес-правила впливають на бізнес-процеси шляхом створення певного глосарію, накладання обмежень, запуску певних дій та управління тим, як виконуються обчислення. Одне бізнес-правило може застосовуватися до кількох ручних або автоматизованих процесів, що є однією з причин, чому краще розглядати бізнес-правила як окремий набір інформації. Існує п’ять категорій бізнес-правил: факти, обмеження, тригери, висновки, розрахунки.

Факти - це просто твердження, які є правдивими щодо бізнесу в певний момент часу. Факт описує асоціації або відношення між важливими бізнес-термінами. Факти про сутності даних, важливі для системи, можуть відображатися в моделях даних. *«За кожне замовлення стягується плата за доставку». «Книги, вищі за 40 см, розміщуються на полицях у розділі бібліотеки для книг великого розміру».*

Обмеження — це свідчення, яке обмежує дії, які дозволено виконувати системі або її користувачам. Хтось, описуючи обмежувальне бізнес-правило, може сказати, що певні дії потрібно, не потрібно або не можна виконувати, або що лише певні люди чи ролі можуть виконувати ці дії. Приклади: *організаційна політика, державні постанови, галузеві стандарти тощо*.

Оскільки багато обмежувальних бізнес-правил стосуються того, які типи користувачів можуть виконувати які функції, стислим способом документування таких правил є матриця ролей і прав доступу.

Правило-тригер запускає певну дію, якщо виконуються певні умови. Людина може запустити цю дію самостійно. Але, якщо задати правило, то на його основі буде реалізована деяка функціональність, завдяки якій програма буде демонструвати правильну поведінку при появі заданих умов. Таблиця рішень є лаконічним способом документування бізнес-правил, що відносяться до тригерів та мають складну логіку. Твердження у формі «Якщо <деяка умова виконується або сталася якась подія>, то <щось відбувається>» є підказкою, що мається на увазі правило-тригер.

Висновок, який іноді називають виведеним знанням або похідним фактом, створює новий факт з інших фактів. Висновки часто записуються за шаблоном «якщо/тоді», який також зустрічається в правилах-тригерах, але пункт «тоді» висновку просто надає частину знань, а не дію, яку потрібно виконати. *«Якщо платіж не буде отримано протягом 30 календарних днів після того, як він повинен бути сплачений, то рахунок є простроченим». «Якщо постачальник не може доставити замовлений товар протягом п’яти днів після отримання замовлення, товар вважається незамовленим».*

Правила-розрахунки перетворюють існуючі дані в нові за допомогою конкретних математичних формул або алгоритмів. Розрахунки зазвичай базуються на правилах, зовнішніх для підприємства, наприклад, формул утримання податку на прибуток. *«Загальна ціна замовлення дорівнює сумі вартості замовлених товарів за вирахуванням будь-яких знижок, а також податків з продажу для локації, куди відправляється замовлення, плюс плата за доставку та додаткова плата за страхування».*

**Документування бізнес-правил**

Оскільки бізнес-правила можуть впливати на кілька застосунків, організації повинні керувати своїми правилами як активами на рівні підприємства. Для початку буде достатньо простого каталогу бізнес-правил. При використанні інструменту керування вимогами, можна зберігати бізнес-правила як тип вимоги за умови, якщо правила доступні для всіх програмних проектів. Великі організації або організації, чиї операції та інформаційні системи сильно залежать від бізнес-правил, повинні створити базу даних бізнес-правил. В такому випадку використання комерційних інструментів керування правилами є більш доцільним, ніж управління правилами через текстовий процесор, електронну таблицю, Wiki чи інший документ із загальним доступом. Правила, пов’язані з безпекою, фінансами чи дотриманням нормативних вимог, становлять найбільший ризик, якщо ними не керують і не виконують їх належним чином.

При наданні кожному бізнес-правилу унікального ідентифікатору, вимоги будуть пов’язані з певним правилом. Наприклад, деякі шаблони для варіантів використання містять поле для бізнес-правил, які впливають на варіант використання. Замість того, щоб включати визначення правила в опис варіанту використання, просто введіть ідентифікатори для відповідних правил. Кожен ідентифікатор служить вказівником на головний екземпляр бізнес-правила. Таким чином не доведеться турбуватися про те, що специфікація варіанту використання застаріє, якщо правило зміниться.

Таблиця 2 - Деякі приклади записів в каталозі бізнес-правил

| id | Визначення | Тип | Властивість | Джерело |
| --- | --- | --- | --- | --- |
| Замовлення-5 | Якщо клієнт замовив книгу автора, який написав кілька книг, запропонуйте клієнту інші книги автора, перш ніж завершити замовлення. | Тригер | Статичне | Маркетингова політика XX |
| Доступ-8 | Усі зображення веб-сайту повинні містити альтернативний текст, який використовуватиметься електронними пристроями для читання, щоб відповідати вимогам доступності для користувачів з вадами зору. | Обмеження | Статичне | Стандарти доступного дизайну |
| Знижка-13 | Знижка розраховується на основі кількості товарів в поточному замовленні, як зазначено в таблиці БВ-060. | Розрахунок | Динамічне | Корпоративна цінова політика XX |

Так само, як запитання «Які у Вас є вимоги до системи?» не дуже допомагає при виявленні вимог користувачів, запитувати «Які у Вас є бізнес-правила?» також не має сенсу. Нижче наведено кілька поширених місць і способів пошуку правил:

* «Загальновідома інформація» стосовно організації, часто отримана від осіб, які працювали в компанії протягом тривалого часу та знають деталі її роботи.
* Застарілі системи (legacy systems), бізнес-правила яких “вбудовані” у вимоги та код.
* Моделювання бізнес-процесу, яке змушує аналітика шукати правила, які можуть впливати на кожен крок процесу: обмеження, події для тригерів, правила для розрахунків та релевантні факти.
* Аналіз існуючої документації, включаючи специфікації вимог з попередніх проектів, нормативні акти, галузеві стандарти, документи корпоративної політики, контракти та бізнес-плани.
* Аналіз даних, таких як різні стани, які може мати об’єкт даних, і умови, за яких користувач або системна подія можуть змінити стан об’єкта. Ці повноваження також можуть бути представлені у вигляді матриці ролей і прав доступу для надання інформації про правила щодо рівнів привілеїв користувачів і безпеки.

Те, що ви знайшли деякі бізнес-правила в цих різних джерелах, не означає, що вони обов’язково застосовуються до поточного проекту або що вони навіть досі дійсні. Обчислювальні формули, реалізовані в коді застарілих програм, можуть також бути застарілими. Обов’язково перевірте, чи потрібно оновлювати правила, отримані зі старих документів і програм. Оцініть сферу застосування виявлених правил. Чи є вони локальними для проекту, чи охоплюють бізнес-домен або всю компанію?

Часто зацікавлені сторони проекту вже знають про бізнес-правила, які впливатимуть на систему. Деякі працівники іноді мають справу з певними типами або класами правил. Варто залучити таких людей до обговорення.

Таблиця 3 - Пошук бізнес-правил з різних перспектив

| **Джерело** | **Типове питання** |
| --- | --- |
| Політика компанії | Чому ми повинні це робити таким чином? |
| Моделі даних | Як зв’язані ці дані? |
| Користувацькі рішення | Що користувач може зробити далі? |
| Події | Що має статися? Чого не повинно статися? |
| Системні рішення | Як система визначає, що робити далі? |
| Життєві цикли об'єктів | Що провокує зміни в стані об'єкта? |
| Розрахунки | Як розраховується цей показник? |
| Нормативні акти | Чого вимагає уряд? |

Щоб запобігти надмірності, не дублюйте правила з каталогу бізнес-правил у документації вимог. Натомість поверніться до конкретних правил як до джерела певних функцій або алгоритмів. Визначте зв’язки між функціональними вимогами та їх батьківськими бізнес-правилами. Ці посилання зберігають вимоги в актуальному стані при зміні правил. Якщо правило змінюється, можна шукати по ідентифікатору зв’язаного правила, щоб знайти вимоги або реалізовані функції, які, можливо, також доведеться змінити. Використання подібних посилань полегшує повторне використання того самого правила в кількох місцях і проектах, оскільки правила не ховаються в документації окремої програми.

**Документування вимог**

Результатом розробки вимог є задокументована угода між зацікавленими сторонами щодо продукту, який буде створено. Документ про концепцію та обсяг містить бізнес-вимоги, а вимоги користувачів можна зафіксувати у формі варіантів використання або історій користувачів. Функціональні та нефункціональні вимоги продукту часто зберігаються в специфікації вимог до програмного забезпечення, або SRS, яка надається тим, хто повинен сконструювати, розробити та перевірити рішення.

SRS зазвичай описується як документ, але він не обов’язково має бути у формі традиційного текстового документа. Альтернативно можна зберігати інформацію в електронній таблиці (яка має багато тих самих обмежень, що й документ), базі даних або за допомогою інструменту керування вимогами.

Специфікація вимог до програмного забезпечення має багато назв у різних організаціях, хоча організації використовують ці терміни не однаково. Її іноді називають документом бізнес-вимог (BRD - Business Requirements Document), функціональною специфікацією, специфікацією продукту, специфікацією системи або просто документом вимог. Оскільки «специфікація вимог до програмного забезпечення» є галузевим стандартом, його також називають ISO/IEC/IEEE 2011.

SRS визначає функції та можливості, які повинна надавати система програмного забезпечення, її характеристики та обмеження, яких вона повинна дотримуватися. Специфікація має описувати так повно, наскільки це необхідно, поведінку системи за різних умов, а також бажані якості системи, такі як продуктивність, безпека та зручність використання. SRS є основою для подальшого планування проекту, проектування та кодування, а також основою для тестування системи та документації користувача. Однак документ не повинен містити деталей проектування, розробки, тестування або управління проектом, крім відомих обмежень проектування та реалізації. Навіть людям, які працюють над agile-проектами, використовують SRS. Зазвичай вони не збирають всю цю інформацію в один документ, але шаблон SRS надає потрібне уявлення про те, яку інформацію потрібно шукати.

Не обов’язково писати SRS для всього продукту перед початком розробки, але треба охопити вимоги для кожної ітерації перед її проходженням. Поступова розробка доречна, коли потрібно швидко отримати деякі функції для користувачів. Відгуки про використання ранніх ітерацій впливатимуть на решту проекту. Однак перед тим, як команда втілить їх у життя, у кожному проекті слід призначити базові лінії для узгодження кожного набору вимог. Призначення і проходження базових ліній — це процес перетворення SRS, що розробляється, на таку, що була переглянута і затверджена. Робота на основі узгодженого набору вимог мінімізує непорозуміння та непотрібне перероблення.

Важливо організувати та написати SRS так, щоб її могли зрозуміти різні стейкхолдери. Існують наступні пропозиції для покращення читабельності:

* Використовуйте відповідний шаблон, щоб упорядкувати всю необхідну інформацію.
* Послідовно позначайте та стилізуйте розділи, підрозділи та окремі вимоги.
* Використовуйте візуальний акцент (жирний шрифт, підкреслення, курсив, колір і шрифти) послідовно та доречно.
* Пам’ятайте, що виділення кольором може бути невидимим для людей із дальтонізмом або під час друку в градаціях сірого.
* Створіть зміст, щоб допомогти читачам знайти потрібну інформацію.
* Усі рисунки і таблиці пронумеруйте, підпишіть і посилайтеся на них за номерами.
* Якщо ви зберігаєте вимоги в документі, зробіть перехресні посилання за допомогою інструментів текстового процесора замість жорстко закодованих номерів сторінок або розділів для посилання на інші місця в документі.
* Якщо ви використовуєте документи, визначте гіперпосилання, щоб дозволити читачеві переходити до пов’язаних розділів у SRS або в інших файлах.
* Якщо ви зберігаєте вимоги за допомогою певного інструменту, використовуйте посилання, щоб дозволити читачеві перейти до відповідної інформації.
* Додайте візуальне представлення інформації, коли це можливо, щоб полегшити розуміння.
* Залучіть кваліфікованого редактора, щоб переконатися, що документ є зрозумілим і використовує несуперечливий словниковий запас і макет.

**Рекомендації щодо написання вимог**

Представлені нижче зразки вимог завжди можна вдосконалити, і крім того завжди є альтернативні способи їх формулювання. Двома важливими цілями написання вимог є наступні:

* Кожен, хто читає вимогу, інтерпретує її так само, як і будь-який інший читач.
* Інтерпретація кожного читача збігається з тим, що мав на увазі автор.

Досягнення таких результатів набагато важливіше, ніж чистота стилю чи догматичне підпорядкування правилам чи конвенціям написання вимог.

**Точка зору системи або користувача**

Можна написати функціональні вимоги з точки зору того, що робить система, або що може зробити користувач. Оскільки ефективна комунікація є головною метою, можна поєднувати ці стилі, формулюючи кожну вимогу в тому стилі, який є більш зрозумілим. Формулювання вимог має бути узгодженим, наприклад «Система повинна» або «Користувач повинен». Загальний шаблон для вимоги: [*необов’язкова передумова*] [*необов’язкова подія*] система повинна [*очікувана відповідь системи*]. Приклад:

*Якщо запитуваний хімікат знайдено на складі хімікатів,* система повинна *відобразити список усіх контейнерів з хімікатом, які зараз знаходяться на складі.*

Цей приклад містить передумову, але не тригер. Також тут деякі автори вимог пропускають фразу «система повинна». Аргументують це тим, що вимоги описують поведінку системи, тому немає потреби постійно говорити «система повинна» зробити те чи інше.

Під час написання функціональних вимог з точки зору користувача слід дотримуватися наступного шаблону: [*Клас користувача або ім’я актора*] повинен мати можливість [*щось робити*] [*з деяким об’єктом*] [*умова, час відповіді або вимоги до якості*].

Альтернативні формулювання: «Система повинна надати можливість (або дозволити, надати доступ, включити тощо) [*певний клас користувачів*] [*робити щось*]». Нижче наведено приклад функціональної вимоги, написаної з перспективи користувача:

*Хімік повинен мати можливість повторити замовлення будь-якого хімікату, який він замовив у минулому, шляхом отримання та редагування деталей замовлення.*

**Стиль написання вимог**

Варто сформулювати вимогу в активному стані, щоб було зрозуміло, яка сутність виконує описану дію. Багато ділових і наукових записів сформульовані в пасивному стані, але активний стан робить речення більш чітким та зрозумілим. У пасивному стані записується така вимога: *Після доставки оновленого продукту серійний номер буде оновлено в рядку контракту.*

Фраза «буде оновлено» вказує на пасивний стан. Він позначає одержувача дії (порядковий номер), але не виконавця дії. Тобто ця фраза не дає підказки щодо того, хто або що оновлює серійний номер. Чи зробить це система автоматично, чи це оновлення очікується від користувача? Перефразування цієї вимоги в активному стані робить актора явним і також пояснює ініційну подію: *Коли надійшло підтвердження про оновлення продукту, система повинна оновити договір клієнта із серійним номером нового продукту.*

Уникайте написання великих абзаців, які містять декілька вимог. Читачі не повинні шукати індивідуальні вимоги у довгому пространому описі. Чітко відокремлюйте індивідуальні вимоги від довідкової або контекстної інформації.

Уникайте використання «і/або» у вимозі, це залишає інтерпретацію на розсуд читача, як у цьому випадку: *Система повинна дозволяти пошук за номером замовлення, номером рахунку-фактури та/або номером замовлення клієнта.*

Ця вимога дозволить користувачеві вводити одну, дві або три цифри одночасно при виконанні єдиного пошуку. Це може відрізнятися від того, що було задумано.

Слова «якщо», «крім» і «але» також вказують на наявність кількох вимог: *Оплата буде стягнена з кредитної картки Покупця, якщо термін дії кредитної картки не закінчився.*

Не вказано, що відбувається, коли умова не виконується, що призводить до неповної вимоги. Треба розділити це на дві вимоги, щоб задати поведінку для двох умов - кредитна картка активна та прострочена:

*Якщо зареєстрована кредитна картка Покупця активна, система стягує платіж із цієї картки.*

*Якщо термін дії зареєстрованої кредитної картки Покупця закінчився, система дозволить Покупцеві оновити дані поточної кредитної картки або ввести нову кредитну картку для оплати.*

Кожна вимога, яка визначає, як система повинна працювати, якщо все правильно, також повинна за необхідності мати супровідні вимоги для опису, як система повинна реагувати, коли трапляються виключення. Наприклад: *Якщо користувач працює з існуючим файлом і вирішує зберегти файл, система збереже його під тим же ім’ям.*

Сама по собі ця вимога не вказує на те, що повинна робити система, якщо вона не може зберегти файл із тим же ім’ям. Доречною другою вимогою для першої може бути: *Якщо система не може зберегти файл під певним іменем, система надасть користувачеві можливість зберегти його під іншою назвою або скасувати операцію збереження.*

**Моделювання вимог**

Візуальні моделі вимог можуть допомогти визначити відсутні, недоречні та неузгоджені вимоги. Враховуючи обмеження людської короткочасної пам’яті, проаналізувати список із тисячі вимог на невідповідності, дублювання та недоречність майже неможливо. Коли ви досягнете п’ятнадцятої вимоги, ви, швидше за все, забудете кілька перших, які прочитали. Навряд чи вийде знайти всі помилки, просто переглянувши текстові вимоги.

З ключових слів, які можна зустріти серед вимог, що представляють клієнти, бізнес-аналітик може вибрати конкретні елементи моделей.

Таблиця 4 - Зв’язок вимог клієнта з компонентами моделей аналізу

| **Тип** | **Приклад** | **Компонент моделі** |
| --- | --- | --- |
| Іменник | Люди, організації, програмні системи, елементи даних або об'єкти, які існують | Зовнішні сутності, сховища даних або потоки даних (DFD)  Актори (діаграма варіантів використання)  Сутності або їх атрибути (ERD)  Доріжки/актори (Діаграма послідовності)  Об'єкти зі станами (Діаграма станів) |
| Дієслово | Дії, речі, які може зробити користувач або система, або події, які можуть відбутися | Процеси (DFD)  Етапи процесу (Діаграма діяльності або послідовності)  Варіанти використання  Відносини між сутностями (ERD)  Переходи (Діаграма станів)  Діяльність (Діаграма діяльності)  Події (Таблиця подія-відповіді) |
| Умовне твердження | Умовні логічні твердження, такі як якщо/то | Рішення (дерево рішень, таблиця рішень або діаграма діяльності)  Розгалуження (Діаграма діяльності або послідовності) |

Таблиця 5 - Вибір найбільш підходящої техніки представлення

| **Інформація** | **Техніка представлення** |
| --- | --- |
| Зовнішні інтерфейси системи | *Контекстна діаграма та діаграма варіантів використання* ідентифікують об’єкти поза системою, які пов'язані з нею. *Контекстна діаграма та діаграми потоку даних* ілюструють входи та виходи системи на високому рівні абстракції. *Карта екосистеми* визначає можливі системи, які взаємодіють з поточною, але також містить деякі, які не взаємодіють з поточною системою безпосередньо. *Діаграми діяльності та послідовності* показують, що відбувається в процесі взаємодії між системами. |
| Потік бізнес-процесу | *Діаграма потоку даних* верхнього рівня показує, як бізнес-процес обробляє дані на високому рівні абстракції. *Діаграми послідовності* показують ролі, які беруть участь у виконанні різних етапів потоку бізнес-процесу.  Діаграми потоків даних нижчих рівнів або діаграми послідовності можуть представляти потоки бізнес-процесів у значних деталях. Подібним чином, *блок-схеми та діаграми діяльності* можна використовувати як на високому, так і на низькому рівнях абстракції, хоча найчастіше вони використовуються для визначення деталей процесу. |
| Визначення даних і зв’язки між об’єктами даних | *Діаграма сутність-зв’язок* показує логічні зв’язки між об’єктами даних (сутностями). *Діаграми класів* показують логічні зв’язки між класами об’єктів і пов’язаними з ними даними.  *Словник даних* містить докладні визначення структур даних і окремих елементів даних. Складні об’єкти даних поступово розбиваються на складові елементи. |
| Стани системи та об'єктів | *Діаграми стан-перехід та таблиці станів* представляють можливі стани системи чи об’єкта на високому рівні абстракції, а також зміни між станами, які можуть відбутися за певних обставин. Ці моделі використовуються, коли кілька варіантів використання можуть маніпулювати станом певних об’єктів.  Деякі аналітики створюють *таблицю «подія-відповідь»* як інструмент визначення обсягу, ідентифікуючи зовнішні події для визначення границь обсягу продукту. Також за допомогою цієї таблиці можна вказати індивідуальні функціональні вимоги, деталізуючи, як система повинна поводитися у відповідь на кожну комбінацію зовнішньої події та стану системи.  *Функціональні вимоги* надають детальний опис поведінки користувача та системи, що призводить до змін стану. |
| Складна логіка | *Дерево рішень* показує можливі результати для набору пов’язаних рішень або умов. *Таблиця рішень* визначає унікальні функціональні вимоги, пов’язані з різними комбінаціями істинних і хибних результатів для послідовності рішень або умов. |
| Користувацькі інтерфейси | *Карта діалогу* (dialog mapping) забезпечує високорівневий огляд запропонованого або фактичного інтерфейсу користувача, показуючи різні елементи відображення та можливі шляхи навігації між ними.  *Макети інтерфейсу* доповнюють діалогову карту, показуючи, що міститиме кожен екран, без зображення точних деталей. Моделі «відображення-дія-відповідь» описують вимоги до відображення та поведінки кожного екрана.  *Деталізовані макети* показують, як саме виглядатимуть елементи екрану. Визначення даних і описи елементів керування інтерфейсу користувача можуть містити додаткові деталі. |
| Опис завдань користувача | *Історії користувачів, сценарії та специфікації варіантів використання* описують завдання користувача з різними рівнями деталізації.  *Діаграми діяльності та послідовності* ілюструють бізнес-процеси або взаємодію між кількома учасниками та системою. *Блок-схеми та діаграми діяльності* наочно зображують потік варіанту використання та розгалуження на альтернативні потоки та винятки.  *Функціональні вимоги* надають детальний опис того, як система та користувач взаємодіятимуть для досягнення потрібних результатів. *Тест-кейси* надають альтернативний погляд із низьким рівнем абстракції, описуючи, яку саме поведінку системи слід очікувати за конкретних умов для вхідних даних, стану системи та дій. |
| Нефункціональні вимоги | Атрибути якості та обмеження зазвичай записуються у формі тексту природною мовою, але це часто призводить до недостатньої точності та повноти. |

**Діаграма потоків даних (DFD)**

Діаграма потоку даних є основним інструментом структурного аналізу. DFD визначає трансформаційні процеси системи, колекції (сховища) даних або фізичних матеріалів, якими система маніпулює, а також потоки даних або матеріалів між процесами, сховищами та зовнішнім світом. Моделювання потоків даних використовує підхід функціональної декомпозиції до системного аналізу, розбиваючи складні проблеми на прогресивні рівні деталізації. Це добре підходить для систем обробки транзакцій та інших програм з великим набором функцій.

Цей інструмент часто використовується під час опитування клієнтів, оскільки легко намалювати DFD на дошці при обговоренні, як працює бізнес користувача. DFD можна використовувати як техніку для виявлення відсутніх вимог до даних. Дані, що передаються між процесами, сховищами даних і зовнішніми об’єктами, також повинні бути змодельовані в ERD і описані в словнику даних. Крім того, DFD надає контекст функціональним вимогам щодо того, як користувач виконує певні завдання.

Діаграми потоків даних можуть представляти системи в широкому діапазоні абстракції. Високорівневі DFD забезпечують цілісне бачення даних і компонентів обробки в багатоетапній діяльності, що доповнює детальне уявлення, втілене у функціональних вимогах.

Кожен процес, який відображається як окреме коло на діаграмі рівня 0, може бути розгорнутий в окремий DFD для розкриття деталей його функціонування. Це поступове вдосконалення продовжується, поки діаграми найнижчого рівня не будуть містити лише примітивні операції процесу, які можна чітко представити у вигляді описового тексту, псевдокоду, діаграм послідовності або діаграм діяльності. Функціональні вимоги точно визначатимуть, що відбувається в кожному примітивному процесі. Кожен рівень DFD має бути збалансованим і узгодженим з рівнем вище, щоб усі вхідні та вихідні потоки на дочірній діаграмі збігалися з потоками на батьківській діаграмі. Складні структури даних у високорівневих діаграмах можуть бути розділені в DFD нижчого рівня на складові елементи, які визначено в словнику даних.

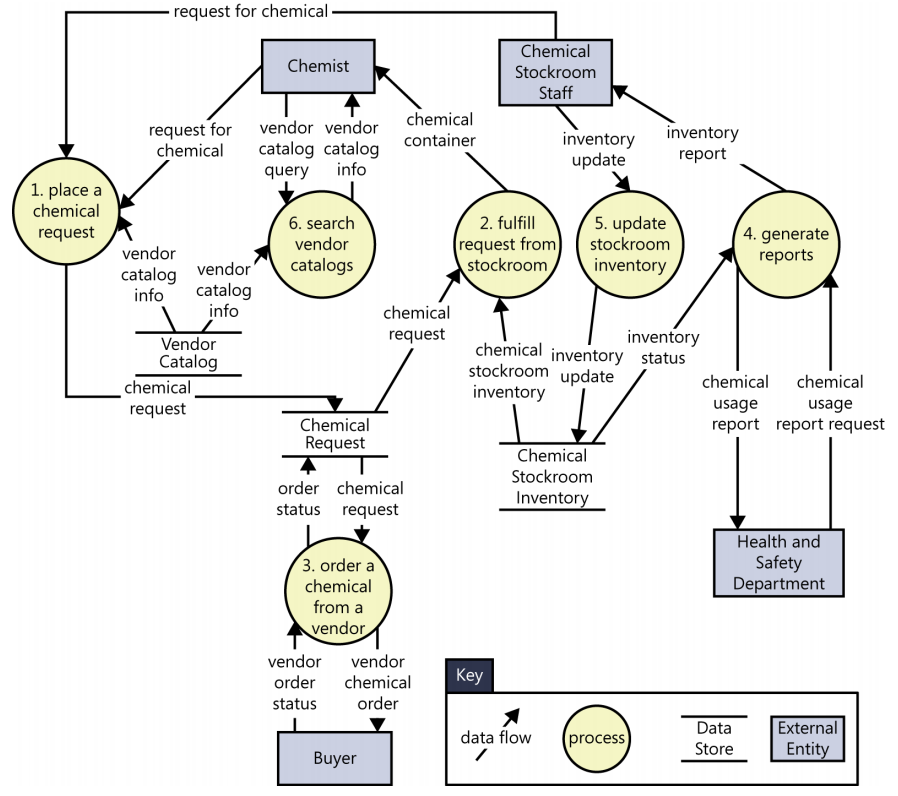


Рисунок 1 - Часткова діаграма потоку даних рівня 0 для системи відстеження хімічних речовин

**Таблиці подія-відповідь**

Іншим підходом до визначення вимог користувача є ідентифікація зовнішніх подій, на які система повинна реагувати. Подія — це певна зміна або діяльність, що відбувається в середовищі користувача, яка стимулює реакцію програмної системи. У таблиці подія-відповідь (яка також називається таблицею подій або списком подій) зазначається поведінка, яку система повинна демонструвати у відповідь на кожну таку подію. Існує три класи системних подій:

* **Бізнес-подія** — це дія користувача-людини, яка стимулює діалог з системою, наприклад, коли користувач ініціює варіант використання. Послідовності подія-відповідь відповідають етапам потоку варіанта використання або діаграмі послідовності.
* **Сигнальна подія** реєструється, коли система отримує контрольний сигнал, сигнал початку читання даних або сигнал переривання від зовнішнього апаратного пристрою чи іншої програмної системи, наприклад, коли замикається перемикач, змінюється напруга, інша програма запитує послугу або користувач проводить пальцем по екрану планшета.
* **Часова подія** ініціюється певною датою та/або часом, коли годинник комп’ютера досягає визначеного часу (наприклад, для запуску операції автоматичного експорту даних опівночі) або коли попередньо встановлена тривалість минула з попередньої події (як у системі що реєструє температуру, яку зчитує датчик кожні 10 секунд).

Аналіз подій особливо добре працює для специфікації систем реального часу. Щоб ідентифікувати події, розгляньте всі стани, пов’язані з об’єктом аналізу, і визначте будь-які події, які можуть перевести об’єкт у ці стани. Перегляньте діаграми контексту для будь-яких зовнішніх об’єктів, які можуть ініціювати активність/подію або вимагати автоматичної відповіді.

Інша інформація, яку можна додати до таблиці подія-відповідь:

* Частота подій (скільки разів подія має місце за певний період часу або обмеження кількості разів, коли вона може відбуватися).
* Елементи даних, необхідні для обробки події.
* Стан системи після виконання відповідей на події.

Таблиця 6 - Часткова таблиця подія-відповідь для системи склоочисника авто

| id | Подія | Стан системи | Відповідь |
| --- | --- | --- | --- |
| 1 | Переведення склоочисника на низьку швидкість | Склоочисник вимкнений, працює на високій швидкості або з перервами | Склоочисник працює на низькій швидкості |
| 2 | Переведення склоочисника на високу швидкість | Склоочисник вимкнений, працює на низькій швидкості або з перервами | Склоочисник працює на високій швидкості |
| 3 | Вимкнення склоочисника | Склоочисник працює на високій, низькій швидкості або з перервами | 1. Завершити поточний цикл витирання  2. Вимкнути склоочисник |
| 4 | Перевести склоочисник на режим роботи з перервами | Склоочисник вимкнений | 1. Зчитати налаштування інтервалу часу витирання  3. Ініціалізувати таймер витирання |
| 5 | Перевести склоочисник на режим роботи з перервами | Склоочисник працює на високій або низькій швидкості | 1. Завершити поточний цикл витирання  2. Зчитати налаштування інтервалу часу витирання 3. Ініціалізувати таймер витирання |

Таблиця подія-відповідь, яка визначає кожну можливу комбінацію події, стану та відповіді, включаючи виняткові умови, може служити частиною функціональних вимог для системи. Можна перетворити таблицю подій-відповідей у таблицю рішень, щоб забезпечити аналіз усіх можливих комбінацій подій і станів системи. Однак бізнес-аналітик має надати додаткові функціональні та нефункціональні вимоги. Наприклад, скільки циклів за хвилину виконує склоочисник у режимі повільного та швидкого стирання.

**Атрибути якості програмного забезпечення**

Кілька десятків характеристик продукту можна назвати атрибутами якості, хоча зазвичай проектна команда повинна ретельно розглянути лише декілька з них. Якщо розробники знають, які з цих характеристик є найважливішими для успіху, вони можуть вибрати відповідні підходи до проектування та розробки для досягнення цілей якості. Атрибути якості були класифіковані відповідно до широкого спектру схем (IEEE 1998; ISO/IEC 2007; ISO/IEC 2011 тощо). Деякі автори побудували розгорнуті ієрархії, які групують пов’язані атрибути в кілька основних категорій. Один зі способів класифікації атрибутів якості базується на тому, чи є вони помітні при використанні програмного забезпечення: зовнішні та внутрішні атрибути якості. Зовнішні атрибути в першу чергу важливі для користувачів, тоді як внутрішні більш важливі для персоналу з розробки та обслуговування. Внутрішні атрибути якості опосередковано сприяють задоволенню клієнтів, полегшуючи розширення, виправлення, тестування продукту та його перенесення на нові платформи.

Існує низка внутрішніх і зовнішніх аспектів якості, які варто розглянути для кожного проекту. Певні атрибути особливо важливі для певних типів проектів:

* Вбудовані системи: продуктивність, ефективність, надійність, стійкість, безпека, безпека праці, зручність використання.
* Веб-застосунки і корпоративні програми: доступність, цілісність, сумісність, продуктивність, масштабованість, безпека, зручність використання.
* Десктоп та мобільні системи: продуктивність, безпека, зручність використання.

**Доступність**

Доступність – це показник запланованого часу роботи, протягом якого послуги системи доступні для використання та повністю функціонують. Формально доступність дорівнює відношенню часу роботи до суми часу роботи та простою. Ще більш формально, доступність дорівнює середньому часу між відмовами (середній час напрацювання на відмову) для системи, поділеному на суму середнього часу між відмовами і середнього часу відновлення системи після виявлення збою. Періоди планового технічного обслуговування також впливають на доступність.

*Система має бути доступною щонайменше на 95 відсотків у будні з 6:00 ранку і опівночі за східним часом, і щонайменше 99 відсотків доступна в будні між 15:00 та 17:00 за східним часом.*

*Час простою, який виключається з розрахунку доступності, складається з технічного обслуговування, запланованого в години з 18:00 у неділю до 3:00 ранку понеділка за тихоокеанським часом.*

**Інстальованість**

*Непідготовлений користувач може успішно виконати початкове встановлення програми в середньому за 10 хвилин.*

*При встановленні оновленої версії програми всі налаштування з профілю користувача буде збережено та за потреби конвертовано у формат даних нової версії.*

*Програма інсталяції перед початком інсталяції перевіряє коректність проведеного завантаження даних.*

*Для встановлення програмного забезпечення на сервері потрібні права адміністратора.*

*Після успішного встановлення програма інсталяції видаляє всі тимчасові, резервні, застарілі та непотрібні файли, пов’язані з програмою.*

**Цілісність**

*Після виконання резервного копіювання файлу система звіряє резервну копію з оригіналом і повідомляє про будь-які розбіжності.*

*Система повинна захищати від неавторизованого додавання, видалення або зміни даних.*

*Система відстеження хімічних речовин підтверджує, що закодована хімічна структура, імпортована зі сторонніх графічних редакторів формул, представляє дійсну хімічну структуру.*

*Система повинна щодня підтверджувати відсутність доданого несанкціонованого коду у виконуваних файлах програми.*

**Сумісність**

Сумісність вказує на те, наскільки легко система може обмінюватися даними та сервісами з іншими програмними системами та наскільки легко вона може інтегруватися із зовнішніми апаратними пристроями. Щоб оцінити сумісність, потрібно знати, які інші програми користувачі використовуватимуть у поєднанні з поточною системою і якими даними вони очікують обмінюватися.

*Система відстеження хімічних речовин повинна мати можливість імпортувати будь-яку дійсну хімічну структуру з ChemDraw (версія 21.0 або раніше) і MarvinSketch (версія 19.21 або раніше).*

*Система хімічного відстеження повинна мати можливість імпортувати будь-яку хімічну структуру, закодовану за допомогою нотації SMILES (simplified molecular-input line-entry system).*

**Продуктивність**

*Авторизація запиту на зняття коштів у банкоматі займає не більше 2,0 секунд.*

*Датчики швидкості антиблокувальної гальмівної системи повідомляють про швидкість коліс кожні 2 мілісекунди з варіацією, яка не повинна перевищувати 0,1 мілісекунди.*

*Веб-сторінки мають повністю завантажуватися в середньому за 3 секунди або менше при підключенні до Інтернету зі швидкістю 30 мегабіт/с.*

*Система торгових операцій повинна оновлювати відображення статусу транзакції протягом 1 секунди після завершення кожної операції щонайменше в 98% відсотках випадків.*

**Надійність**

Проблеми з надійністю можуть виникнути через неправильне введення даних, помилки в самому програмному коді, компоненти, які недоступні, коли це необхідно, і апаратні збої. Стійкість і доступність тісно пов'язані з надійністю. Способи визначення та вимірювання надійності програмного забезпечення включають відсоток операцій, які виконано правильно, середню тривалість часу, протягом якого система працює до збою (середній час між відмовами, або середній час напрацювання на відмову), і максимально прийнятну ймовірність збою протягом певного періоду часу.

*Не більше 5 експериментальних запусків із 1000 можна втратити через збої програмного забезпечення.*

*Середній час між відмовами карт-рідера має становити не менше 90 днів.*

**Стійкість**

Стійкість — це ступінь, до якого система продовжує функціонувати належним чином, коли вона стикається з недійсними вхідними даними, дефектами підключеного програмного чи апаратного забезпечення, зовнішніми атаками, фізичними ушкодженнями або неочікуваними умовами роботи.

*Якщо текстовий редактор виходить з ладу до того, як користувач збереже файл, при наступному запуску редактор має відновити вміст редагованого файлу в тому стані, який був щонайбільше за одну хвилину до збою.*

*Для всіх параметрів графіка мають бути вказані значення за замовчуванням, які графічний редактор буде використовувати, якщо вхідні дані параметра відсутні або недійсні.*

**Безпека праці**

*Користувач повинен мати можливість побачити список усіх інгредієнтів у будь-якому пункті меню з виділеними інгредієнтами-алергенами.*

*Якщо температура корпусу реактора підвищується швидше ніж на 5°C за хвилину, система керування хімічним реактором повинна вимкнути джерело тепла та подати попередження оператору.*

*Терапевтична радіаційна машина повинна дозволяти опромінення лише за наявності належного фільтра.*

*Система повинна припинити будь-яку роботу протягом 1 секунди, якщо виміряний тиск у баку перевищує 90 відсотків від зазначеного максимального тиску.*

**Безпека**

*Система блокує обліковий запис користувача після чотирьох послідовних невдалих спроб входу протягом п’яти хвилин.*

*Система повинна реєструвати всі спроби доступу до захищених даних користувачами з недостатнім рівнем доступу.*

*Користувач повинен змінити призначений тимчасовий пароль на пароль, який раніше не використовувався, одразу після першого успішного входу з тимчасовим паролем.*

*Розблокування дверей, яке є результатом успішного зчитування бейджа безпеки, повинно тримати двері розблокованими протягом 8,0 секунд з допуском 0,5 секунди.*

*Резидентне програмне забезпечення для захисту від зловмисного програмного забезпечення має ізолювати будь-який вхідний Інтернет-трафік, який демонструє характеристики відомих або ймовірних сигнатур вірусів.*

*Магнітометр повинен виявляти щонайменше 99,9 відсотка заборонених об’єктів, при цьому рівень помилкових позитивних результатів не повинен перевищувати 1%.*

**Зручність використання (юзабіліті)**

Юзабіліті стосується факторів щодо зручності та легкості у використанні (user-friendliness, human engineering).

Таблиця 7 - Можливі підходи до дизайну для простоти навчання та простоти використання

| **Простота навчання** | **Простота використання** |
| --- | --- |
| Детальні підказки | Комбінації клавіш |
| Майстри (wizards) | Меню та панелі інструментів, які можна налаштувати |
| Видимі параметри меню | Кілька способів доступу до однієї функції |
| Змістовні зрозумілі повідомлення | Автодоповнення тексту, що вводиться |
| Екрани довідки та підказки | Автовиправлення помилок |
| Подібність до інших загальновикористовуваних систем | Можливості запису макросів і створення сценаріїв |
| Обмежена кількість параметрів і віджетів, що відображаються | Можливість перенесення інформації з попередньої транзакції |
|  | Автоматичне заповнення полів форми |
|  | Інтерфейс командного рядка |

*Тренований користувач може подати запит на хімічну речовину з каталогу постачальника в середньому за три хвилини та максимум за п’ять хвилин у 95% випадків.*

*Для всіх функцій у меню «Файл» мають бути визначені клавіші швидкого доступу, які використовують клавішу Ctrl, натиснуту одночасно з однією іншою клавішею. Команди меню, які також відображаються в MS Word, мають використовувати ті самі стандартні комбінації клавіш, які використовує Word.*

**Ефективність**

Ефективність тісно пов'язана із зовнішнім атрибутом якості продуктивності. Ефективність — це міра того, наскільки добре система використовує потужність процесора, дисковий простір, пам'ять або пропускну здатність зв'язку.

*Принаймні 30% потужності процесора та пам’яті, доступної для програми, не повинні використовуватися в умовах запланованого пікового навантаження.*

*Система повинна надавати оператору попереджувальне повідомлення, коли навантаження на використання перевищує 80% максимальної запланованої потужності.*

**Модифікованість**

Модифікованість стосується того, наскільки легко дизайн та код програмного забезпечення можна зрозуміти, змінити та розширити:

* Виправлення дефектів.
* Розширення та модифікація функціональності для задоволення нових бізнес-потреб і вимог.
* Модифікація системи для функціонування в зміненому операційному середовищі без додавання нових можливостей.
* Виправлення несправностей, обслуговування пристроїв або ремонт пристроїв у їх робочому середовищі.

*Виклики функцій не повинні бути вкладеними більше ніж на два рівні.*

*Принтер має відобразити повідомлення про помилку, якщо змінні картриджі не були вставлені у відповідні слоти.*

**Портативність**

Зусилля, необхідні для міграції програмного забезпечення з одного операційного середовища в інше, є показником портативності. Іноді ще включають можливість інтернаціоналізації та локалізації продукту. Підходи до проектування, які роблять програмне забезпечення портативним, подібні до тих, що роблять його придатним до повторного використання. Портативність стає все більш важливою, оскільки програми повинні працювати в різних середовищах: Windows, Mac і Linux; iOS і Android; ПК, планшети і телефони. Вимоги до портативності даних також важливі.

*Користувач повинен мати можливість переносити закладки браузера до та з Firefox, Internet Explorer, Opera, Chrome і Safari.*

*Інструмент міграції платформи має перенести налаштовані профілі користувачів до нової інсталяції без залучення користувача.*

**Повторне використання**

Програмне забезпечення придатне до повторного використання має бути модульним, добре задокументованим, незалежним від конкретної програми та операційного середовища та бути в певній мірі узагальненим. Численні артефакти проекту пропонують потенціал для повторного використання, включаючи вимоги, архітектури, проекти, код, тести, бізнес-правила, моделі даних, описи класів користувачів, профілі зацікавлених сторін і глосарій термінів. Зробити програмне забезпечення придатним для повторного використання легше при наявності ретельних специфікації вимог і проектування, суворого дотримання стандартів кодування, набору релевантних тест-кейсів і стандартної бібліотеки повторно використовуваних компонентів.

*Функції введення хімічної структури мають бути придатними до повторного використання на рівні об’єктного коду в інших програмах.*

*Алгоритми ціноутворення мають бути придатними до повторного використання в майбутніх програмах для менеджменту магазинів.*

**Масштабованість**

Вимоги до масштабованості стосуються здатності системи розвиватися, щоб обслуговувати більше користувачів, даних, серверів, географічних розташувань, транзакцій, мережевого трафіку, пошуку та інших сервісів без шкоди для продуктивності чи коректності. Масштабованість має як апаратне, так і програмне значення. Масштабування системи може означати придбання більш швидких комп’ютерів, додавання пам’яті або дискового простору, додавання серверів, баз даних або збільшення пропускної здатності мережі. Програмні підходи можуть включати розподіл обчислень на декілька процесорів, стиснення даних, оптимізацію алгоритмів, та інші методи налаштування продуктивності.

*Пропускну здатність системи екстреного телефонного зв'язку необхідно збільшити з 500 викликів на день до 2500 викликів на день протягом 12 годин.*

*Веб-сайт повинен витримувати темпи зростання кількості переглядів сторінок на 30% на квартал протягом принаймні двох років без помітного для користувача погіршення продуктивності.*

*Дистрибуційна система повинна мати можливість вмістити до 20 нових складських центрів.*

**Перевіряємість**

У більш вузькому сенсі її можна назвати здатністю до тестування, а перевіряємість означає, наскільки добре можна оцінити компоненти програмного забезпечення або інтегрований продукт, щоб продемонструвати, чи система функціонує належним чином. Перевіряємість важлива, якщо продукт буде часто змінюватися, оскільки він проходитиме часте регресійне тестування, щоб визначити, чи зміни пошкодили будь-яку існуючу функціональність.

*Конфігурація середовища розробки має бути ідентичною середовищу конфігурації тестування, щоб уникнути невідтворюваних помилок тестування.*

*Тестер повинен мати можливість налаштувати, які результати виконання реєструватимуться під час тестування.*

*Розробник повинен мати можливість налаштувати обчислювальний модуль для відображення проміжних результатів будь-якої визначеної групи алгоритмів для цілей налагодження.*

**Встановлення пріоритетів**

Пріоритезація відповідних атрибутів знадобиться не тільки при обговореннях щодо виявлення вимог. Таблиця парних порівнянь може підійти для невеликого списку елементів. На рисунку 2 представлена така таблиця для атрибутів якості кіоску реєстрації в аеропорту.

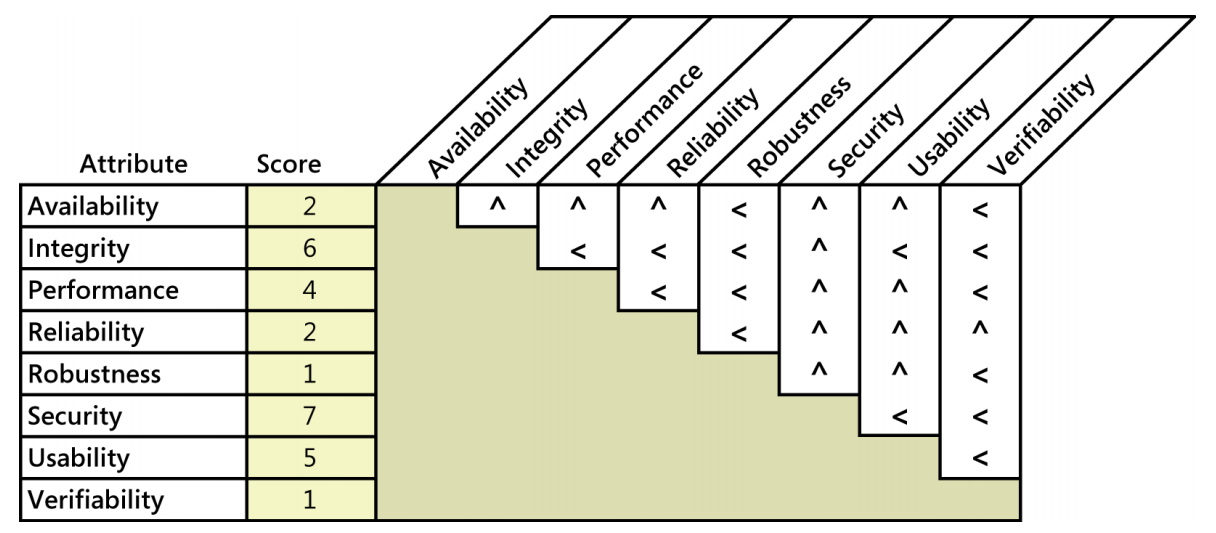


Рисунок 2 - Пріоритети атрибутів якості для реєстрації в аеропорту

Символи в кожній комірці таблиці показують вліво, якщо атрибут в рядку більш важливий, ніж атрибут в стовпці, і навпаки, показують вгору, якщо атрибут в стовпці більш важливий, ніж атрибут в рядку.

В другому стовпці таблиці обчислюється відносний бал для кожного атрибута. На рисунку безпека є найважливішою (з оцінкою 7), за нею йдуть цілісність (6) і зручність використання (5). Хоча й інші фактори важливі для успіху — це погано, якщо кіоск недоступний для пасажирів або якщо він виходить з ладу на півдорозі під час реєстрації — справа в тому, що не всі атрибути якості можуть мати найвищий пріоритет.

Визначення пріоритетів потрібно, щоб знати як реагувати, коли є суперечливі вимоги до якості. У прикладі кіоску реєстрації в аеропорту є певні вимоги до продуктивності, а також до безпеки. Ці два атрибути можуть конфліктувати, оскільки додавання рівнів безпеки може уповільнити транзакції. Однак, оскільки визначення пріоритетів вказує на те, що безпека є важливішою (7), ніж продуктивність (4), слід вирішити подібні конфлікти на користь безпеки.

Декілька типових питань для розуміння очікувань користувачів щодо продуктивності системи:

1. Яким має бути час відповіді на запит?

2. Що користувачі вважають неприйнятним часом відповіді для типового запиту?

3. Скільки одночасних запитів від користувачів ви очікуєте в середньому?

4. Яку максимальну кількість одночасних запитів від користувачів ви очікуєте?

5. У який час доби, тижня, місяця чи року запитів набагато більше, ніж зазвичай?

Також слід запитати користувачів, що вважатиметься неприйнятним рівнем продуктивності, безпеки чи надійності. Тобто треба вказати властивості системи, які будуть порушувати очікування користувача щодо якості, наприклад дозвіл неавторизованому користувачеві видаляти файли.

Пріоритезація може включати міркування щодо цінності для споживача, цінності для бізнесу, ділового чи технічного ризику, вартості, складності впровадження, часу виходу на ринок, відповідності нормативним вимогам або вимогам політики компанії, конкурентних переваг на ринку та договірних зобов’язань. Успішне визначення пріоритетів вимагає розуміння наступного:

* потреби клієнтів;
* відносна важливість вимог до клієнтів;
* коли необхідно надати певні можливості та інші види дедлайнів;
* вимоги, які служать базовими/батьківськими для інших вимог та інші зв'язки між вимогами;
* які вимоги необхідно виконувати групою;
* витрати на задоволення кожної вимоги.

Для визначення вимог з низьким пріоритетом можна поставити такі питання, як:

* Чи є якийсь інший спосіб задовольнити потребу, якої стосується ця вимога?
* Якими будуть наслідки пропуску або відстрочення виконання цієї вимоги?
* Якщо ця вимога не буде реалізована протягом кількох місяців, як це вплине на бізнес-цілі проекту?
* Чому клієнт може бути незадоволений, якщо цю вимогу буде відкладено на пізніший реліз?
* Чи наявність цієї функції настільки важлива, щоб відкласти реліз усіх інших функцій із таким самим пріоритетом?