ТЕМА 5. АТРИБУТИ ЯКОСТІ ПЗ

Атрибути якості ПЗ – є додатковим описом функцій продукту, опис характеристик важливих для користувачів або розробників.

До атрибутів якості можна віднести декілька дюжин характеристик продукту (Charette, 1990), хоча для більшості проектів вистачило б всього декілька. Якщо розробникам відомо, які характеристики найбільш важливі для успіху проекту, вони можуть вибрати відповідні прийоми, працюючи над архітектурою, дизайном і програмним наповненням ПЗ, які дозволять досягти певної якості (Glass, DeGrace і Stahl). Для класифікації атрибутів якості застосовуються різні схеми (Boehm, Brown і Lipow, 1976; Cavano і McCall, 1978; IEEE 1992; DeGrace і Stahi, 1993).

У табл. 1 перераховано декілька атрибутів якості ПЗ, які необхідно брати до уваги в будь-якому проекті.

Таблиця 1.

Атрибути якості ПЗ

Важливі для користувача	Важливі для розробника
• Доступність • Ефективність • Гнучкість • Цілісність • Здатність до взаємодії • Надійність • Стійкість до збоїв • Зручність і простота використання	• Простота в експлуатації • Простота переміщення • Можливість повторного використання • Тестованість

Деякі мають велике значення для вбудованих систем (ефективність і надійність), тоді як інші особливо важливі для Інтернет-застосувань і застосувань для мейнфреймів (доступність, цілісність і легкість в експлуатації) або для настільних систем (здібність до взаємодії і зручність і простота використання). Для вбудованих систем часто враховують і інші важливі атрибути якості, наприклад безпека, легкість і, простота встановлення і обслуговування. Для Інтернет-застосувань також важливий ще один атрибут - масштабованість.

5.1. Атрибути, важливі для користувачів

- **Доступність.** Під доступністю розуміється запланований *час доступності* (uptime), протягом якого система доступна для використання і повністю працездатна.

Формально доступність дорівнює *середньому часу до збою* (mean time to failure, MT-TF) системи, діленому на суму середнього часу до збою і очікуваного часу до відновлення системи після збою.

На доступність впливають періоди планового технічного обслуговування.

Доступність як сукупність надійності, легкості в експлуатації і цілісності (Gilb, 1988).

- **Ефективність -** називається показник того, наскільки ефективно система використовує продуктивність процесора, місце на диску, пам'ять або смугу пропускання з'єднання (Davis, 1993).

Якщо система витрачає дуже багато доступних ресурсів, користувачі відмітять зниження продуктивності.

Недостатня продуктивність дратує користувачів ставить під удар безпеку, наприклад, при перевантаженні системи контролю процесів реального часу.

Визначіть мінімальну конфігурацію устаткування, при якій вдається досягти заданих ефективності, пропускній спроможності і продуктивності. Щоб дозволити нижню межу у разі непередбачених умов і визначити подальше зростання, ви можете скористатися таким формулюванням:

Приклад. Як мінімум 25% пропускної спроможності процесора і оперативної пам'яті, доступної застосуванню, не повинно використовуватися в умовах запланованого пікового навантаження.

Типові користувачі не формулюють вимоги до ефективності, вони можуть задати час відгуку або заповнення простору на диску.

Справа аналітика – поставити питання, які виявлять очікування користувачів про прийнятне зниження продуктивності, можливі піки навантаження і очікуване зростання.

- **Гнучкість –** показує з якою легкістю в продукт вдається додати нові можливості.

Якщо очікується, що при розробці доведеться вносити безліч поліпшень, варто вибрати такі рішення, які дозволять збільшити гнучкість ПЗ.

Цей атрибут важливий для продуктів, як модель розробки яких вибрано поліпшення і повтор успішних випусків, або розвиток прототипу.

Приклад. Програміст по технічному обслуговуванню, що не менше шести місяців працює з продуктом, повинен уміти підключати новий пристрій для створення друкарських копій, що передбачає зміну коду і тестування, не більше ніж за годину робочого часу.

Проект не можна вважати невдалим, якщо програмістові вимагається 75 хвилин, щоб встановити новий принтер, отже, це вимога допускає деяку міру свободи.

Якби ми не вказали цю вимогу, можливо, розробники вибрали б такий варіант дизайну, при якому установка нового пристрою в системі зайняла б дуже багато часу.

Записуйте вимоги до якості у форматі, який передбачає можливість їх вимірювання.

- **Цілісність –** включає безпеку пов'язану з блокуванням неавторизованого доступу до системних функцій, запобіганням втраті інформації, антивірусним захистом ПЗ і захистом конфіденційності і безпеки даних, введених в систему.

Цілісність дуже важлива для Інтернет-застосувань. Користувачі систем електронної комерції хочуть забезпечити дані своїх кредитних карток.

Відвідувачі Web-сайтів не бажають, щоб приватна інформація про них або список відвідуваних ними сайтів використовувалися не за призначенням, а

постачальники послуг доступу до Інтернету хочуть захиститися від атак типу «відмова в обслуговуванні» і інших хакерських атак.

У вимогах до цілісності немає місця помилкам. Використовуйте наступні точні терміни для формулювання вимог до цілісності:

перевірка ідентифікації користувача

рівні привілеїв користувача, обмеження доступу або певні дані, які повинні бути захищені.

Приклад. Тільки користувачі, що володіють привілеями рівня Аудитор, повинні мати можливість проглядати транзакції клієнтів.

Уникайте вказувати вимоги до цілісності у вигляді обмежень дизайну, як, скажімо, вимоги до пароля для контролю доступу. Реальна вимога повинна обмежувати доступ до системи неавторизированих користувачів; паролі - всього лише один із способів (хоча і найпоширеніший) виконання цього завдання, Засноване на вибраному підході до ідентифікації користувачів, це базова вимога до цілісності вплине на певні функціональні вимоги, які реалізують функції аутентифікації в системі.

- **Здатність до взаємодії** показує, яким чином система обмінюється даними або сервісами з іншими системами.

Щоб оцінити здатність до взаємодії, вам необхідно знати, які застосування клієнти застосовуватимуть спільно з вашим продуктом і обмін яких даних передбачається.

Користувачі Chemical Tracking System звикли малювати хімічні структури, за допомогою декількох комерційних інструментів, тому вони висунули наступну вимогу до здатності взаємодії:

Приклад. Chemical Tracking System повинна мати можливість імпортувати будь-які допустимі хімічні структури з пакетів ChemiDraw (версії 2.3 або ранішою) і Chem-Struct (версії 5 або ранішою).

Ви також могли б вказати дану вимогу як вимогу до зовнішнього інтерфейсу і визначити стандартні формати файлів, які здатна імпортувати Chemical Tracking System.

Як альтернативу можна визначити декілька функціональних вимог, що відносяться до операції імпорту. Проте іноді, розглядаючи систему з погляду атрибутів якості, ви з'ясовуєте, що деякі певні вимоги не задані. Клієнти не вказали, дану потребу при обговоренні зовнішнього інтерфейсу або функціональності системи. Як тільки аналітик поставив питання про інші системи, з якими доведеться взаємодіяти Chemical Tracking System, прихильник продукту негайно згадав два пакети для малювання хімічних структур.

- **Надійність –** називається вірогідність роботи ПЗ без збоїв протягом певного періоду часу (Musa, lannino і Okumoto, 1987).

Іноді однією з характеристик надійності вважають стійкість до збоїв.
Вимірюють надійність ПЗ:

· як відсоток успішно завершених операцій;

· середній період часу роботи системи до збою.

Визначають кількісні вимоги до надійності, ґрунтуючись на тому, наскільки серйозними виявляться наслідки збоїв і чи виправдана ціна підвищення надійності.

Системи, для яких потрібна висока надійність, слід проектувати з високим ступенем можливості тестування, щоб полегшити виявлення недоліків, що негативно впливають на надійність.

Приклад: Для ПЗ управління лабораторним устаткуванням, призначеним для дослідів з рідкісними дорогими хімікатами, що тривають цілий день, користувачам був потрібний програмний компонент, який забезпечив би надійність проведення експериментів, інші системні функції, такі як періодичний запис в журнал даних про температуру, були не такими важливими. *Умова надійності* – Не більше п'яти з тисячі початих експериментів можуть бути втрачені через збої в ПЗ.

- **Стійкість до збоїв (**відмовостійкість, fault tolerance) **–** розуміють рівень, до якого система продовжує коректно виконувати свої функції, не дивлячись на не вірне введення даних, недоліки підключених програмних компонентів або компонентів устаткування або несподівані умови роботи.

Стійке до збоїв ПЗ легко відновлюється після різних проблем і «не помічає» помилок користувачів.

З'ясувавши вимоги до стійкості роботи ПЗ, запитаєте користувачів, які помилкові ситуації можливі при роботі з системою і як система повинна на них реагувати.

Приклад 1. Якщо при роботі з редактором відбувся збій і користувач не встиг зберегти файл, то редактор повинен відновити всі зміни, внесені раніше, ніж за хвилину до збою, при наступному запуску програми даним користувачем.

Приклад 2. Для всіх параметрів, що описують графіки, повинні бути вказані значення за умовчанням, які ядро Graphics Engine використовуватиме у випадку, якщо дані вхідного параметра вказані невірно або відсутні.

Виконання цієї вимоги дозволить уникнути збою програми, якщо, наприклад, застосування запрошує колір, який плотеру не вдається відтворити. Graphics Engine використовує значення за умовчанням - чорний колір - і продовжить роботу. Проте це можна розглядати як збій ПЗ, оскільки кінцевий користувач не отримав бажаний колір. Проте такий підхід понизив серйозність наслідків збоїв - замість краху програми отриманий неправильний колір, що є прикладом відмовостійкої.

5.2. Атрибути, важливі для розробників

- **Легкість в експлуатації.** Цей атрибут показує, наскільки зручно виправляти помилки або модифікувати ПЗ. Легкість в експлуатації залежить від того, наскільки просто розібратися в роботі ПЗ, змінювати його і тестувати, і тісно пов'язано з гнучкістю і тестованістю. Цей показник украй важливий для продуктів, які часто змінюють, і тих, що створюються швидко (і, можливо, з економією на якості). Легкість в експлуатації вимірюють, використовуючи такі терміни, як *середній час потрібний для дозволу проблеми* і *відсоток коректних виправлень*.

Приклад. Для Chemical Tracking System одна з вимог до легкості в експлуатації сформульовано таким чином:

Легкість в експлуатації 1. Програміст, що займається технічним обслуговуванням ПЗ, повинен модифікувати існуючі звіти, щоб привести їх в .соответствие із змінами в

положеннях федерального уряду в області хімії, витративши на розробку не більше 20 робочих годин.

При роботі над проектом Graphics Engine ми розуміли, що нам доведеться часто вносити виправлення ПЗ, щоб воно відповідало потребам користувачів. Ми вказали приблизно такі критерії, щоб розробникам вдалося створити ПЗ, більш легке в експлуатації:

Легкість в експлуатації-2. Вкладеність функцій, що викликаються, не повинна перевищувати два рівні.

Легкість в експлуатації-3. Для кожного програмного модуля непорожні коментарі в співвідношенні до початкового коду повинні складати як мінімум 0,5. Ставте такі завдання розробникам обережно, інакше ті, позбувшись самовладання, почнуть діяти формально, виконуючи букву, але не суть завдання. Попрацюйте з програмістами, що займаються технічним обслуговуванням, щоб зрозуміти, які якості початкового |коди полегшать їм внесення змін або виправлення недоліків.

Для апаратних пристроїв з вбудованим ПЗ часто є вимоги до легкості в експлуатації. Одні з них відносяться до вибору дизайну ПЗ, тоді як інші впливають на дизайн устаткування. Наприклад останнє можна сформулювати так:

Легкість в експлуатації-4. Дизайн принтера дозволяє сертифікованому ремонтникові замінити кабельний шнур друкуючої головки не більше ніж за 10 хвилин, датчик стрічки не більше ніж за 5 хвилин і привід стрічки не більше ніж за 5 хвилин.

- **Легкість переміщення.** Мірою її вимірювання можна вважати зусилля, необхідні для переміщення ПЗ з одного операційного середовища в іншу. Деякі практики вважають можливість інтернаціоналізації і локалізації продукту вищим ступенем його мобільності Прийоми розробки ПЗ, які роблять легким його переміщення,, дуже схожі з тими, що застосовують, щоб зробити ПЗ багатократного використовуваним (Glass, 1992). Зазвичай мобільність продукту або не має ніякого значення, або украй важлива для успіху проекту. Важливо визначити ті частини продукту, які необхідно легко переміщати в інші середовища, і описати ці цільові середовища. Потім розробники виберуть способи розробки і кодування, які збільшать мобільність продукту.

- **Можливість повторного використання.** Постійне завдання розробки ПЗ - можливість повторного використання - показує зусилля, необхідні для перетворення програмних компонентів; з метою їх подальшого застосування в інших застосуваннях. Витрати на розробку ПЗ з можливістю повторного використання значно вище, ніж на створення компоненту, який працюватиме тільки в одному застосуванні. Воно повинне бути модульним, добре задокументованим, не залежати від конкретних застосування і операційного середовища, а також володіти деякими універсальними можливостями. Цілі багатократного використання складно кількісно зміряти. Вкажіть, які елементи нової системи необхідно спроектувати так, щоб спростити їх повторне застосування, або вкажіть бібліотеки компонентів багатократного використання, які необхідно створити додатково до проекту. Наприклад:

Можливість повторного використання-1. Функції введення хімічних структур повинні бути спроектовані так, щоб їх вдавалося повторно використовувати на рівні об'єктної коди в інших застосуваннях, побудованих з урахуванням міжнародних стандартів для представлення хімічних структур.

- **Тестованість.** Цей атрибут також називають *проверяемостью,* він показує легкість, з якою програмні компоненти або інтегрований продукт можна перевірити на предмет дефектів. Такий атрибут украй важливий для продукту, в якому використовуються складні алгоритми і логіка або є тонкі функціональні взаємозв'язки. Тестіруємость також важлива в тому випадку, якщо продукт необхідно часто модифікувати, оскільки передбачається піддавати його частому регресивному тестуванню, щоб з'ясувати, чи не погіршують внесені зміни існуючу функціональність

Оскільки я і команда розробників твердо знали, що нам доведеться тестувати Graphics Engine багато раз в період його неодноразових удосконалень, ми включили в специфікацію наступну директиву, тестируемости, що стосується, ПЗ:

Тестованість-1. Максимальна цикломатическая складність модуля не повинна перевищувати 20.

Цикломатична складність (cyclomatic complexity) - це кількість логічних відгалужень в модулі початкового коду (McCabe. 1982).

Чим більше відгалужень і циклів в модулі, тим важче його тестувати, розуміти і підтримувати. Звичайно, проект не буде провалений. якщо значення цикломатической складності одного з модулів досягне 24, проте наша директива вказала розробникам рівень якості, до якої слід прагнути. Якби її (вона представлена як вимога до якості) не було, не факт, що розробники при написанні своїх програм взяли б до уваги таку характеристику, як цикломатическая складність. В результаті код програми міг опинитися такий заплутаний, що його ретельне тестування і доповнення виявилося б практично неможливим, а відладка взагалі стала б кошмаром.

5.3. Визначення нефункціональних вимог за допомогою мови Planguage

Деякі з атрибутів якості, перерахованих в цьому розділі, є неповними або неспеціалізованими - дуже важко виразити їх одной-двумя короткими фразами. Вам не вдасться оцінити, чи відповідає продукт неточно сформульованим вимогам до якості чи ні. Крім того, спрощені завдання, пов'язані з якістю продуктивністю, можуть виявитися нездійсненними. Максимальний час відгуку, рівний двом секундам для запиту до БД, чудово працює при нескладному пошуку в локальній БД, але абсолютно нездійсненно при з'єднанні шести зв'язаних таблиць, размещенных на серверах, які розташовані в різних місцях.

Для вирішення проблеми неявних і неповних нефункціональних вимог консультант Tom Gilb (1988; 1997) розробив *Planguage* |мова планування з великим набором ключових слів, який дозволяє точно встановлювати атрибути якості і другу завдання проекту (Simmons, 2001). Далі показано, як виразити вимогу до продуктивності за допомогою всього лише декілька з безлічі ключових слів Planguage.

Приклад є версією вимог на мові Planguage до продуктивності з «95% запитів БД каталога повинні бути виконані протягом 3 секунд на комп'ютері Intel Pentium 1,1 Ггц під управлінням Microsoft Windows XP, коли доступні як мінімум 60% системних ресурсів».

TAG. Продуктивність. Час відгуку на запит.

AMBITION. Швидкий відгук на запити БД на призначеній для користувача платформі.

SCALE. Час (у секундах) між натисненням клавіші Enter і клацанням ОК для відправки запиту і початком відображення результатів запиту

METER. Тестування з використанням секундоміра, виконане на 250 тестових запитах, що представляють певний операційний профіль використання.

MUST. Не більше *10* секунди на 98% запитів. (Фахівець виїзний служби підтримки).

PLAN. Не більше 3 секунд для запитів категорії 1. 8 секунд для всіх запитів

WISH. Не більше двох секунд для всіх запитів.

*Основна призначена для користувача платформа DEFINED***.** Процесор Intel Pentium 4 1,1 Ггц, оперативна пам'ять 128 Мб, під управлінням ОС Microsoft Windows XR зі встановленим пакетом QueryGen версії 3.3, однопользовательский комп'ютер, вільно як мінімум 60% системних ресурсів, інші застосування не виконуються

У кожній вимозі присутній унікальний **Тег** (tag).

Мета (ambition) – встановлює мету або завдання для системи, яка породжує дану вимогу.

Масштаб (scale) – визначає одиниці вимірювання.

Лічильник (meter) – описує точно, як виконати вимірювання. Всі зацікавлені сторони повинні однаково добре розуміти, яким чином вимірюється продуктивність. Припустимо, користувачу простіше сприймає систему вимірювань, яка заснована на часі від натиснення клавіші Enter до виведення всіх результатів запиту, чим до початку відображення результатів, як вказано в прикладі, Розробник може заявити, що вимога задоволена, а користувач стверджуватиме зворотне. Точно виражені вимоги до якості і системи вимірювань допоможуть запобігти такого роду непорозумінням.

Ви можете вказати декілька бажаних кількісних величин, які необхідно вимірювати.

Критерій **must** (зобов'язання) – визначає найменший досяжний рівень. Вимога не задоволена до тих пір, поки не будуть задоволені всі умови must, таким чином, ця умова повинна бути обґрунтована в бізнес-термінах. Можна вказати вимогу must іншим способом. Для цього потрібно визначити умову **fait** (невдача) (ще одне ключове слів мови Planguage), наприклад: «Більше 10 секунд очікування для більше 2% всіх запитів». Величина **plan** (план) указує номінальне значення, **wish** (побажання) є ідеальним результатом. Крім того, покажіть джерело необхідної продуктивності, наприклад, згадану вище умову must (зобов'язання) вказав фахівець виїзної служби підтримки. Для полегшення читання будь-які спеціалізовані терміни, використовувані в операторах мові Planguage, задані як **defined** (визначувані).

Planguage визначає безліч додаткових ключових слів, за допомогою яких вдається добитися гнучкішою і точнішою специфи-кации вимог. Оскільки мова Planguage, його словник і синтаксис ще розвиваються, рекомендую звернутися за свіжою інформацією на Web-сайт *http://www.gilb.com*. Planguage є могутнім засобом для точного формулювання атрибутів якості і вимог до продуктивності. Вказавши декілька рівнів для очікуваного результату, ви отримаєте ширше уявлення про вимоги до .качеству, чим за допомогою простих конструкцій, таких, як «чорне -біле» і «так- ні».

5.4. Реалізація нефункціональних вимог

Дизайнери і програмісти повинні визначити якнайкращий спосіб задоволення вимоги для кожного атрибуту якості і продуктивності. Хоча атрибути якості відносяться до нефункціональних вимог, вони допомагають сформулювати функціональні вимоги, визначити напрями розробки або технічну інформацію, яка дозволить реалізувати продукт бажаної якості. У табл. 2 перераховані деякі категорії технічної інформації, які можуть бути виведені за допомогою атрибутів якості.

Таблиця 2.
Перетворення вимог до якості в технічну документацію

Типи атрибутів якості	**Категорія технічної інформації**
Цілісність, здібність до взаємодії, стійкість до збоїв, легкість і простота використання, безпека	Функціональні вимоги
Доступність, ефективність, гнучкість, производительность, надійність	Архітектура системи
Здібність до взаємодії, легкість і простота використання	Обмеження дизайну
Гнучкість, легкість в експлуатації, легкість перемещения. надійність, можливість повторного використання, тестируемость, легкість і простота використання	Керівництво ПЗ дизайну
Легкість переміщення	Обмеження реалізації

Наприклад, в медичний пристрій із строгими вимогами до доступності може входити джерело резервного живлення (архітектура) і функціональні вимоги для візуального або звукового сповіщення, коли продукт працює на резервному живленні. Це перетворення вимог до якості, що мають значення для користувачів або для розробників, у відповідну технічну інформацію є частиною процесу розробки вимог і дизайну високого рівня.