Лекція 7

Управління якістю та тестування

Анотація: Стандартизація якості. Методи забезпечення якості ПЗ. Поняття тестування. Тестування чорного ящика. Тестування білого ящика. Інструменти тестування. Критерії тестування. Види тестування. Робота з помилками. Засоби контролю помилок (bug tracking systems).

Управління якістю

Стандартизація в сучасному бізнесі і промисловості.

Розвиток світового ринку призвів до того, що багато товарів і послуг почали розповсюджуватися в усьому світі, набули розвитку глобальні сервіси, зокрема, телекомунікаційні, банківські. Для того, щоб усунути технічні бар'єри в промисловості, торгівлі і бізнесі, які виникли унаслідок того, що в різних країнах для одних і тих самих технологій і товарів діяли різнорідні стандарти, почали створюватися національні і міжнародні комітети із стандартизації. Зупинимося на найвідоміших міжнародних комітетах.

1. 1865 рік – створений комітет, який зараз має назву ***ITU*** (*International* Telecommunication *Union*). На сьогоднішній день його штаб-квартира знаходиться в Женеві (Швейцарія), а ITU є частиною ООН. Його основне завдання – стандартизація телекомунікаційних протоколів і інтерфейсів з метою підтримки і розвитку глобальної світової *телекомунікаційної мережі*. Найвідомішими стандартами *ITU* є:
   - ***ISDN*** (цифровий телефонний зв'язок, який об'єднує телефонні сервіси і передачу даних);
   - ***ADSL*** (широко відома модемна технологія, що дозволяє використовувати телефонну лінію для виходу в Інтернет і при цьому не блокує звичайний телефонний сервіс);
   - ***OSI*** (модель відкритого 7-рівневого мережевого протоколу, на якій базуються всі сучасні стандартні мережеві інтерфейси і протоколи; також є стандартом ISO);
   - мови візуального проектування телекомунікаційних систем, SDL і MSC, які пізніше влилися в UML.

Багато стандартів ***ITU*** перекладаються державними мовами країн світу і перетворюються на національні стандарти.

2. 1946 рік – створена організація ***ISO*** (*International Organization*for *Standardization*). Мета організації – сприяння розвитку у світі стандартизації, а також суміжних видів діяльності з метою забезпечення міжнародного обміну товарами і послугами, сприяння і розвиток співпраці в інтелектуальній, науково-технічній і економічній сферах. До теперішнього часу створено близько 17 000 стандартів у

самих різних галузях промисловості – продовольчі і інші товари, різне устаткування, банківські сервіси і так далі. Наведемо деякі стандарти.

- Серія стандартів ***ISO 9000***. Ця серія націлена на стандартизацію якості товарів і послуг. Стандарти надають визначення якості, визначення системи підтримки якості на всіх життєвих фазах виробу, товару, послуги (проектування, розробка, комерціалізація, встановлення і обслуговування), опис процедур з поліпшення діяльності компаній, промислового виробництва.
- ***ISO/IEC 90003:2004*** – адаптація стандартів ISO 9000 до виробництва ПЗ з метою забезпечення якості в життєвому циклі ПЗ.
- ***ISO 9126:2001*** – визначення якісного ПЗ і різних атрибутів, що описують цю якість.

Багато стандартів ISO перекладаються українською мовою і перетворюються на українські стандарти у вигляді ДСТУ. Є багато стандартів в галузі інформаційних технологій, а також декілька – в галузі програмної інженерії. На відповідність стандартам ISO існує сертифікація. Зокрема, компанії сертифікуються на відповідність стандартам ISO 9000, тобто на якісний процес розробки ПЗ.

3. 1988 рік, утворення організації ***ETSI*** (European *Telecommunications* Standards Institute), штаб-квартира в технологічному парку на північному заході від Антіб й на південному заході від Ніцци у Франції. ***ETSI*** є незалежною, некомерційною, організацією із стандартизації в телекомунікаційній промисловості (виробники устаткування і оператори мереж) в Європі. Найвідоміші стандарти – ***GSM***, система професійного мобільного радіозв'язку ***TETRA***.

Міжнародні комітети, які безпосередньо пов'язані з розробкою ПЗ.

1. 1984 рік – створення ***SEI*** (*Software Engineering* Institute) на базі університету Карнегі-Меллон в м. Пітсбурзі (США). Ініціатор і головний спонсор – міністерство оборони США. Основне завдання – стандартизація в галузі програмної інженерії, розробка критеріїв для сертифікації надійних і зрілих компаній (що в першу чергу цікавить Міноборони США для виконання його замовлень). Найвідоміші продукти:
   - ✓ стандарт ***CMM***,
   - ✓ стандарт ***CMMI***,
   - ✓ розробки в галузі сімейства програмних продуктів (*product* lines).

   Ці продукти зробили крок далеко за межі військових розробок США, їх використання і розвиток стало міжнародною діяльністю. Деякі продукти SEI стандартизовані також і ISO. На відповідність CMM/CMMI виконується сертифікація.

2. 1963 рік – створення ***IEEE*** (Institute of *Electrical* and *Electronics Engineers*). Веде історію з кінця XIX століття, в контексті промислової стандартизації в США. Зараз *IEEE* міжнародна некомерційна асоціація фахівців в галузі техніки, світовий лідер в галузі розробки стандартів для радіоелектроніки і електротехніки. Штаб-квартира знаходиться в США, існують численні підрозділи в різних країнах, включаючи і Українську Секцію IEEE (IEEE Ukraine Section). IEEE видає третю частину світової технічної літератури, що відноситься до застосування радіоелектроніки, комп'ютерів, систем управління, електротехніки, зокрема (січень 2008) 102 реферативних наукових журналів і 36 галузевих журналів для фахівців, проводить в рік більше 300 великих конференцій, брала участь в розробці близько 900 стандартів, що діяли або діють і дотепер.

3. 1989 рік – група американських IT-компаній (зокрема Hewlett Packard, Sun Microsystems, Canon) організували ***OMG*** (*Object Management Group*). Зараз включає близько 800 компаній-членів. Основний напрямок – розробка і просування об'єктно-орієнтованих технологій і стандартів, зокрема для створення незалежних програмних додатків рівня підприємств. Відомі стандарти ***CORBA***, ***UML***, ***MDA***.

Всі ці комітети і організації включають програмну інженерію в сферу своєї діяльності, співробітничають, видають спільні стандарти, використовують напрацювання один одного і так далі.

Стандартизація якості.

З погляду тестування ПЗ в цих стандартах слід звернути увагу на стандартизацію якості (в контексті тестування) продукції, що спочатку випускається, а потім поширюються і на процеси з її розробки. Тут спрацьовує ідея про те, що якісного результату неможна створити без якісного процесу. Забезпечення якості є більш загальним контекстом для тестування.

Якість продукту або сервісу, який призначений для споживача, визначається в стандарті ISO 9000:2005, як ступінь відповідності його характеристик вимогам – обов'язковим або таким, що мається на увазі.

Методи забезпечення якості ПЗ.

Не претендуючи на абсолютну повноту, перерахуємо різні способи контролю якості, які використовуються на практиці підчас розробки ПЗ.

- ***Налагоджування якісного процесу***, іншими словами – це вдосконалення процесу. Для комплексного поліпшення процесів в компанії (підхід ***technology push***) компаніями-розробниками ПЗ використовуються стандарти CMM/CMMI та стандарти серії ISO 9000 (з подальшою офіційною сертифікацією). Застосовуються і локальні стратегії, які є менш затратними і більш спрямовані на вирішення окремих проблем (підхід ***organization pull***).

- ***Формальні методи*** – використання математичних формалізмів для доказу коректності, специфікації, перевірки формальної відповідності, автоматичної генерації і т.д. Серед них:
  - доказ правильності роботи програм;
  - перевірка на моделях певних властивостей (model cheking);
  - статичний аналіз коду за деревом розбору програми (наприклад, перевірка коректності коду за певними критеріями – акуратна робота з пам'яттю, пошук мертвого коду і ін.);
  - модельно-орієнтоване тестування (model-based testing): автоматична генерація тестів і тестового оточення за формальними специфікаціями вимог до системи і так далі.

  На практиці застосовуються обмежено через необхідність серйозної математичної підготовки користувачів, складність в освоєнні, великі обсяги робіт з розгортання. Ефективні для систем, що мають підвищені вимоги до надійності. Також є випадки ефективного використання засобів, які засновані на цих методах, з боку висококваліфікованих фахівців.

- ***Дослідження і аналіз динамічних властивостей ПЗ***. Як приклад, широко використовується профілізація – дослідження використання системою пам'яті, її швидкодія і ін. характеристик шляхом запуску і безпосередніх спостережень у вигляді графіків, звітів і ін. Зокрема, цей підхід використовується у випадку розпаралелювання програм, під час пошуку "вузьких" місць. Ще приклад – галузь, яка має назву "моделювання і аналіз продуктивності" (performance modeling and analysis). В цьому випадку моделюється навантаження системи з боку оточення (кількість одночасних користувачів системи, мережевий трафік і ін.) та спостерігається поведінка системи.

- ***Забезпечення якості коду.*** Сюди відноситься цілий комплекс різних заходів і методів. Нижче наведені найбільш відомі з них.
  - *Розробка стандартів оформлення коду* в проекті і контроль за дотриманням цих стандартів. Сюди входять правила на створення ідентифікаторів змінних, методів і імен класів, на оформлення коментарів, правила використання стандартних для проекту бібліотек і так далі.
  - *Регулярний рефакторінг* для запобігання створення з коду «вермішелі». Існує тенденція погіршення структури коду у випадку внесення до нього нової функціональності, виправлення помилок і ін. З'являється надмірність, утворюються невживані або слабо використовувані фрагменти, структура стає заплутаною і важкою для розуміння. Рефакторінг – це регулярна діяльність з переписуванню коду, але не з метою додавання нової функціональності, а для поліпшення його структури.

Рефакторінг з'явився в контексті "гнучких" методів, в даний момент активно підтримується різними середовищами розробки ПЗ.

- Різні *варіанти інспекції коду*, наприклад, техніка *peer code review*. Останній полягає в тому, що код кожного учасника проекту, вибірково, розглядається і обговорюється на спеціальних зустрічах (code review meetings), і робиться це регулярно. Практика показує, що в цілому код поліпшується.
- Є ще такий підхід, як *"вичитування" коду*, що використовується, наприклад, під час розробки критичних систем реального часу. Ним також займаються розробники, але їхня роль в даному проекті – вичитування, а не розробка.

- ***Тестування***. Найпоширеніший спосіб контролю якості ПЗ, який існує, фактично, в кожному програмному проекті.

Тестування

Тестування – це перевірка відповідності між реальною поведінкою програми і її очікуваною поведінкою за спеціально заданих, штучних умов. Розберемо це визначення за складовими.

Очікувана поведінка програми. Початковою інформацією для тестування є знання про те, як система повинна поводитися, тобто вимоги до неї або до її окремої частини. Найпоширенішим способом тестування є тестування методом **чорного ящика**, тобто коли реалізація системи є недоступною для тестувальників, а тестується лише її інтерфейс. Часто це закріплюється і організацією колективу – тестувальники виявляються окремими співробітниками і в деяких компаніях вони навіть принципово не спілкуються з розробниками, щоб мінімально знати деталі реалізації і максимально повно виступити в ролі перевіряючої інстанції. Існує тестування методом **білого ящика**, коли код програм є доступним тестувальникам і використовується як джерело інформації про систему. Його схема наведена на Рис. 7.1.

На Рис. 7.1 видно, що на основі вимог до системи створюється реалізація і тестова модель системи. Тестування є зіставлення цих двох уявлень з метою виявити їх невідповідності. Чим більш незалежні один від одного будуть ці уявлення, тим більше пуття від їх зіставлення. Інакше, якщо тестувальники істотно використовують інформацію про реалізацію системи підчас складання тестів, то вони можуть мимоволі ввести у тести помилки реалізації. Знайдена при тестуванні невідповідність – це ще не помилка, оскільки самі тестувальники могли неправильно зрозуміти вимоги, тому і в тестах, і в засобах тестування могли бути власні помилки.

Даний підхід закріплюється також і в організації колективів програмістів – тестувальники, як правило, відокремлені від розробників. Це різні люди, що виконують несумісні ролі в MSF. Існує приклад однієї

американської компанії де розробники і тестувальники розміщувались на різних поверхах, ходили в різному одязі (тестувальники в костюмах, розробники – в светрах) і начальство не заохочувало неробочі відносини між цими групами. Це, звичайно ж, крайність, але вона ще раз підкреслює, як важливо, щоб точка зору на систему у тестерів відрізнялася від точки зору розробників. Але, звичайно, і та, і інша точки зору повинні виходити із загального бачення системи – її вимог.

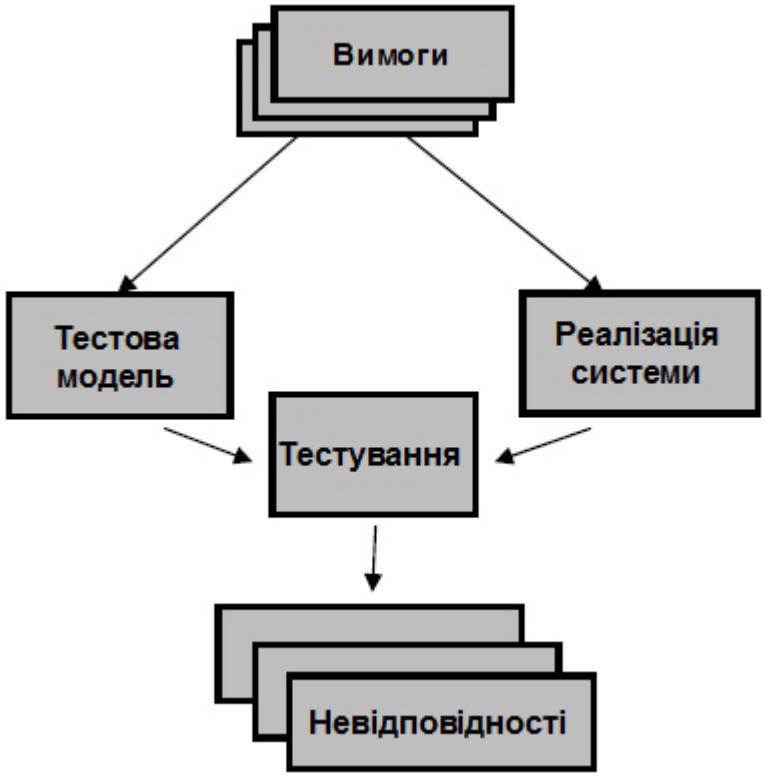

Рис. 7.1. Схема тестування методом білого ящика

Спеціально задані, штучні умови, – це ті умови, за яких виконується тестування. При цьому ключовим аспектом тут є наявність **тестів –** відтворених кроків маніпуляції з системою, які призводять до її некоректної роботи. Концепція тесту дуже важлива, оскільки необхідно не просто виявити некоректну поведінку системи, а створити і зафіксувати алгоритм відтворення помилки – щоб повторити його для розробника, або щоб розробник сам зміг відтворити помилку. Якщо помилка не відтворюється, то не існує можливості її виправити.

Тести можуть бути "ручними" і автоматизованими.

"Ручний" тест – це послідовність дій тестувальника, яку він (або розробник) може відтворити і помилка станеться. Як правило, в засобах контролю помилками такі послідовності дій містяться в описі помилки.

Автоматизований тест – це деяка програма, яка впливає на систему і перевіряє ту або іншу її властивість. Автоматизований тест, в порівнянні з "ручним", можна легко відтворювати без участі людини.

Можна створювати набори тестів і "проганяти" їх дуже часто, наприклад, в режимі регресійного тестування. Крім того, автоматизовані тести можна генерувати за більш високорівневими специфікаціями, наприклад, за формально описаними вимогами до системи. А тести для компіляторів можна генерувати за формальним описом мови програмування.

Таким чином, переваги автоматизованих тестів перед "ручними" є очевидними. Звернемо увагу на **труднощі автоматизованого тестування**.

По-перше, для того, щоб автоматично запускати тести, потрібні відповідні програмні продукти, які також є невід'ємною частиною спеціально заданих, штучних умов. Їх називають **інструментами тестування**. У їх завдання входить запуск тесту на системі, "прогін" цілого пакету тестів, а також аналіз отриманих результатів і їх обробка.

Крім того, важливим завданням інструментів тестування є забезпечення доступу тесту до системи через деякий її інтерфейс. Доступ до системи може виявитися складним, наприклад, через політичні обставини, коли сторонніми розробниками створюється підсистема деякої стратегічної системи, і доступ до цієї глобальної системи у субпідрядників сильно обмежений. Або через апаратні обмеження – важко "залізти" на "сервер", де працює цільовий код системи.

Крім того, часто важко виконати "безшовне" тестування системи, коли створюється на систему мінімальне навантаження, а при цьому добираються до всіх аспектів її функціонування. В цілому, налагоджування і розгортання готових, сторонніх тестових інструментів часто виявляється дорогим і нетривіальним завданням. Розробка своїх власних тестових інструментів також є непростою.

По-друге, часто виникає проблема ресурсів для автоматичного тестування. Особливо у разі автоматичної генерації тестів: часто є можливість автоматично згенерувати дуже велику кількість тестів, і якщо їх ще й регулярно виконувати в режимі безперервної інтеграції, то може не вистачити наявних системних ресурсів. При цьому якість тестування може виявитися незадовільною – помилки знаходяться не часто, або взагалі не знаходяться. Річ у тому, що кількість всіх можливих станів програмної системи дуже велика, і тестування не може перевірити їх всіх.

На практиці, в реальних проектах, визначають **критерії тестування**, які визначають ту "планку" якості, яку необхідно досягти в цьому проекті. Адже гарна якість коштує дорого і, вочевидь, різне ПЗ має різну якість, наприклад, система управління ядерним реактором і текстовий редактор. На практиці часто якість ПЗ визначається бюджетом проекту підчас його розробки.

Через обмеженість ресурсів на тестування часто доцільно буває визначити ті аспекти ПЗ, які найбільш важливі – як для загальної працездатності системи, так і для замовника. Наприклад, у разі тестування WEB-додатку, що надає послугу зі створення оголошень про продаж нерухомості, такими критеріями могли би бути:

- відповідність переходів пошукового майстра, зокрема, і можливість переходів назад, до попередніх переглядів;
- цілісність введених користувачем даних про створювані оголошення.

Нарешті, окрім обмеження кількості тестів для відбору, важливим є їх прогін на деяких (не на всіх можливих!) вхідних даних. Часто тут застосовують **принцип факторизації –** безліч всіх можливих вхідних значень розбивають на значущі з погляду тестування класи і "проганяють" тести не на всіх можливих вхідних значеннях, а беруть лише один набір значень з кожного класу. Наприклад, тестують деяку функцію системи на її граничних значеннях – дуже великі значення параметрів, дуже маленькі і ін. Часто факторизацію зручно виконувати, виходячи з вимог до даної функції. Також буває корисним подивитися на її реалізацію і "пройтися" тестами різними її логічними гілками (які породжуються, наприклад, умовними операторами).

Види тестування.

Можна виділити такі види тестування (не повний перелік).

- **Модульне тестування –** тестується окремий модуль, у відриві від решти системи. Найпоширеніший випадок застосування – тестування модуля самим розробником, перевірка того, що окремі модулі, класи, методи виконують дійсно те, що від них очікується. Різні середовища розробки широко підтримують засоби модульного тестування – наприклад, є популярною вільно поширювана бібліотека Nunit для Visual Studio, Junit – для Java і так далі. Створені розробником модульні тести часто включаються в пакет регресійних тестів і можуть запускатися багато разів.
- **Інтеграційне тестування –** два і більше компонентів тестуються на сумісність. Це дуже важливий вид тестування, оскільки різні компоненти можуть створюватися різними людьми, в різний час, на різних технологіях. Цей вид тестування, безумовно, повинен застосовуватися самими програмістами, щоб, як мінімум, упевнитися, що все разом працює, хоча б у першому наближенні. Далі тонкощі інтеграції можуть досліджувати тестувальники. Необхідно відзначити, що такого роду помилки – "помилки на стиках" – непросто виявляти і усувати. Під час розробки всі компоненти одночасно можуть не бути готові, інтеграція відкладається, а в кінці проекту виявляються важкі

помилки (в тому сенсі, що їх усунення вимагає істотної роботи). В такому випадку рішенням проблеми може стати рання інтеграція системи, а надалі використання практики постійної інтеграції.

- **Системне тестування –** це тестування всієї системи в цілому, як правило, через призначений для її користувача інтерфейс. При цьому тестувальники, менеджери і розробники акцентують увагу на тому, як ПЗ виглядає і працює в цілому, чи зручне воно, чи задовольняє воно очікуванням замовника. При цьому можуть відкриватися різні дефекти, такі як незручність у використанні тих або інших функцій, забуті або такі, що були не до кінця зрозумілими, вимоги.
- **Регресійне тестування –** тестування системи в процесі її розробки і супровід на регрес. Тобто перевіряється, що зміни у системі не погіршили вже існуючої функціональності. Для цього створюються пакети регресійних тестів, які запускаються з певною періодичністю, – наприклад, в пакетному режимі, який пов'язаний з процедурою постійної інтеграції.
- **Тестування навантаженням –** тестування системи на коректну роботу з великими об'ємами даних. Наприклад, перевірка баз даних на коректну обробку великого (граничного) об'єму записів, дослідження поведінки серверного ПЗ у разі великої кількості клієнтських з'єднань, експерименти з граничним трафіком для мережевих і телекомунікаційних систем, одночасне відкриття великої кількості файлів, проектів і так далі.
- **Стресове тестування –** тестування системи на стійкість до непередбачених ситуацій. Цей вид тестування потрібний далеко не для кожної системи, оскільки має на увазі високу планку якості.
- **Приймальне тестування –** тестування, що виконується на етапі прийманні системи замовниками. Більш того, різні стандарти часто включають набори приймальних тестів. Наприклад, існує великий пакет тестів, що підтримуються компанією SUN Microsystems і є обов'язковими для прогону для всіх нових реалізацій Java-машини. Вважається, що лише після того, як всі ці тести успішно проходять, нова реалізація має право називатися Java.

Робота з помилками

Між програмістами і тестувальниками необхідний спеціальний **інтерфейс спілкування**. Адже помилок знаходиться багато, їх виправлення вимагає часу, і після їх виправлення розробниками тестувальники повинні упевнитися, що вони дійсно виправлені. Крім того, менеджерам потрібна статистика щодо знайдених і виправлених помилок – це хороший інструмент контролю проекту. Все це зображено

на Рис. 7.2. Щоб обробити такий потік інформації і забезпечити необхідні в роботі зручні сервіси, існує спеціальний клас програмних засобів – **засоби контролю помилок** (*bug tracking systems*).

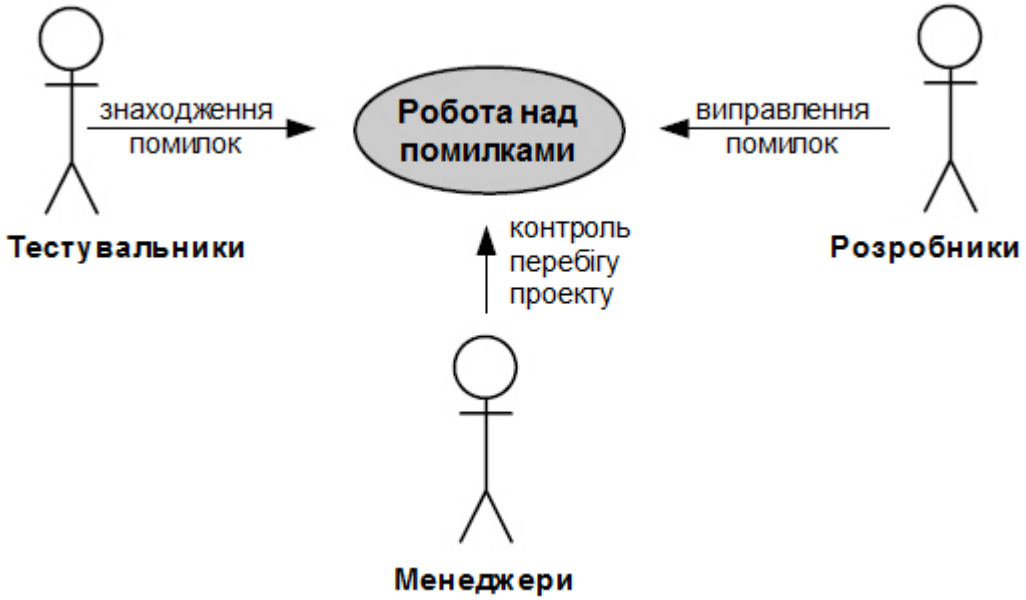

Рис. 7.2. Схема роботи з помилками ПЗ

Як правило, опис помилки в системі контролю помилок має такі основні атрибути:

- відповідального за її перевірку – тестувальника, який її знайшов і який перевіряє, що виправлення, які були виконані розробником, дійсно усувають помилку;
- відповідального за її виправлення – розробника, якому помилка відправляється для виправлення;
- стан, наприклад, помилка знайдена, помилка виправлена, помилка закрита, помилка знов виявилася і так далі.

Цей список істотно доповнюється в різних програмних засобах контролю помилок, але це основні атрибути.

Використання цих систем давно стало загальною практикою підчас розробки ПЗ, нарівні із засобами версійного контролю і багатьма іншими інструментами. Вони включають:

- базу даних для зберігання помилок;
- інтерфейс до цієї бази даних для внесення нових помилок і визначення їх численних атрибутів, для перегляду помилок на основі різних фільтрів – наприклад, всі знайдені помилки за останній місяць, всі помилки, за які відповідає даний розробник і т.д.;
- мережевий доступ, оскільки проекти все частіше виявляються розподіленими;

- програмний інтерфейс для можливостей програмної інтеграції таких систем з іншим ПЗ, що підтримує розробку ПЗ (наприклад, із засобами безперервної інтеграції – вони можуть автоматично вносити до бази даних знайдені підчас автоматичного прогону тестів помилки).

Дуже важливим для роботи з помилками виявляються різні звіти.