Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук
(повна назва)

Кафедра Програмної інженерії
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка
рівень вищої освіти – другий (магістерський)

Дослідження методів управління багтрекінговими системами з метою їх оптимізації
(тема)

Виконав: студент 2 курсу, групи ІПЗм-18-3
Кузін В.С.
(прізвище, ініціали)

спеціальності 121 – Інженерія програмного забезпечення
(код і повна назва спеціальності)

Освітньо-наукова програми
(тип програми)

Інженерія програмного забезпечення
(повна назва освітньої програми)

Керівник доц. Лановий О.Ф.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____ З.В.Дудар

2020 р.

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ РАДІОЕЛЕКТРОНІКИ

Факультет _Комп'ютерних наук_____

Кафедра _Програмної інженерії_____

Рівень вищої освіти – другий (магістерський)

Спеціальність _121 – Інженерія програмного забезпечення_____
(код і повна назва)

Тип програми _освітньо-наукова програма_____

Освітня програма _Інженерія програмного забезпечення_____

ЗАТВЕРДЖУЮ:

Зав. кафедри _____
(підпис)
«____»_____20__ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові _Кузіну В'ячеславу Сергійовичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи _Дослідження методів управління багтрекінговими системами з метою їх оптимізації
затверджена наказом університету від "__"_____ 20__ р. № _____
заповнюється вручну після отримання наказу
2. Термін подання студентом роботи до екзаменаційної комісії
10 травня 2020 р.
3. Вихідні дані до роботи: Баг, баг-трекінгова система, життєвий цикл, управління проектами, критерії оптимізації управління
4. Перелік питань, що потрібно опрацювати в роботі *аналіз предметної області, дослідження систем управління проектами та відстеження помилок, постановка задачі, визначення критеріїв, порівняльна характеристика баг-трекінгових систем, висновки.*

5. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім’я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи (проекту)	Термін виконання етапів проекту (роботи)	Примітка
1.	Аналіз предметної галузі	27 січня 2020	виконано
2.	Огляд існуючих методів та моделей	13 лютого 2020	виконано
3.	Експериментальне дослідження ефективності існуючих моделей	6 квітня 2020	виконано
4.	Підготовка пояснювальної записки	16 квітня 2020	виконано
5.	Підготовка презентації та доповіді	2 травня 2020	виконано
6.	Попередній захист	7 травня 2020	виконано
7.	Нормоконтроль, рецензування	8 травня 2020	виконано
8.	Занесення диплома в електронний архів	9 травня 2020	виконано
9.	Допуск до захисту у зав. кафедри	10 травня 2020	виконано

Дата видачі завдання: «30» березень 2020 р.

Студент _____ Кузін В.С.

(підпис)

Керівник роботи _____ доц. Лановий О.Ф.

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної магістерської роботи містить 80 сторінок, 16 рисунки. 12 таблиць.

Об’єкт дослідження – Системи управління проектами і відстеження дефектів.

Мета дослідження – аналіз існуючих систем відстеження дефектів, переваги, недоліки та її особливості, рекомендації щодо виправлення проблемних місць цих систем.

В результаті роботи був проведений аналіз предметної області, були досліджені і проаналізовані різні статті пов'язані з управлінням і відстеження помилок, дано загальне поняття про системи управління і відстеження і наведено їх переваги при використанні в розробки програмного забезпечення.

РОЗРОБКА ПРОГРАМНОГО ПРОДУКТУ, СИСТЕМИ ВІДСТЕЖЕННЯ ДЕФЕКТІВ, ТЕСТУВАННЯ, JIRA.

.

The explanatory note to the attestation master's thesis contains 82 pages, 16 figures. 12 tables.

Object of study - Defective management and tracking systems.

The purpose of the study is to analyze the existing defect tracking systems, advantages, disadvantages and their features, recommendations for fixing the problem areas of these systems,

As a result of the research practice, a domain analysis was conducted, various articles related to management and error tracking were investigated and analyzed, a general concept of management and tracking systems was given and their advantages when used in software development.

DEVELOPMENT OF SOFTWARE, DEFECTS MONITORING SYSTEMS, TESTING, JIRA

ЗМІСТ

ВСТУП

Однією з неминучих проблем, з якою стикаються підприємства, що займаються розробкою і впровадженням програмного забезпечення, є виникнення різних помилок. Вони виникають як на етапах розробки, так і при використанні програмного продукту користувачем.

Безумовно, кожен розробник прагне до того, щоб мінімізувати кількість помилок в роботі свого програмного продукту і тим самим поліпшити його якість. Ці проблеми найчастіше з'являються в командах розробників, які знаходяться в різних філіях компанії або навіть в різних компаніях. Проблеми у розробників можуть виникнути різні. Наприклад, два співробітники можуть виконати одну і ту ж роботу; може вийти так, що один співробітник занадто багато працює, а інший простоює, так як для керівництва може виявитися складним відстеження зайнятості всіх співробітників.

На сьогоднішній день, у багатьох компаніях приймається рішення про те, що необхідно наймати фахівців, які б займалися тестуванням продуктів, що розробляються програмістами. Однак дані фахівці, найчастіше займаються так званим «ручним» тестуванням.

Важливо, щоб інформація, представлена в звітах про помилки, була актуальною і повної, щоб допомогти швидко усувати помилки.

Однак часто така інформація надходить розробникам після кількох ітерацій спілкування між розробниками і творців звітів про помилки. Погано спроектовані системи стеження за вадами частково винні в тому, що цей обмін інформацією розтягується з плином часу.

Робота всіх цих систем ґрунтується на роботі з помилками або дефектами, інформація про яких зберігається в окремій базі даних. Ці відомості можуть складатися з ідентифікатора помилки; короткого опису помилки; того, хто повідомив про помилку; коли була виявлена помилка; версія продукту, в якій виявлена помилка; критичність помилки і пріоритет її розгляду; відтворення

неправильної поведінки програми для виявлення помилки; очікуваний результат і фактичний результат роботи над помилкою; на кому лежить відповідальність за усунення помилки; обговорення можливих рішень; поточний стан помилки; номер версії продукту, в якій помилка виправлена. Крім того, найбільш просунуті системи надають можливість прикріплювати файли, що описують проблему (скріншот, відео матеріал, лог-файл).

У багатьох програмних проектах системи відстеження помилок відіграють центральну роль: вони дозволяють користувачам спілкуватися з розробниками, щоб вони могли знати про будь-які проблеми та вимагати нових функцій. Крім того, розробники можуть відслідковувати будь-які невирішені помилки та вимагати додаткову інформацію від користувачів. Однак більшість систем відстеження помилок далеко не ідеальні. Багато з них є просто кращим інтерфейсом до бази даних, в якій зберігаються всі повідомлення про помилки. Як результат, вони часто занадто багато запитують у кінцевих споживачів, які не знайомі з практикою розвитку. У той же час вони викликають розчарування у розробників, які розчаровуються щодо якості звітів про помилки, що надсилаються користувачами.

Метою роботи є дослідження методів управління багтрекінговими системами з метою їх оптимізації, для покращення процесу розробки програмного забезпечення.

В роботі буде проведено аналіз предметної галузі, досліджені основні проблеми присутності помилок в програмних проектах, також буде проведено аналіз аналогів існуючих систем відстеження за помилками та побудована модель порівняння існуючих аналогів, наприкінці будуть описані проблемні місця цих систем і можливі варіанти їх вирішення.

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Цілі виявлення помилок і критерії їх ранжування

Під час процесу управління дефектами деякі зі звітів можуть містити описи помилкових спрацьовувань, а не фактичних збоїв через дефекти. Наприклад, тест міг пройти не успішно при збої мережевого з'єднання або таймауті. Така поведінка не є наслідком дефекту об'єкта тестування, але аномалією, яку необхідно досліджувати. Тестувальники повинні спробувати звести до мінімуму кількість помилкових спрацьовувань, прийнятих за дефекти[1].

Оскільки однією з цілей тестування є виявлення дефектів, необхідно реєструвати дефекти, виявлені під час тестування. Спосіб реєстрації дефектів може варіюватися в залежності від контексту тестованого компонента або системи, рівня тестування і моделі життєвого циклу. Будь-які виявлені дефекти повинні бути вивчені й відстежуватися від моменту виявлення і класифікації до прийняття рішення по ним (наприклад, виправлення дефектів і успішного підтверджуючого тестування рішення, відстрочки до наступного релізу, трактування як постійного обмеження продукту). Щоб управляти дефектами до прийняття рішення по ним, в організації повинен існувати процес управління дефектами, який включає в себе життєвий цикл і правила класифікації дефектів. Цей процес повинен бути узгоджений з усіма, хто бере участь в управлінні дефектами, включаючи проектувальників, розробників, тестувальників і власників продуктів. У деяких організаціях реєстрація та відстеження дефектів можуть бути дуже слабо формалізованими.

Дефекти можуть бути виявлені під час кодування, статичного аналізу, рецензування, динамічного тестування або експлуатації програмного продукту. Дефекти можуть повідомляти про проблеми в коді або експлуатованих системах, в документації будь-якого типу, включаючи вимоги, призначені для користувача історії та критерії приймання, документацію розробки, документацію тестування, керівництва користувача або керівництва по установці. Щоб мати продуктивний та

ефективний процес управління дефектами, організації можуть визначати стандартний набір атрибутів, правила класифікації та життєвий цикл дефектів.

Звіти про дефекти мають наступні цілі:

– надавати розробникам та іншим сторонам інформацію про негативні події, що відбулися, щоб вони могли визначити побічні ефекти, ізолювати проблему з мінімальними витратами на відтворення і виправити потенційні дефекти в міру необхідності, або розв'язувати проблеми іншими способами;

– забезпечити керівників тестування інструментами відстеження якості продукту і впливу на тестування (наприклад, якщо повідомляється про велику кількість дефектів, то тестувальники будуть змушені витрачати багато часу на звітність по знайденим дефектам замість того, щоб запускати тести; отже, потрібно більше підтверджуючого тестування);

– надати ідеї для вдосконалення процесів тестування та розробки.

Повідомлення про дефект, створене під час динамічного тестування, зазвичай включає:

– ідентифікатор дефекту;

– заголовок і короткий опис знайденого дефекту;

– дату повідомлення про дефект, інформацію про автора повідомлення;

– ідентифікацію елемента тестування (перевіряється елемента конфігурації) і середовища;

– фазу життєвого циклу розробки, в якій виявлено дефект;

– опис дефекту, достатній для його відтворення і прийняття рішення, включаючи системні журнали, скріншоти, дампи бази даних або записи (якщо вони створені під час виконання тесту);

– очікувані та фактичні результати;

– область або ступінь впливу дефекту на інтереси зацікавленої сторони (критичність);

– терміновість або пріоритет для виправлення;

– статус дефекту (наприклад, відкритий, Відкладений, дублікат, очікує виправлення, очікує перевірки, повторно відкритий, закритий);

– висновки, рекомендації та узгодження;

– глобальні проблеми, наприклад, області, які будуть порушені виправленням дефекту;

– історія змін, наприклад, послідовність дій членів команди проекту, щоб ізолювати дефект, виправити і підтвердити виправлення дефекту;

– посилання, включаючи посилання на тестовий сценарій, який виявив дефект.

Звіт про дефект (defect report) - документ, що описує і пріорітізірующій виявлений дефект, а також сприяє його усуненню.

Як випливає з самого визначення, звіт про дефект пишеться з наступними основними цілями:

– надати інформацію про проблему - повідомити проектну команду та інших зацікавлених осіб про наявність проблеми, описати суть проблеми;

– пріоритизувати проблему - визначити ступінь небезпеки проблеми для проекту і бажані терміни її усунення;

– сприяти усуненню проблеми-якісний звіт про дефект не тільки надає всі необхідні подробиці для розуміння суті того, що сталося, але також може містити аналіз причин виникнення проблеми і рекомендації щодо виправлення ситуації.

На останній мети слід зупинитися докладніше. Є думка, що добре написаний звіт про дефект – половина розв'язання проблеми для програміста. І дійсно, як ми побачимо далі (і особливо в розділі «типові помилки при написанні звітів про дефекти), від повноти, коректності, акуратності, подробиці і логічності звіту про дефект залежить дуже багато-одна і та ж проблема може бути описана так, що програмісту залишиться буквально виправити пару рядків коду, а може бути описана і так, що сам автор звіту на наступний день не зможе зрозуміти, що ж він мав на увазі.

Звіт про дефект (і сам дефект разом з ним) проходить певні стадії життєвого циклу, які схематично можна показати так (рисунок 1.1):

– виявлено – (submitted) - початковий стан звіту (іноді називається «Новий» (new)), в якому він знаходиться відразу після створення. Деякі засоби також дозволяють спочатку створювати чернетку (draft) і лише потім публікувати звіт;

– призначений – (assigned)-в цей стан звіт переходить з моменту, коли хтось із проектної команди призначається відповідальним за виправлення дефекта. Призначення відповідального проводиться або рішенням лідера ко-Манди розробки, або колегіально, або за добровільним принципом, або іншим прийнятим в команді способом або виконується автоматично на основі певних правил.

– виправлений (fixed) – в цей стан звіт переводить відповідальний за ІС-правління дефекту член команди після виконання відповідних дій по виправленню.

– перевірений (verified) – в цей стан звіт переводить Тестувальник, удостовірився, що дефект насправді був усунений. Як правило, таку перевірку виконує Тестувальник, спочатку написав звіт про дефект.

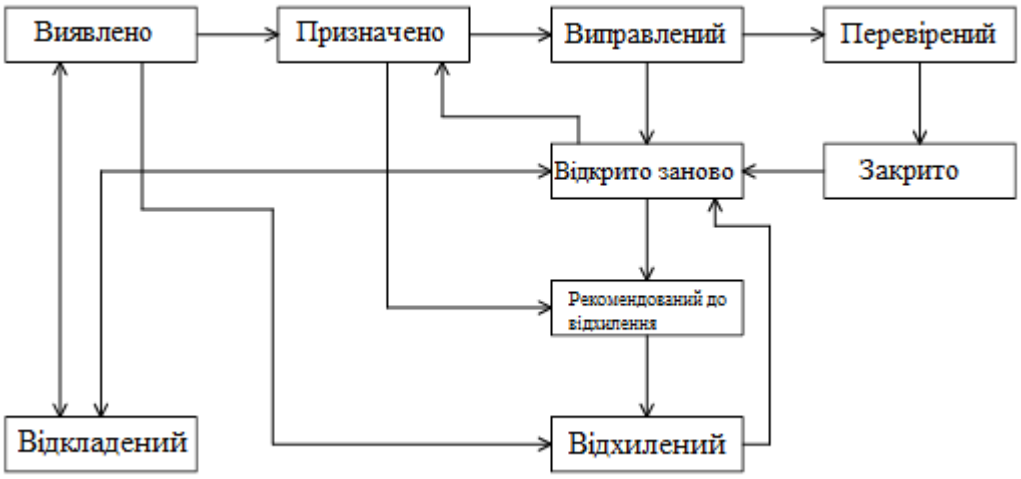

Рисунок 1.1 - життєвий цикл звіту про дефект з найбільш типовими переходами між станами

Критерії серйозності дефектів ранжуються наступним чином:

– критична (critical) – існування дефекту призводить до масштабних наслідків катастрофічного характеру, наприклад: втрата даних, розкриття конфіденційної інформації, порушення ключової функціональності додатка і тощо;

– висока (major) – існування дефекту приносить відчутні незручності багатьом користувачам в рамках їх типової діяльності, наприклад: недо-ступность вставки з буфера обміну, непрацездатність загальноприйнятих клавіатурних комбінацій, необхідність перезапуску програми при виконанні типових сценаріїв роботи;

– середня (medium) – існування дефекту слабо впливає на типові схеми роботи користувачів, і/або існує обхідний шлях досягнення мети, наприклад: діалогове вікно не закривається автоматично після натискання кнопок «OK» або «Cancel», при роздруківці декількох документів поспіль не зберігається значення поля «Двосторонній друк», переплутані спрямування сортувань по якомусь полю таблиці;

– низька (minor) – існування дефекту рідко виявляється незначним відсотком користувачів і (майже) не впливає на їх роботу, наприклад: помилка в глибоко вкладеному пункті меню налаштувань, якесь вікно відразу при відображенні розташоване незручно (потрібно перетягнути його в зручне місце), неточно відображається час до завершення операції копіювання файлів.

Критерії пріоритету дефектів ранжуються наступним чином:

– найвища (ASAP, as soon as possible) терміновість вказує на необхідність усунути дефект настільки швидко, наскільки це можливо. Залежно від контексту "настільки швидко, наскільки можливо «може варіюватися від» в найближчому білді" до одиниць хвилин;

– висока (high) терміновість означає, що дефект слід виправити позачергово, тому що його існування або вже об'єктивно заважає роботі, або почне створювати такі перешкоди в самому найближчому майбутньому;

– звичайна (normal) терміновість означає, що дефект слід виправити в порядку загальної черговості. Таке значення терміновості отримує більшість дефектів;

– низька (low) терміновість означає, що в доступному для огляду майбутньому виправлення даного дефекту не зробить істотного впливу на підвищення якості продукту.

Один з найбільш частих питань, який між ними зв'язок. Ніякий. Для кращого розуміння цього факту можна порівняти важливість і терміновість з координатами X і Y точки на площині. Хоч на побутовому рівні і здається, що дефект з високою важливістю слід виправити в першу чергу, в реальності ситуація може виглядати зовсім інакше.

Щоб проілюструвати цю думку докладніше, повернемося до переліку градацій: чи помітили ви, що для різних ступенів важливості приклади наведені, а для різних ступенів терміновості – ні? І це не випадково.

Пояснимо на життєвому прикладі: наскільки для життя людини важлива вода? Дуже важлива, без води людина вмирає. Значить, важливість води для людини можна оцінити як критичну. Але чи можемо ми відповісти на питання «Як швидко людині потрібно випити води?», не знаючи, про яку ситуацію йдеться? Якщо розглянута людина вмирає від спраги в пустелі, терміновість буде найвищою. Якщо він просто сидить в офісі і думає, чи не попити чи чаю, терміновість буде звичайною або навіть низькою.

Повернемося до прикладів з розробки програмного забезпечення і покажемо чотири випадки поєднання важливості і терміновості в таблиці 1.1.

Таблиця 1.1 – Приклади поєднання важливості і терміновості дефектів

Терміновість	Важливість	
	Критична	Низька
Найвища	Проблеми з безпекою під введеному в експлуатацію	На корпоративному сайті ушкодилася картинка з фірмовим логотипом.
Низька	На самому початку розробки проекту виявлена ситуація, при якій можуть бути пошкоджені або зовсім втрачені для користувача дані.	У керівництві користувача виявлено кілька помилок, які не впливають на зміст тексту.

Знаючи суть проекту і суть дефекту, його важливість визначити досить легко, тому що ми можемо простежити вплив дефекту на критерії якості, ступінь виконання вимог тієї чи іншої важливості та тощо, але терміновість виправлення дефекту можна визначити тільки в конкретній ситуації.

1.2 Проблема присутності помилок в програмних проектах

Тестуванням називається процес, який гарантує справну роботу пристроїв програми і показує відсутність помилок в програмному продукті. Можна помітити, що дане визначення не зовсім коректно і навіть неправильно. Людина з деяким досвідом прикладного програмування знає, що повна відсутність помилок в програмі виявити і показати неможливо. Більш правильним буде визначити процес тестування і налагодження як – завершальний етап створення програмного продукту, який полягає у виконанні програми з метою виявлення збоїв і помилок програмного коду. Замість того, щоб гарантувати відсутність помилок в новій програмі, розумніше буде хоча б продемонструвати їх наявність. Якщо додаток

коректно працює при виконанні безлічі різних тестів, це надає деяку впевненість, але ще не гарантує відсутність в ній помилок. Це лише показує, що нам поки невідомо, в яких випадках програма може дати збій. Виходить «парадокс тестування». В його основі лежать два протилежних твердження: з одного боку, тестування дозволяє переконатися, що продукт працює добре; а з іншого - виявляє помилки в ПЗ, показуючи, що продукт не працює. Друга мета тестування є більш продуктивною з точки зору поліпшення якості, через те, що не дозволяє ігнорувати недоліки ПО.

Дефект – розбіжність очікуваного і фактичного результату.

Очікуваний результат – поведінка системи, описане у вимогах.

Фактичний результат – поведінка системи, що спостерігається в процесі тестування.

У силлабусі ISTQB написано[2], що людина робить помилки, які призводять до виникнення дефектів в коді, які, в свою чергу, призводять до збоїв і відмов додатки (однак збої і відмови можуть виникати і через зовнішніх умов, таких як електромагнітний вплив на обладнання і тощо.)

На рисунку 1.2 зображена спрощена схема взаємозв'язку проблем в розробці

.

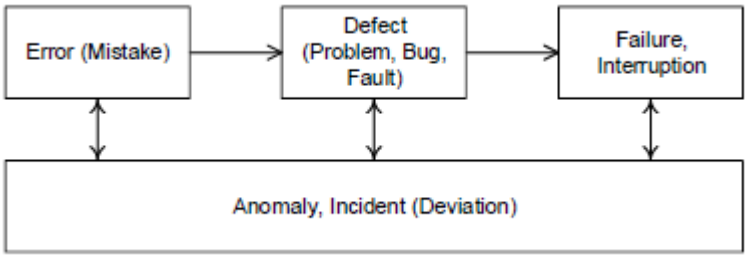

Рисунок 1.2. – Взаємозв'язок проблем в розробці програмних продуктів

Розглянемо основні терміни тестування.

Помилка (error, mistake) – дія людини, що приводить до некоректного результата.

Термін «дефект» дуже часто використовують як найбільш універсальний, що описує будь-які проблеми («помилка людини», «помилка в коді», «помилка в

документації», «помилка виконання операції», «помилка передачі даних», «помилковий результат» і тощо) більш того, куди частіше ви зможете почути «звіт про помилку», ніж «звіт про дефект». І це нормально, так склалося історично, до того ж термін «помилка» насправді дуже широкий.

Цей термін також розуміють досить широко, кажучи про дефекти в документації, Налаштуваннях, вхідних даних і тощо. Цей термін якраз стоїть посередині – безглуздо писати звіти про людські помилки, так само як і майже марно просто описувати прояви збоїв і відмов – потрібно докопатися до їх причини, і першим кроком в цьому напрямку є саме опис дефекту.

Збій (interruption) або відмова (failure) – відхилення поведінки системи від очікуваного.

Аномалія (anomaly) або інцидент (incident, deviation) — будь-яке відхилення спостережуваного (фактичного) стану, поведінки, значення, результату, властивості від очікувань спостерігача, сформованих на основі вимог, специфікацій, іншої документації або досвіду і здорового глузду.

Помилки, дефекти, збої, відмови і тощо. є проявом аномалій — відхилень фактичного результату від очікуваного. Варто відзначити, що очікуваний результат дійсно може ґрунтуватися на досвіді і здоровому глузді, так як. поведінка програмного засобу ніколи не доходить до рівня базових елементарних прийомів роботи з комп'ютером.

Звідси логічно випливає, що дефекти можуть зустрічатися не тільки в коді програми, але і будь – якій документації, в архітектурі і дизайні, в настройках тестованого додатки або тестового оточення-де завгодно.

1.3 Загальна характеристика баг-трекінгових систем

Зміна є постійною рисою розробки програмного забезпечення. Всі проекти мають своєю головною метою щось змінити. Система або її частини

модернізуються, виправляються або замінюються, імовірно, для забезпечення більшої або кращої функціональності, простоти використання, скорочення експлуатаційних витрат і тощо. Однак, зміна не завжди сприятливо і має контролюватися при його впровадженні в проект. При неправильній обробці змін можуть вплинути на графік розробки, вплинути на якість і навіть вбити проект. У міру наближення проекту до його завершення наслідки змін стають більш серйозними. Зрозуміло, що для контролю над змінами потрібен механізм.

Інструментальних засобів управління звітами про дефекти (bug tracking system, defect management tool) дуже багато, до того ж багато компаній розробляють свої внутрішні засоби вирішення цього завдання [3]. Найчастіше такі інструментальні засоби є частинами інструментальних засобів управління тестуванням.

Системи служать центральним сховищем для спостереження за ходом звітів про помилки, запитуючи додаткову інформацію у творця звіту про помилки та обговорюючи можливі рішення для виправлення помилки. Розробники використовують інформацію, надану у звітах про помилки, для виявлення причини дефекту та звуження правдоподібних файлів, які потребують виправлення.

Щоб забезпечити те ж саме, на ринку QA протягом багатьох років з'являлися різні системи відстеження помилок або інструменти управління дефектами.

Як правило, всі інструменти, що належать до певного «жанру», складаються з певних загальних та схожих функцій, на які ми можемо розраховувати.

Для програмного забезпечення для відстеження помилок важливо мати:

– засіб звітності-містить поля, які дозволять вам надати інформацію про помилку, середовище, модуль, серйозність, знімки екрана тощо;

– призначення-що хорошого в помилку, коли все, що ви можете зробити, це знайти її і зберегти при собі;

– прогресування на всіх етапах життєвого циклу-робочий процес;

– історія, робочий журнал, коментарі;

– звіти-графіки або діаграми;

– зберігання і витяг - кожна сутність в процесі тестування повинна бути унікально ідентифікованої, це ж правило може бути застосовано і до помилок;

– таким чином, інструмент відстеження помилок повинен забезпечувати спосіб мати ідентифікатор, який можна використовувати для зберігання, пошуку (пошуку) та організації інформації про помилки.

Згадані вище особливості сутності, що означає, що вони абсолютно необхідні для будь-якої системи, яка претендує на те, щоб бути системою відстеження помилок. Крім цього, можуть бути додаткові зручні функції, такі як перегляд, збереження результатів пошуку і тощо, а також деякі гарантії, такі як голосування, відображення інформації про помилки в прямому ефірі і тощо.

Хоча «зручність і впевненість» в тому, що вони корисні, це саме ті особливості, які змінюють гру під час оцінки і роблять вибір щодо того, який інструмент використовувати. Потім, є економіка, щоб розглянути теж.

Ми знаємо, що доступних на ринку інструментів незліченно - деякі з них ідеально підходять для вас, а інші просто не підійдуть. Далі в цій роботі буде приділена деяким з доступних інструментів відстеження помилок і описані за певними критеріями.

Перевага інструменту відстеження дефектів полягає не тільки в ефективному управлінні, але чи знаєте ви, що інструменти відстеження дефектів можуть допомогти вам стати кращим тестером?

У цій частині розглянемо, як це зробити.

По-перше, навіщо використовувати інструмент відстеження дефектів?

За відсутності інструменту відстеження помилок команди використовують електронні таблиці для складання звітів, відстеження та транспортування своїх помилок. Хоча це може бути хорошим тимчасовим рішенням для невеликих груп і проектів, цей метод не є стійким.

Електронні таблиці (наприклад, Excel) створюють масу проблем, коли ви використовуєте їх в якості основного методу відстеження та управління дефектами.

Деякі з них перераховані нижче:

– занадто багато громіздких листів: це дзвонить у дзвін? Листи Excel з вкладеними скріншотами іноді перевищують кілька МБ. Я часто прикріплював ці електронні листи до електронної пошти, сидячи в моїй поштовій скриньці, чекаючи відправлення або отримуючи повне поштове сповіщення, як тільки я їх отримав;

– відсутність в реальному часі інформації про виявлення помилок і їх розвиток або стан: ми не чуємо про проблему, як тільки вона виявлена. Ми також не знаємо, чи була проблема повторно перевірена або повернута і тощо. в режимі реального часу;

– оскільки немає автоматичної системи оповіщення, дефекти не вимагають до себе ніякої уваги, якщо хтось не дивиться на них навмисно;

– проблеми з призначенням роботи: ми не знаємо, у кого є які проблеми і що вони роблять. Якщо він був обраний для дозволу, то який пріоритет встановлений і тощо., ніколи не буде так легко помітно, як хотілося б;

– можливо, вам доведеться зателефонувати або написати через електронну пошту або відправити миттєве повідомлення, щоб дізнатися, що відбувається;

– відсутність центрального сховища: занадто багато папок, випусків, модулів або щось інше.

Якщо ви хочете повернутися до дефекту, про який повідомлялося в попередньому випуску або, можливо, в декількох випусках, який був певним чином прокоментований розробником ви просто граєте в здогад про те, де може бути дефект.

Інструменти управління дефектами або відстеження помилок пропонують єдину точку правди для всіх дефектів, надають оновлення в режимі реального часу, допомагають співпраці з членами команди, відстежують дефекти відповідно до вимог і генерують звіти в режимі реального часу .

Гарантовано, що дефекти у звіті будуть більш значними, допустимими і простішими для розуміння і матимуть більш високий показник «вибрано для вирішення».

Ми не говоримо про щільність дефектів або дефектів на вимогу і тощо. ми говоримо про більш глибоке розуміння тестованої системи.

Припустимо, ви новачок в тестуванні програмного забезпечення. Коли ви перебуваєте в процесі розуміння системи, перевірте свій інструмент відстеження помилок на предмет раніше виявлених дефектів.

Зверніть увагу на деякі з наступних моментів:

– чи є компонент або модуль або функціональна область програми, в якій записано більше помилок, ніж в інших?;

– чи були проблеми з платформою або сумісністю раніше?;

– чи дозволено командам тестування робити пропозиції щодо поліпшення?;

– чи були якісь проблеми, пов'язані з навколишнім середовищем, і чи розглядаються вони як типові дефекти цією командою?;

– що за дефект обернувся? Скільки часу пройшло між звітом про дефекти та виправленням або закриттям?;

– який середній вік дефектів?.

Тепер кожна компанія, кожен проект, кожна команда і кожна людина різні. Таким чином, незважаючи на те, що існує кілька загальних рекомендацій про те , як складати звіти про дефекти, ніщо не готує вас так, як це роблять ваші власні дослідження.

Перевірте свій інструмент відстеження дефектів на наступне:

– які звіти про дефекти повертаються як «недостатньо інформації»?;

– які дефекти були повністю відхилені розробниками як «не дефект »або «працює як задумано». і чому?;

– які пропозиції щодо поліпшення були розглянуті?;

– які дефекти ще відкриті?;

– звіти з скріншотами мали більш високий рівень виправлення?;

– для дефекту, якщо розробники змінили серйозність, перевірте чому? це може дати вам зрозуміти, що» серйозно " для команди, а що ні.;

Ключова частина управління конфігурацією – це управління вимогами. Управління вимогами передбачає встановлення та підтримку згоди між замовником та розробником щодо технічних та нетехнічних вимог. Ця угода є основою для оцінки, планування, виконання та відстеження проектної діяльності протягом усього проекту, а також для підтримки та вдосконалення розробленого програмного забезпечення.

Основні заходи включають:

– планування фази вимог;

– встановлення процесу вимог;

– контроль змін вимог;

– мінімізація додавання нових вимог (повзучість);

– відстеження прогресу;

– вирішення проблем із замовниками та розробниками;

– проведення перевірок вимог.

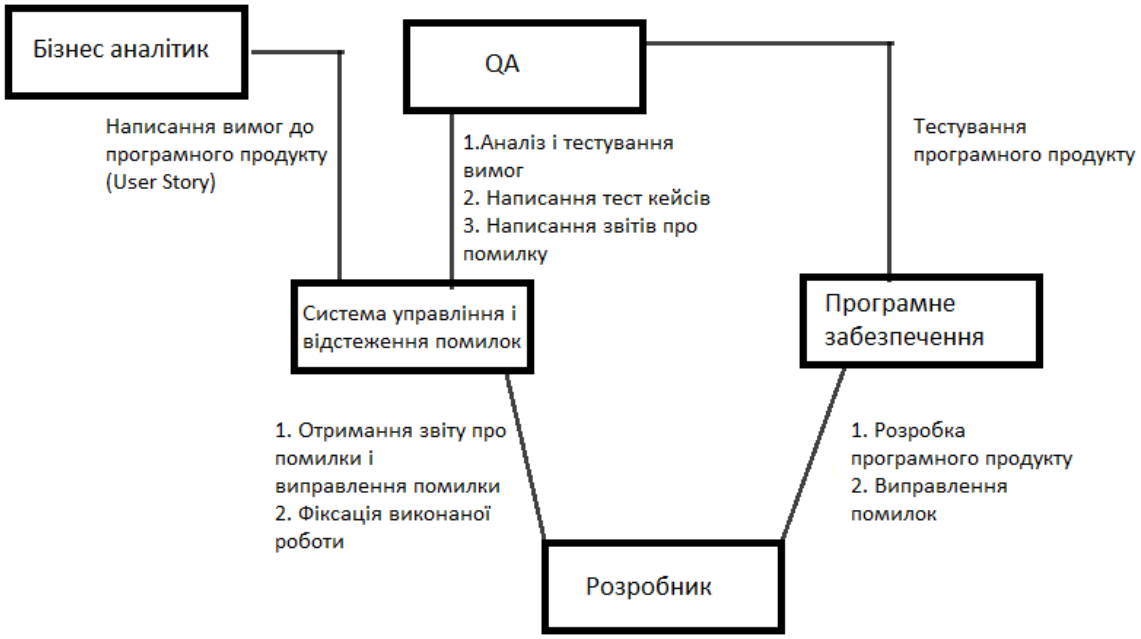

Рисунок 1.3 – Робота системи управління проектами і відстеження помилок

Іншими словами, хороший інструментальний засіб управління життєвим циклом звітів про дефекти не тільки позбавляє людину від необхідності уважно виконувати велику кількість рутинних операцій, але і надає додаткові можливості, що полегшують роботу тестувальника і роблять її більш ефективною.

1.4 Постановка завдання дослідження

В рамках написання атестаційної роботи буде проведено дослідження щодо вивчення методів управління багтрекінговими системами. Мета дослідження – визначення ключових критеріїв їх використання, які дозволять оптимізувати їх для прискорення процесу розробки програмного забезпечення, підвищення його якості та покращення рівня взаємодії між учасниками проекту. Для досягнення цієї мети в роботі планується:

- проаналізувати аналоги;
- визначити критерії;
- провести системний аналіз;
- розробити методологію застосування систем.

На основі аналізу предметної галузі основною задачею є модифікація методів використання баг-трекінгових систем на підставі аналізу даних та критеріїв їх ранжування, також на підставі отриманих результатів, будуть описані рекомендації щодо поліпшення систем, на основі отриманої інформації команді розробників буде легше робити вибір яку систему управління проектами та відстеження помилок вибрати для проекту.

2. ПОРІВНЯЛЬНИЙ АНАЛІЗ СИСТЕМ ВІДСТЕЖЕННЯ ПОМИЛОК

2.1 Критерії порівняння баг-трекінгових систем

Щоб вирішити всі вимоги, про які згадувалося, люди створили різні системи відстеження випусків. У цьому розділі будуть описані найпопулярніші з них. є багато інших систем стеження.

Оскільки всі наведені нижче системи мають свої особливості, і їх треба певним чином порівнювати, були обрані конкретні критерії за якими буде проводитися порівняння:

– установка та початкова конфігурація – коротке введення в вимоги до системного середовища та пояснення того, наскільки складно налаштувати систему відстеження для початкового використання;

– користувацький інтерфейс – деякі користувацькі інтерфейси системи більш зручні для користувачів, ніж інші. Показані (якщо можливо скріншоти) - типові випадки використання «створення нового і пошук питання»;

– робочий процес – очікується, що система дозволяє налаштувати робочий процес і додати спеціальні атрибути. Тут розслідування покаже, наскільки складно змінити робочі процеси за замовчуванням, чи можливо додати або змінити спеціальні атрибути та які кроки необхідні для цього;

– управління проектами – можливість обробляти кілька проектів в одному екземплярі системи відстеження;

– аналіз та моніторинг – будуть показані види статистики, графіки та інші ресурси моніторингу, які надаються системою відстеження;

– інтеграція систем управління вихідним кодом – було надано короткий підсумок про можливість підключення різних версій систем управління (SVN, CVS тощо) до системи відстеження;

– доступність та розширюваність – читач побачить можливості розширення базової функціональності системи відстеження за допомогою плагінів та дізнається про її доступність від інших систем;

– спеціальність – ця частина підсумовує додаткову цінність щодо інших систем, як рекламована розробниками системи відстеження, або як показана протягом цього розслідування.

2.2 Особливості застосування системи Trac

Trac – це, мабуть, найвідоміша та найбільш використовувана система відстеження випусків із відкритим кодом (та безкоштовна). Він пропонує дуже інтуїтивно зрозумілий і вишуканий користувальницький інтерфейс, тісний зв'язок із системами управління версіями (переважно Subversion), контролем редагування та вікі для вмісту. Завдяки цим чинникам він широко розповсюджується у спільноті з відкритим кодом та використовується у багатьох невеликих колективах та компаніях. Є багато сайтів (наприклад, Assembla.com), які пропонують хостинг Trac & Subversion безкоштовно [7].

Установка Trac поширюється у вигляді пакету на блискавці. Перед установкою самого Trac користувачеві необхідно вручну підготувати робоче середовище та встановити:

– Python;

– Genshi (мова шаблонів XML для Python);

– SetupTools (колекція вдосконалень для легшого створення та розповсюдження пакетів Python);

– база даних (Sqlite, MySQL або PostgreSQL);

– веб-сервер із підтримкою модуля Python (найчастіше це Apache HTTPD-сервер з модулем mod_python);

– субверсія (або інша підтримувана система управління версіями).

Після цього користувач розпаковує завантажений пакет у вибраний каталог та запускає інсталятор Trac на основі Python.

Після завершення встановлення користувач налаштовує Trac за допомогою утиліти командного рядка під назвою «trac-admin». Усі зміни в конфігурації повинні бути внесені цією утилітою, хоча для виконання найпоширеніших завдань адміністратора є можливість встановити та використовувати плагін «webadmin».

Користувацький інтерфейс Trac є прекрасним прикладом зручності використання та простоти. Ви можете досягти будь-якої цілі за три кліки миші. Кнопок та опцій не надто багато, тому навіть початківці можуть швидко зрозуміти, як ним користуватися.

Вітальна сторінка – це стандартна вікі-сторінка і може використовуватися для проектної документації. Вся система заснована на вікі, тому користувач швидко розуміє, що t досить просто гіперпосилання інформації між базою даних проблем, контролем редагування та вмістом вікі.

Робочий процес – коли створюється нове середовище, у trac.ini конфігурується робочий процес за замовчуванням. Цей робочий процес є основним робочим процесом (див. рис 2.1).

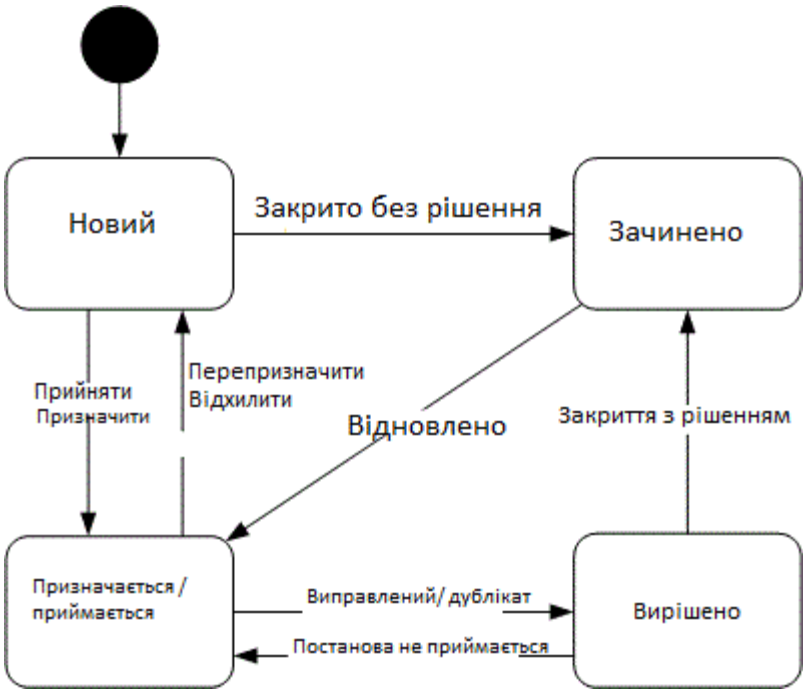

Рисунок 2.1 – Життєвий цикл в Trac

Цю модель робочого процесу можна змінити, замінивши trac.ini з іншими заздалегідь підготовленими заздалегідь файлами моделі робочого процесу або налаштувавшись шляхом прямого редагування цього файлу.

Trac також підтримує додавання спеціальних полів до квитків, редагуючи файл конфігурації trac.ini; однак будь-які квитки, створені до того, як визначено спеціальне поле, не матимуть значення для цього поля, і необхідно вручну оновити базу даних.

Управління проектами Trac не підтримує управління кількома проектами; отже, наявність власного примірника для кожного проекту є єдиним реальним робочим рішенням. Однак таким чином втрачається можливість контролювати всі проекти одночасно, керувати завданнями, спільними для декількох проектів та інших.

Аналіз та моніторинг у базовій установці Trac підтримує лише основні функції моніторингу. Ви можете лише розділити запити між випусками і побачити, скільки запитів вже виконано; однак є додатки, які на основі доданих спеціальних полів підтримують додатковий моніторинг.

Інтеграція систем управління вихідним кодом Trac в основному підтримує системи управління версіями Subversion, Git, Mercurial, Bazaar і Darcs. Для інших систем є додатки, які доступні або їх можна зробити

Доступність Trac сам не підтримує RPC, проте є плагін XML-RPC для Trac версії 0.10 і новішої, але є дві проблеми.

Перша проблема полягає в тому, що цей плагін не входить до базового інсталяційного пакета, і його установка вимагає певного часу.

Друга проблема полягає в тому, що плагін все ще знаходиться в стадії розробки, і його API все ще здається, що він змінюється.

Якщо ви не можете порахувати наявність плагіна та не може залежати від API, важко розробити будь-які віддалені програми Trac для виклику. Це важко, але можливо. Наприклад, плагін Mylyn для Eclipse розв'язувати цю проблему шляхом аналізу HTML-коду інтерфейсу Trac та виклику методів HTTP POST та GET.

Розширюваність – завдяки популярності у спільноті з відкритим кодом існує широкий спектр плагінів та розширень для інтеграції Trac із сторонніми додатками, розширення функціональності або просто спрощення загальних завдань.

Таким чином, Trac не намагається бути складним інструментом розробки та управління проектами. Він зосереджений на одному: видачі відстеження та виконує роботу дуже добре. У базовій установці він не забезпечує багато інших функціональних можливостей; однак існує маса розширень, які роблять Trac більш складним інструментом. Зважаючи на все, Trac як і раніше залишається інструментом, призначеним для використання в окремих проектах.

2.3 Особливості застосування системи JIRA

JIRA - система відстеження помилок та проблем та управління проектами, розроблена Atlassian Software Systems. Він добре відомий своєю складністю, але в водночас, зберігаючи чіткий і простий у користуванні інтерфейс. Він також відомий своєю спільнотою, яка робить безліч розширень та плагінів, а також широким спектром навчальних ресурсів та документації. Останньою великою перевагою є його легка інтеграція з корпоративним wiki Confluence та іншими інструментами Atlassian [9].

Код JIRA є відкритим кодом. JIRA – хороший приклад того, що власне програмне забезпечення з відкритим вихідним кодом перевіряється та розширюється спільнотою. Atlassian надає JIRA безкоштовно для проектів з відкритим кодом та організацій, які є некомерційними, неурядовими, неакадемічними, некомерційними, неполітичними та світськими. Для інших організацій Atlassian стягує від 1200 до 4800 доларів США, залежно від версії, зі знижкою 50% на академічну ліцензію. Починаючи з версії 3.13 JIRA, безкоштовна особиста ліцензія також доступна для некомерційного використання. Ця ліцензія

не включає підтримку Atlassian і вона обмежена трьома повноцінними користувачами.

JIRA поставляється у вигляді інсталяційного файлу, який підтримує установку на основі GUI (більш зручну для робочого столу) або текстову установку (більш зручну для віддалених серверів). Ніяких спеціальних препаратів не потрібно; інсталяційний файл включає всі необхідні частини.

Існує також можливість не встановлювати JIRA у власному середовищі, але розмістити його в Atlassian.

Користувацький інтерфейс JIRA – одна з найбільших переваг. Коли спостерігається складність JIRAs, він залишається дуже чистим і легко зрозумілим завдяки користувальницькому інтерфейсу, але його можна легко налаштувати. Кожен користувач може визначити унікальну конфігурацію інтерфейсу користувача, як найкращу для кожної людини.

Деякі завдання за замовчуванням встановлюють занадто багато кроків. Для цієї дилеми існує безліч плагінів, які дозволяють користувачеві виконувати ці завдання легше. Наприклад, за замовчуванням, якщо користувач хоче перенести квиток з одного випуску проекту на інший, дії необхідно виконати всередині квитка. Це не дуже зручно, якщо користувач повинен обробляти 50 квитків. Плагін GreenHopper існує саме з цієї причини. Не сама JIRA, але ці розширення та налаштування роблять JIRA першокласним продуктом.

JIRA забезпечує робочий процес за замовчуванням – загальний набір кроків, які починаються негайно (див. рис. 2.2). Налаштувати робочий процес JIRA в інтерфейсі легко. Гнучка архітектура робочого процесу JIRA дозволяє користувачеві визначати індивідуальні робочі процеси для відділів, проектів і навіть типів видань. Кожен робочий процес може мати стільки або кілька кроків, скільки потрібно.

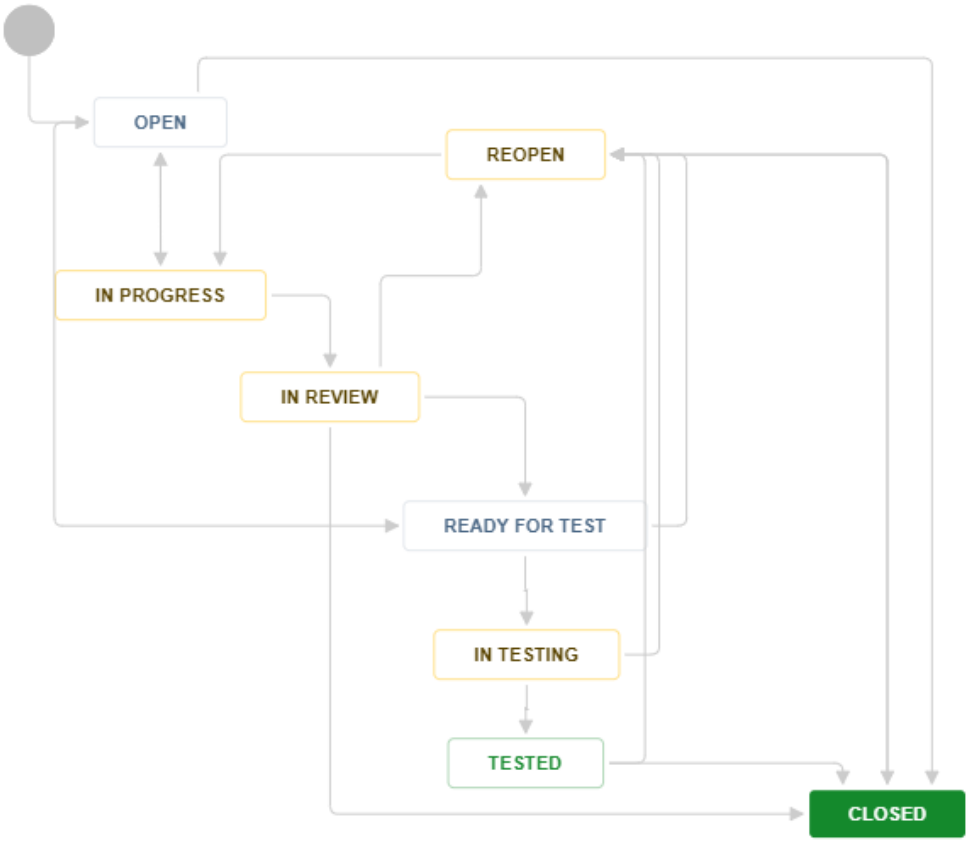

Рисунок 2.2 – Життєвий цикл в Jira

JIRA дозволяє користувачеві налаштувати спеціальні поля квитків на загальносистемній основі, що стосуються проекту та або типу випуску. JIRA поставляється з більш ніж 20 типами користувацьких полів і користувач також може розробляти окремі типи користувацьких полів в інтерфейсі користувача.

JIRA також дозволяє створювати власні типи квитків та керувати ними. Користувач може дійсно налаштувати JIRA для будь-якого середовища, не тільки для розробки програмного забезпечення, але й, наприклад, для «управління життям».

JIRA розроблена як система багатопроєктного відстеження запитів та управління проектами. Проекти можна сортувати за категоріями тощо, як показано на зображенні нижче. Кожен проект може мати індивідуальні налаштування (робочий процес, фільтри, метрики, звіти, типи видань тощо).

На рисунку 2.3 представлена концепція моделі Jira.

Рисунок 2.3 – Концепція моделі Jira

JIRA підтримує моніторинг та звітування на основі фільтрів.

Фільтр – це щось на кшталт налаштованого пошуку. Користувач визначає, на яких умовах є квитки, щоб їх виконати, і називає це. Також можна встановлювати фільтри для всіх проектів, присутніх у JIRA, не лише для кожного проекту окремо.

Потім фільтр може використовуватися не тільки для управління квитками, але і як джерело інструментів для моніторингу та звітності тощо. Користувач може, наприклад, декількома клацаннями миші створити фільтр, який відображає запити, «призначені йому для поточного випуску». Встановіть, щоб це відображалося як кругова діаграма з сортуванням на основі статусу запиту. Крім того, ця діаграма може відображатися як портлет на приладовій панелі.

Приладова панель зазвичай є початковою точкою. Це дуже налаштовується, розміщуючи на ньому так звані портлети. Вони є підключеними компонентами інтерфейсу користувача. Портлети не є класичними портлетами сервера порталів на основі специфікації Java Portlet - вони призначені лише для JIRA. Існує безліч нестандартних портлетів для моніторингу та звітності; однак, як було зазначено, користувачі можуть визначити свої власні або налаштувати нестандартні портлети на основі фільтрів.

JIRA підтримує підключення до різноманітних систем управління вихідним кодом через плагіни. Звичайно, найпоширеніші з них, наприклад, роз'єм Subversion, є нестандартними.

Atlassian також пропонує продукт FishEye, який допомагає командам розробників вести вкладки про те, що відбувається у сховищі вихідного коду у веб-інтерфейсі. Цю можливість також повністю інтегрувати в JIRA.

JIRA 3.0 і вище постачається з плагіном RPC, який забезпечує віддалений доступ через XML-RPC та SOAP. Також можливо використовувати інтерфейс Java API та розширити його на JIRA.

Існує досить велика спільнота, яка працює над розширеннями плагінів та розширень JIRA. Їх роботу можна знайти на веб-сторінці JIRA.

Особливості плагінів для JIRA:

– можливість індивідуального налаштування інтерфейсу користувача для кожного користувача, навіть анонімного;

– це дозволяє використовувати JIRA навіть для служби підтримки та обслуговування клієнтів;

– забезпечує дрібнозернисту безпеку на рівні підприємства;

– можуть бути встановлені права доступу не тільки для окремих проектів, але і для запитів, навіть для одиночних коментарів до квитків;

– запити можна створювати автоматично з електронної пошти (знову корисно для довідкової служби тощо);

– запити можна розділити на підзадачі;

– підтримка за замовчуванням для відстеження часу (реєстрація роботи, претензії, оцінки часу);

– інтеграція з іншими інструментами Atlassian, такими як вікі підприємства Confluenc.

JIRA – найкращий продукт у відстеженні запитів та моніторингу проектів. Вона має дуже доброзичливу політику розповсюдження та завдяки підтримці спільноти з відкритим кодом; це робить додатки та розширення, щоб зробити JIRA ще кращою. Це легко засвоїти і легко настроюється.

Особливості застосування плагіна для гнучких команд в Jira.

Незважаючи на те, що JIRA дуже добре моделює інформацію про проект та має чудові функції звітування, для керівників проектів це не дуже привабливе середовище. Інтерфейс орієнтований на пошук та перелік питань, але редагування та управління проектом за замовчуванням є досить недоброзичливим. Тут представлений плагін GreenHopper. Основні цілі цього плагіна – надати користувачам JIRA наступне:

– інтерактивний та простий інтерфейс для управління своїми проектами;

– інструменти для підвищення видимості та відстеження поточних версій.

Він був розроблений, щоб зробити інтерфейс більш корисним як для розробників, так і для керівників проектів. Використовуючи метафору білої дошки та покажчика, вона дає миттєвий впізнаваний і дуже редагований передній кінець. Насправді він дає лише три додаткові точки зору:

– planning Board - для глобального перегляду всіх версій та всіх компонентів;

– дошка завдань;

– дошка діаграм – діаграми, які допоможуть менеджеру проаналізувати хід проекту.

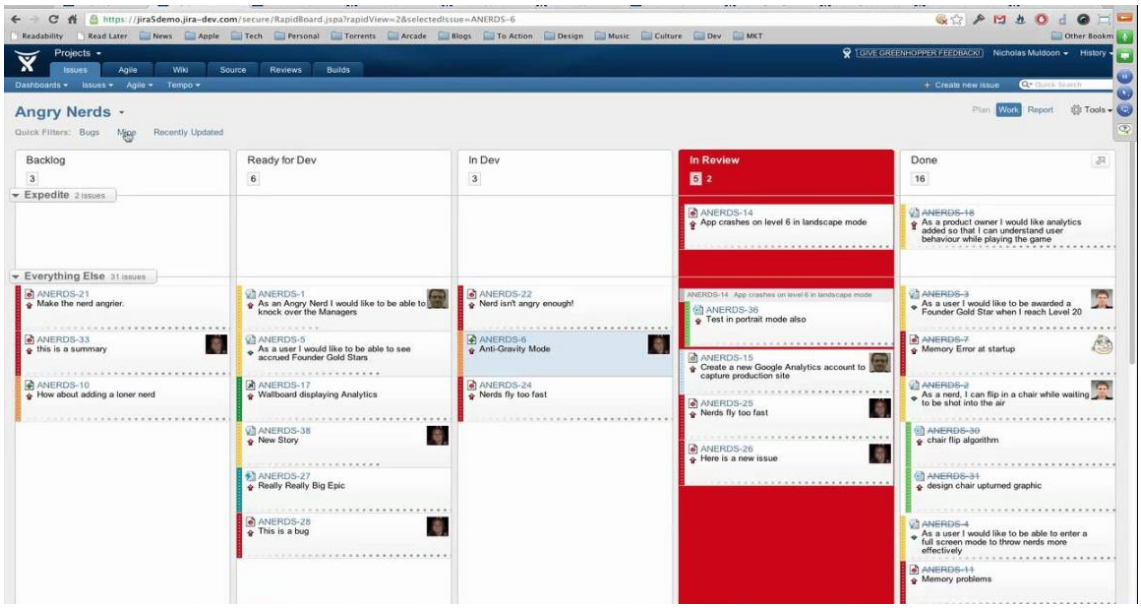

Рисунок 2.4 – Planning Board

Planning Board дає користувачеві загальний огляд усіх версій та всіх компонентів. Він розділений на дві частини.

Основна панель називається «Випуск навігатора». Користувач бачить список проблем у вибраному вмісті і, використовуючи метод перетягування, він може легко замовити та спланувати.

Права панель містить вікна, що можна випускати, і пропонує вибір різних видів подання. Існує три способи управління проблемами:

– огляд (усі невипущені версії виправлень та позапланові версії об'єднані в один перегляд) - це дуже корисно для ранжирування;

– версії - за невипущеними версіями виправлень;

– компоненти – за компонентами.

Тобто, управління проблемами між версіями та компонентами насправді стало цікавим завданням JIRA з GreenHopper. За допомогою простого перетягування та опускання проблеми можна пов’язати із відповідною версією або компонентом.

Кожна версія або компонентне поле також містить таку інформацію та дозволяє виконувати наступні операції:

«Рядок ходу», який показує хід роботи вибраної версії або компонента. Клацання на кольоровому стані приведе навігатор проблем, попередньо заповнений проблемами в цьому стані.

Атрибут «Основний» у кожній з версій або компонентів. За допомогою цього налаштування можна організувати версії та компоненти «ієрархічно». Це дуже корисно і допоможе користувачеві мати глобальний вигляд версій та більш детальний вигляд під-версій.

Статистика та підрахунки – дата випуску, кількість випусків за типом, кількість невирішених та вирішених питань тощо.

Кнопка «Випустити версію», яка дозволить користувачеві випускати версії безпосередньо з вікон версій. Якщо деякі проблеми не вирішені, GreenHopper пропонує два варіанти:

– замініть невирішені питання на обрану версію;

– ігноруйте ці проблеми і все одно випустіть.

Отже, підсумовуючи, дошка планування надає користувачеві:

– глобальний вигляд усіх неопублікованих версій;

– глобальний погляд на всі компоненти;

– вигляд картки та списку;

– багаторівневе планування (ієрархічне);

– видавати замовлення за допомогою перетягування;

– планування випуску за допомогою перетягування;

– багаторазовий вибір у плані планування;

– узгоджене вирішення підзадач;

– видавати фільтрацію та сортування;

– швидкий доступ до навігатора випусків.

Дошка завдань дозволяє користувачам легко відстежувати запити у версіях. Екран розділений на кілька стовпців, які відповідають переходу JIRA. Користувачі можуть додавати, видаляти та перейменувати ці стовпці, щоб зробити дошку зручнішою. Кожен користувач може персоналізувати власну дошку завдань, сховавши або показавши конкретні стовпці.

За допомогою простого перетягування питань між плавальними смугами, користувач може легко змінити статус проблеми. Видача картки в стовпчик автоматично відобразить список усіх можливих переходів. Звичайно, всі події сповіщення, постфункції тощо будуть запускатись. Якщо для переходу питань потрібні деякі обов'язкові записи, GreenHopper попросить користувача ввести ці необхідні дані через модальний екран у панелі завдань.

Підсумовуючи це, Task board передбачає:

– інтуїтивне відстеження версій;

– окреслена дошка завдань;

– персональна дошка;

– регульовані смуги плавання;

– оновлення стану випуску за допомогою перетягування;

– швидкий журнал роботи користувача;

– швидкий доступ до навігатора випуску;

– видають фільтрування та сортування.

Chart Board пропонує безліч діаграм і діаграм, щоб допомогти менеджерам проаналізувати хід проекту. Більш детально про доступні діаграми та їх конфігурацію можна ознайомитись у документації, доступній на веб-сайті.

Підсумовуючи, Chart Board передбачає:

– динамічна робота та вимога відсутності налаштувань;

– відображення кривої вигорання;

– відображення кривої зусиль команди;

– відображення кривої точності оцінки;

– діаграма спасання на основі спеціального поля;

– діаграма спалювання на основі спеціального поля;

– діаграма значень на основі спеціального поля;

– видавати фільтрацію; і

– налаштована дата початку та дата закінчення.

2.4 Особливості застосування системи CodeBeamer

CodeBeamer – це не просто проста система відстеження проблем, але це платформа розвитку співпраці з інтегрованим управлінням життєвим циклом додатків. Під цим заголовком ми можемо представити набір таких служб, як управління документами, пов'язаний з проектом, Wiki, Форум, Інтернет-чат і, звичайно, відстеження випусків, інтегрований з контролем версій (SVN).

Він доступний безкоштовно для студентів та оцінювачів; однак для використання в бізнесі це комерційно. Як приклад публічного використання, ми

можемо назвати, наприклад, спільноту JavaForge.com, яка розміщує декілька проектів з відкритим кодом Java та працює над CodeBeamer.

CodeBeamer – програма Java EE, що працює на контейнері сервлетів Apache Tomcat. Він поставляється у вигляді двійкового інсталяційного файлу або для Windows, або для Linux. Файли встановлення Tomcat включені в пакет; однак, немає жодної проблеми встановити CodeBeamer у свій власний екземпляр Tomcat. Закінчивши установку, потрібно встановити підключення до бази даних (та деякої іншої конфігурації) у файлі властивостей.

Здається, інтерфейс користувача має кращу структуру, ніж інтерфейс Bugzilla; однак це все одно не виграє ціну за зручність використання. Як ви бачите нижче у сценарії використання, для додавання нових запитів на випуск потрібно досить глибоко скористатися інтерфейсом. Це допоможе перенести деякі найбільш використовувані функції на деякі додаткові панелі чи інші подібні реорганізації. Під час користування інтерфейсом CodeBeamer інтерфейс нічого не знайшов, лише багато речей передбачали занадто багато кроків.

CodeBeamer забезпечує робочий процес за замовчуванням, який можна налаштувати в інтерфейсі користувача. Посібник можна знайти на сторінці WikiBeamer «Workflows» на CodeBeamer (http://cb.esast.com/cb/wiki/5713).

Діаграма робочого процесу нижче містить такі типи вузлів:

– жовтий вузол – позначає стартовий стан. У цьому статусі з’явиться новий елемент відстеження;

– сині вузли – позначають регулярні стани. Він переміститься серед них протягом більшої частини життєвого циклу елемента трекера;

– помаранчеві вузли – оскільки для них залишаються лише переходи, ви не можете переміщувати елементи до них, але можна переміщувати елементи з них до звичайних елементів. Ці вузли в основному використовуються для правильної міграції зі застарілих трекерів.

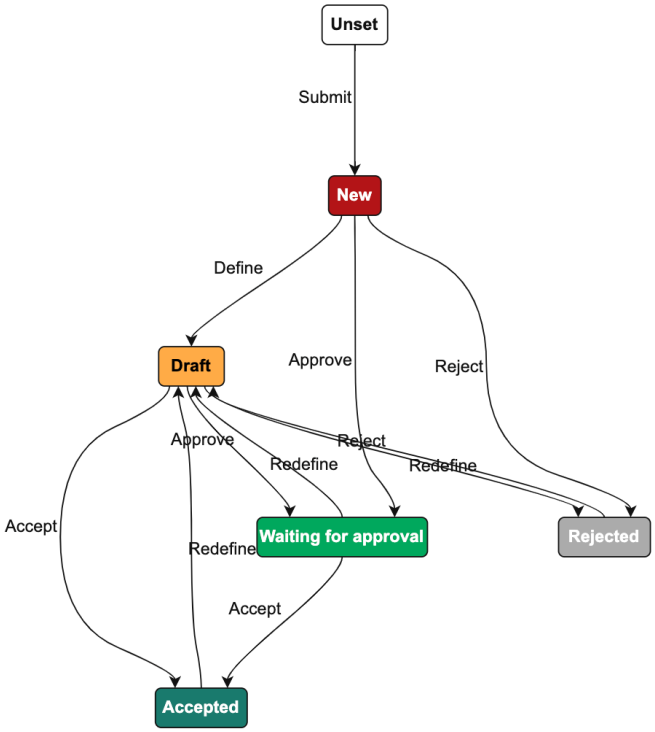

Рисунок 3.2 – Життєвий цикл в Code Beamer

CodeBeamer також підтримує додавання спеціальних полів до квитків в інтерфейсі користувача.

За замовчуванням CodeBeamer підтримує управління кількома проектами. Проекти можна також згрупувати у так звані «Робочі набори», що дозволяють їх легше керувати. Так, наприклад, проекти можна сортувати за робочими наборами, названими на честь замовника, для якого вони є.

Кожен проект може мати власні налаштування, права доступу, робочий процес квитків тощо. Він працює таким чином, що налаштування проекту застосовуються в першу чергу, після цього приймаються налаштування робочого набору (якщо проект знаходиться в якомусь робочому наборі) і, нарешті, налаштування CodeBeamer за замовчуванням.

CodeBeamer має налаштовані системи звітності. За замовчуванням він забезпечує:

– завдання в реальному часі та звітування про помилки з діаграмами;

– діаграма Ганта;

– звіти про якість вихідного коду та зміни коду з тенденціями;

– звіти з користувацькими звітами, що дозволяють підвищити гнучкість та автоматично створювати нотатки до випусків та інші звіти;

– вбудовані звіти CMMI.

Звіти працюють над списком елементів трекера від одного або декількох трекерів, що належать до одного або декількох проектів. Прості звіти містять одну таблицю для кожного трекера. Об’єднані звіти представляють елементи трекера в єдиному табличному макеті.

Після запуску звіту результати можна експортувати у формати Excel, PDF, CSV, XML та Wiki.

CodeBeamer має вбудовані інтерфейси для систем CVS, Subversion, PVCS, SourceSafe, CM Synergy та ClearCase SCM, і їх можна легко розширити для підтримки інших систем.

Доступність та розширюваність CodeBeamer має API веб-служб, заснований на двійковій веб-службі Caucho Hessiann.

Як згадувалося на початку, CodeBeamer – це не просто проста система відстеження проблем, але це платформа розвитку співпраці з інтегрованим управлінням життєвим циклом додатків. Тут вводяться такі особливості:

– вбудована система управління документами;

– внутрішня корпоративна система wiki, яка об'єднує всі компоненти, такі як трекер, менеджер документів, система scm разом, в єдину програму, де всі артефакти можуть бути пов'язані та посилатися на розмітки wiki;

– підтримка періодичних побудов з постійною інтеграцією;

– вбудовані дискусійні форуми;

– вдосконалена платформа безпеки;

– інтеграція електронної пошти з вхідними;

– підтримка для надання статистики та показників якості.

Отже, CodeBeamer – це платформа розвитку співпраці. Він надає різноманітні інструменти для підтримки як управління, так і розвитку, а крім того, також забезпечує його інтеграцію плагінів у найбільш поширені IDE. Він також легко налаштовується та ремонтується. Користувацький інтерфейс розглядається як невеликий мінус, коли найпоширеніші завдання не довершені і залишаються «прихованими»

2.5 Особливості застосування системи Bugzilla

Сама Bugzilla поширюється як gZip-пакет. Перш ніж встановлювати Bugzilla, користувач повинен вручну підготувати робоче середовище - встановити Perl, базу даних (MySQL або PostgreSQL) та веб-сервер з підтримкою модуля Perl (найпоширеніший - сервер Apache HTTPD з модулем mod_perl). Після цього користувач розпаковує завантажений пакет gZip у вибраний каталог та редагує файли конфігурації веб-сервера та файли конфігурації програми [10].

Як легко сказати, інтерфейс Bugzilla є строго функціональним. У цьому немає нічого приємного, він надає безліч функцій в межах невеликого простору, і на початку користувач може відчувати себе досить незручно і загублено; однак, виявивши його, користувач дізнається, що це не дуже складно, і працювати з ним просто.

Bugzilla підтримує досить повну, хоча і жорстко закодовану модель робочого процесу (див. рис 3.6). Хоча цю модель робочого процесу неможливо налаштувати в Bugzilla, вона зазвичай виявляється достатньою для більшості організацій.

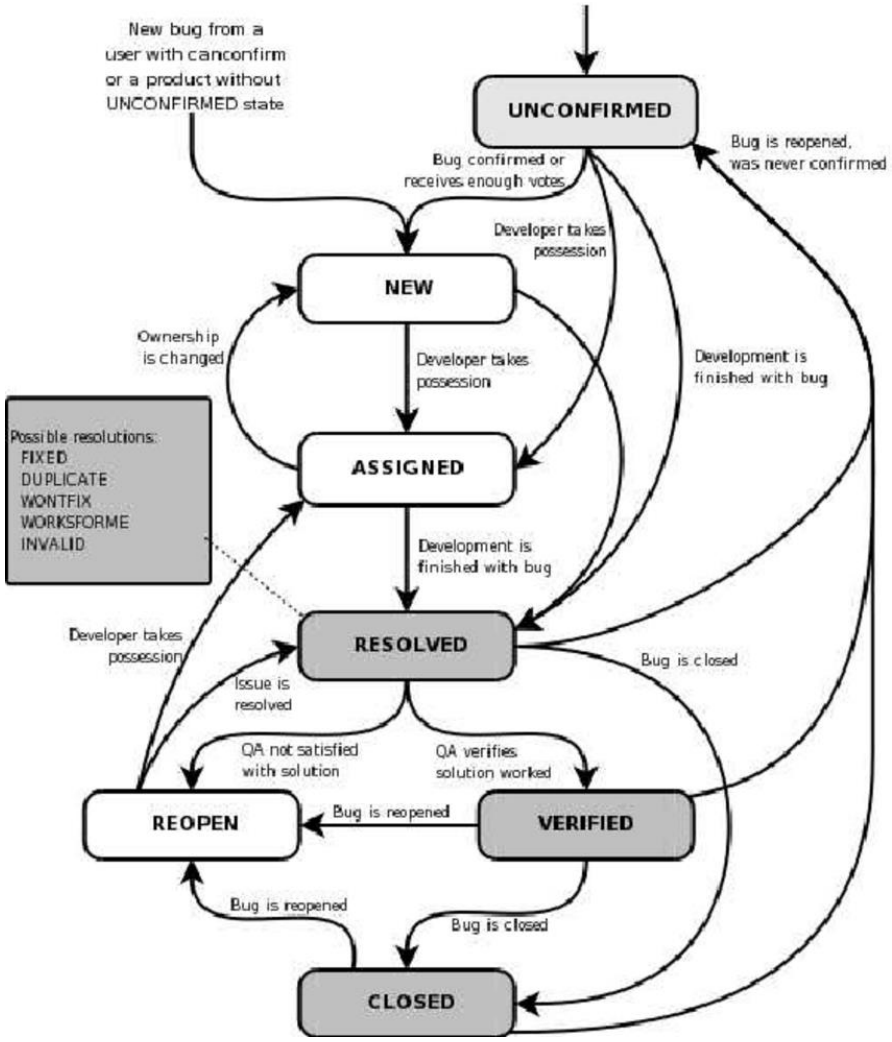

Рисунок 3.6 – Життєвий цикл в Bugzilla

Bugzilla також підтримує додавання спеціальних полів у базу даних про помилки для збору та пошуку даних, унікальних для організації.

Bugzilla дозволяє адміністраторам створювати та змінювати так звані «дерева класифікації» продуктів. Це означає, що немає жодної проблеми створити дерево, де, наприклад, на першому рівні були б назви проектів, а на другому рівні – назви продуктів, що редагуються. Ніяких спеціальних злому не потрібно.

Bugzilla має дуже просунуту систему звітування, яку на перший погляд досить важко почати використовувати – користувальницький інтерфейс не дуже дружній.

Щоб побачити, як зараз виглядає база даних про помилки, можна створити таблицю, використовуючи будь-які два поля як вісь X та Y, та використовуючи

будь-які критерії пошуку для обмеження цих проблем. Наприклад, Продукт може бути представлений віссю X, а Статус - віссю Y, і тоді звіт показує, скільки помилок було в кожному стані, у кожному продукті.

Ця ж таблиця може розглядатися як лінійний графік, гістограма або кругова діаграма. «Вісь Z» також може бути задана для генерації декількох таблиць або графіків. Ці звіти можна експортувати у форматі CSV, щоб їх можна було редагувати в електронній таблиці. Нарешті, щоб побачити, як змінилася установка Bugzilla з часом, Bugzilla також підтримує графічну систему, яка може створювати графіки, які відслідковують зміни в системі з часом.

Інтеграція Bugzilla із системами управління вихідним кодом забезпечується за допомогою плагінів. Офіційно підтримуються CVS, Subversion, Bonsai, Teamtrack Performance та Tinderbox. Ці та інші можна знайти на сторінці ресурсів додатків.

До Buggilla можна отримати доступ та змінити через інтерфейс веб-служб XML-RPC. Це дає можливість легко писати зовнішні інструменти, які взаємодіють з Bugzilla.

Існує також багато плагінів та розширень для Bugzilla. З їх переліком можна ознайомитись за посиланням[14].

UI Bugzilla створюється з шаблонів HTML; тому налаштувати інтерфейс користувача досить легко.

Існує велика кількість локалізацій.

Bugzilla повідомляє користувачів про будь-які нові або оновлені помилки електронною поштою.

Bugzilla підтримує базове відстеження часу.

Bugzilla також підтримує систему голосування, в якій користувачі можуть голосувати за питання або функції, які хочуть бачити реалізованими.

Тобто, Bugzilla – перевірене рішення, яке підтримує великі проекти та бази користувачів. Особливості його робочого процесу більш ніж достатньо для більшості організацій. З іншого боку, Bugzilla особливо складний в установці та обслуговуванні, а користувальницький інтерфейс не виграє жодних призів за

дизайн або зручність. Його функції звітування зроблять роботу, хоча вони не особливо зручні для користувачів.

2.6 Результати порівняння баг-трекінгових систем

В даному розділі були проведені дослідження чотирьох систем відстеження помилок (Trac, JIRA, CodeBeamer, Bugzilla) за такими критеріями:

– установка та початкова конфігурація;

– користувацький інтерфейс;

– робочий процес;

– управління проектами;

– аналіз та моніторинг;

– інтеграція систем управління вихідним кодом;

– доступність та розширюваність;

– спеціальність.

Отже, коротко підвести підсумки отримані під час дослідженнях систем можна так:

– Trac написаний за допомогою Python і є веб-сайтом. Коли ви інтегруєте Trac з системою SCM, ви можете використовувати його для перегляду коду, перегляду змін, перегляду історії і тощо, проблеми та інциденти в Trac називаються «квитками», і система управління квитками також може бути використана для управління дефектами, якщо ви цього хочете;

– Atlassian JIRA, в першу чергу інструмент управління інцидентами, також широко використовується для відстеження помилок. Він надає повний набір записів, звітів, робочих процесів та інших функцій, пов'язаних із зручністю. Це інструмент, який безпосередньо інтегрується з середовищами розробки коду, що робить його ідеальним вибором і для розробників. Крім того, завдяки своїй

здатності відстежувати будь-які і всі види проблем, він не обов'язково зосереджений тільки на індустрії розробки програмного забезпечення і досить ефективно надає себе довідковим службам, системам управління відпустками. JIRA також підтримує гнучкі проекти. Це комерційно ліцензований продукт з багатьма надбудовами, які підтримують розширюваність;

– CodeBeamer – це платформа розвитку співпраці. Він надає різноманітні інструменти для підтримки як управління, так і розвитку, а крім того, також забезпечує його інтеграцію плагінів у найбільш поширені IDE;

– Bugzilla вже досить давно є провідним інструментом відстеження помилок, широко використовуваним багатьма організаціями. Це дуже простий у використанні веб-інтерфейс. Він володіє всіма ознаками сутності, зручності і впевненості. Він є повністю відкритим вихідним кодом і вільний у використанні.

Використання програмного забезпечення для відстеження помилок може допомогти в усуненні помилок при тестуванні і розробці процесів. Маючи можливість надавати вичерпні звіти, документацію, можливості пошуку, відстеження помилок і проблем, програмне забезпечення для відстеження помилок є відмінним інструментом для цих потреб в розробці програмного забезпечення.

Управління запитами є важливою частиною кожного програмного проекту, і необхідно використовувати системи стеження за вадами, що дає нам такі переваги:

– поліпшення якості програмного забезпечення;

– підвищення задоволеності користувачів і клієнтів;

– забезпечення підзвітності запитів;

– поліпшення спілкування в команді, а також з клієнтами;

– підвищення продуктивності команди;

– скорочення витрат.

Детальну інформацію про дослідження можна побачити на таблицях 2.1 та 2.2

Таблиця 2.1 – Підсумкова інформація порівняння TRAC та JIRA

Критерії	Системи відстеження помилок	
	Trac	JIRA
Установка	Необхідність підготовки середовища. Установка скриптами оболонки та пізніше редагування файлів конфігурацій	Дуже легко, запустивши покроковий інсталятор. Не потрібно підготовка навколишнього середовища
Користувацький інтерфейс	Дуже простий, інтуїтивно зрозумілий та простий у навчанні інтерфейс користувача	Базовий інтерфейс інтуїтивно зрозумілий та простий у навчанні користуватися, але деякі завдання не дуже зручні, однак інтерфейс інтерфейсу дуже налаштований, і існує широкий спектр розширень
Робочий цикл питань	Можливість редагування моделі робочого процесу запиту (за допомогою редагування файла конфігурації)	Дуже легко налаштувати в інтерфейсі користувача
Спеціальні поля для запиту	Додавання можливо шляхом редагування файлу конфігурації та оновлення бази даних вручну	Дуже легко налаштувати в інтерфейсі користувача
Багатопроектне середовище	Не підтримується. Один екземпляр Trac	Так
Доступність	Модуль інтерфейсу XML-RPC доступний лише для певних версій	Вбудований інтерфейс XML-RPC та SOAP; Java API доступний
Додаткове значення	Вбудований Wiki Різноманітність плагінів, які підвищують функціональність Trac	Настроюється та настроюється Електронна пошта може бути джерелом видачі квитків Різноманітні плагіни

Таблиця 2.2 – Підсумкова інформація порівняння Bugzilla та CodeBeamer

Критерії	Системи відстеження помилок	
	Bugzilla	CodeBeamer
Установка	Необхідність підготовки середовища, ручне налаштування файлів конфігурацій	Дуже легко, запустивши покроковий інсталятор. Не потрібно підготовка навколишнього середовища
Користувацький інтерфейс	Дуже простий, на початку не дуже зручний у користуванні, але дуже простий і функціональний	Складний інтерфейс користувача. Не такий інтуїтивно зрозумілий на початку, але ефективний
Робочий цикл питань	Модель робочого процесу з жорстким кодом запиту	Легко налаштувати в інтерфейсі користувача
Спеціальні поля для запиту	Додавання можливо в інтерфейсі користувача	Додавання можливо в інтерфейсі користувача
Багатопроектне середовище	Так, за «Класифікацією»	Так
Доступність	Вбудований інтерфейс XML-RPC	Вбудований інтерфейс двійкового веб-сервісу Caucho Hessiann
Додаткове значення	Відстеження часу Система голосів, які просять звернутися першими	Складна платформа розробки із вбудованою Wiki, DMS, дискусійною дошкою тощо. Звіти CMMI

Далі на підставі отриманих результатів проведемо системний аналіз, використовуючи метод парних порівнянь.

3. ПОРІВНЯЛЬНА ХАРАКТЕРИСТИКА БАГ-ТРЕКІНГОВИХ СИСТЕМ

3.1 Модель порівняння баг-трекінгових систем

Дослідимо задачу о виборі системи управління проектами та відстеження помилок. Проблема буде вирішуватися за наступними критеріями:

- критерій 1 (К1): установка та початкова конфігурація;
- критерій 2 (К2): користувацький інтерфейс;
- критерій 3 (К3): робочий процес;
- критерій 4 (К4): управління проектами;
- критерій 5 (К5): аналіз та моніторинг;
- критерій 6 (К6): інтеграція систем управління вихідним кодом;
- критерій 7 (К7): доступність та розширюваність;
- критерій 8 (К8): спеціальність.

Вибирати будемо із множини альтернатив:

- альтернатива 1(А1): Jira;
- альтернатива 2 (А2): Trac;
- альтернатива 3 (А3): CodeBeamer;
- альтернатива 4 (А4): Bugzilla.

Побудуємо ієрархічну структуру, використовуючи метод парних порівнянь.

3.2 Оцінка векторів пріоритетів незадовільності методом аналізу ієрархій

Далі побудуємо матрицю попарних порівнянь для кожного рівня ієрархії, скориставшись відповідною шкалою Т. Сааті [13]. Ця шкала надає можливість поставити у відповідність ступеням переваги одного порівняльного об’єкта над іншим деяке число. На основі цих даних знаходяться вектори локальних

пріоритетів, індекси узгодженості та відношення узгодженості. Останні два допомагають визначити міру узгодженості результатів, отриманих експертним шляхом. Допустимим вважається значення відношення узгодженості менше двох . Більші значення свідчать про нелогічність суджень та спонукають до перегляду даних.

Таблиця 3.1 – Матриця попарних порівнянь критеріїв

Критерії оцінювання	К1	К2	К3	К4	К5	К6	К7	К8	Оцінки компо-нентів	Вектор пріоритетів
К1	1	$\frac{1}{5}$	$\frac{1}{8}$	$\frac{1}{7}$	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{2}$	0.293	0.025
К2	5	1	$\frac{1}{3}$	$\frac{1}{2}$	2	4	2	4	1.643	0.140
К3	8	3	1	2	6	7	6	7	4.130	0.352
К4	7	2	$\frac{1}{2}$	1	5	6	5	6	2.984	0.255
К5	4	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{4}$	1	2	1	2	0.847	0.072
К6	2	$\frac{1}{4}$	$\frac{1}{7}$	$\frac{1}{6}$	$\frac{1}{2}$	1	$\frac{1}{2}$	1	0.483	0.042
К7	4	$\frac{1}{2}$	$\frac{1}{6}$	$\frac{1}{5}$	1	2	1	2	0.847	0.072
К8	2	$\frac{1}{4}$	$\frac{1}{7}$	$\frac{1}{6}$	$\frac{1}{2}$	1	$\frac{1}{2}$	1	0.483	0.042
Усього									11.71	1

Індекс узгодженості (ІУ) = 0.094

Випадкова узгодженість = 1,41.

Відносна узгодженість (ВУ) = $\frac{0,094}{1,41} = 0,067 = 6,7\%$

З отриманих проміжних результатів можна зробити висновок, що матриця попарних порівнянь заповнена правильно, тобто жодне з тверджень не суперечить

загальній логіці моделі, а дані узгоджені між собою на допустимому рівні. Вектор локальних пріоритетів відносно проблеми вибору набуває вигляду:

$$\vec{p}^{\,k} = (0,025; 0,140; 0,352; 0,255; 0,072; 0,042; 0,072; 0,042)\,. \qquad (3.1)$$

Для прийняття рішення о використовуванні метода, необхідно провести порівняльний аналіз альтернатив. Оцінив їх стосовно кожної з альтернатив, отримаємо данні, котрі зображені на таблицях 3.2 – 3.9. Випадкова узгодженість для матриць являється 0,9.

Таблиця 3.2. – Порівняння по установці і початкової конфігурації

Критерій 1	A1	A2	A3	A4	Власний вектор	Вектор пріоритетів
A1	1	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{2}$	2.25	0.111
A2	4	1	2	2	9	0.445
A3	2	$\frac{1}{2}$	1	1	4.5	0.222
A4	2	$\frac{1}{2}$	1	1	4.5	0.222
Усього					20.25	1

Індекс узгодженості(ІУ) = 0.001.

Відносна узгодженість(ВУ) = $\frac{0,001}{0,9} = 0,001 = 0,1\%$

На підставі дослідження «Установка та початкова конфігурація» були розставлені пріоритети альтернатив один до одного. На таблиці 3.2 видно що у JIRA інсталяційний процес вийшов краще по відношенню CodeBeamer і Bugzilla і набагато краще ніж Trac. CodeBeamer і Bugzilla гірше JIRA і не набагато краще Trac

Таблиця 3.3. Порівняння по користувацьким інтерфейсом

Критерій 2	A1	A2	A3	A4	Власний вектор	Вектор пріоритетів
A1	1	$\frac{1}{2}$	3	4	1.565	0.048
A2	2	1	3	5	2.340	0.072
A3	$\frac{1}{3}$	$\frac{1}{4}$	1	2	0.638	0.02
A4	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{2}$	1	0.398	0.012
Усього					32.601	1

Індекс узгодженості = 0.027.

Відносна узгодженість = $\frac{0,027}{0,9} = 0,003 = 0,3\%$

На підставі дослідження «Користувацький інтерфейс» були розставлені пріоритети альтернатив один до одного. На таблиці 3.3 видно що у JIRA інтерфейс вийшов набагато краще по відношенню CodeBeamer і Bugzilla і краще ніж Trac. Trac гірше JIRA і краще CodeBeamer і Bugzilla

Таблиця 3.4. – Порівняння по робочому процесу

Критерій 3	A1	A2	A3	A4	Власний вектор	Вектор пріоритетів
A1	1	$\frac{1}{3}$	1	$\frac{1}{2}$	0.639	0.141
A2	3	1	3	2	2.06	0.455
A3	1	$\frac{1}{3}$	1	$\frac{1}{2}$	0.639	0.141
A4	2	$\frac{1}{2}$	2	1	1.19	0.263
Усього					4.528	1

Індекс узгодженості = 0.003

Відносна узгодженість = $\frac{0,003}{0,9} = 0,003 = 0,3\%$

На підставі дослідження «Робочий процес» були розставлені пріоритети альтернатив один до одного. На таблиці 3.4 видно що у JIRA робочий процес вийшов набагато краще по відношенню CodeBeamer і Trac і краще ніж Bugzilla. Bugzilla гірше JIRA і краще чим CodeBeamer і Trac.

Таблиця 3.5 – Порівняння по управлінням проектами

Критерій 4	A1	A2	A3	A4	Власний вектор	Вектор пріоритетів
A1	1	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{1}{2}$	0.537	0.122
A2	3	1	2	2	1.861	0.424
A3	2	$\frac{1}{2}$	1	1	1	0.227
A4	2	$\frac{1}{2}$	1	1	1	0.227
Усього					4.398	1

Індекс узгодженості = 0.007,

Відносна узгодженість = $\frac{0,007}{0,9} = 0,007 = 0,7\%$

На підставі дослідження «Управління проектами» були розставлені пріоритети альтернатив один до одного. На таблиці 3.5 видно що у JIRA управління проектами вийшов краще по відношенню CodeBeamer і Bugzilla і набагато краще ніж Trac. CodeBeamer і Bugzilla гірше JIRA і не набагато краще Trac

Таблиця 3.6 – Порівняння по аналізу і моніторінгу

Критерій 5	A1	A2	A3	A4	Власний вектор	Вектор пріоритетів
A1	1	$\frac{1}{4}$	$\frac{1}{2}$	$\frac{1}{3}$	0.45	0.095
A2	4	1	3	2	2.213	0.467
A3	2	$\frac{1}{3}$	1	$\frac{1}{2}$	0.76	0.160
A4	3	$\frac{1}{2}$	2	1	1.316	0.278
Усього					4.739	1

Індекс узгодженості = 0.023,

Відносна узгодженість = $\frac{0,023}{0,9} = 0,026 = 2,6\%$

На підставі дослідження «Аналіз та моніторинг» були розставлені пріоритети альтернатив один до одного. На таблиці 3.6 видно що у JIRA аналізу та моніторингу вийшов гораздо краще по відношенню CodeBeamer і Trac і краще ніж Bugzilla.

Bugzilla гірше JIRA і краще чим Trac і CodeBeamer .

Таблиця 3.7 – Порівняння по інтеграції з системами управління кодом

Критерій 6	A1	A2	A3	A4	Власний вектор	Вектор пріоритетів
A1	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0.595	0.142
A2	2	1	1	1	1.189	0.286
A3	2	1	1	1	1.189	0.286
A4	3	$\frac{1}{2}$	2	1	1.316	0.278
Усього					4.739	1

Індекс узгодженості = 0.001,

Відносна узгодженість = $\frac{0,001}{0,9} = 0,001 = 0,1\%$

На підставі дослідження «Інтеграція систем управління вихідним кодом» були розставлені пріоритети альтернатив один до одного. На таблиці 3.7 видно що JIRA, CodeBeamer та Bugzilla однаковий власний вектор в роботі с вихідним кодом, та вони краще чим система Trac.

Таблиця 3.8 – Порівняння по доступністю та розширюваністю

Критерій 7	A1	A2	A3	A4	Власний вектор	Вектор пріоритетів
A1	1	$\frac{1}{2}$	$\frac{1}{2}$	$\frac{1}{2}$	0.595	0.142
A2	2	1	1	1	1.189	0.286
A3	2	1	1	1	1.189	0.286
A4	2	1	1	1	1.189	0.286
Усього					4.162	1

Індекс узгодженості = 0.001,

Відносна узгодженість = $\frac{0,001}{0,9} = 0,001 = 0,1\%$

На підставі дослідження «Доступність та розширюваність» були розставлені пріоритети альтернатив один до одного. На таблиці 3.8 видно що JIRA, CodeBeamer та Bugzilla однаковий власний вектор як це було з таб. 3.7 та вони краще чим система Trac.

Таблиця 3.9 – Порівняння по спеціальністю

Критерій 8	A1	A2	A3	A4	Власний вектор	Вектор пріоритетів
A1	1	$\frac{1}{3}$	1	1	0.76	0.167
A2	3	1	3	3	2.28	0,499
A3	1	$\frac{1}{3}$	1	1	0.76	0.167
A4	1	$\frac{1}{3}$	1	1	0.76	0.167
Усього					4.56	

Індекс узгодженості = 0.002,

Відносна узгодженість = $\frac{0,002}{0,9} = 0,002 = 0,2\%$

На підставі дослідження «Спеціальність» були розставлені пріоритети альтернатив один до одного. На таблиці 3.9 видно що у JIRA вийшла набагато краще по відношенню CodeBeamer,Bugzilla Trac.

3.3 Модель порівняння

На основі отриманих результатів розрахуємо вектор глобальних пріоритетів та відповідні показники. Зазначимо, що в даному випадку індекс узгодженості – це сума індексу першого рівня ієрархії та скалярного добутку векторів пріоритетів критеріїв з вектором індексів узгодженості відповідного критерію.

Як бачимо з таблиці 3.11 усі дані відповідають нормам узгодженості, а максимальна компонента вектора глобальних пріоритетів відповідає другій альтернативі.

Таблиця 3.11 – Конечні данні

Альтернативи	Критерії								
	К1	К2	К3	К4	К5	К6	К7	К8	Узагальнені пріоритети
А1	0.111	0.048	0.141	0.122	0.095	0.142	0.142	0.167	0.135
А2	0.445	0.072	0.455	0.424	0.467	0.286	0.286	0,499	0.427
А3	0.222	0.02	0.141	0.227	0.160	0.286	0.286	0.167	0.191
А4	0.222	0.012	0.263	0.227	0.278	0.286	0.286	0.167	0.247

Яку систему відстеження помилок використовувати, залежить від типу проекту, над яким ви працюєте, і від доступних ресурсів проте, з точки зору результатів проведеного аналізу, Atlassian JIRA на цей час є кращим доступним продуктом на ринку - не тільки через своїх можливостей настройки, розширення і співробітництва, але також через своєї політики поширення, яка намагається зробити JIRA доступним для всіх. Система JIRA має найменше недоліків, значить що рекомендації по поліпшенню являються найбільш актуальні. В наступному розділі будуть описані методи управління системою.

3.4 Методи управління системою JIRA

На підставі аналізу результатів порівняння систем стеження за вадами, описаних в попередньому розділі, було зроблено рішення описати докладно методи управління JIRA

Панель управління JIRA надає безліч корисних можливостей і функцій, що дозволяють легко зібрати і впорядкувати всі знайдені проблеми.

Система JIRA складається з наступного ряду компонентів, кожен з яких можна налаштувати:

– робочий процес (workflow);

– типи завдань (issue types);

– користувальницькі робочі простори (custom fields);

– вікна (screens);

– налаштування робочих просторів (field configuration);

– повідомлення (notification);

– рішення (permissions).

JIRA дозволяє відстежувати баги і завдання, що лежать в основі проекту. Як тільки ви імпортуєте проект в JIRA, ви можете створювати завдання.

У вкладці «завдання» (Issues) можна виявити наступні функції:

– типи завдань (issue types);

– робочий процес (workflow);

– вікна (screens);

– робочі простори (fields);

– властивості завдань (issue attributes).

У вкладці «типи завдань»відображені всі типи завдань, які можна створювати і відстежувати в JIRA. Як можна побачити на рис. 3.1, завдання класифікуються різними видами функцій, підзадач, багів і тощо.

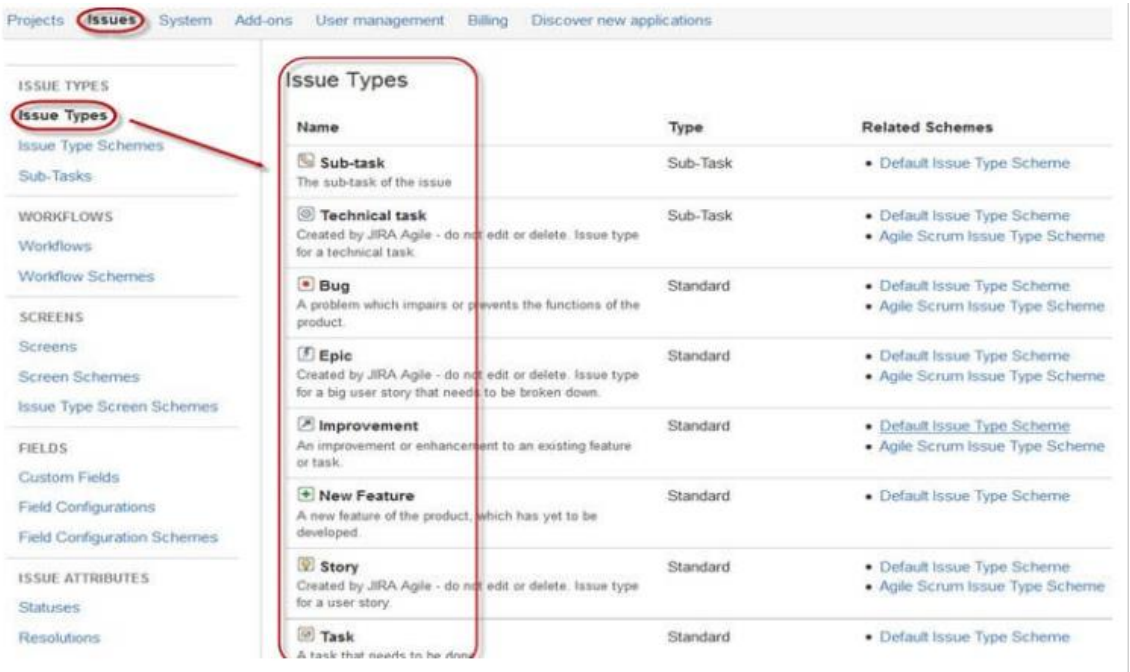

Рисунок 3.1 – Класифікація різних видів функцій

У JIRA є дві системи організації типів завдань.

Стандартна система організації типів завдань (Default Issue Type Scheme).

У стандартній системі організації типів завдань всі нові створені завдання автоматично додаються на схему.

Agile Scrum система організації типів завдань (Agile Scrum Issue Type Scheme).

Завдання та проекти, які асоціюються з Agile Scrum, використовують цю систему.

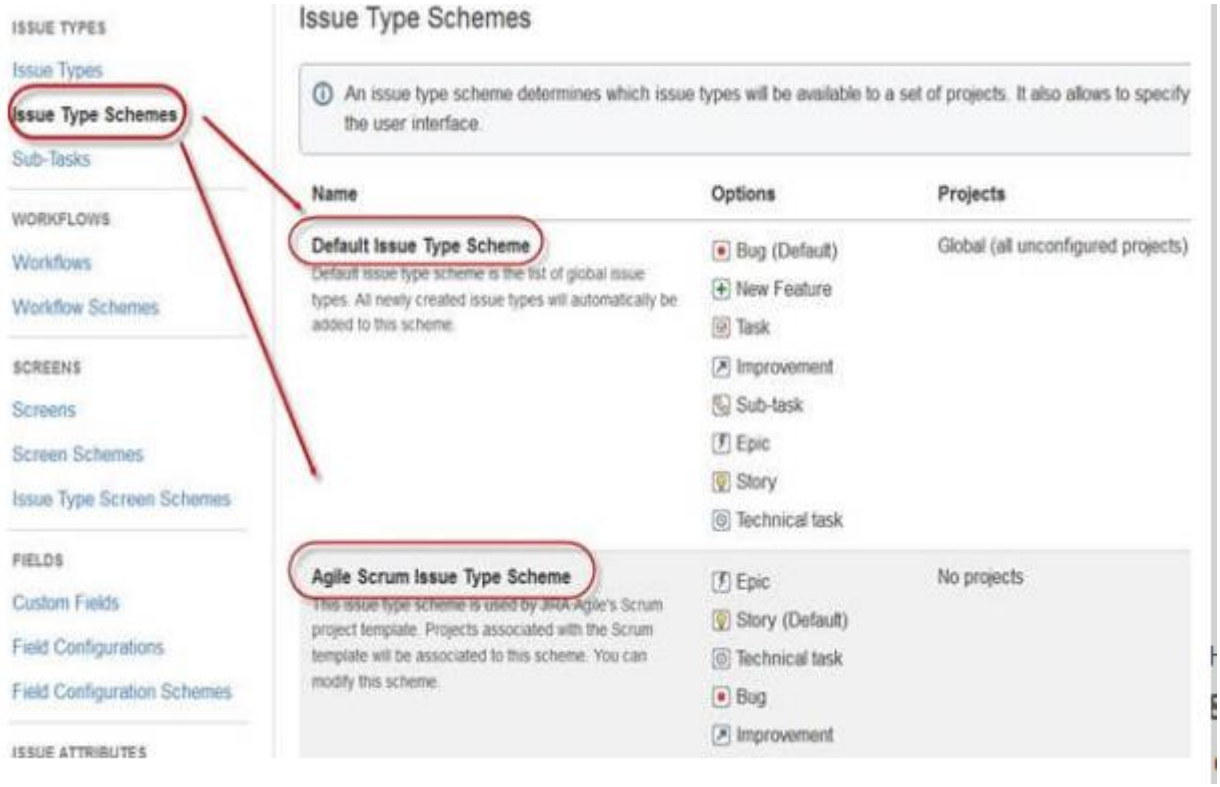

Рисунок 3.2 – Системи організації типів завдань Jira

Крім цих двох схем ви можете створювати власні типи завдань, налаштовуючи функціонал під себе. Наприклад, ми можемо створити схему IT і підтримка (IT & Support), перетягнувши типи завдань з вкладки «Доступні типи завдань» (Available Issue type) на вкладку «типи завдань для поточної схеми» (Issue type for current scheme)..

Компоненти – це підрозділи проекту. Вони використовуються, щоб комбінувати завдання поточного проекту в малих розділах. Компоненти додають проекту структурованість, розбиваючи його на функції, групи, модулі, під проекти та інше. Використовуючи компоненти, Ви можете генерувати звіти, збирати статистику, відображати її на панелях управління.

Якщо завдання створено в JIRA, воно буде організовано та представлено на різних робочих просторах, які називаються екранами. Ці робочі простори можуть переводитися і редагуватися в ході робочого процесу. Як можна побачити на скріншоті, кожному завданню ви можете призначити тип екрану. Щоб асоціювати здійснення завдання з певним екраном, потрібно зайти в головне меню, клікнути завдання (Issues), клікнути схеми (Schemes), після цього клікнути асоціювати здійснення завдання з екраном (Associate an issue operation with a screen) і додати екран, відповідний вимогам.

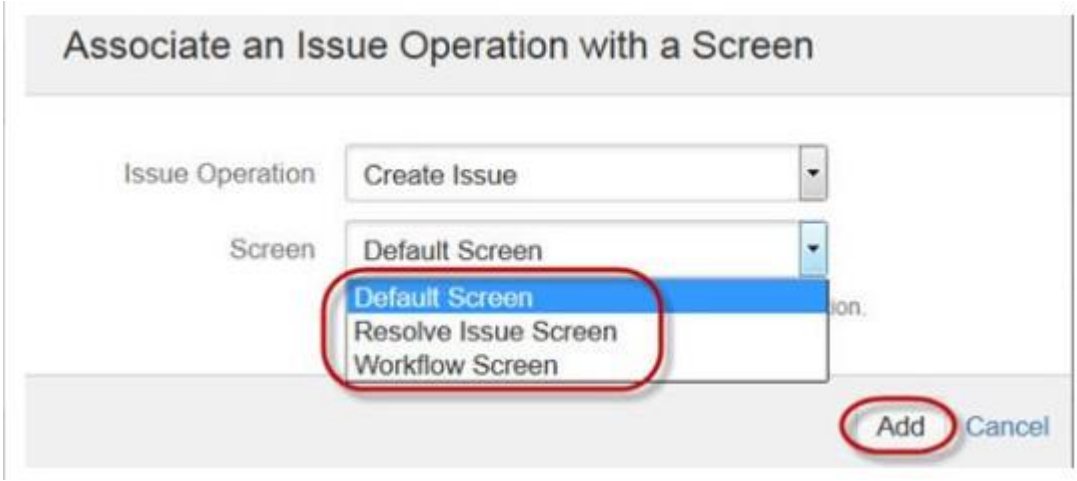

Рисунок 3.3 – Вікна Jira

У властивості завдань (Issue Attributes) входять:

– статуси (Statuses);

– рішення (Resolutions);

– пріоритети (Priorities).

Різні статуси використовуються, щоб позначити прогрес проекту: очікує виконання (Todo), виконується (In Progress), Відкритий (Open), закритий (Closed),

Перевідкритий (ReOpened), вирішений (Resolved). Також є рішення і пріоритети. Рішення позначають прогрес виконання завдання: Виправлено (Fixed), не буде виправлено (Won't fix), дублікат (Duplicate), не завершено (Incomplete), не відтворюється (Cannot reproduce), виконано (Done). Також можно вказати пріоритети завдань: критичний (Critical), високий (Major), малозначимий (Minor), блокуючий (Blocker) і тривіальний (Trivial).

Функція «схеми захисту завдань» JIRA (Issue Security Schemes) дозволяє вам контролювати доступ до завдань. Вона включає в себе кілька рівнів доступу, які розподіляються між користувачами і групами. Можно вказати рівень доступу до завдання під час її створення або редагування.

Також є стандартна схема захисту (Default permission Scheme), яка буде призначена будь-якому новому проекту. Схеми захисту дозволяють створювати набори рівнів доступу та застосовувати їх до будь-якого проекту.

Є декілька корисних функцій, які JIRA надає адміністратору.

Логи ревізій (Audit Log). У цій вкладці ви можете побачити деталі створеного завдання, а також зміни, внесені в завдання.

Зв'язування завдань (Issue Linking). Тут вказується чи пов'язана ваше завдання з якоюсь іншою, існуючої в даному проекті. Також в цій панелі можна скасувати даний зв'язок.

Система пошти JIRA (Mail in JIRA). Використовуючи систему пошти в якості адміністратора, ви можете пересилати завдання на поштові сервера POP і IMAP, а також відправляти їх у вигляді повідомлень на зовнішні поштові скриньки.

Події (Events). У цій вкладці описано статус, стандартний шаблон, схеми оповіщення та передача відповідальності за подію. Події розділені на два типи: Системні події (System event, ті, що встановлені в JIRA за замовчуванням) і призначені для користувача події (Custom event, відповідно, ті, що були створені користувачами).

Контрольний список (Watch list). Дозволяє переглядати певні завдання, бачачи повідомлення, пов'язані з ними. Щоб переглянути завдання, клікніть

"перегляд" у вікні завдання, а якщо ви хочете побачити, хто ще переглядає це завдання, ви можете натиснути на число в дужках.

Лічильник завдань (Issue Collectors). Дозволяє збирати інформацію з будь-якого сайту. Будучи адміністратором, можна клікнути по лічильнику завдань, після чого з'явиться опція, що дозволяє його додати. Як тільки ви налаштуєте зовнішній вигляд лічильника, автоматично згенерований JavaScript можна перенести на сайт для передачі інформації.

Інструменти розробки (Development Tools). Ви можете також підключити ваші інструменти розробки ПЗ до JIRA, використовуючи функції адміністратора. Вам необхідно ввести URL програми для підключення його до JIRA.

Як створити завдання в JIRA (How to create an issue in JIRA)

Панель завдань JIRA відкриється, як тільки ви введете свій ID і пароль. Під панеллю керування ви виявите вкладку проекти (Project). Клікнувши по ній, ви відкриєте вікно зі списком таких опцій, як просте відстеження завдань (Simple Issue Tracking), управління проектами (Project Management), Agile Kanban, класична JIRA (Jira Classic), відповідно скріншоту.

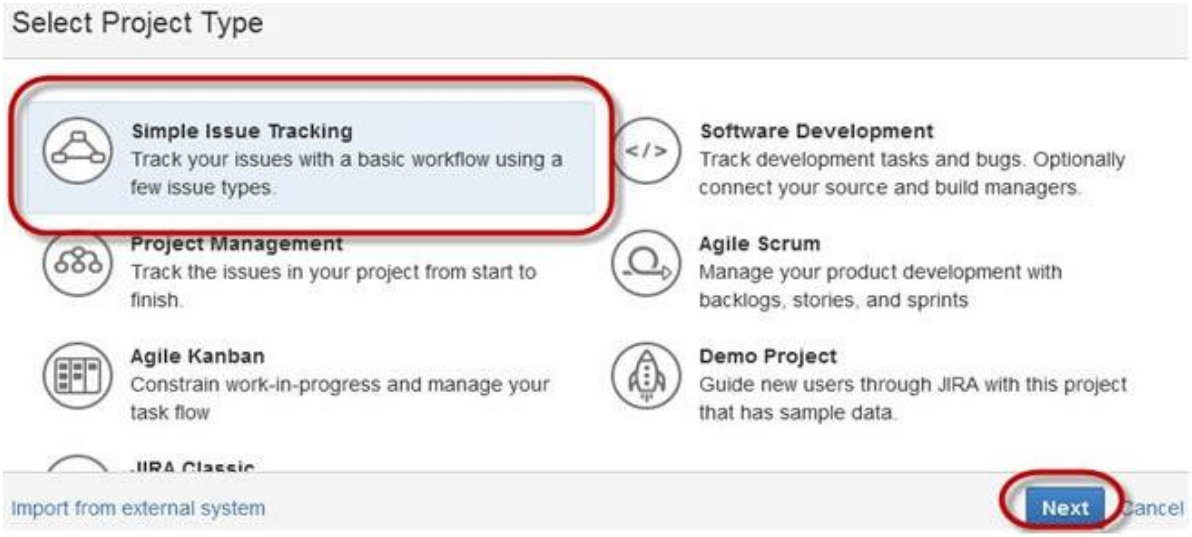

Рисунок 3.4 – Створення завдання у Jira

Якщо клікнуть по опції «просте відстеження завдань» (Simple Issue Tracking), відкриється інше вікно, в якому згадуються деталі завдання, а також призначення певній відповідальній особі.

Після натискання кнопки «підтвердити» (Submit) відкриється вікно, в якому можна виконати ряд дій, на зразок створення і призначення завдань, перевірок їх статусу і тощо.

Можна побачити на рис 3.5 що після завершення створення завдання на екрані з'явиться спливаюче вікно з оповіщенням про те, що завдання успішно створена.

Тепер, якщо необхідно відредагувати завдання або експортувати його у вигляді XML або Word документа, можно навести курсор на головну панель і клікнути завдання (Issues). У списку виберіть «Пошук завдань» (Search for issues), після чого відкриється вікно, за допомогою якого можно виявити ваші завдання і виконати інші дії.

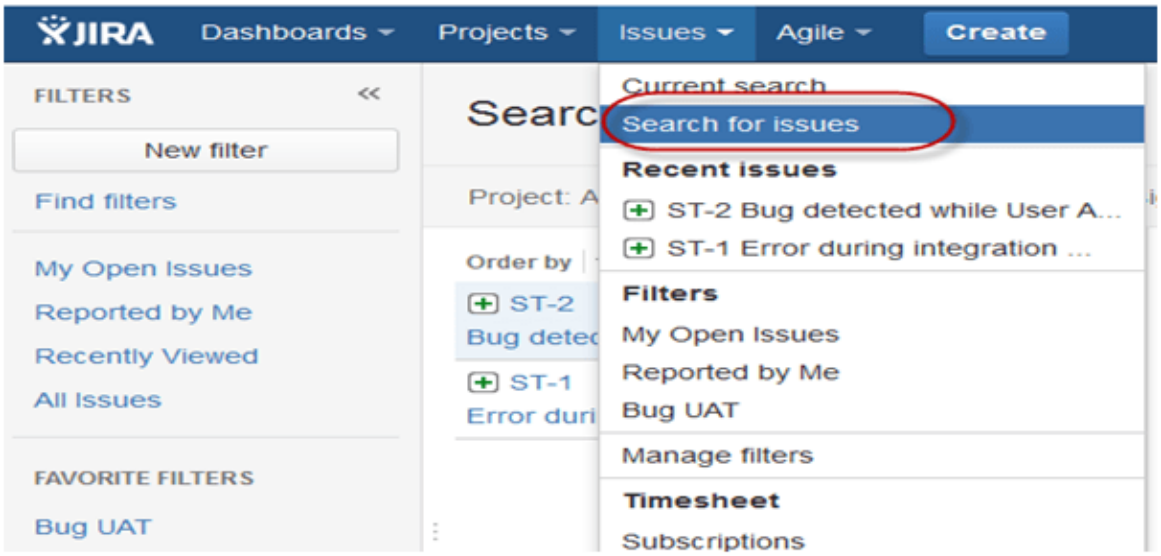

Рисунок 3.5 – Пошук завдань у Jira

Коли виберете «пошук завдань» (search for Issues), відкриється таке ж вікно, як на рис 3.6, на ньому можна побачити завдання «Баг», виявлений під час призначеного для користувача оціночного тестування» (Bug detected while User Acceptance Testing) і всі деталі, що стосуються її. Звідси ви можете виконати різні

дії, наприклад, призупинити роботу над завданням (stop the progress on issues), відредагувати завдання (edit the issues), прокоментувати завдання (comment on the issues), призначити завдання на когось (assigning issues) .

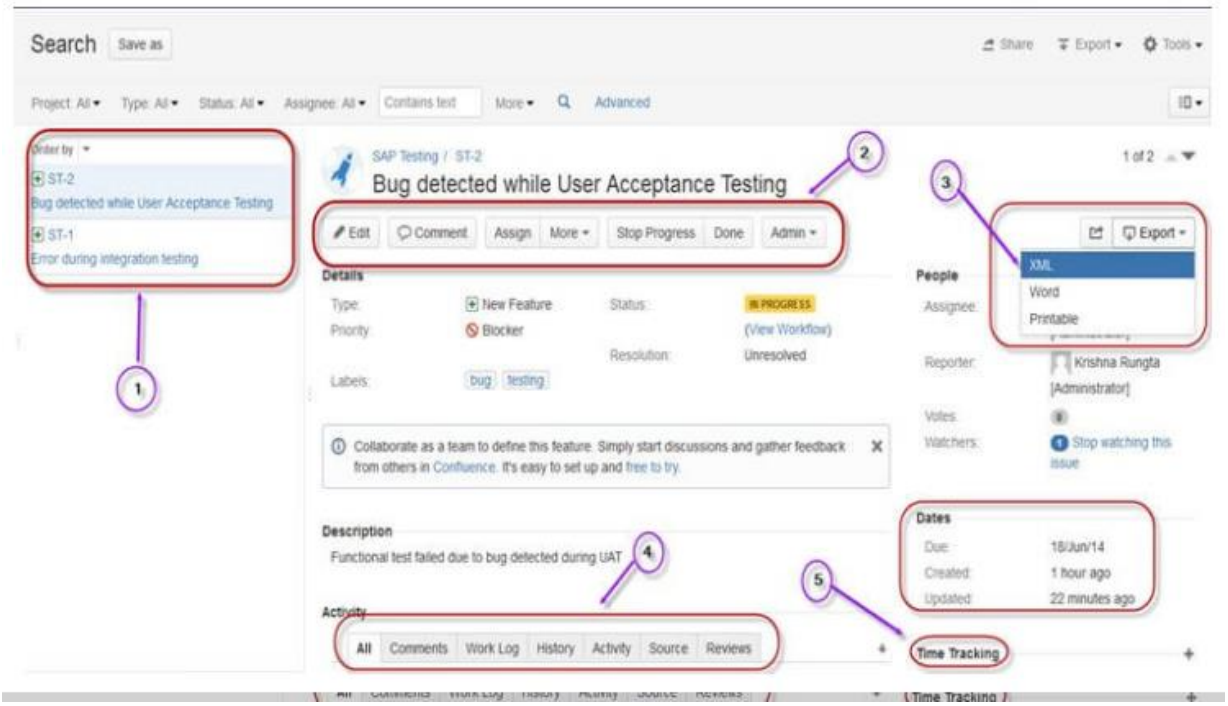

Рисунок 3.6 – Результат пошуку у Jira

У цьому ж вікні можна встановити фільтр для завдання і зберегти його в «Обрані фільтри» (Favorite Filters), так що якщо потрібно знайти це завдання, це легко можна зробити, скориставшись фільтром.

За допомогою функцією «зведення» (Summary), можна відкрити вікно з діаграмою, на якій зображені всі деталі, пов'язані з вашим проектом, і прогрес роботи над ним. У правій частині вікна зведення можна побачити «Журнал активності» (Activity Stream), на якому відображаються деталі, пов'язані із завданням, і коментарі, залишені відповідальним за завдання людиною.

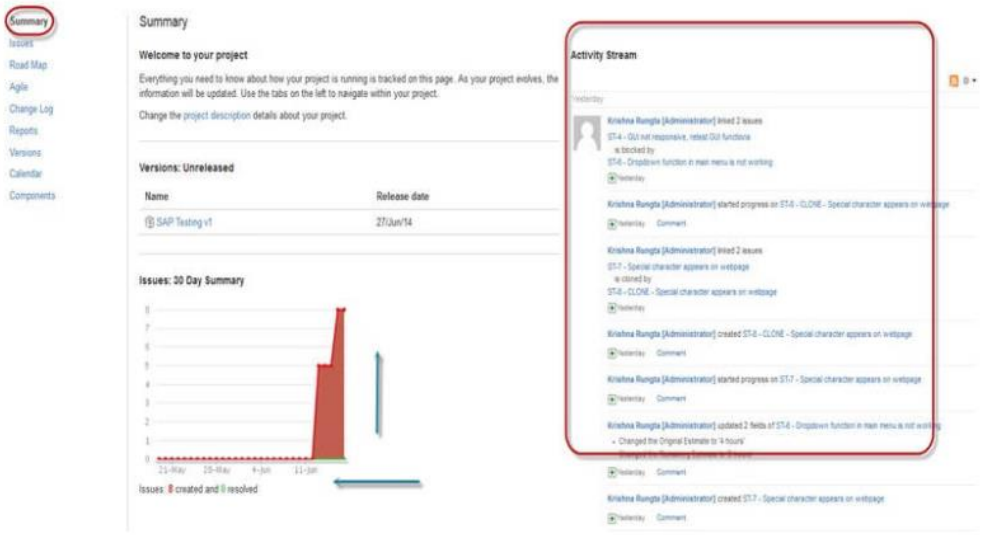

Рисунок 3.7 – Сводка у Jira

Невеликі підзадачі (Sub Task) корисні, коли потрібно розбити основне завдання на ряд окремих, які точно так само можуть бути відстеженні. Це дозволяє всебічно підійти до основного завдання, розподіляючи навантаження між декількома працівниками.

Підзадача може бути створена двома способами:

– через опцію «підзадачі» (sub-task) на вікні основного завдання;

– при створенні завдання можна вказати, що вона є підзадачею.

Щоб створити підзадачу в JIRA, потрібно вибрати завдання, до якого хочете її прикріпити. У вікні завдання виберіть опцію призначення. Після цього виберіть Створити підзадачу (Create sub-task), як показано на скріншоті. Також можна конвертувати завдання в підзадачу (convert to sub-task), вибравши відповідну опцію.

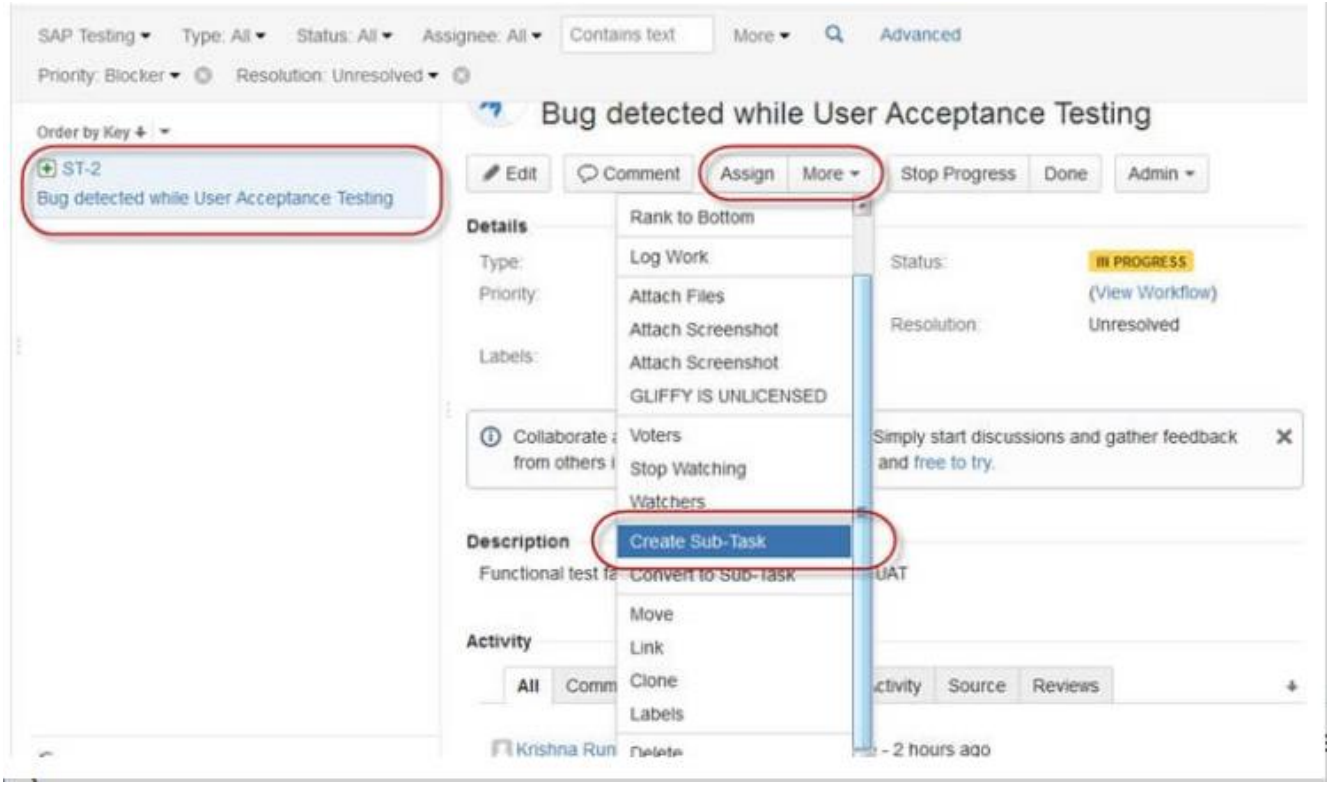

Рисунок 3.8 – Створення саб-таски у Jira

Таким чином, можна створити підзадачу, прикріплену до основного завдання, а на сторінці «Підзадач» можна побачити час, відведений на її виконання.

Кілька важливих речей, які потрібно пам'ятати, створюючи підзадачі:

– можна створити необмежену кількість в рамках основного завдання;

– не можна створити підзадачу для підзадачі;

– якщо для завдання була створена підзадача, то вона вже не може бути конвертована в підзадачу;

– підзадача, однак, може бути конвертована в звичайну задачу;

– можна працювати з підзадачами, не закриваючи вікно основного завдання.

В наступному розділі будуть описані проблеми та рекомендації по покращенню проблемних місць у системи відстеження помилок JIRA

3.5 Модифікація методів управління баг-трекінгових систем на основі обраних критеріїв

В ході дослідження систем стеження за вадами, було з'ясовано що в них є проблемні місця які заважають команді розробників працювати над проектів. Далі будуть описані проблемні місця в системах і варіанти їх вирішення.

Одна з найважливіших частин якої немає в системах відстеження за помилками це – інструменти, які допомагають їм збирати інформацію, необхідну для подачі звіт про помилки. ідеалі такі інструменти повинні бути інтегровані в саме програмне забезпечення або в його систему плагінів.

Надання тестувальникам інструментальну підтримку які дозволяють робити скріншоти, запис відео, отримати логи дефектів для збору та підготовки необхідної розробникам інформації, приведе до поліпшення і прискорення надання інформації про помилки для звітів

Наступна проблема це – різні рівні знань, які мають тестувальники програмного забезпечення. Новачки в тестуванні часто роблять помилку і не знають як правильно збирати інформацію потрібну для правильного звіту про помилки і так само яку інформацію потрібно надати звіту, яка допоможе розробникам програмного забезпечення зрозуміти в чому проблема і швидко розв'язати цю проблему.

Потрібно дати підказки недосвідченим репортерам, яку інформацію вони повинні надати і як вони можуть її зібрати. Так само новачкам в тестування буде корисно пройти курси сертифікації ISTQB, які дозволять тестувальникам отримати всі необхідні знання для своїх ключових активностей.

Також під час пошуку та аналізу інформації про проблемні місця систем стеження за помилками і управлінням проектами, була виявлена така інформація, що тестувальники, які добре відомі, або особисто, або через добре написані минулі повідомлення про помилки, отримають більше уваги.

Одним з поліпшень в системах відстеження помилок було б введення репутації в профілі користувачів. Це допоможе розробникам швидко виявити досвід роботи репортера, навіть якщо вони не знають його особисто.

В ідеалі репутація повинна складатися з двох компонентів: незалежний від проекту, який розповідає про досвід роботи зі звітами про помилки в цілому, і специфічний для проекту, який говорить про те, що досвід для даного проекту.

Для вирішення даної ситуації слід інтегрувати систему репутацію в профілі користувачів, яка дозволяє виставляти бали репутації тестувальників на підставі виконаної роботи. Виставлення балів в системі, бажано проводити між «Спринтами» при догляді з проекту, що б не з'являлися проблеми в колективі під час роботи.

Наступна проблемне місце в системах відстеження помилок це дублікати, їх проблема, полягає в тому, що через них витрачається багато часу. Вони помилково надають інформацію про нову проблему і можуть змусити різних розробників несвідомо працювати над виправленням однієї і тієї ж помилки. Щоб уникнути такої втрати часу, часто спочатку перевіряють помилку, якщо вона повторюється. Якщо помилка визначена як така, системи стеження за порушеннями дозволяють тестувальникам або розробникам позначати їх як дублікати, щоб інші знали, що їх можна ігнорувати. Системи відстеження помилок мають обмежену функціональність пошуку. Як наслідок, у багатьох звітах описується проблема, про яку вже повідомлялося в минулому.

Для розв'язання даної проблеми потрібно забезпечити потужну, але просту і зручну у використанні функцію пошуку повідомлень про помилки. Ця система повинна аналізувати новий «звіт про помилку» і порівнювати його з вже існуючими. Для більш кращого і швидкого порівняння, можна додати обов'язковий поле «компонент» в якому тестувальник буде вказувати, в якій частині програми, відбулося невідповідність фактичного і очікуваного результату при перевірки функціоналу.

Інформація в дублікатах помилок може бути корисною. Багато рекомендацій щодо складання звітів про помилки вважають дублікати звітів про помилки

шкідливими. Коли звіт про помилку ідентифікується як дублікат, він просто закривається і інформація відкидається, що в довгостроковій перспективі перешкоджає користувачам відправляти звіти про помилки. Вони неохоче надають додаткову інформацію, як тільки бачать, що звіт про помилку вже поданий.

Заохочуйте користувачів надавати додаткові відомості, в ідеалі до вже існуючого звіту про помилку. Для вирішення цієї проблеми можна реалізувати спеціальні інструмент по злиттю двох існуючих звітів про помилки, де користувач зможе вибирати які розділи звітів він хоче поєднати, а що хоче прибрати с звіту.

Підбиваючи підсумки, крім використання систем відстеження помилок, команда повинна навчитися правильно оформляти звіти про помилки та дотримуватися таких правил а інструменти які надає система допоможе організувати ефективну взаємодію учасників процесу:

– будьте конкретні. У детальному описі повинно бути вказано, що є очікуваним результатом і що відбувається замість цього - що можна вважати помилкою;

– покажіть якомога більше. Якщо програма пише повідомлення про помилку, воно має бути скопійовано в звіт. Якнайбільше додавати скріншоти в звіти про помилки дуже важливо;

– опишіть, як відтворити проблему. Програміст повинен буде відтворити проблему на своєму власному комп'ютері. Після запису, як це зробити, користувач повинен сам пройти ці інструкції. Повинна бути описане середовище, в якому виникла проблема – конфігурація комп'ютера, операційна система, мережеве з'єднання, браузер, версія Java. Все, що можливо пов'язано з помилкою, має бути записано. Гарне практичне правило – писати більше, ніж менше;

– чітко розрізняйте факти і припущення. Іноді помилка прихована в іншому місці і з припущеннями може привести програміста до помилкового сліду. Основну увагу слід приділити симптомів, а діагноз слід залишити фахівцям.

ВИСНОВКИ

В результаті атестаційної роботи було досліджено таку проблему як методи управління багтрекінговими системами з метою їх оптимізації.

В роботі виконано дослідження різних систем відстеження помилок з використанням певних критеріїв. На підставі зібраної інформації було зроблено докладне порівняння цих систем.

Проведений системний аналіз надав можливість обрати найкращу систему відстеження помилок на основі отриманої інформації команді розробників буде легше робити вибір яку систему управління проектами та відстеження помилок вибрати для проекту.

JIRA показала себе кращою за всіма критеріями крім здатності інтеграції з системами управління віхіднім кодом, в цьому критерії вона була нарівні з CodeBeamer і Bugzilla випереджаючи систему Trac.

В кінці роботи були описані проблемні місця систем управління проектами та стеження за помилками і були написані рекомендації щодо поліпшення цих систем .

Вибір інструментів відстеження помилок дуже великий. Якщо ви використовуєте інструмент управління тестами, у вас також буде доступ до відстеження дефектів.

Нарешті, якщо вашій команді потрібен інструмент для відстеження дефектів і якщо все тестування як і раніше ведеться вручну, найкраще використовувати систему управління проектами та відстеження помилок – JIRA..

Отримана інформація під час дослідження позволяє оптимізувати процеси управління проектів і систем відстеження за помилками.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Браткевич В.В., Бутов М.В. Быстрое тестирование. Вильямс 2012. – 384.
2. International Software Testing Qualifications Board. 2018. – 50-70 с.
3. Тестирование программного обеспечения. Базовый курс: практ. пособие. / С. С. Куликов. – Минск: Четыре четверти, 2015. – 294 с.
4. Рекс Блэк. Ключевые процессы тестирования. Планирование, подготовка, проведение, совершенствование. Лори, 2016. – 544 с.
5. Роман Савин .Тестирование Дот Ком, или Пособие по жестокому обращению с багами в интернет-стартапах. Дело, 2017. – 312 с.
6. Сэм Канер, Джек Фолк, Енг Кек Нгуен. Тестирование программного обеспечения. Фундаментальные концепции менеджмента бизнесприложений ДиаСофт, 2011. – 544 с.
7. Груздо І.В., Соловйова К.О., Піщухіна О.О, Данилов А.Д. Аналіз, розробка та управління вимогами: Навч. Посібник – Харків: ХНУРЕ, 2015 – 276 с.
8. Duenas JC, Navarro JM, Parada HA, Andion J, Cuadrado F. Applying event stream processing to network online failure prediction. Commun Mag. 2018. – 166–70 с.
9. Черняк, М. Оценка эффективности автоматизации тестирования [Электронный ресурс] / М. Черняк. – 2015. – Режим доступа : http://www.a1qa.ru/blog/otsenka-effektivnosti-avtomatizatsiitestirovaniya/. – Дата доступа : 25.04.2020.
10. Груздо І.В., Федосенко Н. Prerequisites for creation and main ideas of the semantic web. of the eleventh international scientific-practical conference internet-education-science-2018. ukraine vinnytsia vntu 22-25 may, 2018. 254-255 p.
11. How to use Jira Software – оновлення 2020. URL: https://www.atlassian.com/software/jira/guides (дата звернення: 10.03.2020)
12. Mozilla bug tracking system.- оновлення 2017 URL: https://bugzilla.mozilla.org/. (дата звернення: 20.03.2020).

13. S. Breu, J. Sillito, R. Premraj, and T. Zimmermann. Frequently asked questions in bug reports. Technical report, University of Calgary, 2009. – 2 с.

14. G. Leban and M. Grobelnik, “Displaying email-related contextual information using Contextify,” International Semantic Web Conference, Shanghai, China, 2010. – 181-184 с.

15. X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, “An approach to detecting duplicate bug reports using natural language and execution information,” in Proceedings of the 30th international conference on Software engineering, 2018, pp. 461–470.

16. A. Chowdhury, O. Frieder, D. Grossman, and M. C. McCabe, “Collection statistics for fast duplicate document detection,” ACM Transactions on Information Systems (TOIS), vol. 20, no. 2, 2012. – 171–191 с.

17. S. Just, R. Premraj, and T. Zimmermann. Towards the next generation of bug tracking systems. In VL/HCC’08: Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing, 2008. – 82–85 с.