МАГІСТЕРСЬКА РОБОТА

МР. ІПм - 03.00.00.000 ПЗ

Група ІПм-24-1

Б`єсик Володимир

2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Б`єсик Володимир Михайлович

(прізвище, ім'я, по батькові)

УДК 004.9

(індекс)

МАГІСТЕРСЬКА РОБОТА

Моделі та методи документування інноваційних програмних проектів

(назва роботи)

Інженерія програмного забезпечення

(назва освітньої програми)

121 - Інженерія програмного забезпечення

(шифр і назва спеціальності)

Б'єсик В.М.

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Шекета Василь Іванович, д.т.н., професор

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

доц. Бандура В.В.

(посада) (підпис) (дата) (ініціали та прізвище)

Нормоконтроль

доц. Вовк Р.Б.

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

Івано-Франківськ – 2025

Івано-Франківський національний технічний університет нафти і газу

Факультет інформаційних технологій

Кафедра інженерії програмного забезпечення

Освітній рівень магістр

Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедрою **ІПЗ**

доц. В.В. Бандура

“ 04 ” вересня 2025 р.

ЗАВДАННЯ

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТУ

Б`єсику Володимиру Михайловичу

(прізвище, ім’я, по-батькові)

1. Тема магістерської роботи “ Моделі та методи документування інноваційних програмних проектів”

керівник проекту (роботи) Шекета Василь Іванович, д.т.н., професор

затверджені наказом закладу вищої освіти від “ 05 ” листопада 2025 р. № 695/7

2. Строк подання студентом проекту (роботи) 15 грудня 2025 р.

3. Вихідні дані до проекту (роботи) Теоретичні концепції та формальні моделі побудови та функціонування інформаційних технологій документування проектів

4. Зміст розрахунково - пояснювальної записки(перелік питань, які потрібно розробити)

1. Аналіз та дослідження предметної області документування програмних проектів
2. Моделі і метрики якості та відстежування проблем в програмних проектах
3. Імплементація методів документування інноваційних програмних проектів
4. Оцінка пропонованого рішення управління інноваціями в умовах гнучкої розробки

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)

1. Схема CMMI (рис. 2.1)
2. Поступове представлення CMMI та області процесів (рис. 2.2)
3. Поетапні рівні зрілості CMMI (рис. 2.3)
4. Категорії областей процесів CMMI (рис. 2.4)
5. Робочий процес запиту в Bugzilla (рис. 2.7)

6. Консультанти розділів проекту (роботи)

Розділ	Консультант	Підпис, дата
Перевірка на плагіат	доц., к.т.н. Вовк Р.Б.	

7. Дата видачі завдання 04 вересня 2025 р.

Керівник _____
(підпис)

Завдання прийняв до виконання_____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назви етапів магістерської роботи	Строк виконання етапів роботи	Примітка
1	Підбір і вивчення літератури по темі магістерської роботи	15.09.2025	виконано
2	Аналіз та дослідження предметної області документування програмних проектів	29.09.2025	виконано
3	Моделі і метрики якості та відстежування проблем в програмних проектах	15.10.2025	виконано
4	Імплементація методів документування інноваційних програмних проектів	08.11.2025	виконано
5	Оцінка пропонованого рішення управління інноваціями в умовах гнучкої розробки	20.11.2025	виконано
6	Реалізація функціональності пропонованої технології	01.12.2025	виконано
7	Затвердження пояснювальної записки роботи завідувачем кафедри	15.12.2025	виконано

Студент – магістр _____
(підпис)

Керівник роботи _____
(підпис)

АНОТАЦІЯ

Магістерська робота: 77 с., 28 рис., 2 табл., 35 джерел.

Тема: Моделі та методи документування інноваційних програмних проектів

Об'єкт дослідження: процеси управління документуванням і відстеженням у життєвому циклі інноваційних програмних проєктів.

Мета роботи: дослідження та обґрунтування моделей і методів документування інноваційних програмних проєктів, що забезпечують підвищення якості управління вимогами, змін і дефектами в умовах застосування гнучких методологій розробки.

Предмет дослідження: моделі, методи та інструменти документування та відстеження задач, змін і дефектів у програмних проєктах.

Результати дослідження

В роботі здійснено дослідження проблематики документування та відстеження інноваційних програмних проєктів із урахуванням сучасних методологій управління, стандартів якості та прикладних інструментів підтримки життєвого циклу програмного забезпечення.

Висновок

Запропоновані методики дозволяють покращити прозорість процесів документування; забезпечити оперативне управління змінами та підвищити якість кінцевого продукту.

ДОКУМЕНТУВАННЯ ПРОГРАМНИХ ПРОЄКТІВ, МОДЕЛЬ, УПРАВЛІННЯ ВИМОГАМИ, СИСТЕМИ ВІДСТЕЖЕННЯ ЗАДАЧ, ІННОВАЦІЙНІ ПРОЄКТИ, УПРАВЛІННЯ ЗМІНАМИ

ABSTRACT

Master Thesis: 77 pp., 28 fig., 2 tab., 35 sources.

Topic: Models and methods of documenting innovative software projects

Object of research: processes of documentation and tracking management in the life cycle of innovative software projects.

Purpose of work: research and justification of models and methods of documenting innovative software projects, which ensure the improvement of the quality of requirements, changes and defects management in the conditions of flexible methodological developments.

Subject of research: models, methods and tools of documenting and tracking tasks, changes and defects in software projects.

Research results

The study of the issues of documenting and tracking innovative software projects was effectively carried out, taking into account modern management methodologies, quality standards and applied tools for supporting the software life cycle.

Conclusion

The proposed methods cannot improve the transparency of documentation processes; ensure operational change management and improve the quality of the final product.

SOFTWARE PROJECT DOCUMENTATION, MODELS, REQUIREMENTS MANAGEMENT, TASK TRACKING SYSTEMS, INNOVATIVE PROJECTS, CHANGE MANAGEMENT

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ALM - Application Lifecycle Management

DSR - Design science research

CMMI - Capability Maturity Model Integration

FEI - Front-end of innovation

IPM - Ideation portfolio management

NCD - New concept development

NPD - New product development

SaaS - Software-as-a-service

S3M-i - Strategic management maturity model for innovation

ВСТУП

Актуальність теми.

Сучасний розвиток інформаційних технологій супроводжується зростанням складності програмних систем, підвищенням вимог до їхньої надійності, якості та гнучкості. В умовах цифрової трансформації організацій особливого значення набуває процес документування інноваційних програмних проєктів, що забезпечує прозорість життєвого циклу розробки, ефективне управління змінами та підтримку комунікації між усіма учасниками проєктної діяльності.

Управління вимогами, контроль змін і відстеження дефектів є критично важливими для успішного впровадження інноваційних рішень у сфері програмної інженерії. Використання сучасних моделей зрілості процесів (зокрема CMMI), міжнародних стандартів управління якістю (ISO) та інструментів відстеження задач (Bugzilla, CodeBeamer, Rational ClearQuest, Trac, JIRA) дозволяє підвищити ефективність розробки програмного забезпечення та забезпечити його відповідність бізнес-цілям.

Незважаючи на широкий спектр доступних інструментів, проблемою залишається їхня інтеграція у гнучкі методології управління проєктами (Agile, Scrum), а також адаптація до потреб інноваційних компаній. Саме тому актуальним є дослідження моделей і методів документування, розробка рекомендацій та впровадження практичних рішень у реальному бізнес-середовищі.

Актуальність дослідження зумовлена потребою у підвищенні якості управління інноваційними програмними проєктами, де швидкі зміни ринку, зростання конкуренції та ускладнення програмних систем висувають нові вимоги до процесів документування та відстеження. Традиційні підходи часто не забезпечують достатньої гнучкості та адаптивності, що ускладнює реалізацію проєктів в умовах невизначеності.

Впровадження систем відстеження задач та інтеграція з Agile-практиками дозволяє досягти балансу між формалізованістю процесів та їхньою гнучкістю. Крім того, узгодженість із міжнародними стандартами ISO та моделями зрілості (CMMI) забезпечує системність і підвищує конкурентоспроможність програмних продуктів.

Таким чином, розробка й удосконалення моделей та методів документування інноваційних програмних проєктів є необхідною умовою для забезпечення їхньої результативності, якості та відповідності потребам сучасного бізнес-середовища.

Метою магістерської роботи є дослідження та обґрунтування моделей і методів документування інноваційних програмних проєктів, що забезпечують підвищення якості управління вимогами, змін і дефектами в умовах застосування гнучких методологій розробки.

Завдання дослідження:

- Проаналізувати предметну область документування та відстеження програмних проєктів.

- Визначити роль управління вимогами в життєвому циклі програмного забезпечення.

- Дослідити сучасні моделі та стандарти управління якістю (CMMI, ISO) у контексті документування та відстеження проблем.

- Провести порівняльний аналіз систем відстеження та документування програмних проєктів (Bugzilla, CodeBeamer, Rational ClearQuest, Trac, JIRA).

- Імплементувати запропоновані методи документування на прикладі реальної компанії та оцінити їхню ефективність.

Об’єкт дослідження - процеси управління документуванням і відстеженням у життєвому циклі інноваційних програмних проєктів.

Предмет дослідження - моделі, методи та інструменти документування та відстеження задач, змін і дефектів у програмних проєктах.

Методи дослідження:

- системний аналіз і синтез для дослідження предметної області;

- порівняльний аналіз для оцінки можливостей сучасних систем відстеження;

- методи моделювання для побудови моделей документування;

- кейс-стаді для аналізу прикладної проблеми на прикладі конкретної компанії;

- експертна оцінка для визначення зрілості процесів та оцінки впровадженого рішення.

Наукова новизна отриманих результатів полягає в удосконаленні підходів до документування інноваційних програмних проєктів шляхом інтеграції моделей зрілості процесів (CMMI) та стандартів ISO з гнучкими методологіями управління; запропоновано методику застосування системи JIRA та її розширень як комплексного інструмента управління вимогами, дефектами та змінами.

Практичне застосування результатів

Результати дослідження можуть бути використані у компаніях, що займаються розробкою програмного забезпечення, для підвищення ефективності управління проєктами. Запропоновані методики дозволяють покращити прозорість процесів документування; забезпечити оперативне управління змінами та підвищити якість кінцевого продукту.

Структура магістерської роботи. Робота складається зі вступу, трьох розділів та висновків. Загальний обсяг роботи становить 77 сторінок, і містить 28 рисунків, 2 таблиці, список використаних джерел із 35 найменувань.

РОЗДІЛ 1. АНАЛІЗ ТА ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ДОКУМЕНТУВАННЯ ТА ВІДСТЕЖЕННЯ ПРОГРАМНИХ ПРОЕКТІВ

1.1. Роль та інструменти управління вимогами в життєвому циклі розробки програмного забезпечення

Зміни є невід'ємною та постійною характеристикою процесу розробки програмного забезпечення. Основною метою будь-якого проєкту є внесення певних змін. Система або її компоненти піддаються модернізації, виправленню помилок або заміні, як правило, з метою покращення чи розширення функціональності, спрощення використання, скорочення операційних витрат тощо. Проте, зміни не завжди є безумовно корисними та потребують контролю під час їх впровадження у проєкт. За відсутності належного управління, неконтрольовані зміни можуть призвести до зриву графіку, погіршення якості продукту та навіть до повного провалу проєкту. З наближенням проєкту до завершення наслідки від внесення змін стають дедалі серйознішими. Отже, виникає очевидна необхідність у механізмі для контролю змін.

Частиною загального підходу до управління змінами є управління конфігурацією (Configuration Management). Управління конфігурацією — це процес контролю та документування змін у системі, що розробляється. Зі збільшенням масштабу проєкту зростає і необхідність впровадження ефективного управління конфігурацією. Цей підхід дозволяє великим командам розробників взаємодіяти у стабільному середовищі, забезпечуючи при цьому гнучкість, необхідну для творчої діяльності.

Ключовим компонентом управління конфігурацією є управління вимогами (Requirements Management). Управління вимогами передбачає встановлення та підтримку згоди між замовником і розробником щодо технічних та нетехнічних вимог. Ця згода є основою для оцінки, планування, виконання та відстеження всіх робіт протягом життєвого циклу проєкту, а

також для подальшого супроводу та вдосконалення розробленого програмного продукту. Основні види діяльності в рамках цього процесу включають:

- планування етапу роботи з вимогами;
- побудова процесу управління вимогами;
- контроль за змінами у вимогах;
- мінімізація додавання нових вимог (розповзання меж проєкту або "scope creep");
- відстеження прогресу виконання;
- вирішення спірних питань із замовниками та розробниками;
- проведення аналізу та затвердження вимог.

Дана робота присвячена програмним інструментам, що застосовуються в управлінні вимогами та документуванні проектів. Вони мають різні назви: системи відстеження задач (Issue Tracking Systems), системи відстеження проблем (Trouble Tracking Systems), системи відстеження помилок (Bug Tracking Systems), системи відстеження вимог (Requirements Tracking Systems) тощо. Однак їхнє призначення залишається незмінним: збір вимог, управління ними та відстеження процесу їх реалізації.

На початку цієї роботи розглядається проблематика вищезазначеного процесу відстеження. Білі буде описано життєвий цикл вимоги та ролі, залучені до цього процесу. Однією з цілей цього розділу є обґрунтування важливості систем відстеження задач. Наприкінці аналізуються методології розробки та управління, такі як CMMI та стандарти ISO (що можуть слугувати метриками якості розробки), та їхній зв'язок з даною темою.

Згодом досліджуються найпопулярніші на сьогоднішній день інструменти для відстеження задач/помилок, після чого читач ознайомиться з можливостями їх інтеграції в середовища розробки.

Також в роботі буде представлено процес розробки плагіна для IDE NetBeans, призначеного для спрощення використання переваг системи відстеження JIRA безпосередньо розробниками.

1.2. Системи відстеження і документування задач та вимог

Процес розробки програмного забезпечення є ітеративним і рідко завершується остаточною версією продукту. Завжди існують аспекти, що потребують додавання, модифікації або виправлення. Документ, що містить запит на внесення таких коректив до системи, називається запитом на зміну (Change request).

1.2.1. Функціональне призначення систем відстеження задач

В умовах постійного надходження нових вимог виникає потреба у використанні спеціалізованих інструментів, що дозволяють ефективно управляти цим процесом. Такі системи повинні забезпечувати наступні ключові можливості:

- ефективний обмін інформацією між членами команди;
- отримання миттєвого огляду стану програмного продукту;
- прийняття обґрунтованих рішень щодо випуску нових версій (релізів);
- визначення та оновлення пріоритетів для окремих завдань та виправлень;
- ведення та збереження повної історії змін.

По суті, основна функція таких систем полягає у фіксації та зберіганні вичерпної інформації щодо кожного запиту на зміну. Централізована база даних повинна містити такі відомості:

- що саме потребує виправлення чи реалізації;
- симптоми та прояви помилки, а також опис некоректної роботи;
- опис очікуваної (коректної) поведінки системи;
- хто є автором запиту, хто його підтвердив, проаналізував, реалізував рішення та перевірив результат;
- хронологія запиту: дата створення, дата виправлення та дата верифікації;

- обґрунтування вибору конкретного способу вирішення проблеми порівняно з альтернативними;

- які зміни були внесені до програмного коду;

- час, витрачений на обробку запиту.

1.2.2. Способи збору запитів на зміну

У практичній діяльності збір запитів на зміну може здійснюватися різними способами, зокрема через:

- інструменти звітування про збої (crash reporting tools, напр., BreakPad, Bug Buddy, Windows Error Reporting);

- системи відстеження задач (issue tracking systems, напр., Bugzilla, JIRA, Rational ClearQuest, Trac);

- контактні форми (веб-форми, інтегровані модулі);

- дискусійні форуми;

- електронна пошта та інші канали комунікації.

Як видно з наведеної класифікації, системи відстеження задач виділені окремо, оскільки вони є спеціалізованими інструментами, розробленими саме для виконання цих функцій та задоволення вищезазначених вимог. Інші ж методи є допоміжними або замінниками, оскільки їхнє первинне призначення є іншим.

1.2.3. Переваги використання систем відстеження та документування задач

За умови коректного впровадження та використання, системи відстеження задач забезпечують низку суттєвих переваг:

- підвищення якості програмного забезпечення;

- зростання задоволеності користувачів та замовників;

- забезпечення підзвітності та контролю за виконанням запитів;

- покращення комунікації всередині команди та із замовниками;

- підвищення продуктивності команди розробників;

- скорочення операційних витрат.

У наступних розділах роботи буде детальніше розглянуто методології та стандарти управління розробкою ПЗ, такі як CMMI та ISO, що також можуть слугувати метриками якості процесу розробки. Впровадження цих стандартів, у свою чергу, надає значні додаткові переваги.

1.3. Дослідження особливостей і характеристик запитів на зміни

1.3.1. Структура запиту на зміну

Запит на зміну (Change request) — це формалізований документ, який містить вимогу щодо внесення коректив у функціонування системи. Він є ключовим елементом у процесі управління змінами. Запит на зміну має декларативний характер, тобто він визначає, що саме потрібно досягти, але не вказує, яким чином це має бути реалізовано.

Важливими компонентами запиту на зміну є:

- Ім'я або ID ініціатора. Ця інформація необхідна для зв'язку в разі виникнення додаткових питань.

- Тип запиту. Класифікує запит як виявлену помилку (bug), вдосконалення (enhancement), нову вимогу (new requirement) тощо.

- Короткий опис проблеми. Приклад: "Критична помилка при друці в альбомній орієнтації". Цей опис має бути точним.

- Номер збірки або версія коду. Важливо вказати, чи стосується проблема кінцевого продукту, чи тестової/розробницької версії. Деякі помилки можуть проявлятися лише в комерційній версії, що робить цю інформацію критично важливою.

- Компонент. Визначення компонента, в якому виявлено проблему, спрощує її призначення відповідному розробнику.

- Детальний опис та кроки для відтворення. Ця частина включає інформацію про конфігурацію програмного та апаратного забезпечення, а також покрокові інструкції для відтворення помилки. Важливо вказувати

кожен крок, оскільки навіть незначні деталі (наприклад, використання клавіатури замість миші) можуть впливати на результат.

- Пріоритет запиту. Визначає ступінь критичності проблеми. Зазвичай використовується така градація:

- Блокувальник (Blocker): блокує подальшу розробку та/або тестування, унеможливлює роботу продукту.

- Критичний (Critical): спричиняє збої, втрату даних, значні витоки пам'яті.

- Значний (Major): призводить до суттєвої втрати функціональності.

- Незначний (Minor): спричиняє незначну втрату функціональності або має просте рішення.

- Тривіальний (Trivial): стосується дрібних проблем (орфографічні помилки, неправильне вирівнювання тексту).

- Відповідальний за запит. Вказується особа, яка буде відповідальна за виправлення помилки. У деяких компаніях за призначення запитів відповідає окрема особа.

1.3.2. Типи запитів на зміни

Запити на зміни зазвичай класифікуються на основі джерела їх походження, зокрема:

- Проблемні звіти, що містять інформацію про помилки, які необхідно виправити (найпоширеніший тип).

- Запити на вдосконалення системи від користувачів.

- Вплив від розробки інших систем.

- Зміни в базовій структурі або стандартах (наприклад, перехід на нову операційну систему).

- Вимоги вищого керівництва.

На основі цього запити на зміни можуть бути поділені на такі категорії:

- Проблема (Issue): описує "нестандартну" або "неочікувану" поведінку системи. Цей тип часто використовується, коли неможливо точно класифікувати проблему, і згодом змінюється на інший (помилка/вдосконалення) або закривається.

- Помилка (Bug): помилка, вада або несправність у програмному коді, що призводить до неналежного функціонування програми (наприклад, некоректний результат). Більшість помилок є наслідком недоліків у коді або дизайні, і лише деякі викликані некоректною роботою компіляторів.

- Вдосконалення (Enhancement): запити на покращення або розширення існуючих функцій.

- Вимога (Requirement): включає запити на нову функціональність або нові можливості системи, що можуть бути визначені як внутрішніми, так і зовнішніми факторами.

- Завдання (Task): загальна категорія для функціональних і нефункціональних завдань, що вимагають виконання. Наприклад, оновлення апаратного забезпечення або середовища.

Класифікація запитів на зміни може варіюватися залежно від внутрішніх стандартів компанії.

1.3.3. Життєвий цикл запиту на зміну

Життєвий цикл запиту на зміну починається з його створення. Зазвичай, ініціатор запиту не знає, хто є відповідальним за його опрацювання, тому він призначається провідному експерту, який потім перенаправляє його відповідному фахівцю.

Верифікація та призначення

Після отримання запиту виконується його верифікація. Верифікатор намагається відтворити помилку, слідуючи інструкціям, наданим у детальному описі.

Якщо помилка відтворюється, запит передається розробнику для її виправлення.

Якщо інформація в запиті є недостатньою або помилку не вдалося відтворити, запит повертається ініціатору для надання додаткових даних.

Статуси запиту

"Виправлено як дублікат" (Fixed Duplicates): вказує, що проблему вже було виправлено в іншому запиті.

Обробка розробником

Розробник приймає запит, підтверджуючи свою відповідальність за його вирішення. На основі аналізу проблеми розробник призначає один із наступних статусів:

Проблема не буде виправлена:

- Відкладено (Postponed): виправлення відкладається до наступної версії продукту через технічні причини, обмеження часу тощо.

- Відома проблема (Known problem): проблема або подібний запит вже відомі; можливо, існує обхідний шлях, або виправлення заплановано на майбутнє.

- Відхилено (Rejected):

- "Відхилено з інших причин" (Rejected for other reasons): наприклад, через брак інформації або неможливість відтворення.

- "Працює як задумано" (Working as Designed): поведінка, яка сприймається як помилка, насправді відповідає початковому задуму.

- "Відтерміновано" (Deferred): вартість виправлення перевищує потенційну вигоду.

- Проблема буде виправлена:

- Виправлено (Fixed): зміна внесена в код і буде доступна в наступній збірці.

- Дублікат (Duplicate): проблема є дублікатом існуючого запиту.

- Вирішено (Resolved): проблема вирішена без зміни коду (наприклад, через оновлення драйверів).

Перевірка та закриття

Після виправлення запит повертається ініціатору або тестувальнику для повторної перевірки.

Якщо помилка успішно виправлена, запит отримує статус "Перевірено" (Verified).

Якщо проблема залишається, запит повертається розробнику зі статусом "Відкрито" (Re-opened).

Завершений запит закривається, що означає завершення його життєвого циклу. Однак проблема може виникнути знову, і тоді запит може бути повторно відкритий.

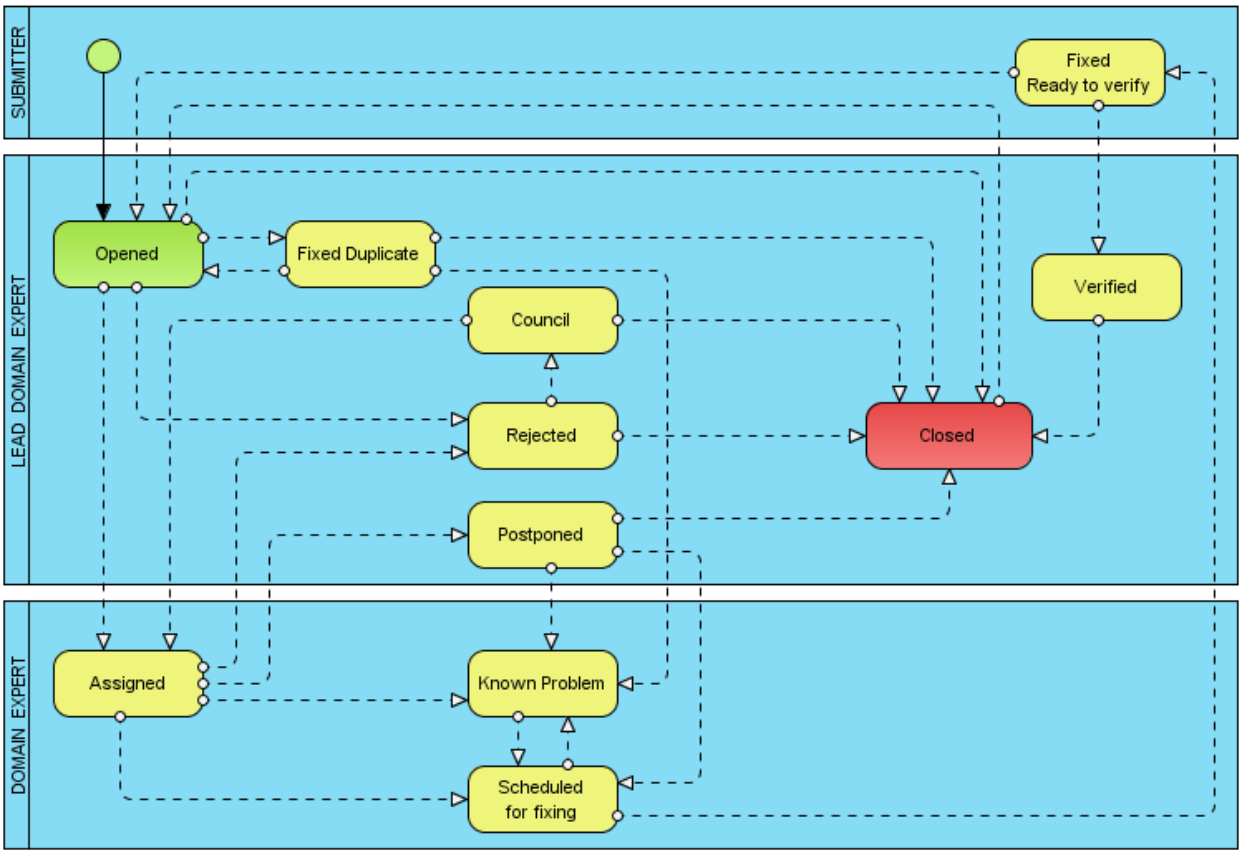

Рис. 1.1. Перехід стану запиту

Діаграма життєвого циклу запиту також включає обговорення проблеми представниками розробників і тестувальників для прийняття рішення щодо її подальшого статусу.

1.3.4. Рекомендації щодо складання звітів

Щоб збільшити ймовірність виправлення проблеми, рекомендується дотримуватися наступних правил:

Правило №1 – Пишіть якісний заголовок

Однорядковий опис проблеми є найважливішою частиною звіту. Ідеальний заголовок повинен містити:

- Короткий, але конкретний опис, що дозволяє уявити проблему.

- Вказівку на обмеження або залежності (наскільки специфічними є умови виникнення).

- Загальну оцінку серйозності (без використання формального рейтингу).

Правило №2 – Поясніть, як відтворити проблему

- Детально опишіть проблему та кроки для її відтворення.

- Почніть з відомої початкової точки (наприклад, запуск програми).

- НУМЕРУЙТЕ кроки. Описуйте кожен крок, вказуючи на будь-яку незвичайну поведінку.

- Опишіть некоректну поведінку та, якщо необхідно, поясніть, що мало статися.

- Надайте інформацію про системну конфігурацію.

- Виділіть ключові кроки та опишіть, як вони впливають на результат. Додавайте примітки з додатковою інформацією.

Правило №3 – Один звіт, одна помилка

Об'єднання кількох помилок в одному звіті призводить до:

- Нечітких заголовків (наприклад, "не працює").

- Занадто довгих і складних для розуміння описів.

- Ризику, що одна з проблем буде виправлена, а інша — ні.

- Якщо проблеми пов'язані, можна зробити перехресні посилання.

Правило №4 – Усуньте непотрібні кроки

Скорочуйте послідовність кроків до мінімально необхідної для відтворення помилки. Звертайте увагу на:

- Повідомлення про помилки, затримки або незвично швидку реакцію.

- Візуальні аномалії (некоректне відображення, мерехтіння).

- Звукові сигнали, що відрізняються від норми.

- Показники пристроїв (індикатор використання).

- Повідомлення відлагоджувального монітора.

Правило №5 – Варіації в окремому розділі

Якщо поведінка помилки змінюється в різних умовах, опишіть ці варіації в окремому розділі "ДОДАТКОВІ УМОВИ" (ADDITIONAL CONDITIONS). Почніть з найкоротшого і найпростішого шляху відтворення, а потім деталізуйте інші сценарії та їхній вплив на результат.

Подолання заперечень

Програмісти можуть уникати виправлення помилок з різних причин. Найпоширеніші з них:

- Неможливість відтворити дефект.

- Складність і незрозумілість опису.

- Занадто великий обсяг роботи.

- Виправлення не є реалістичним.

- Виправлення не вплине на користувача.

- Відсутність зацікавленості з боку керівництва.

- Особиста недовіра до автора запиту.

Мотивацією для виправлення можуть бути:

- Висока критичність проблеми.

- Цікавість, що викликана складністю помилки.

- Вплив на велику кількість користувачів.

- Простота виправлення.

- Репутаційні ризики для компанії.

- Вказівка керівництва.

- Особисті стосунки або довіра до автора запиту.

Ці рекомендації, хоч і написані переважно для звітів про помилки, є актуальними і для інших типів запитів. Їх дотримання значно підвищує ефективність процесу управління змінами.

Висновки до розділу

У результаті аналізу предметної області встановлено, що управління вимогами є ключовим процесом у життєвому циклі програмного забезпечення, оскільки забезпечує зв’язок між бізнес-цілями та технічними рішеннями. Досліджено структуру і класифікацію запитів на зміни та визначено їхній вплив на розвиток програмних систем. Використання систем відстеження й документування задач дозволяє підвищити прозорість процесу розробки, покращити контроль змін та мінімізувати ризики втрати критичної інформації.

РОЗДІЛ 2. МОДЕЛІ І МЕТРИКИ ЯКОСТІ ТА ВІДСТЕЖУВАННЯ ПРОБЛЕМ В ПРОГРАМНИХ ПРОЕКТАХ

Для об'єктивної оцінки ефективності та зрілості організації в очах замовника використовуються стандартизовані метрики. Ці інструменти слугують для підтвердження якості виробничих процесів, оптимізації внутрішньої діяльності та мінімізації витрат. Наявність відповідних сертифікатів, таких як CMMI та ISO, може стати вирішальним фактором у виборі підрядника, оскільки вони демонструють спроможність компанії працювати на певному рівні якості. Крім того, прагнення до таких сертифікацій є ефективною маркетинговою стратегією, сприяє економії ресурсів через оптимізацію процесів та слугує орієнтиром для оцінки програм управління якістю.

2.1. Інтегрована модель зрілості можливостей (CMMI)

Інтегрована модель зрілості можливостей (CMMI) є набором продуктів, розроблених Інститутом програмної інженерії (SEI) Університету Карнегі-Меллона, для покращення бізнес-процесів. Модель складається з "найкращих практик", які охоплюють весь життєвий цикл продукту — від проектування до підтримки. Хоча CMMI переважно застосовується в розробці програмного забезпечення, її загальний характер дозволяє використовувати її для широкого спектра процесів у різних сферах.

Модель вказує, що саме потрібно робити (наприклад, відстежувати зміни у вимогах), але не диктує конкретні методи виконання. Це дозволяє організаціям встановлювати цілі та пріоритети для покращення процесів, забезпечуючи їхню стабільність, передбачуваність та зрілість.

Області процесів у CMMI об'єднують індивідуальні практики зі схожими характеристиками (наприклад, управління вимогами, планування проєкту). Кожна область процесу містить специфічні цілі та практики, що

стосуються її конкретного напрямку, а також загальні цілі та практики, які застосовуються до всіх областей.

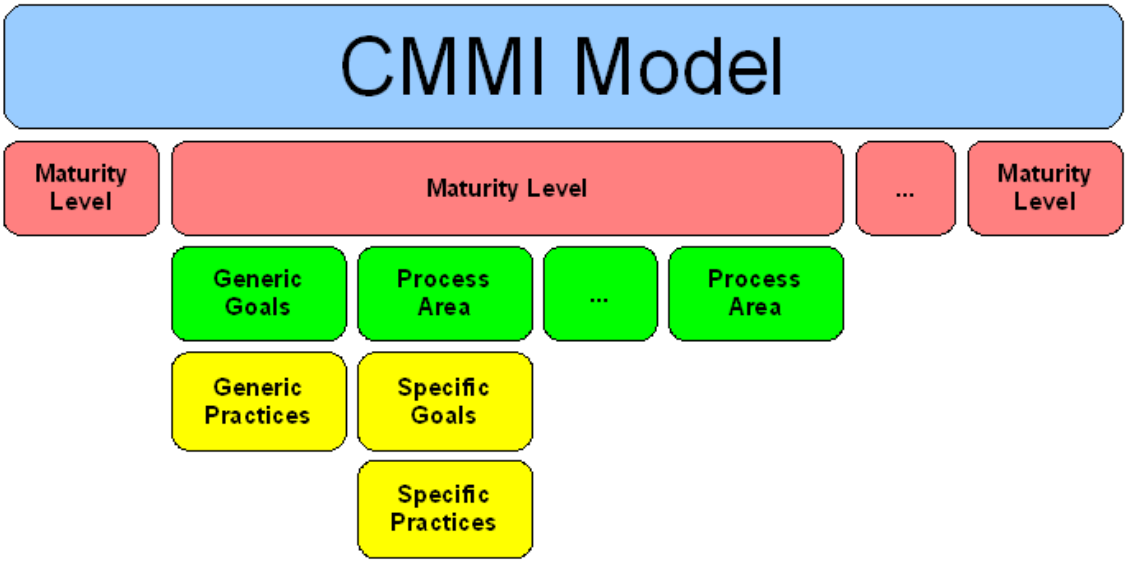

Рис. 2.1. Схема CMMI

2.1.1. Поступове та безперервне представлення CMMI

CMMI існує у двох представленнях, які дозволяють організації контролювати різні аспекти процесу покращення. Обидва представлення мають однаковий зміст, але різняться за структурою.

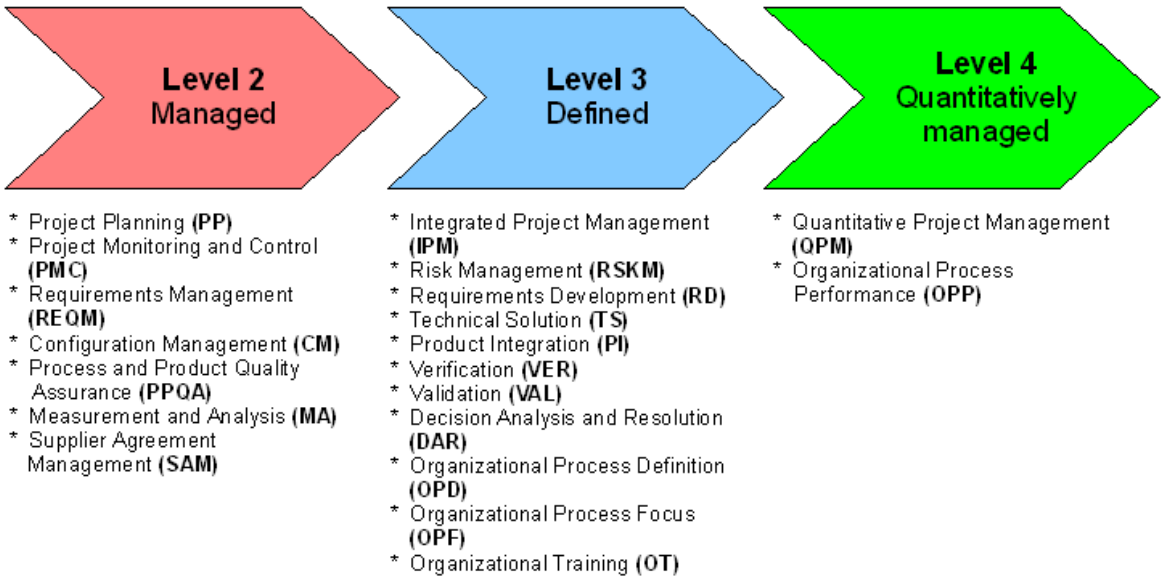

Рис. 2.2. Поступове представлення CMMI та області процесів

1. Поступове представлення (5 рівнів зрілості)

Цей підхід пропонує покрокову методологію покращення, де кожен наступний рівень ґрунтується на досягненнях попереднього. Пропуск рівнів не допускається. Ця модель дозволяє здійснювати загальне порівняння між організаціями на основі їхнього рівня зрілості.

Рівень 1: Початковий (Initial). Процеси є хаотичними, не документуються і залежать від "героїчних зусиль" окремих осіб.

Рівень 2: Керований (Managed). Організація починає визначати та документувати основні процеси. Дії контролюються і виконуються згідно з політикою.

Рівень 3: Визначений (Defined). Більшість процесів формалізовані та інтегровані в єдину модель. Це дозволяє швидко ідентифікувати та вирішувати проблеми.

Рівень 4: Кількісно керований (Quantitatively Managed). Процеси не лише визначені, але й контролюються за допомогою кількісних метрик. Дані використовуються для оптимізації.

Рівень 5: Оптимізований (Optimized). Процеси перебувають у стані постійного вдосконалення. Організація зосереджена на запобіганні помилкам та інноваціях.

2. Безперервне представлення (6 рівнів можливостей)

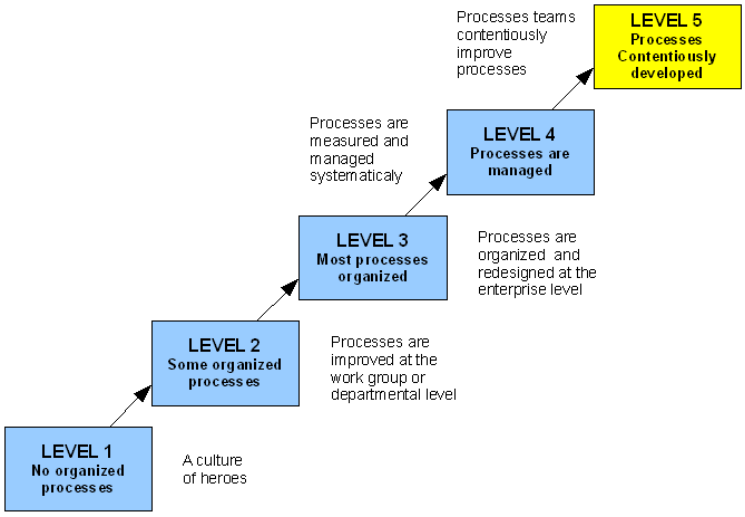

Рис. 2.3. Поетапні рівні зрілості CMMI

Ця модель дозволяє компаніям адаптувати процес покращення до своїх бізнес-цілей та зосередитися на проблемних ділянках. Кожен процес оцінюється індивідуально за 6 рівнями можливостей.

Рівень 0: Незавершений (Incomplete). Процес не виконується або виконується частково.

Рівень 1: Виконуваний (Performed). Процес задовольняє специфічні цілі, але є хаотичним та непередбачуваним.

Рівень 2: Керований (Managed). Процес планується та контролюється, має базову інфраструктуру для підтримки.

Рівень 3: Визначений (Defined). Процес адаптований зі стандартних організаційних процедур, що дозволяє синхронізувати діяльність.

Рівень 4: Кількісно керований (Quantitatively Managed). Процес контролюється кількісними аналітичними техніками.

Рівень 5: Оптимізований (Optimizing). Процес постійно вдосконалюється через інкрементальні та інноваційні зміни.

Area	Maturity level	Process Area	Abbreviation
Project Management	2	Project Planning	PP
	2	Project Monitoring and Control	PMC
	3	Integrated Project Management	IPM
	3	Risk Management	RSKM
	4	Quantitative Project Management	QPM
Engineering	2	Requirements Management	REQM
	3	Requirements Development	RD
	3	Technical Solution	TS
	3	Product Integration	PI
	3	Verification	VER
	3	Validation	VAL
Support	2	Configuration Management	CM
	2	Process and Product Quality Assurance	PPQM
	2	Measurement and Analysis	MA
	3	Decision Analysis and Resolution	DAR
Process Management	3	Organizational Process Definition	OPD
	3	Organizational Process Focus	OPF
	3	Organizational Training	OT
	4	Organizational Process Performance	OPP

Рис. 2.4. Категорії областей процесів CMMI

Це представлення дозволяє порівняння між організаціями лише на основі областей процесів. Області процесів у безперервному представленні поділяються на чотири категорії: планування проекту, підтримка, інженерія та управління процесами.

2.2. Стандарти ISO. Роль відстежування проблем в стандартах управління якістю

ISO 9000 — це сімейство стандартів, що стосуються систем управління якістю. Для розробки програмного забезпечення найбільш важливими є:

ISO 9000:2005. Містить основи та термінологію для систем управління якістю.

ISO 9001:2000. Визначає вимоги, яким організація повинна відповідати для досягнення задоволеності клієнтів. Включає вимоги до постійного покращення.

ISO 9004:2000. Надає рекомендації щодо покращення продуктивності системи управління якістю.

Ключові принципи, що містяться в ISO 9001:2000, включають:

- Офіційна політика якості, що відповідає бізнес-цілям.
- Прийняття рішень на основі документованих даних.
- Відстеження продуктів та проблем до джерела.
- Документаційний контроль та управління ресурсами.
- Моніторинг та вимірювання ключових процесів.
- Планування розробки нових продуктів.
- Регулярні внутрішні аудити та аналіз для постійного вдосконалення.
- Чіткі вимоги до закупівель та системи комунікації з клієнтами.
- Процедури для роботи з невідповідностями.

ISO/IEC 20000 є міжнародним стандартом для управління IT-сервісами, заснованим на британському стандарті BS 15000. Він складається з двох частин:

- ISO 20000-1: Специфікація для управління сервісами, яка використовується для сертифікації.

- ISO 20000-2: Кодекс практики, що описує найкращі практики для управління сервісами.

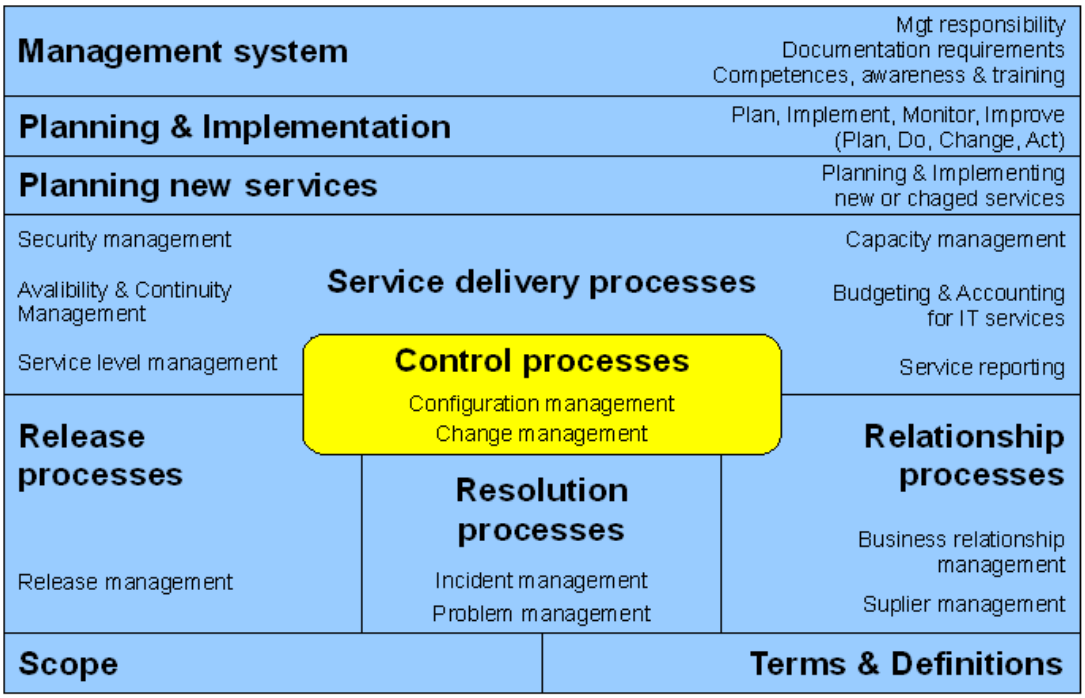

Рис. 2.5. Схема розділів ISO 20000

Відстеження проблем є невід'ємним елементом усіх вищезгаданих стандартів. Воно забезпечується через такі ключові області та процеси:

- Управління конфігурацією (Configuration Management): процес контролю та документування змін у системі.

- Управління вимогами (Requirements Management): встановлення та підтримка домовленостей між замовником та розробником, що є основою для планування та відстеження проєкту. Цей процес включає контроль змін, мінімізацію "розширення обсягу" (scope creep) та вирішення проблем.

- Управління проблемами (Problem Management), управління інцидентами (Incident Management) та управління змінами (Change Management) є окремими випадками управління вимогами.

- Моніторинг та контроль проєкту (Project Monitoring and Control): передбачає відстеження поточного стану проєкту, аналіз статистичних даних (наприклад, витрати часу на функції) для покращення оцінок та розподілу ресурсів.

Для ефективного виконання цих функцій необхідні інструменти, які збирають та аналізують дані, забезпечуючи кількісну оцінку та підтримку прийняття рішень.

2.3. Огляд та дослідження ключових характеристик систем відстеження та документування програмних проектів

Для задоволення вимог, окреслених у попередньому розділі, розроблено різноманітні системи відстеження проблем (Issue Tracking Systems, ITS). У цьому розділі буде представлено огляд найпопулярніших з них. Слід зазначити, що існує безліч інших систем, які часто є клонами проаналізованих нижче (наприклад, Java-порт jTrac для Trac, що базується на Python). П'ять обраних для дослідження систем вважаються еталонними зразками для інших рішень.

2.3.1. Методологія порівняльного аналізу

З огляду на унікальні переваги кожної з обраних ITS, їх порівняння буде проводитися за низкою уніфікованих критеріїв. Ці критерії розширюють та деталізують вимоги, зазначені в попередньому розділі, та включають:

- Інсталяція та початкова конфігурація. Аналіз вимог до середовища функціонування системи та оцінка складності її первинного налаштування для подальшої експлуатації.

- Інтерфейс користувача та крива навчання. Оцінка ергономічності інтерфейсу користувача та швидкості його освоєння. За можливості, буде наведено ілюстрації типових сценаріїв використання, таких як створення та пошук задач (issue).

- Система керування "задачами". Дослідження можливостей налаштування робочого процесу та кастомізації атрибутів "задач". Буде оцінено складність модифікації стандартних робочих процесів, а також можливість та процедура додавання або зміни користувацьких атрибутів.

- Управління проектами. Аналіз спроможності системи керувати декількома проектами в рамках одного екземпляру.

- Аналіз та моніторинг. Огляд наявних інструментів для аналізу даних, включаючи статистичні звіти, графіки та інші ресурси моніторингу.

- Інтеграція з системами керування вихідним кодом. Оцінка можливостей підключення до популярних систем контролю версій (CVS, SVN, Git тощо).

- Доступність та розширюваність. Аналіз можливостей розширення базового функціоналу за допомогою плагінів, а також можливості взаємодії з іншими системами.

- Особливості. Узагальнення додаткових функціональних переваг, які відрізняють аналізовані системи від конкурентів, згідно з офіційними матеріалами розробників або результатами цього дослідження.

В рамках даного дослідження особлива увага приділяється ITS JIRA. Ця система є найбільш ефективним рішенням, і вона була обрана для подальшої інтеграції та використання в середовищі розробки NetBeans IDE.

2.3.2. Дослідження системи Bugzilla

Bugzilla, як одна з найвідоміших систем відстеження проблем з відкритим вихідним кодом, є інструментом, що широко застосовується в масштабних проєктах, зокрема в розробці Mozilla, Eclipse та багатьох дистрибутивах операційної системи Linux. Ця система, що поширюється за ліцензією вільного програмного забезпечення, написана на мові програмування Perl і доступна за посиланням http://www.bugzilla.org/.

Аналіз функціональних та технічних аспектів

1. Інсталяція та конфігурація

Процес інсталяції Bugzilla є багатоетапним і вимагає ручного налаштування. Спершу користувачеві необхідно підготувати робоче середовище, системи керування базами даних (MySQL або PostgreSQL) та веб-сервера (Apache HTTPD). Після розпакування дистрибутиву у вибрану директорію, необхідно вручну редагувати конфігураційні файли як веб-сервера, так і самої програми.

2. Інтерфейс користувача та ергономіка

Інтерфейс користувача Bugzilla орієнтований виключно на функціональність. Він не має привабливого дизайну, однак надає велику кількість функцій на обмеженому просторі. Це може викликати початкові труднощі в освоєнні, але після ознайомлення з логікою системи, робота з нею стає досить ефективною.

3. Система відстеження запитів

Bugzilla використовує повну, але жорстко визначену модель робочого процесу. Незважаючи на відсутність можливості її кастомізації, ця модель є достатньою для потреб більшості організацій. Система також підтримує додавання користувацьких полів до бази даних для збору та пошуку специфічних для організації даних.

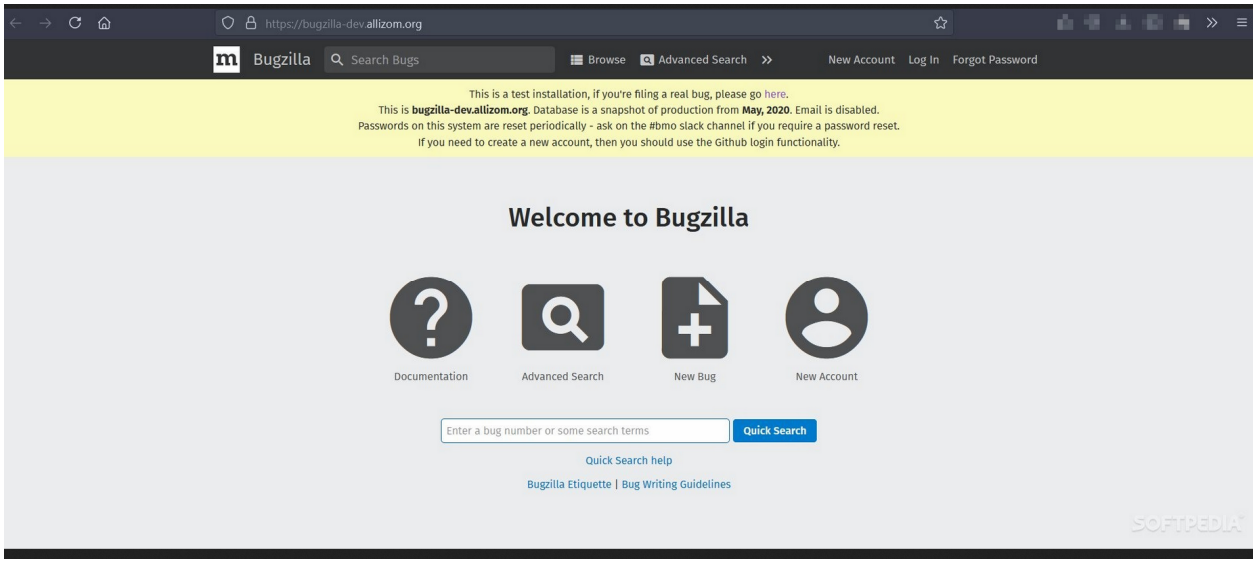

Рис. 2.6. Головна сторінка в Bugzilla

Bugzilla підтримує доволі повну, хоча й жорстко визначену, модель робочого процесу (рис. 2.7). Хоча цю модель не можна налаштувати, вона зазвичай є достатньою для більшості організацій.

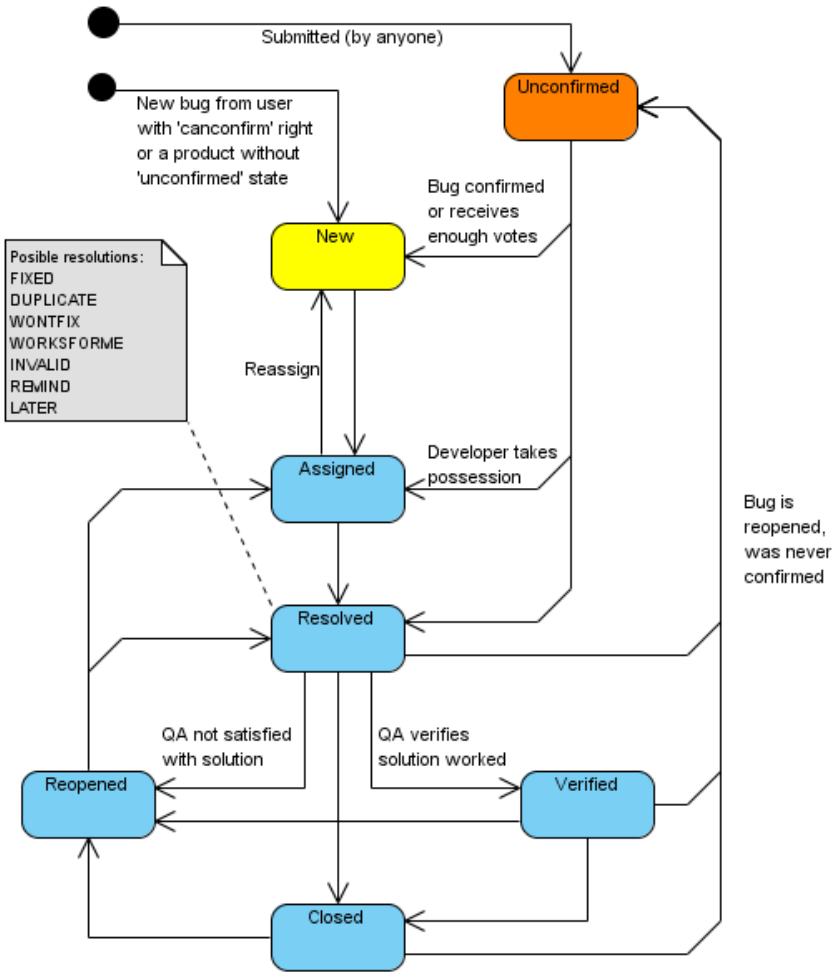

Рис. 2.7. Робочий процес запиту в Bugzilla

Крім того, Bugzilla дозволяє додавати до бази даних користувацькі поля для збору та пошуку даних, які є унікальними для конкретної організації.

4. Управління проєктами

Система дозволяє адміністраторам створювати та модифікувати так звані «дерева класифікації» продуктів. Це забезпечує можливість ієрархічної структуризації, де верхній рівень може відповідати назвам проєктів, а наступні — назвам продуктів, що перебувають у розробці, без необхідності застосування нестандартних рішень.

Bugzilla надає розвинену систему звітності, хоча її інтерфейс може здатися складним для початківців. Система дозволяє генерувати двовимірні табличні звіти, де осі X та Y можуть бути призначені будь-яким двом полям. Наприклад, можна візуалізувати кількість запитів за їхнім статусом у кожному продукті. Ці дані можуть бути представлені у вигляді лінійних, стовпчастих або кругових діаграм. Також можлива тривимірна візуалізація. Звіти експортуються у формат CSV для подальшого аналізу. Система також включає функцію побудови графіків, що відстежують зміни у базі даних з часом.

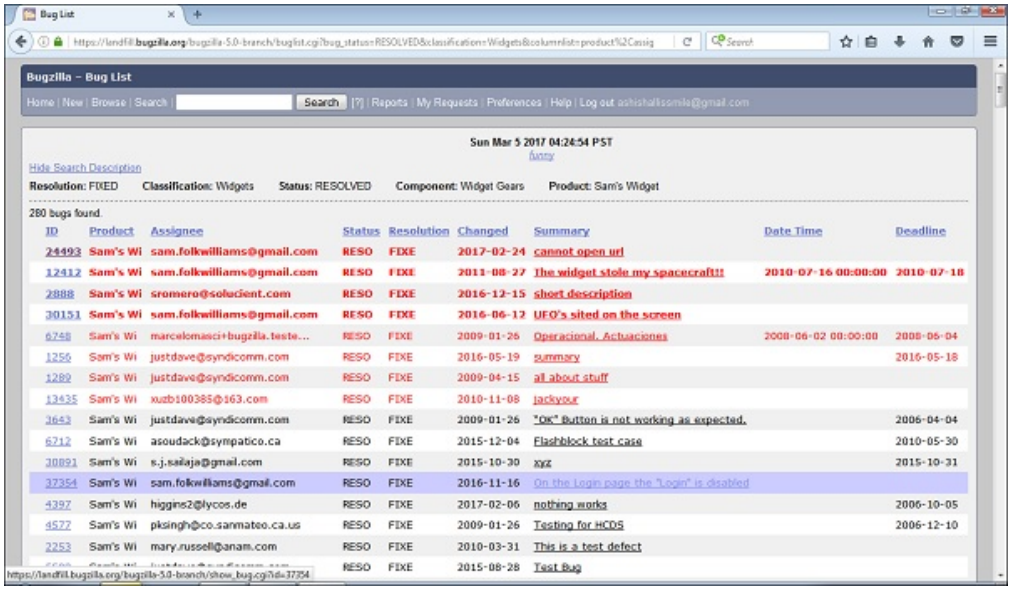

Рис. 2.8. Bug list в Bugzilla

Інтеграція Bugzilla з системами керування вихідним кодом (такими як CVS, Subversion, Bonsai, Teamtrack Performance, Tinderbox) реалізується через систему плагінів, доступних на сторінці додатків.

Доступ до Bugzilla та її модифікація можливі через XML-RPC веб-сервіси, що дозволяє створювати зовнішні інструменти для взаємодії з системою. Крім того, існує велика кількість плагінів та розширень.

Характерні особливості:

- Налаштовуваний інтерфейс: Інтерфейс користувача генерується за допомогою HTML-шаблонів, що полегшує його налаштування.

- Локалізація: Підтримується велика кількість мов.

- Сповіщення: Система автоматично надсилає користувачам електронні листи про нові або оновлені запити.

- Відстеження часу: Bugzilla включає базову функцію відстеження часу, витраченого на виконання завдань.

- Система голосування: Користувачі можуть голосувати за пріоритетність запитів або функцій.

Bugzilla є надійним і перевіреним часом рішенням, що підходить для управління великими проєктами та великою кількістю користувачів. Його функціональність робочого процесу зазвичай задовольняє потреби більшості організацій. Однак, слід враховувати, що Bugzilla є відносно складною в інсталяції та супроводі, а її інтерфейс користувача не відрізняється високим рівнем ергономіки. Незважаючи на це, функціонал звітності є повноцінним, хоч і не надто дружнім до користувача.

2.3.3. Платформа CodeBeamer

CodeBeamer є не просто системою відстеження проблем, а комплексною платформою для спільної розробки програмного забезпечення з інтегрованим управлінням життєвим циклом додатків (ALM). Ця пропрієтарна система, що функціонує на базі мови програмування Java, включає в себе широкий спектр сервісів: управління документацією, Wiki, форум, онлайн-чат та, безумовно, систему відстеження проблем, яка тісно інтегрована з системами контролю версій (наприклад, SVN).

Система є платною для комерційного використання, але доступна безкоштовно для освітніх та оцінювальних цілей. Прикладом її застосування є хостинг спільноти JavaForge.com, де розміщено численні проєкти з відкритим кодом.

Аналіз функціональних та технічних аспектів

1. Інсталяція та конфігурація

CodeBeamer реалізовано як Java EE-додаток, який працює на сервері сервлетів Apache Tomcat. Система постачається у вигляді бінарного інсталятора для операційних систем Windows та Linux, який вже включає файли інсталяції Tomcat. Також можлива інсталяція на існуючий екземпляр Tomcat. Після завершення інсталяції необхідно налаштувати з'єднання з базою даних та інші параметри, що зберігаються у відповідному файлі властивостей.

2. Інтерфейс користувача

Інтерфейс користувача CodeBeamer є більш структурованим у порівнянні з Bugzilla, проте не може вважатися повністю інтуїтивним. Наприклад, для додавання нових запитів на проблеми необхідно виконувати багато кроків. Хоча пошук функцій не є складним, їх розміщення потребує оптимізації, щоб найчастіше використовувані опції були більш доступними, що покращило б загальну ергономіку.

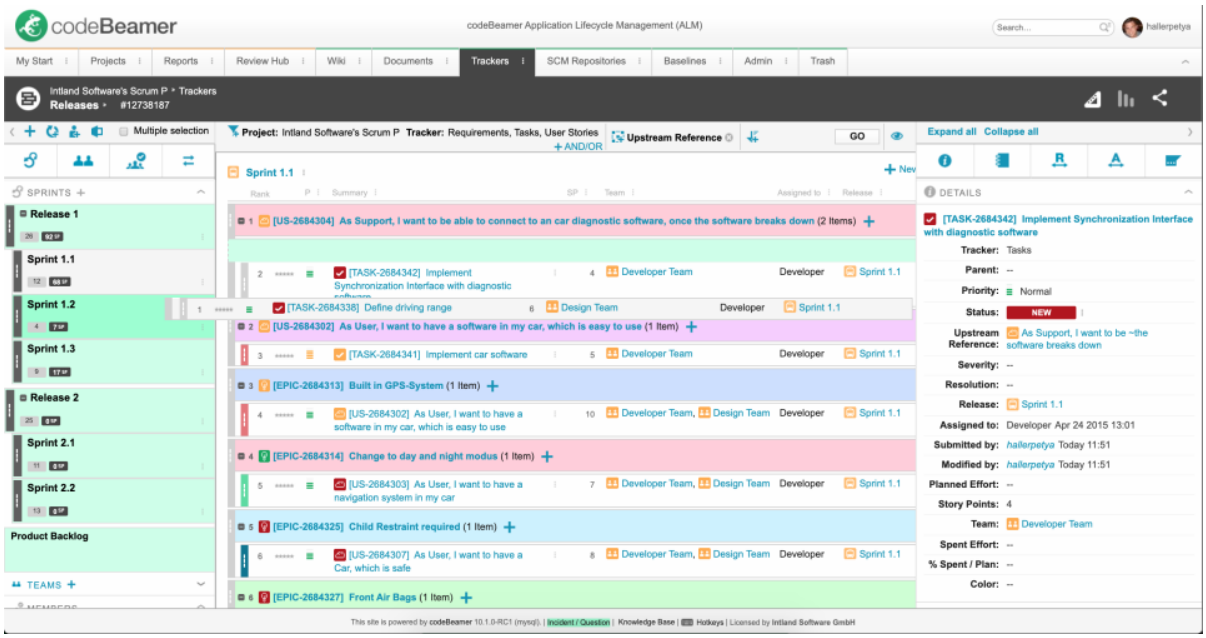

Рис. 2.8. Документування проекту засобами CodeBeamer

3. Система відстеження запитів

CodeBeamer пропонує стандартний робочий процес, який може бути налаштований через інтерфейс користувача. Детальну інформацію про

конфігурацію можна знайти на відповідній сторінці Wiki-системи CodeBeamer «Workflows».

Жовтий вузол: Початковий стан, в якому з'являється новий елемент трекера.

Сині вузли: Звичайні стани, між якими елемент переміщується протягом більшої частини свого життєвого циклу.

Помаранчеві вузли: Використовуються для міграції зі застарілих трекерів, оскільки вони дозволяють лише вихідні переходи.

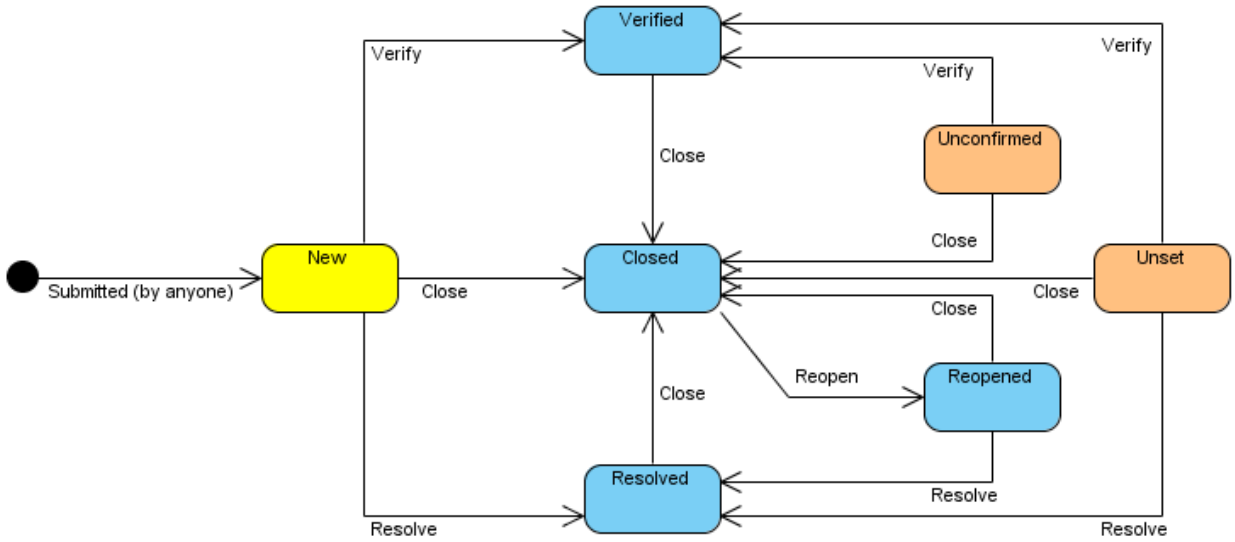

Рис. 2.9. Стандартний робочий процес запиту в CodeBeamer

Крім того, система підтримує можливість додавання користувацьких полів до запитів.

4. Управління проєктами

Система за замовчуванням підтримує управління декількома проєктами, які можуть бути об'єднані в так звані «робочі набори» (work sets) для спрощення адміністрування. Кожен проєкт може мати індивідуальні налаштування, включаючи права доступу та робочий процес. Налаштування застосовуються в ієрархічному порядку: спочатку налаштування проєкту,

потім робочого набору, і в останню чергу — глобальні налаштування системи.

CodeBeamer має гнучку систему звітності, яка за замовчуванням включає:

- Звіти про завдання та помилки з візуалізацією у вигляді діаграм.
- Діаграми Ганта.
- Звіти про якість та динаміку змін вихідного коду.
- Можливість створення користувацьких звітів, включаючи автоматизоване формування нотаток до випусків.

Звіти формуються на основі даних з одного або кількох трекерів та проєктів, а результати можуть бути експортовані у формати Excel, PDF, CSV, XML та Wiki.

CodeBeamer має вбудовані інтерфейси для CVS, Subversion, PVCS, SourceSafe, CM Synergy та ClearCase, з можливістю розширення для підтримки інших систем.

Система надає API веб-сервісів, що базується на бінарному веб-сервісі Caucho Hessian, що дозволяє інтеграцію з іншими додатками.

CodeBeamer, як платформа ALM, надає розширений функціонал, що виходить за рамки стандартного відстеження проблем:

- Вбудована система управління документами.
- Внутрішня Wiki-система для інтеграції всіх артефактів проєкту.
- Підтримка безперервної інтеграції та періодичних збірок.
- Вбудовані дискусійні форуми.
- Розширена платформа безпеки.
- Інтеграція з електронною поштою.
- Надання статистики та метрик забезпечення якості.

CodeBeamer повною мірою відповідає своєму позиціонуванню як платформа для спільної розробки, надаючи широкий спектр інструментів для управління та розробки. Його перевагами є простота конфігурації та

підтримки. Головним недоліком є неоптимальна ергономіка інтерфейсу користувача, де найбільш затребувані функції не є очевидними.

2.3.4. Інструмент Rational ClearQuest

Rational ClearQuest — це гнучкий інструмент для відстеження вимог і змін у процесі розробки програмного забезпечення. Він належить до портфоліо програмних рішень IBM Rational, що охоплює весь життєвий цикл розробки програмного забезпечення, від збору вимог і проектування до кодування та тестування, а також управління змінами та забезпечення якості. Концепція Rational Unified Process (RUP), яка є частиною цього портфоліо, стала еталонною моделлю для управління масштабними проєктами.

ClearQuest вирізняється серед інших систем тим, що він розроблений як "товстий клієнт", що безпосередньо підключається до баз даних, де зберігаються схеми та дані. Основний клієнт доступний для платформи Microsoft Windows, а для інших платформ пропонуються клієнти на базі Eclipse, плагіни або веб-доступ. Система ліцензується як пропрієтарне програмне забезпечення.

Аналіз функціональних та технічних аспектів

1. Інсталяція та конфігурація

Встановлення ClearQuest здійснюється за допомогою бінарного інсталятора на основі Java, який підтримує як графічний, так і текстовий режими. Спеціальна попередня підготовка не потрібна, оскільки інсталяційний пакет містить усі необхідні компоненти.

2. Інтерфейс користувача

Інтерфейс користувача ClearQuest заснований на моделі настільного клієнта, що може бути незвичним для тих, хто звик до веб-систем. Використання програми нагадує роботу зі звичайними настільними додатками (наприклад, текстовим редактором). Наприклад, для створення нового запиту необхідно використовувати меню New -> Defect, а контекстні дії доступні через праву кнопку миші. Незважаючи на це, користувачі та

спільнота часто критикують систему за високе споживання ресурсів, низьку продуктивність, а також посередній дизайн та зручність використання.

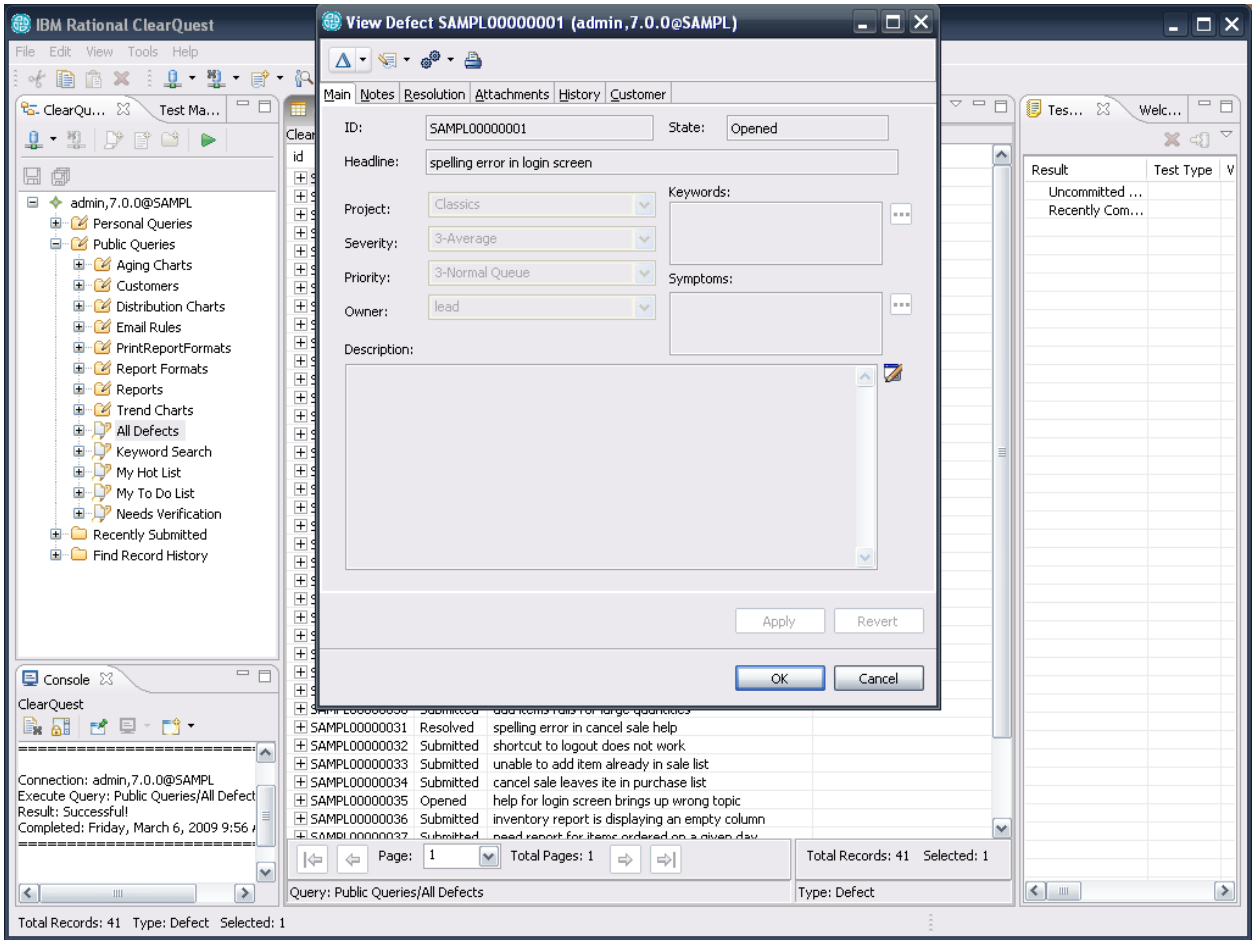

Рис. 2.10. Перегляд деталізації помилки

Крім нативного клієнта для Windows, існує клієнт на базі платформи Eclipse та веб-клієнт. Всі вони мають схожий зовнішній вигляд, що полегшує перехід між ними. Веб-версія, за винятком можливості створення запитів, надає повний функціонал настільного клієнта.

3. Система запитів

Стандартний робочий процес для запитів відповідає рекомендаціям методології RUP, як показано на діаграмі нижче (рис. 2.11).

Модифікація робочого процесу здійснюється за допомогою програми ClearQuest Designer, що робить цей процес гнучким і зручним. Крім того,

система підтримує скрипти на Perl, що дозволяє реалізовувати складніші конфігурації та автоматизувати дії.

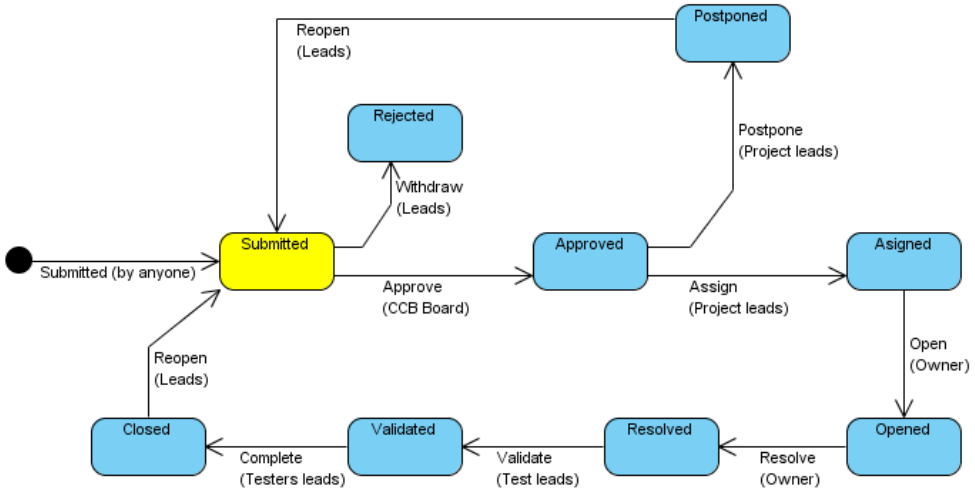

Рис. 2.11. Стандартний робочий процес запиту в IBM Rational ClearQuest

4. Управління проєктами

ClearQuest спроектований для управління кількома проєктами, дозволяючи для кожного з них створювати окрему базу даних з індивідуальними налаштуваннями, робочими процесами та правами доступу.

Аналітичні можливості ClearQuest є винятковими. Завдяки методології RUP, система пропонує велику кількість попередньо визначених шаблонів звітів і графіків, які легко налаштовуються. Наявність покрокових "майстрів" дозволяє навіть початківцям створювати власні запити та звіти.

За замовчуванням ClearQuest тісно інтегрований з IBM Rational ClearCase — інструментом для контролю версій. Інтеграція з іншими системами можлива через плагіни сторонніх розробників.

IBM надає Rational Team API, уніфікований клієнтський API на основі Java для взаємодії з ClearCase та ClearQuest. Крім того, інтерфейс користувача ClearQuest заснований на платформі Eclipse RPC, що забезпечує

високу розширюваність. Скрипти на Perl дозволяють здійснювати складніші конфігурації та автоматизацію.

Характерні особливості:

- Тісна інтеграція з іншими продуктами сімейства Rational.

- Автоматичні сповіщення про події та зміни через електронну пошту.

- Можливість створювати запити, генерувати звіти та аналізувати дані у вигляді графіків.

- Реагування на події та зміни за допомогою "хуків" (hooks).

- Створення та виконання планів і сценаріїв тестування через інтегрований TestManager.

- Інтеграція з інструментами управління проєктами, такими як MS Project та Rational Portfolio Manager.

- Підтримка інтеграції з Microsoft Visual Studio .NET.

Rational ClearQuest — це потужна система відстеження проблем з відмінними можливостями аналізу та звітності. Її справжня цінність розкривається при інтеграції в середовище розробки, що використовує інші інструменти сімейства Rational, де вона стає надзвичайно ефективним компонентом для управління життєвим циклом програмного забезпечення.

2.3.5. Система відстеження задач проектів Trac

Trac — це система відстеження проблем з відкритим вихідним кодом, розроблена на мові програмування Python і ліцензована як вільне програмне забезпечення. Ця система, що користується значною популярністю, особливо після Bugzilla, вирізняється інтуїтивно зрозумілим інтерфейсом, тісною інтеграцією з системами контролю версій (переважно Subversion) та функціоналом Wiki. Завдяки цим особливостям, Trac широко використовується в спільноті відкритого коду, а також в невеликих командах і компаніях. Існують платформи, як-от Assembla.com, що пропонують безкоштовний хостинг Trac разом із Subversion.

Аналіз функціональних та технічних аспектів

1. Інсталяція та конфігурація

Процес встановлення Trac є багатоетапним. Користувачеві необхідно вручну підготувати середовище, що включає встановлення Python, Genshi (мова шаблонів для Python), SetupTools, системи керування базами даних (Sqlite, MySQL або PostgreSQL), веб-сервера з підтримкою модуля Python (найчастіше Apache HTTPD з mod_python), системи контролю версій (Subversion або іншої підтримуваної системи).

Після цього необхідно розпакувати завантажений zip-пакет та запустити інсталятор. Конфігурація системи здійснюється через утиліту командного рядка trac-admin. Для спрощення адміністративних завдань можна встановити плагін webadmin.

2. Інтерфейс користувача

Інтерфейс користувача Trac є прикладом високої зручності та мінімалізму. Користувач може виконати будь-яку дію в межах трьох кліків. Відсутність надмірної кількості кнопок та опцій дозволяє швидко освоїти систему навіть новачкам. Оскільки вся система побудована на концепції Wiki, вона забезпечує просту навігацію та можливість створювати гіперпосилання між базою даних проблем, історією ревізій та вмістом Wiki.

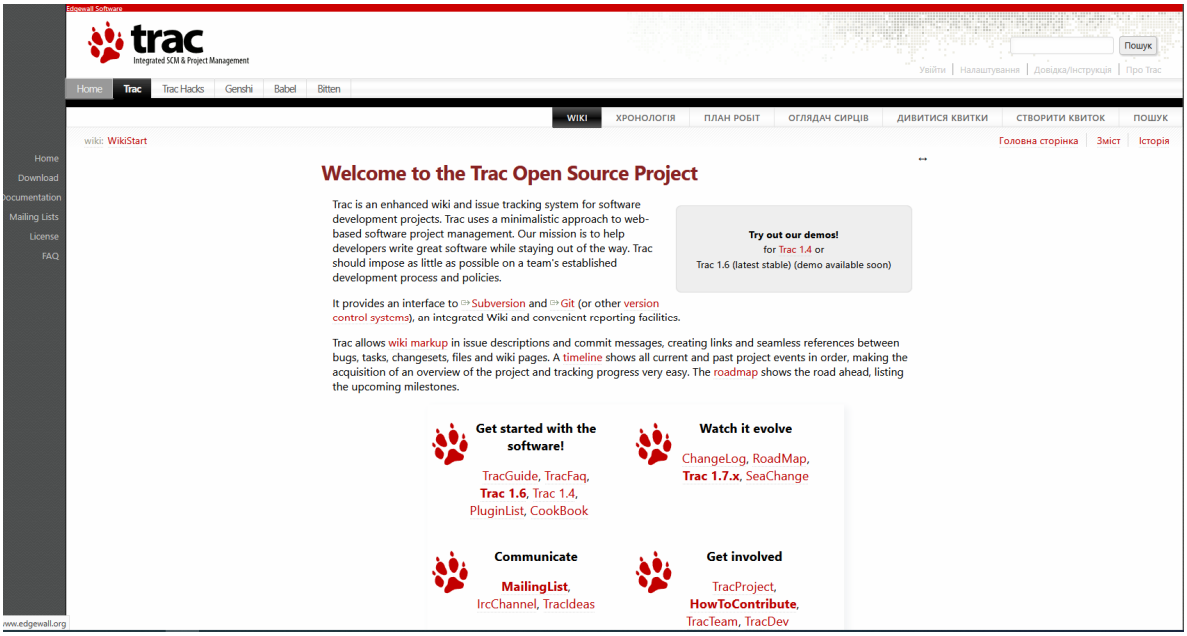

Рис. 2.12. Головна сторінка Trac

3. Система запитів

Стандартний робочий процес у Trac, який конфігурується у файлі trac.ini, є базовим (рис. 2.13).

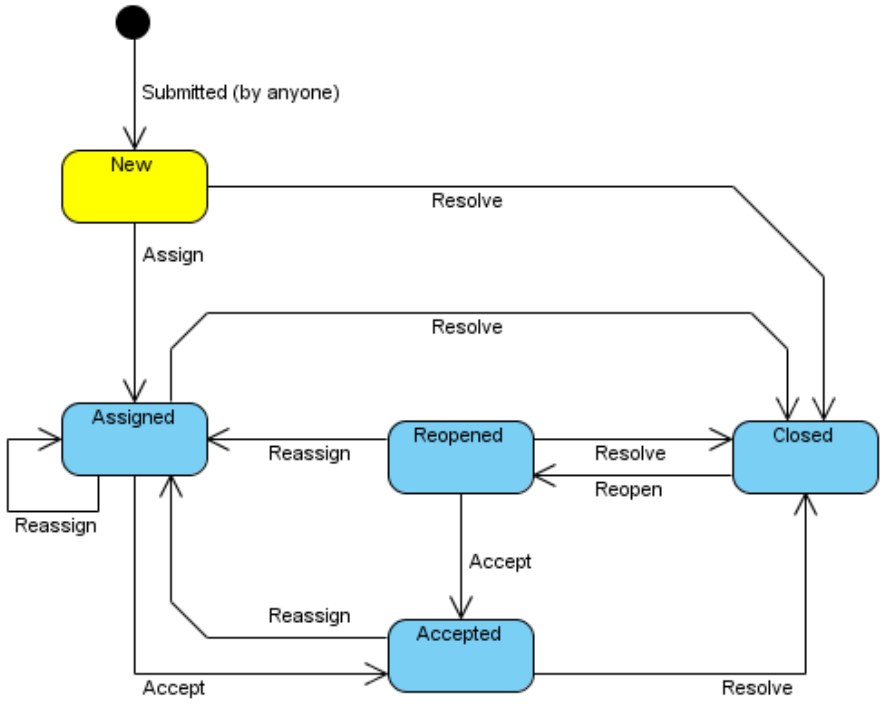

Рис. 2.13. Стандартний робочий процес запиту в Trac

Ця модель може бути модифікована шляхом редагування файлу trac.ini або заміни його іншими, заздалегідь підготовленими конфігураціями. Trac також дозволяє додавати користувацькі поля, але для запитів, створених до їх визначення, необхідно оновлювати базу даних вручну.

4. Управління проєктами

Trac не підтримує управління декількома проєктами в рамках одного екземпляру. Єдиним ефективним рішенням є створення окремого екземпляру для кожного проєкту, що, однак, унеможливлює централізований моніторинг та управління завданнями, спільними для різних проєктів.

Базова установка Trac надає лише обмежені можливості моніторингу. Користувач може відстежувати розподіл запитів за релізами та їх виконання.

Розширені функції моніторингу доступні через плагіни, які дозволяють використовувати додаткові користувацькі поля.

Trac має вбудовану підтримку систем контролю версій Subversion, Git, Mercurial, Bazaar та Darcs. Інтеграція з іншими системами можлива за допомогою плагінів.

Trac не має вбудованої підтримки RPC, але існує XML-RPC плагін для версій 0.10 і вище. Його встановлення потребує додаткових зусиль, і, оскільки API плагіна все ще знаходиться в стадії розробки, він може бути нестабільним. Це ускладнює розробку сторонніх додатків. Наприклад, плагін Mylyn для Eclipse обходить цю проблему шляхом аналізу HTML-коду інтерфейсу Trac.

Завдяки популярності в спільноті відкритого коду, існує велика кількість плагінів та розширень, що інтегрують Trac з іншими додатками та розширюють його функціонал.

Trac не позиціонує себе як складна платформа для управління розробкою. Його основна мета — ефективне відстеження проблем, що він виконує дуже якісно. Хоча базова версія не надає багато додаткового функціоналу, він може бути значно розширений за допомогою численних плагінів. Загалом, Trac є оптимальним інструментом для середовищ, де розробка зосереджена на одному проєкті.

2.3.6. Система управління і документування проектів JIRA

JIRA, розроблена компанією Atlassian, є комплексною системою управління проєктами та відстеженням проблем (issue/bug management system). Вона відома своєю потужністю, що не заважає їй мати чистий та зручний інтерфейс користувача. Система також вирізняється активною спільнотою розробників, яка створює численні розширення та плагіни, а також великою кількістю навчальних матеріалів та документації. Ще однією значною перевагою є легка інтеграція з іншими продуктами Atlassian, зокрема з корпоративною Wiki-системою Confluence.

Хоча JIRA є пропрієтарним програмним забезпеченням, Atlassian пропонує його безкоштовно для некомерційних та освітніх організацій, а також для проєктів з відкритим кодом. Для комерційного використання діє ліцензійна політика, що передбачає знижки для академічних установ. Крім того, з версії JIRA 3.13 доступна безкоштовна персональна ліцензія для некомерційних цілей, що дозволяє використовувати систему до трьох користувачів.

Аналіз функціональних та технічних аспектів

1. Інсталяція та конфігурація

JIRA постачається у вигляді бінарного інсталятора, що підтримує як графічний, так і текстовий режими встановлення, що робить її зручною для різних середовищ. Жодна попередня підготовка не вимагається, оскільки інсталяційний файл включає всі необхідні компоненти. Atlassian також надає послуги хостингу, що дозволяє користувачам обійтися без локальної інсталяції.

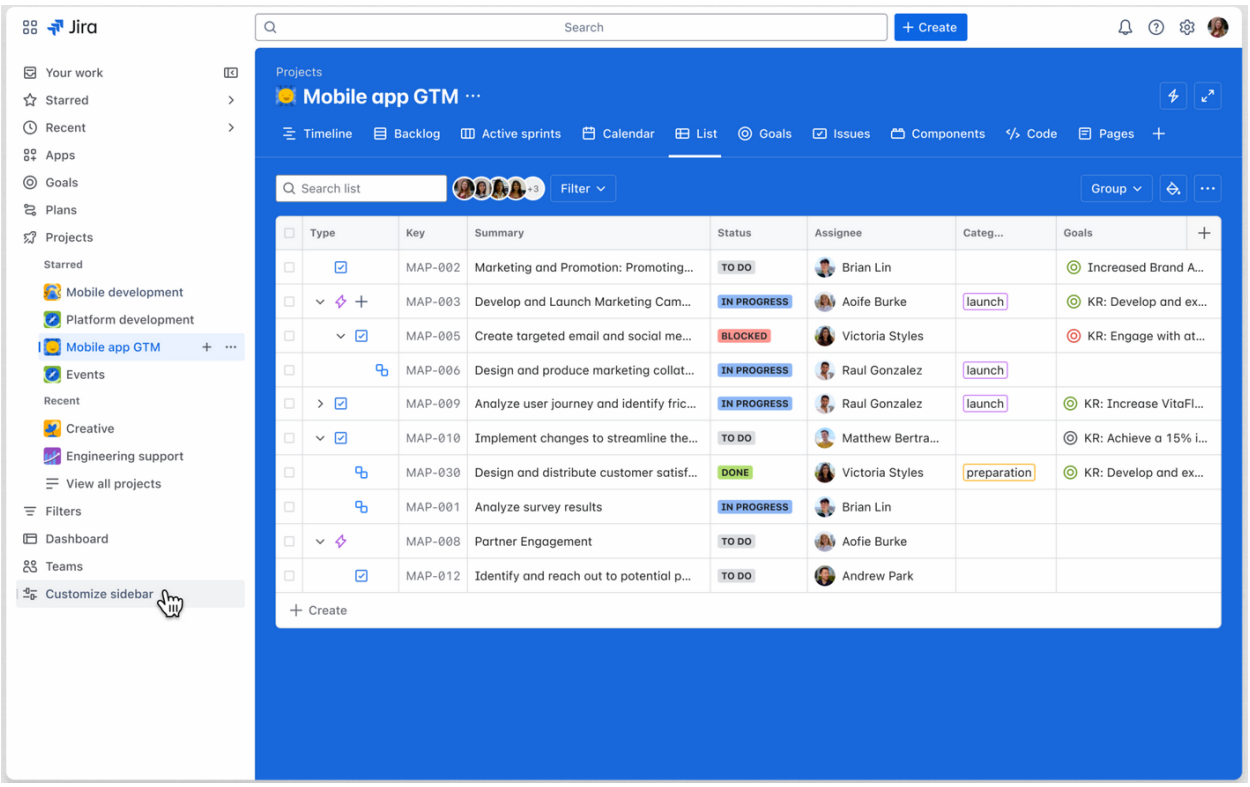

Рис. 2.14. Статус задач в JIRA

2. Інтерфейс користувача

Інтерфейс користувача JIRA є однією з її ключових переваг. Незважаючи на складність системи, інтерфейс залишається чистим, інтуїтивно зрозумілим і гнучким, дозволяючи кожному користувачеві налаштувати його під власні потреби. Деякі типові завдання, що вимагають виконання багатьох кроків, можуть бути спрощені за допомогою численних плагінів, як-от GreenHopper, що дозволяє групове переміщення запитів між релізами. Таким чином, саме розширюваність і налаштування перетворюють JIRA на високоякісний продукт.

3. Система запитів

JIRA надає стандартний робочий процес (див. діаграму нижче), який може бути легко налаштований через інтерфейс користувача.

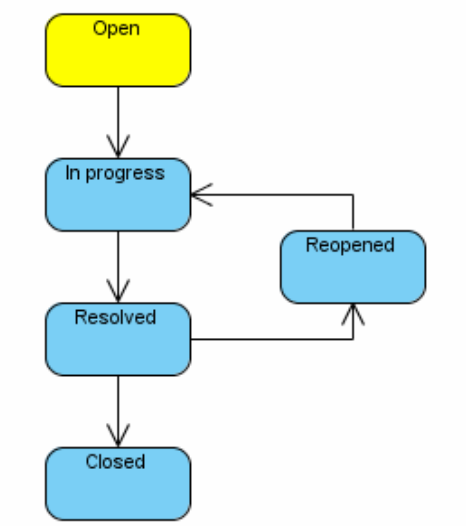

Рис. 2.15. Стандартний робочий процес запиту в JIRA

Гнучка архітектура робочого процесу JIRA дозволяє створювати індивідуальні процеси для різних відділів, проєктів і навіть типів проблем. Кожен робочий процес може містити стільки кроків, скільки необхідно. Система дозволяє створювати та керувати користувацькими полями на системному рівні, а також на рівні проєктів і типів проблем. JIRA постачається з понад 20 типами користувацьких полів і надає можливість

розробляти власні типи. Це дозволяє адаптувати JIRA для управління не лише розробкою програмного забезпечення, але й іншими процесами, наприклад, "управлінням життєвим циклом" (life management).

4. Управління проєктами

JIRA розроблена для управління декількома проєктами. Проєкти можуть бути згруповані за категоріями, і кожен проєкт може мати індивідуальні налаштування, включаючи робочий процес, фільтри, метрики.

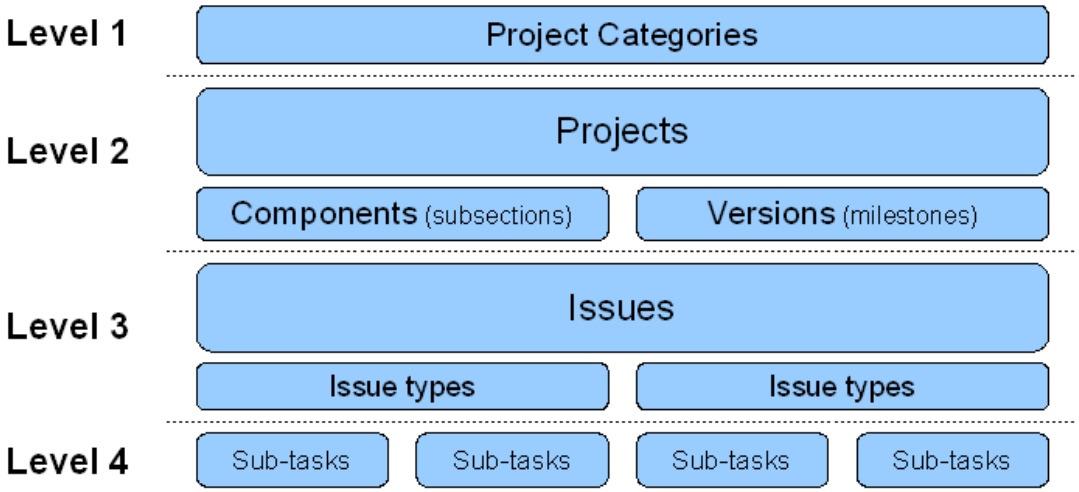

Рис. 2.16. Концепції JIRA

Система моніторингу та звітності в JIRA базується на фільтрах, які є збереженими пошуковими запитами. Користувач може визначити фільтри для управління запитами та використовувати їх як джерело даних для інструментів моніторингу. Наприклад, можна створити фільтр, що відображає всі призначені користувачеві запити для поточного релізу, і візуалізувати їх у вигляді кругової діаграми на інформаційній панелі.

Інформаційна панель є головною сторінкою і може бути налаштована за допомогою портлетів — підключаємих компонентів. Існує безліч готових портлетів для моніторингу та звітності, і користувачі можуть створювати власні на основі фільтрів.

JIRA підтримує підключення до різних систем керування вихідним кодом через плагіни. Стандартні конектори, такі як для Subversion,

постачаються разом із системою. Atlassian також пропонує продукт FishEye, який забезпечує веб-інтерфейс для моніторингу репозиторіїв та повністю інтегрується з JIRA.

JIRA 3.0 і вище має плагін RPC, що дозволяє віддалений доступ через XML-RPC та SOAP. Також можливе розширення функціоналу за допомогою Java API. Активна спільнота розробників постійно створює нові розширення та плагіни, доступні на офіційному сайті Atlassian.

Характерні особливості:

- Кожен користувач може адаптувати інтерфейс під свої потреби, що робить JIRA придатною навіть для служби підтримки.

- Система пропонує корпоративний рівень безпеки, дозволяючи встановлювати права доступу не лише на рівні проєктів, але й для окремих запитів та коментарів.

- Автоматизація. Запити можуть автоматично створюватися з електронних листів.

- Підзадачі. Система дозволяє розділяти запити на підзадачі.

- Вбудована підтримка обліку часу, витраченого на виконання завдань.

- Інтеграція. Тісна інтеграція з іншими продуктами Atlassian.

Отже, JIRA є лідером у сфері відстеження запитів і моніторингу проєктів. Вона пропонує гнучку політику ліцензування, а завдяки підтримці спільноти її функціонал постійно розширюється. Система є простою в освоєнні, при цьому надає широкі можливості для налаштування, що робить її потужним інструментом для управління різноманітними процесами.

2.4. Дослідження основних плагінів та доповнень системи JIRA

2.4.1. Плагін GreenHopper для JIRA

Незважаючи на високу функціональність JIRA у сфері моделювання та звітності, її стандартний інтерфейс не є оптимальним для управління проєктами, особливо з точки зору проєктних менеджерів. Інтерфейс за

замовчуванням орієнтований на пошук та перегляд запитів, тоді як управління та редагування є менш інтуїтивними. Саме цю проблему вирішував плагін GreenHopper, розроблений для підвищення інтерактивності та зручності інтерфейсу для розробників і менеджерів проєктів.

Використовуючи метафору "дошки та карток", GreenHopper надавав три додаткові візуалізації:

- Планувальна дошка
- Дошка завдань
- Дошка діаграм

Раніше, GreenHopper був окремим комерційним продуктом, що інтегрувався в JIRA. Однак, на сьогодні його функціонал повністю інтегрований у Jira Software, і він більше не існує як окремий плагін.

Планувальна дошка надавала користувачеві загальний огляд усіх версій та компонентів. Вона була поділена на дві основні панелі.

"Навігатор проблем" - основна панель, що відображала список проблем у вибраному контексті. Завдяки функції перетягування (drag-and-drop), користувач міг легко впорядковувати та планувати завдання.

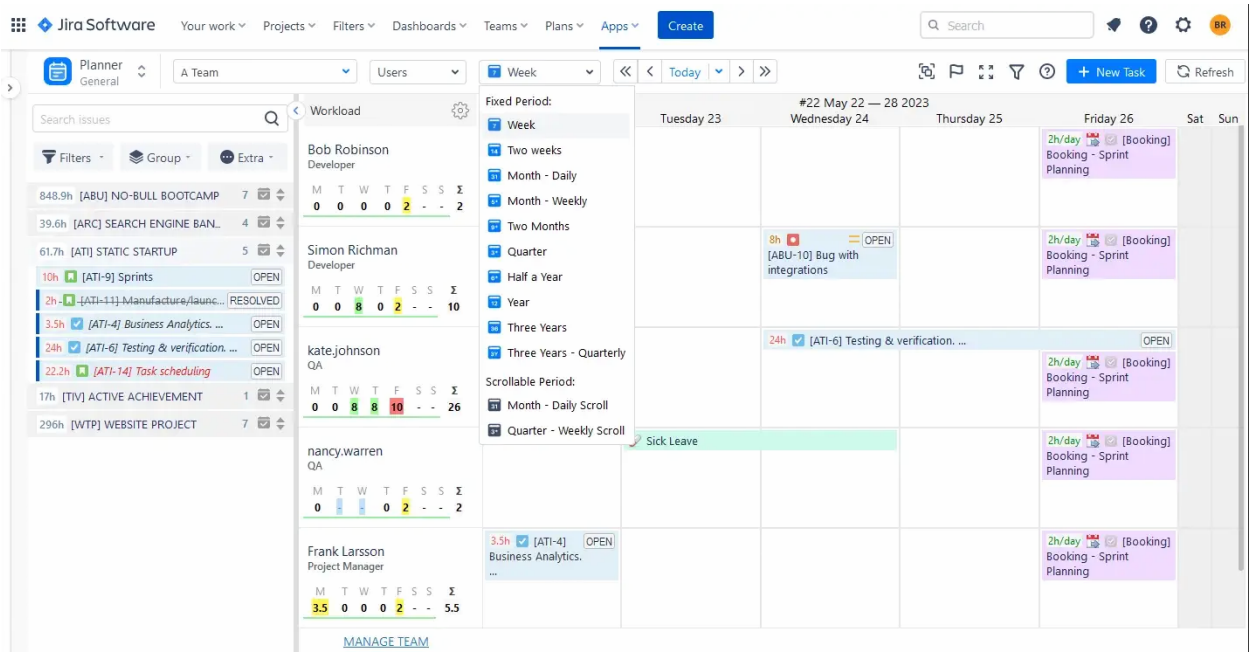

Рис. 2.17. Планувальна дошка JIRA

Огляд об'єднував усі незаплановані версії та надавав можливість для ранжування.

Управління проблемами між версіями та компонентами було значно спрощене завдяки можливості перетягування. Кожен контейнер версії або компонента містив важливу інформацію, таку як індикатор прогресу, статистики (дати випуску, кількість проблем за типом, кількість вирішених/невирішених проблем) та опцію "Випустити версію". Якщо залишалися невирішені проблеми, GreenHopper пропонував їх перенести в іншу версію або проігнорувати.

Таким чином, планувальна дошка надавала:

- Візуальний огляд усіх невипущених версій та компонентів.

- Організацію у вигляді карток та списків.

- Багаторівневе ієрархічне планування.

- Можливість впорядкування та планування проблем за допомогою перетягування.

2.4.2. Atlassian Bamboo

Atlassian Bamboo — це комерційний сервер безперервної інтеграції (Continuous Integration, CI) і безперервної доставки (Continuous Delivery, CD), який автоматизує процес випуску програмного забезпечення від створення коду до його розгортання.

Ця система дозволяє командам розробників автоматично виконувати збірку, тестування та випуск свого коду. Bamboo тісно інтегрується з JIRA, що надає можливість відстежувати стан кожного випуску та пов'язувати його з відповідними проблемами.

Ключові функціональні можливості:

1. Bamboo забезпечує автоматизацію всього життєвого циклу розробки, від збірки та тестування до розгортання. Це сприяє підвищенню ефективності та зменшенню кількості ручних операцій.

2. Тісна інтеграція з JIRA дозволяє централізовано відстежувати стан збірок і випусків, а також пов'язувати їх із конкретними запитами та проблемами. Це забезпечує прозорість процесу розробки.

3. Підтримка паралельного виконання збірок дозволяє значно прискорити процес тестування та інтеграції, що особливо важливо для великих проєктів.

4. Система підтримує різні мови програмування та технології, що робить її універсальним інструментом для команд з різноманітними технологічними стеками.

5. Bamboo надає детальну інформацію про статус збірки та випуску, що дозволяє командам ефективно контролювати процес розробки та швидко реагувати на виявлені проблеми.

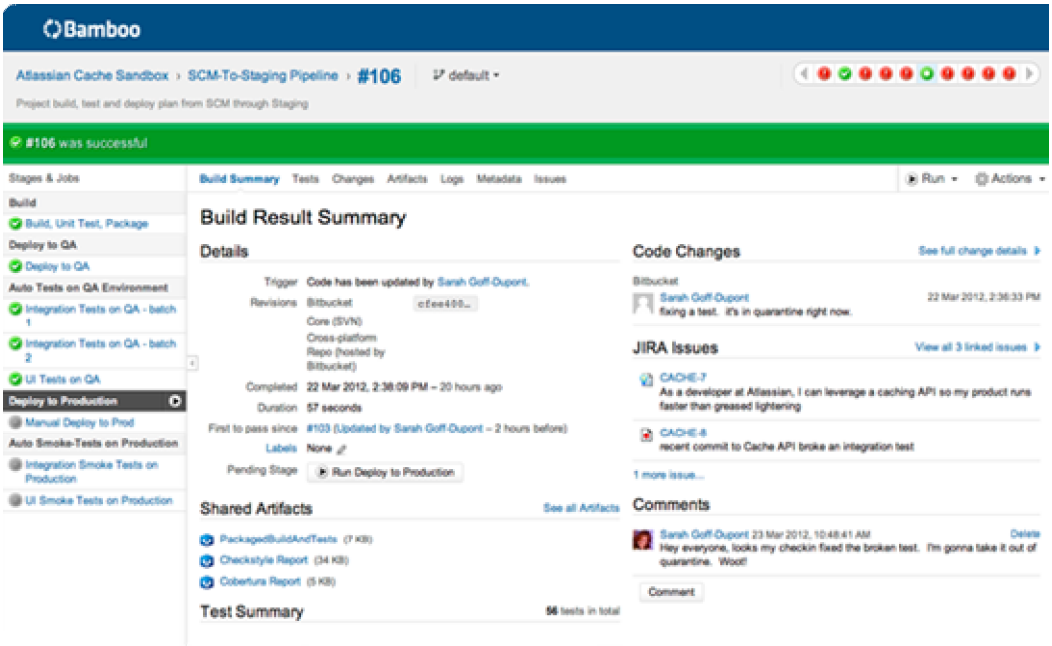

Рис. 2.18. Atlassian Bamboo

Bamboo легко інтегрується з JIRA, дозволяючи командам розробників переглядати стан збірки та випуску безпосередньо в інтерфейсі JIRA. Ця функція сприяє швидкому виявленню та вирішенню проблем, пов'язаних з кодом, що підвищує загальну продуктивність та якість програмного забезпечення.

2.4.3. Atlassian Crucible

Atlassian Crucible — це комерційний інструмент для проведення оглядів коду (code reviews), призначений для спільної роботи та обговорення змін у вихідному коді. Ця система тісно інтегрується з JIRA та низкою популярних систем контролю версій, включаючи Git, Subversion, Mercurial, CVS та Perforce.

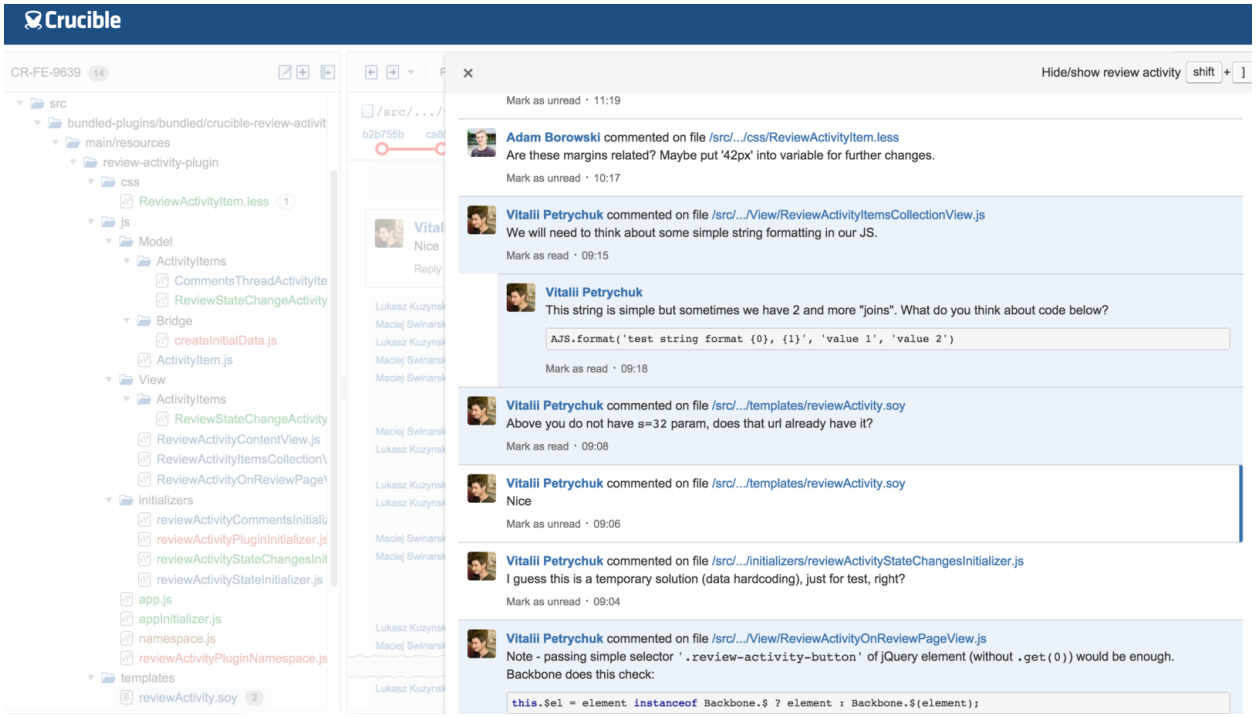

Рис. 2.19. Atlassian Crucible

Ключові функціональні можливості

1. Crucible надає спеціалізоване середовище, що дозволяє командам розробників проводити структуровані огляди коду та ефективно обговорювати внесені зміни.

2. Користувачі можуть додавати коментарі безпосередньо до конкретних рядків коду. Ця функція сприяє точному обговоренню та швидкому вирішенню питань.

3. Завдяки інтеграції з JIRA, огляди коду можуть бути пов'язані з відповідними проблемами або завданнями, що забезпечує повну відстежуваність та прозорість процесу розробки.

4. Система надає детальні звіти про стан огляду коду, його результати, а також статистику щодо виявлених дефектів.

Crucible дозволяє розробникам створювати огляди коду, додавати коментарі та обговорювати зміни. Це сприяє підвищенню якості коду та допомагає виявити потенційні проблеми на ранніх стадіях розробки, перш ніж вони будуть інтегровані в основну кодову базу.

2.5. Порівняльний аналіз систем відстеження та документування програмних проектів

У цьому розділі представлено порівняльний аналіз п'яти популярних систем відстеження проблем (Bugzilla, CodeBeamer, Rational ClearQuest, Trac та JIRA), що базується на низці ключових критеріїв. Порівняння охоплює аспекти ліцензування, технологічної реалізації, складності інсталяції, ергономіки інтерфейсу користувача, гнучкості робочого процесу, можливостей управління проєктами, аналітичного інструментарію, інтеграції з системами керування версіями, а також доступності та розширюваності системи.

Таблиця 2.1.

Порівняльна характеристика систем

Критерій	**Bugzilla**	**CodeBeamer**	**Rational ClearQuest**	**Trac**	**JIRA**
Ліцензія	Відкрите ПЗ	Власницька	Власницька	Відкрите ПЗ	Власницька
Мова реалізації	Perl	Java	Perl / Visual Basic	Python	Java
Складність інсталяції	Висока	Середня	Низька	Висока	Низька
Інтерфейс користувача	Функціональний	Структурований	Складний	Інтуїтивний	Зручний
Налаштування робочого процесу	Обмежене	Гнучке	Гнучке	Обмежене	Гнучке

Управління проєктами	Багатопроєктне	Багатопроєктне	Багатопроєктне	Однопроєктне	Багатопроєктне
Аналіз та моніторинг	Просунутий	Просунутий	Просунутий	Базовий	Просунутий
Інтеграція з SCM	Плагіни	Вбудована	ClearCase	Вбудована	Плагіни
Доступність	XML-RPC	Веб-сервіси	Rational Team API	Обмежена	XML-RPC, SOAP
Розширюваність	Плагіни	Плагіни	Плагіни	Плагіни	Плагіни
Особливості	Голосування, локалізація	ALM, Wiki, Форум	Інтеграція з Rational	Wiki, SCM	Налаштування UI, безпека

Кожна з проаналізованих систем має унікальні характеристики, що визначають її оптимальне застосування. Вибір системи залежить від специфічних вимог і масштабу проєкту.

Bugzilla є надійним, перевіреним часом інструментом, що найкраще підходить для масштабних проєктів з відкритим кодом.

CodeBeamer позиціонується як комплексна платформа для спільної розробки (ALM), що надає широкий спектр можливостей.

Rational ClearQuest є потужним рішенням для великих корпоративних команд, що працюють в екосистемі продуктів IBM Rational.

Trac — це простий та інтуїтивно зрозумілий інструмент, що ідеально підходить для невеликих команд та однопроєктних середовищ.

JIRA є гнучкою та потужною системою, що вирізняється широкими можливостями налаштування, інтеграції та зручним інтерфейсом.

Висновки до розділу

В даному розділі озглянуто моделі та стандарти управління якістю, зокрема CMMI та ISO, що визначають роль відстеження проблем у підвищенні ефективності програмних проєктів. Проведений порівняльний аналіз сучасних систем відстеження (Bugzilla, CodeBeamer, Rational ClearQuest, Trac, JIRA) показав відмінності у їхніх функціональних

можливостях та сферах застосування. Дослідження плагінів JIRA засвідчило доцільність інтеграції інструментів управління вимогами та контролю якості з методологіями Agile. Встановлено, що JIRA у поєднанні з розширеннями має найбільш комплексний набір засобів для супроводу інноваційних програмних проєктів.

РОЗДІЛ 3. ІМПЛЕМЕНТАЦІЯ МЕТОДІВ ДОКУМЕНТУВАННЯ ІННОВАЦІЙНИХ ПРОГРАМНИХ ПРОЕКТІВ

3.1. Методологія дослідження

Це дослідження проводиться з використанням методології дослідження дизайну (Design Science Research, DSR). DSR — це підхід, де артефакти проєктуються та досліджуються в певному контексті. Артефактом може бути, наприклад, система, компонент, метод або бізнес-процес, а контекст описує, з чим цей артефакт взаємодіє. Проблемний контекст поширюється на зацікавлених сторін артефакту, відомих як соціальний контекст, а також на знання, що використовуються для його проєктування, які називаються контекстом знань. Фреймворк DSR проілюстрований на рисунку 3.1. Він показує взаємодію між артефактом і контекстами, а також дві основні діяльності в рамках дослідження дизайну: проєктування та дослідження.

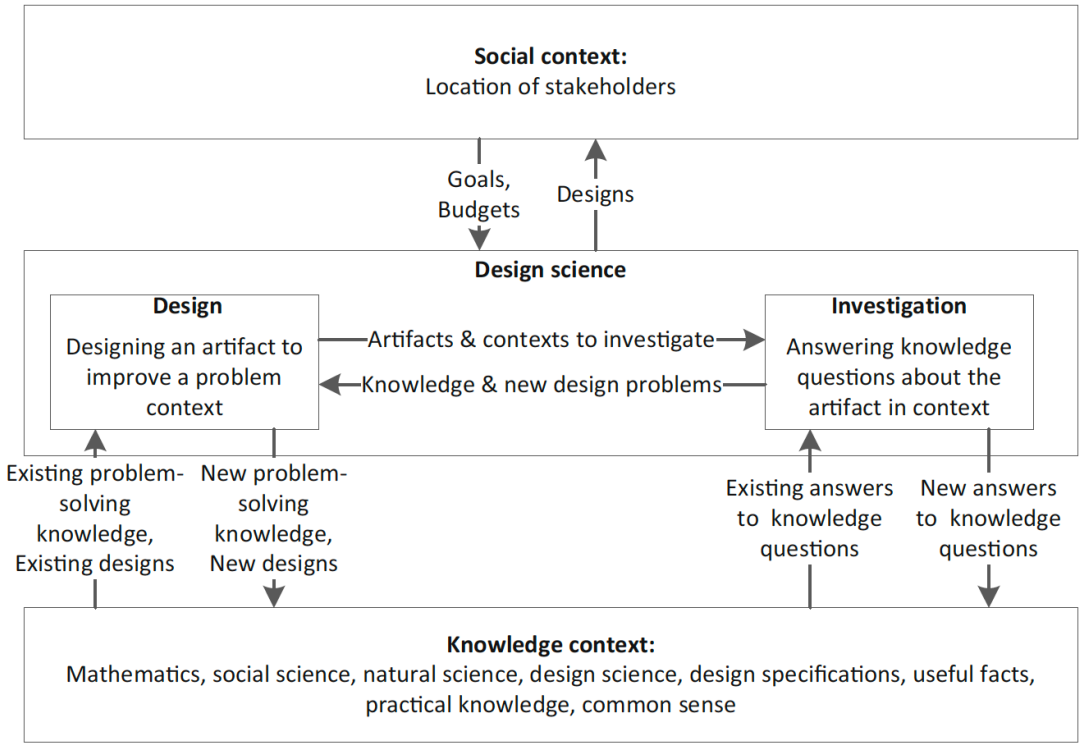

Рис. 3.1. Фреймворк DSR (Design Science Research)

Зазвичай існують два основні підходи до DSR: орієнтований на практику та орієнтований на знання. У цьому дослідженні обрано орієнтований на знання підхід, оскільки початкова проблема може бути адекватно визначена, теорія буде відправною точкою для проєктування, і використовується лише невелика або помірна кількість ітерацій дизайну. Використовуваний процес слідує загальному циклу дослідження дизайну, , який включає чотири кроки: дослідження (explore), синтез (synthesise), створення (create) та оцінку (evaluate). Ці кроки показані на рисунку 3.2 нижче, разом із методами та інструментами, які використовуються на кожному етапі.

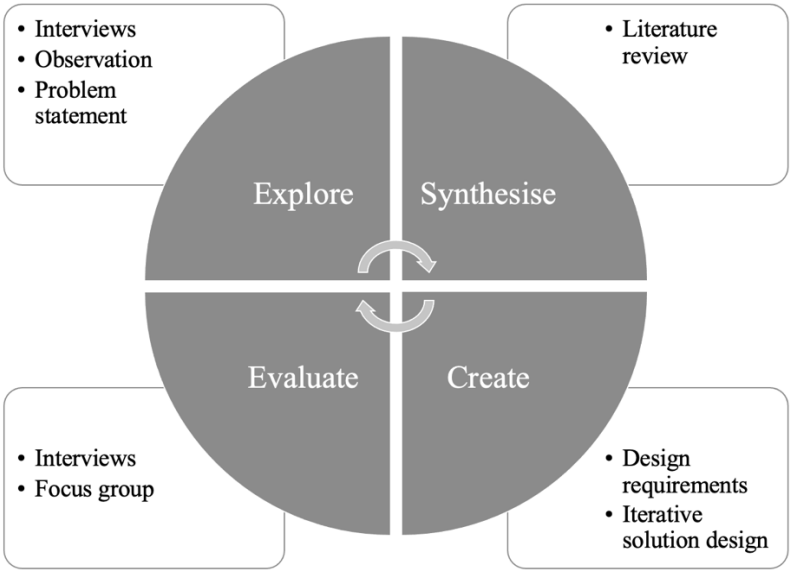

Рис. 3.2. Процес DSR (Design Science Research)

На етапі дослідження проблема визначається за допомогою інтерв'ю та спостережень у компанії. Після того як проблема чітко визначена та погоджена з зацікавленими сторонами компанії, проводиться огляд літератури з метою отримання існуючих знань з контексту знань шляхом використання наукових статей з відповідних галузей. Це дослідження є основою для розробки рішення. Далі, на етапі створення, визначається напрямок рішення та вимоги до дизайну, а розробка рішення відбувається

ітеративно. Кожен запропонований варіант рішення презентується в компанії і оцінюється разом із зацікавленими сторонами, таким чином враховуючи соціальний контекст. На основі цього зворотного зв’язку рішення буде уточнюватися для наступної ітерації. На цьому етапі також може знадобитися додатковий синтез, тому етапи синтезу та створення частково виконуються паралельно. Нарешті, після завершення останньої ітерації рішень, воно оцінюється за допомогою інтерв’ю та фокус-груп з відповідними особами з компанії. Рішення оцінюється на основі того, наскільки добре воно відповідає вимогам до дизайну та наскільки воно приносить користь компанії, згідно з оцінкою зацікавлених сторін.

3.2. Аналіз прикладної проблеми та кейс-стаді

Цей розділ присвячений емпіричному дослідженню, що включає аналіз компанії-кейсу, ідентифікацію ключових проблемних областей та формулювання напрямку для розробки рішення. Опис проблеми базується на результатах інтерв'ю та спостережень, доповнених оцінками зрілості, що забезпечують комплексний погляд на поточний стан інноваційної діяльності в організації.

3.2.1 Опис об’єкту - компанія-кейс

Об'єктом дослідження є постачальник програмного забезпечення як послуги (SaaS), портфоліо якого складається з низки рішень для бізнес-клієнтів. Дослідження сфокусоване на одному з продуктів компанії — програмному забезпеченні для фінансового управління B2B-сегменту, а також на відповідному бізнес-підрозділі, який відповідає за його розробку, маркетинг, продажі та клієнтське обслуговування.

Організаційна структура підрозділу представлена окремими відділами, які, попри взаємодію, мають власне керівництво та штат співробітників. Розробка продукту здійснюється за принципами гнучкої методології Scrum, з

командами, що спеціалізуються на певних функціональних областях продукту. Кожна команда розробки складається з розробників, Scrum-майстра та власника продукту. Загальне керівництво процесом розробки здійснює керівник розробки. Стратегічне бачення та напрямок розвитку продукту визначає команда управління "продуктів та ринків", яку очолює директор з продуктів. Цей підрозділ та команда є основним об'єктом аналізу.

3.2.2. Опис проблеми

Компанія-кейс функціонує в умовах високої конкуренції та динамічного ринку. Попри зрілість, прибутковість продукту та значну клієнтську базу, виникає потреба в нових підходах для розширення ринків та адекватного реагування на конкурентне середовище. Галузь постійно зазнає технологічних змін, що створює як загрози, так і можливості. Водночас, існує необхідність у постійній роботі з технічним боргом продукту.

Компанія визнає потребу в інноваціях для збереження конкурентоспроможності, проте ця потреба не повною мірою трансформується у стратегію та конкретні дії в процесі розробки. Виникає проблема балансування між щоденними операційними завданнями, усуненням технічного боргу та впровадженням нових функцій. На основі цього було ідентифіковано дві ключові проблеми:

- Відсутність структурованого процесу управління інноваціями: У компанії немає формалізованого процесу або фреймворку для управління інноваційними зусиллями. Інновації відбуваються спорадично і неструктуровано. Застосування гнучкої методології Scrum, хоча і ефективне для розробки, не є достатнім для стимулювання та управління інноваціями, як це вказує огляд літератури. Відсутність прозорості в процесі ідейного циклу призводить до дезорієнтації розробників та ускладнює отримання згоди на впровадження змін. Також відсутній чіткий механізм для працівників, щоб висловлювати свої ідеї та відстежувати їх подальшу реалізацію.

- Проблеми з пріоритизацією: Компанія стикається з труднощами у фокусуванні на пріоритетних ініціативах через надмірну кількість ідей, що робить вибір першочергових завдань майже неможливим. Це ілюструє типову дилему початкової фази інновацій, пов’язану з балансуванням між обсягом та якістю ідей. Крім того, виникає складність у балансуванні щоденної розробки (дрібні покращення, запити клієнтів, усунення помилок) та впровадженням інноваційних концепцій.

Таким чином, центральна проблема дослідження полягає в розробці механізму, що дозволить компанії ефективно керувати початковою фазою інновацій та інтегрувати пріоритетні ідеї в план та беклог продукту.

3.2.3. Оцінка зрілості

Для поглибленого аналізу проблемного контексту було проведено два види оцінки зрілості на основі даних, отриманих під час інтерв'ю та спостережень.

Модель S3M-і дозволила оцінити зрілість стратегічного управління в організації за сімома вимірами: лідерство, планування та виконання, процеси та інструменти, структура та модель, люди та культура, управління ефективністю та інновації. Кожен вимір оцінювався за шестирівневою шкалою: невизначений, початковий, запланований, виконаний, оптимізований, відмінний.

Результати оцінки показали, що, попри високий рівень стратегічного управління в цілому, існує значний дефіцит фокусу на інноваціях. Зокрема, виміри "Процеси та інструменти", "Управління ефективністю" та "Інновації" оцінені як "Запланований" або "Початковий/Запланований", що свідчить про відсутність структурованого процесу, метрик та чіткої інтеграції інновацій у загальну стратегію.

На основі фреймворки (рис. 3.3), що описує інноваційні архетипи, було проаналізовано інноваційну спроможність компанії. Фреймворк оцінює здатність компанії усвідомлювати необхідність змін та впроваджувати їх.

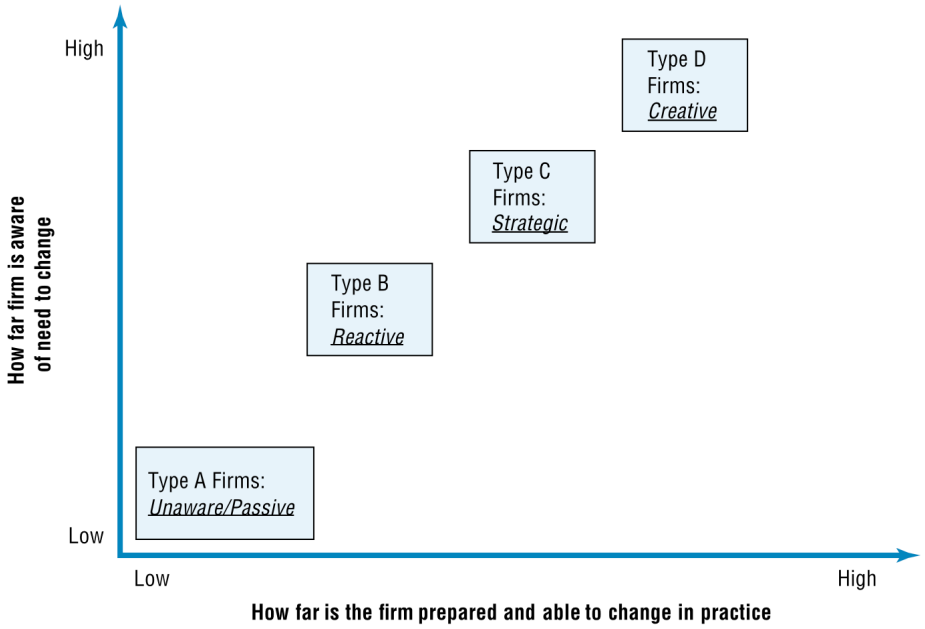

Рис. 3.3. Фреймворк оцінки інноваційної спроможності

Згідно з оцінкою, компанія-кейс наразі перебуває на реактивному рівні інноваційної спроможності. Вона здатна розпізнавати окремі можливості, але їй бракує структури та ясності для систематичного управління інноваційним процесом. Мета дослідження полягає в розробці фреймворку, який дозволить перейти на стратегічний рівень.

Виходячи з ідентифікованих проблем, головною метою дослідження є розробка фреймворку управління для початкової фази інноваційної діяльності, що забезпечить безперервне створення стратегічно узгоджених концепцій для подальшої розробки. Зосередження на початковій фазі обґрунтовано тим, що компанія-кейс вже має ефективні процеси для фази розробки продукту та комерціалізації.

Контекст компанії-кейсу вимагає гнучкості, оскільки гнучкі методи показали свою користь у швидкозмінних галузях, таких як програмна індустрія. Під час огляду літератури було виявлено, що важливою точкою зв'язку між початковим етапом, або відкриттям продукту, та офіційною розробкою є беклог продукту. Також було встановлено, що процеси відкриття та доставки мають бути безперервними та паралельними. Ці

елементи включені в рішення, а процес відкриття продукту розроблений відповідно до принципів Scrum та гнучкої розробки. Дизайн рішення також використовує методи веб-орієнтованих систем генерування ідей та управління портфоліо ідей. Мета полягає в тому, щоб дозволити міжфункціональним командам брати участь у початкових діях та забезпечити прозорість процесу. Крім того, рішення розглядає початковий етап з мультипроєктної перспективи, замість того, щоб зосереджуватися на одному інноваційному проєкті.

Таблиця 3.1.

Принципи дизайну

DP-1	В інноваційному процесі діяльність з ідентифікації та аналізу можливостей, генерування та вибору ідей, а також розробки концепцій має проводитися ітеративно, щоб створити керований та прозорий процес для початкового етапу інновацій, покращуючи інноваційну ефективність.
DP-2	На початковому етапі інновацій слід використовувати метод генерування ідей, керований можливостями, щоб забезпечити стратегічну узгодженість, дозволяючи створювати пріоритетні концепції, пов'язані зі стратегією.
DP-3	У швидкозмінній програмній індустрії гнучкі методології слід застосовувати на початковому етапі, щоб дозволити швидко реагувати на можливості, створюючи конкурентні переваги.
DP-4	На початковому етапі розробки програмного забезпечення, початковий етап має бути пов'язаний з розробкою продукту через беклог продукту, забезпечуючи безперервне та паралельне виконання відкриття та доставки, що призводить до ефективного та гнучкого наскрізного інноваційного процесу.
DP-5	На початковому етапі інновацій слід використовувати веб-орієнтовані системи генерування ідей, щоб уможливити внесок міжфункціональних команд, що призводить до зниження невизначеності та покращення інноваційної ефективності.
DP-6	На початковому етапі інновацій слід застосовувати підхід управління портфоліо ідей, щоб забезпечити мультипроєктну перспективу, що призводить до пріоритетного портфоліо інноваційних проєктів.

3.3 Результати та оцінка дослідження дизайну

Цей розділ містить результати дослідження дизайну, а саме: опис розробленого рішення та його оцінку. На основі результатів оцінювання пропонується переглянута версія рішення, яка є фінальним результатом проєкту.

3.3.1. Опис рішення

Розроблено фреймворк та модель процесу для управління початковою фазою інновацій у компанії-кейсі. Модель поділена на три взаємопов'язані та паралельно діючі складові: Стратегія, Відкриття та Доставка. Ця модель, ілюстрована на рисунку 3.4, не є лінійним процесом, а передбачає ітеративну взаємодію між компонентами.

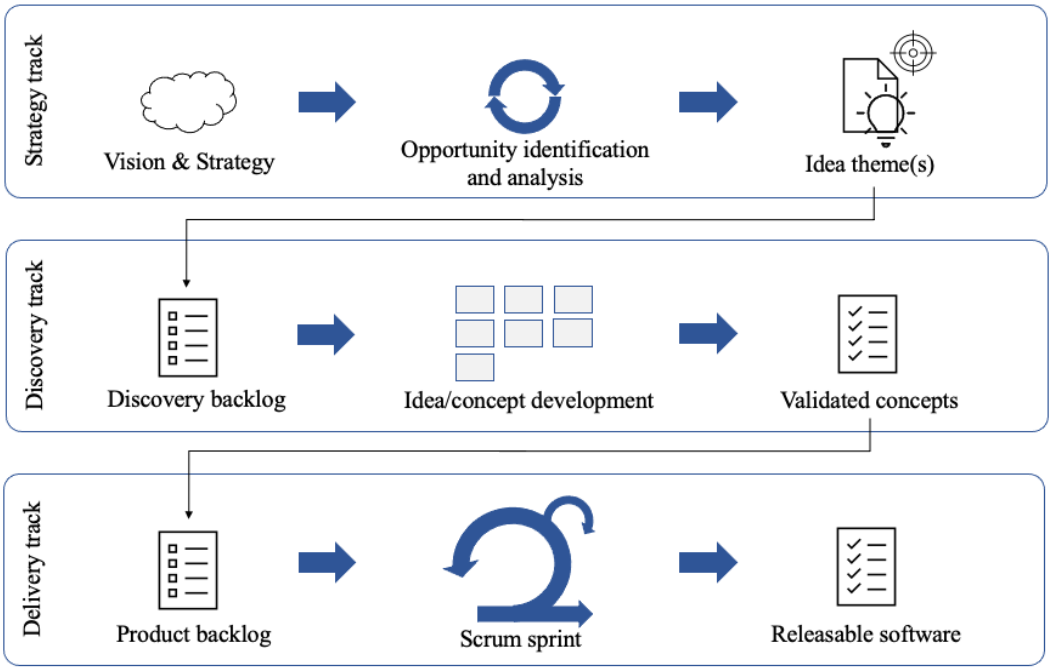

Рис. 3.4. Фреймворк рішення

Трек стратегії є відповідальністю команди "продуктів та ринків". Він розширює існуючий процес розробки стратегії, додаючи діяльність з ідентифікації та аналізу можливостей. У рамках цього треку стратегічне

бачення трансформується в теми для ідей, що забезпечує стратегічну узгодженість на наступній фазі. Для кожного стратегічного періоду рекомендується формулювати від однієї до трьох тем, що сприяє пріоритизації.

Трек Відкриття є новим елементом у процесах компанії, що заповнює прогалину у структурованому управлінні початковою фазою. Цей трек, який очолює команда управління продуктом, передбачає активну участь усіх співробітників. Ідеї, зібрані за заданими темами, проходять ітеративний процес аналізу, відбору, концептуалізації, тестування та валідації. Процес відкритості управляється за допомогою канбан-дошки, що забезпечує прозорість. Інструментом для цього обрано Jira, який вже використовується в компанії, що мінімізує додаткові витрати та час на навчання.

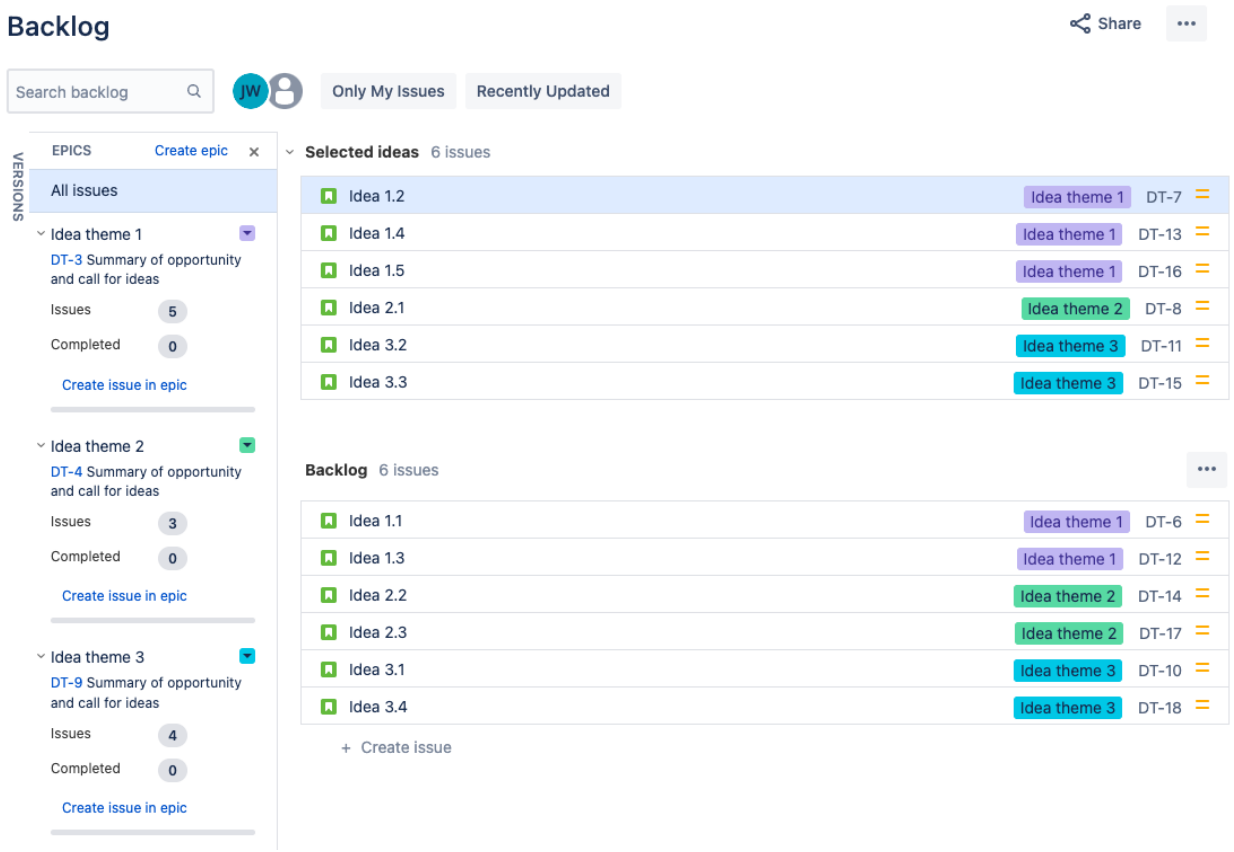

Рис. 3.5. Discovery backlog

Трек Доставки відповідає за перетворення валідованих концепцій на робочий продукт. Він є існуючим процесом розробки продукту, що

здійснюється Scrum-командами. Цей трек пов'язаний з треком "Відкриття" через беклог продукту, куди переносяться валідовані концепції.

Для ведення беклогу відкриття пропонується використовувати Jira, адаптовану для потреб нового процесу. Теми для ідей реєструються як "епіки", а самі ідеї — як окремі завдання.

Процес відкриття візуалізується на канбан-дошці (рис. 3.6), що дозволяє відслідковувати прогрес ідей від стадії "збору" до "готовності для беклогу продукту". На дошці передбачена колонка для відхилених ідей.

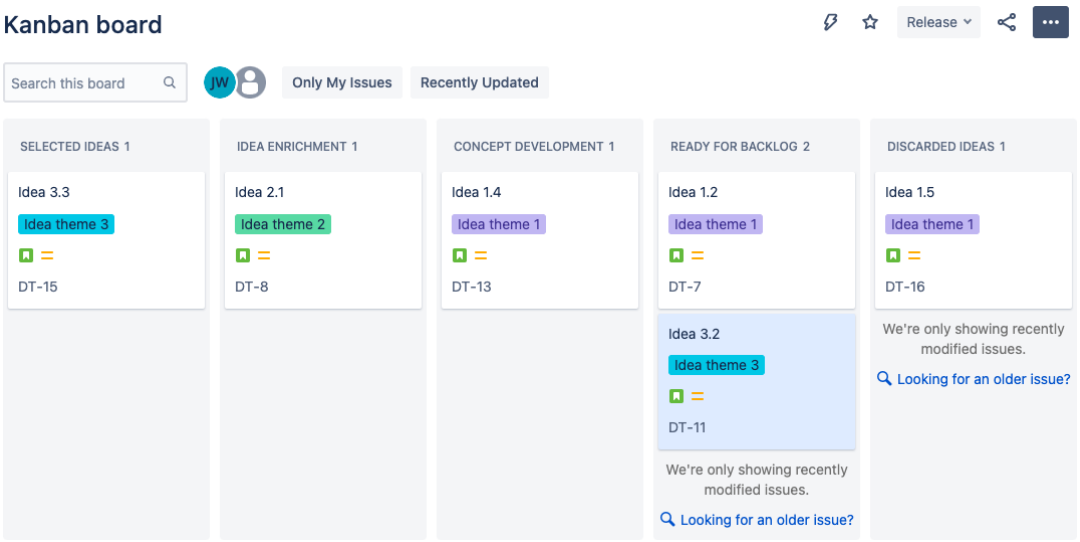

Рис. 3.6. Discovery Kanban дошка

Представимо ключові механізми:

1. Теми для ідей.

Створюються на основі постійного моніторингу зовнішнього та внутрішнього середовища, забезпечуючи стратегічну відповідність інноваційних зусиль.

2. Чемпіони ідей.

Призначення "чемпіона" для обраної ідеї підвищує відповідальність та сприяє її просуванню через етапи валідації.

3. Гнучкість та прозорість.

Модель розроблена для мінімізації бар'єрів для участі, а використання канбан-дошки забезпечує повну прозорість процесу для всіх співробітників.

Запропонована модель має на меті збалансувати творчу свободу з необхідністю фокусування, забезпечуючи, щоб процес генерування ідей залишався керованим та стратегічно узгодженим.

3.4. Оцінка пропонованого рішення управління інноваціями в умовах гнучкої розробки

Оцінка розробленого артефакту дизайну проводилася на основі його відповідності цілям зацікавлених сторін, вимогам до дизайну та здатності вирішувати ідентифіковану проблему в компанії-кейс. Методологія оцінки включала інтерв'ю та фокус-групи, де зацікавленим сторонам було презентовано артефакт дизайну та його теоретичні засади, а також зібрано їхні суб'єктивні відгуки та пропозиції. Важливо зазначити, що на цьому етапі оцінка є концептуальною, а не емпіричною, оскільки практичне впровадження ще не відбулося.

Аналіз показав, що артефакт адекватно досягає основної мети — розробки фреймворку для управління початковою фазою інновацій. Зокрема, було встановлено, що чотири з шести вимог до дизайну (DR-1 до DR-4) були повністю виконані. Проте, дві вимоги — вимірюваність процесу (DR-5) та чітке визначення ролей і обов'язків (DR-6) — були виконані лише частково. Це обумовлено відсутністю формально визначених цілей та нечітким розподілом ролей для всіх учасників процесу.

Крім того, аналіз зворотного зв'язку від зацікавлених сторін виявив дві ключові області для покращення:

1. Потреба у фіксованих термінах. Зацікавлені сторони висловили занепокоєння щодо потенційного відкладання або забуття роботи у гнучкому процесі, заснованому на канбан-дошці. Вони віддали перевагу більш структурованому, часово-обмеженому підходу.

2. Необхідність деталізації ролей. Була висловлена потреба в більш чіткому визначенні, хто саме має бути відповідальним за кожну діяльність, особливо в рамках треку "Відкриття".

На основі результатів оцінки було розроблено фінальний артефакт дизайну, в якому оригінальний трек "Відкриття" на канбан-дошці замінено на модель, що ґрунтується на спринтах управління продуктом. Нова модель, ілюстрована на рисунку 3.7, зберігає структуру з трьох треків — Стратегії, Відкриття та Доставки.

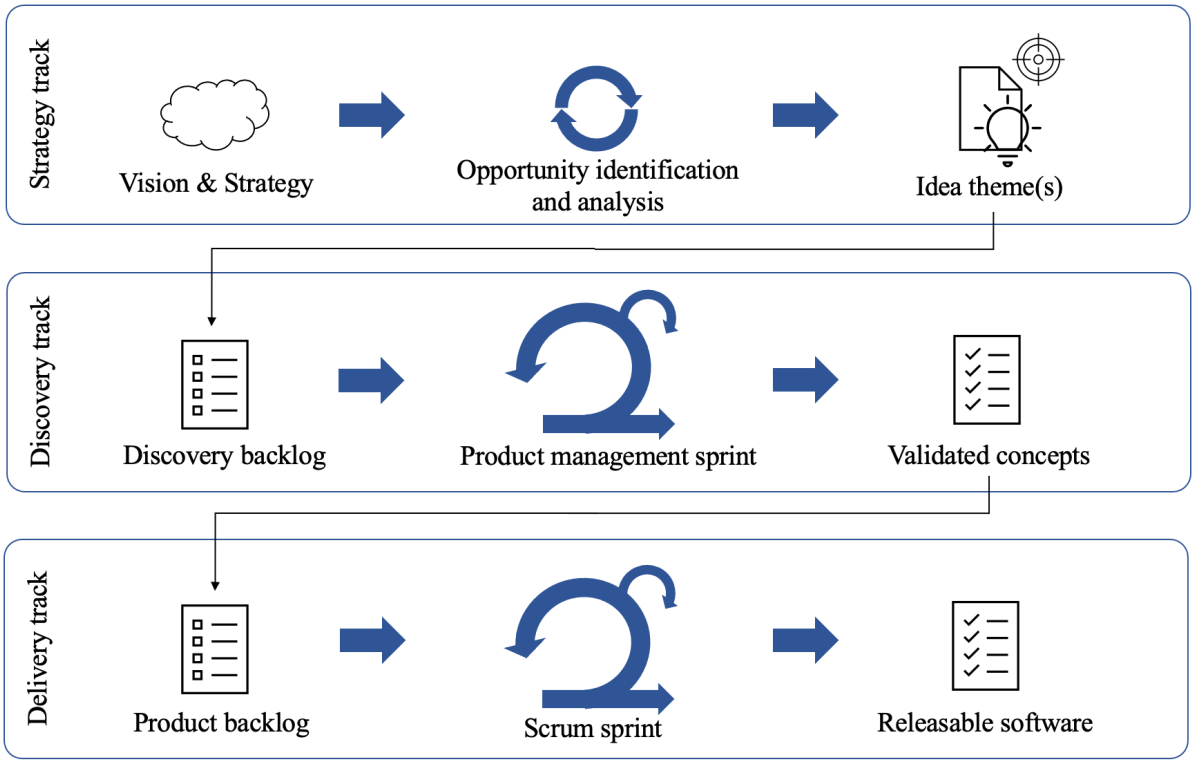

Рис. 3.7. Остаточний артефакт дизайну

Трек стратегії та трек доставки слідують процесу, поясненому в розділі опису рішення раніше, але трек відкриття пояснюється більш детально в цьому розділі. Переглянутий процес "треку відкриття" проілюстрований на рисунку 3.8 . Ця модель процесу слідує гнучкому методу розробки Scrum та процесу Agile SPM.

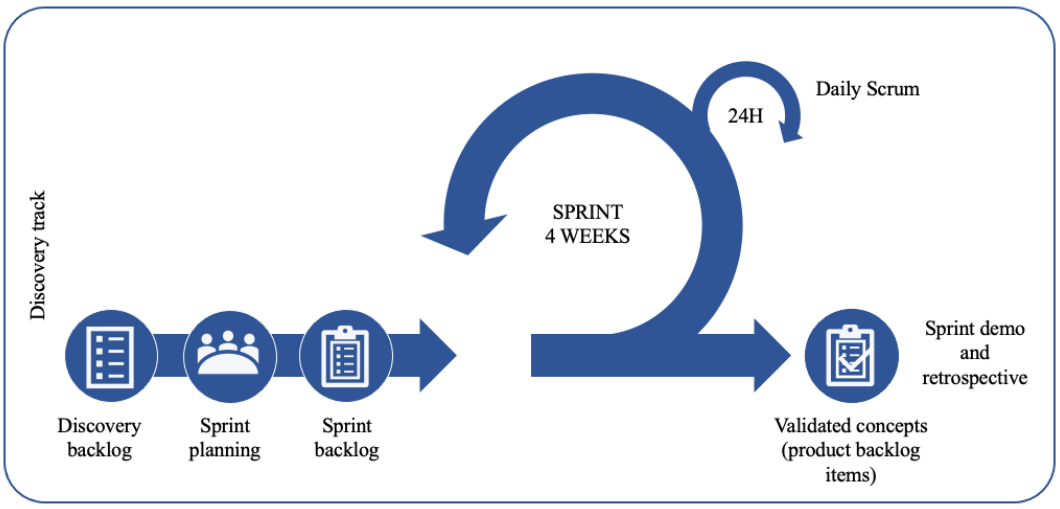

Рис. 3.8. Процес "треку відкриття"

Переглянутий процес треку "Відкриття" (рис. 3.8):

1. Планування спринту.

На початку кожного спринту менеджери з продуктів аналізують беклог відкриття, обирають теми та ідеї для роботи, встановлюють цілі та формують беклог спринту. Це забезпечує фокус та вимірюваність прогресу.

2. Виконання спринту.

Ідеї розвиваються в концепції. Рекомендовано формувати міжфункціональні команди навколо кожної ідеї, залучаючи розробників, дизайнерів, маркетологів та інших фахівців. Тривалість спринту встановлюється на чотири тижні.

3. Оцінка та зворотний зв'язок.

По завершенні спринту його результати оцінюються. Концепції можуть бути перенесені до беклогу продукту, відхилені або повернуті для подальшого аналізу. Прозорість забезпечується через демонстрацію результатів спринту, де також оцінюється готовність концепцій до розробки.

Прогрес треку "Відкриття" можна відстежувати за допомогою діаграми згорання. Для довгострокового аналізу якості концепцій пропонується відстежувати їхню адаптацію та успішність після комерціалізації. Кожен спринт завершується ретроспективною зустріччю для обговорення перешкод та покращення самого процесу.

Впровадження спринтів вирішує проблему відсутності часових рамок та робить процес більш керованим. Водночас, для забезпечення максимальної участі співробітників, менеджерам з продуктів необхідно активно формувати міжфункціональні команди, щоб зберегти переваги спільної творчості, які були в початковій версії з канбан-дошкою.

Отже, метою цього розділу було розроблення рішення для управління початковою фазою інновацій в умовах гнучкої розробки програмного забезпечення. Створений артефакт дизайну, що являє собою фреймворк для процесу відкритості, ефективно досяг цієї мети, інтегрувавши діяльність початкового етапу в існуючу гнучку структуру.

Відповідно до результатів попередніх досліджень, запропоноване розділення процесу на два треки дозволяє керувати генерацією ідей та концептуалізацією, не порушуючи гнучкості та переваг методології. Дане дослідження поглибило розуміння процесу створення валідованих концепцій для продуктового беклогу, а також розширило його, включивши елементи розробки стратегії. Це було досягнуто через впровадження методу генерації ідей, орієнтованого на можливості, що підкреслює важливість стратегічної узгодженості як ключового фактора успіху на початковому етапі інновацій.

Результати свідчать, що програмні компанії можуть отримати значну вигоду від застосування методів управління початковою фазою інновацій у своїх процесах розробки.

Висновки до розділу

Отже, в цьому розділі, у ході імплементації методів документування інноваційних програмних проєктів проведено дослідження практичного кейсу, що дозволило виявити проблеми управління вимогами та змінами в реальній компанії. На основі оцінки зрілості організації було розроблено та впроваджено рішення, яке інтегрує сучасні інструменти документування з гнучкими підходами управління. Результати дослідження підтвердили, що

впроваджене рішення підвищує прозорість, покращує координацію роботи команди та сприяє скороченню кількості дефектів у кінцевому продукті, забезпечуючи ефективне управління інноваціями.

ВИСНОВКИ

У магістерській роботі було здійснено комплексне дослідження проблематики документування та відстеження інноваційних програмних проєктів із урахуванням сучасних методологій управління, стандартів якості та прикладних інструментів підтримки життєвого циклу програмного забезпечення.

У першому розділі проаналізовано предметну область документування й відстеження програмних проєктів. Визначено ключову роль управління вимогами як основоположного процесу, що забезпечує узгодженість між бізнес-цілями, очікуваннями замовників і технічними можливостями розробників. Досліджено функціональні можливості систем відстеження задач, способи збору та фіксації запитів на зміни, а також класифікацію та структуру таких запитів. Показано, що використання спеціалізованих інструментів документування й відстеження дозволяє суттєво підвищити прозорість розробки, зменшити ризики втрати важливої інформації та забезпечити контроль над еволюцією програмних продуктів.

У другому розділі було розглянуто моделі та метрики якості в контексті управління програмними проєктами, зокрема інтегровану модель зрілості можливостей (CMMI) у її поступовому та безперервному поданні. Встановлено, що застосування CMMI сприяє стандартизації процесів, зниженню кількості дефектів та покращенню якості кінцевого продукту. Значну увагу приділено аналізу стандартів ISO, де підкреслено важливість відстеження проблем як одного з ключових елементів забезпечення відповідності вимогам якості. Проведено порівняльний аналіз сучасних систем відстеження та документування (Bugzilla, CodeBeamer, Rational ClearQuest, Trac, JIRA), що дало змогу виділити їхні переваги, обмеження та специфіку застосування в різних типах інноваційних проєктів. Досліджено функціональність розширень та плагінів системи JIRA (GreenHopper, Bamboo, Crucible), які забезпечують інтеграцію процесів управління

вимогами, контролю якості та підтримки гнучких методологій. Порівняльний аналіз підтвердив, що JIRA має найбільш комплексний набір можливостей для супроводу інноваційних проєктів у гнучкому середовищі розробки.

У третьому розділі було здійснено імплементацію методів документування інноваційних програмних проєктів у рамках кейс-стаді конкретної компанії. Проведено оцінку рівня зрілості організації та ідентифіковано проблеми, що перешкоджали ефективному управлінню вимогами та контролю змін. На основі результатів аналізу було запропоновано та впроваджено рішення, орієнтоване на інтеграцію сучасних інструментів управління документуванням і відстеженням із методологіями Agile. Розроблене рішення продемонструвало позитивний вплив на прозорість управління проєктом, покращення координації командної роботи, підвищення швидкості реагування на зміни та скорочення кількості дефектів у програмному продукті. Оцінка результатів упровадження підтвердила його доцільність та ефективність у контексті управління інноваціями.

Загалом проведене дослідження підтвердило, що системний підхід до документування та відстеження програмних проєктів, який базується на інтеграції моделей зрілості процесів, міжнародних стандартів якості та сучасних інструментів підтримки, дозволяє значно підвищити ефективність реалізації інноваційних рішень у сфері програмної інженерії. Розроблені методичні підходи та імплементовані практики можуть бути застосовані у широкому спектрі організацій, що працюють у сфері розробки програмного забезпечення, сприяючи забезпеченню стабільності, гнучкості та конкурентоспроможності програмних продуктів.

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Demir, K. A. (2018). A strategic management maturity model for innovation. Procedia Computer Science, 138, 269-276.
2. Tidd, J., & Bessant, J. (2018). Managing Innovation: Integrating Technological, Market and Organizational Change. 6th ed. John Wiley & Sons.
3. Cooper, R. G. (2001). Winning at new products: Accelerating the process from idea to launch. 3rd ed. Perseus Publishing.
4. Ries, E. (2011). The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses. Crown Business.
5. Maurya, A. (2012). Running Lean: Iterate from Plan A to a Plan That Works. O'Reilly Media.
6. Beck, K., et al. (2001). Manifesto for Agile Software Development. Agile Manifesto.
7. Sutherland, J., & Schwaber, K. (2017). The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. Scrum.org.
8. Highsmith, J. (2010). Agile Project Management: Creating Innovative Products. 2nd ed. Addison-Wesley Professional.
9. Routledge, M. (2019). Agile Product Management with Scrum: A Practical Guide to Product Ownership. Apress.
10. Pressman, R. S. (2014). Software Engineering: A Practitioner's Approach. 8th ed. McGraw-Hill Education.
11. Cockburn, A. (2006). Agile Software Development: The Cooperative Game. 2nd ed. Addison-Wesley Professional.
12. Leffingwell, D. (2011). Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise. Addison-Wesley Professional.

13. Martin, R. C. (2002). Agile Software Development, Principles, Patterns, and Practices. Prentice Hall.
14. Cooper, R. G., & Edgett, S. J. (2008). The product development process: A new way to look at the process and its stages. Management Review, 4(1), 22-31.
15. Kelley, D., & Kelley, T. (2013). Creative Confidence: Unleashing the Creative Potential Within Us All. Crown Business.
16. Brown, T. (2009). Change by Design: How Design Thinking Transforms Organizations and Inspires Innovation. HarperBusiness.
17. Christensen, C. M. (1997). The Innovator's Dilemma: When New Technologies Cause Great Firms to Fail. Harvard Business Review Press.
18. Blank, S. (2013). The Four Steps to the Epiphany: Successful Strategies for Startups that Win. K & S Ranch.
19. Osterwalder, A., & Pigneur, Y. (2010). Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers. John Wiley & Sons.
20. Fitzgerald, B., et al. (2013). Agile and Global Software Engineering: A Research Synthesis. IEEE Transactions on Software Engineering, 39(11), 1629-1644.
21. Larman, C. (2004). Agile and Iterative Development: A Manager's Guide. Addison-Wesley Professional.
22. Pichler, R. (2010). Agile Product Management with Scrum: Driving Innovation and Delivering Value. Addison-Wesley Professional.
23. Cohn, M. (2010). Succeeding with Agile: Software Development Using Scrum. Addison-Wesley Professional.
24. Abrahamsson, P., et al. (2017). Agile software development methods: A systematic literature review. Information and Software Technology, 84, 1-13.
25. Iivari, J., & Iivari, N. (2011). The nature of design science research in information systems. Journal of Information Technology, 26(1), 60-72.

26. Hevner, A. R., et al. (2004). Design science in information systems research. MIS Quarterly, 28(1), 75-105.
27. Wieringa, R. (2014). Design Science Methodology for Information Systems and Software Engineering. Springer.
28. Peffers, K., et al. (2007). A design science research methodology for information systems research. Journal of Management Information Systems, 24(3), 45-77.
29. Winter, R., & Stjepandić, J. (2013). The Role of Design Science Research in Business Process Management. In: Process Management for Business Collaboration and Service Creation. Springer.
30. Munk, T. (2018). Strategic Management of Innovation in Software Companies: A Design Science Study. Master's thesis, Norwegian University of Science and Technology.
31. Chesbrough, H. (2003). Open Innovation: The New Imperative for Creating and Profiting from Technology. Harvard Business School Press.
32. Gassmann, O. (2006). Opening up the innovation process: towards an innovation funnel model. Journal of Product Innovation Management, 23(1), 14-23.
33. O'Reilly, C. A., & Tushman, M. L. (2004). The Ambidextrous Organization. Harvard Business Review, 82(4), 74-81.
34. Trott, P. (2017). Innovation Management and New Product Development. 6th ed. Pearson.
35. Kim, W. C., & Mauborgne, R. (2005). Blue Ocean Strategy: How to Create Uncontested Market Space and Make the Competition Irrelevant. Harvard Business School Press.