ТЕМА 8. ЗНАЙОМСТВО З ІНСТРУМЕНТАМИ ТА ГНУЧКИМИ ПРАКТИКАМИ

1. Agile development

Agile Manifesto

Ми постійно відкриваємо для себе досконаліші методи розробки програмного забезпечення, займаючись розробкою безпосередньо та допомагаючи у цьому іншим. Завдяки цій роботі ми змогли зрозуміти, що:

Люди та співпраця важливіші за процеси та інструменти

Працюючий продукт важливіший за вичерпну документацію

Співпраця із замовником важливіша за обговорення умов контракту

Готовність до змін важливіша за дотримання плану

Тобто, хоча, цінності, що справа важливі, ми все ж цінуємо більше те, що зліва.

Основні принципи Agile-маніфесту

1. Найвищим пріоритетом для нас є задоволення потреб замовника, шляхом завчасного та регулярного постачання програмного забезпечення.

2. Схвальне ставлення до змін, навіть на заключних стадіях розробки. Agile-процеси надають можливість використовувати зміни задля забезпечення конкурентоспроможності замовника.

3. Працюючий продукт слід випускати якомога частіше, з періодичністю від пари тижнів до пари місяців.

4. Впродовж усього проекту розробники і представники бізнесу повинні працювати разом щодня.

5. Над проектом повинні працювати вмотивовані професіонали. Щоб робота була виконана, створіть їм умови, надайте підтримку і повністю на них покладіться.

6. Особиста комунікація – найефективніший та найпрактичніший метод як донести інформацію до команди, так і поширити її всередині.

7. Працюючий продукт – головний показник прогресу.

8. Інвестори, розробники і користувачі повинні мати можливість підтримувати постійний ритм як завгодно довго. Agile допомагає налагодити такий сталий процес розробки.

9. Постійна увага до технічної досконалості і якості проектування підвищує гнучкість проекту.

10. Простота – мистецтво мінімізації зайвої роботи – вкрай необхідна.

11. Найкращі вимоги, архітектурні та технічні рішення виникають у командах, що здатні самоорганізовуватись.

12. Команда регулярно намагається знайти способи підвищення ефективності та відповідно корегує свою роботу.

2. Scrum

Scrum (Скрам) - це не абревіатура, термін взятий із регбі, який означає сутичку навколо м'яча. Сам термін можна визначити як методологію управління проектами, яка побудована на принципах тайм-менеджмету.

Scrum є однією з найбільш популярних «методологій» розробки програмного забезпечення. Згідно із визначенням, скрам - це каркас розробки, із використанням якого люди можуть вирішувати проблеми по мірі надходження, при цьому продуктивно і виробляючи продукти найвищої значущості.

У класичному скрам існує 3 базові ролі:

- Product Owner;
- Scrum Master;
- Команда розробки (Development Team).

Product Owner (PO) є сполучною ланкою між командою розробки та замовником. Завдання PO – максимальне збільшення цінності продукту, що розробляється і роботи команди. Одним з основних інструментів PO є Product Backlog. Product Backlog містить необхідні для виконання робочі завдання (такі як Story, Bug, Task тощо), відсортовані у порядку пріоритету (терміновості).

Scrum Master (SM) є «службовим лідером» (англ. Servant-leader). Завдання Scrum Master - допомогти команді максимізувати її ефективність за допомогою усунення перешкод, допомоги, навчання та мотивації команди, а також допомоги PO.

Команда розробки (Development Team, DT) складається з фахівців, які роблять безпосередню роботу над продуктом. Згідно The Scrum Guide (документу, що є

офіційним описом Scrum від його авторів), DT повинна володіти такими якостями і характеристиками:

- бути самоорганізованою. Ніхто (включаючи SM і PO) не може вказувати команді яким чином перетворити Product Backlog в працюючий продукт;
- бути багатофункціональними, володіти усіма необхідними навичками для випуску працюючого продукту;
- за виконувану роботу відповідає вся команда, а не її індивідуальні члени.

Рекомендований розмір команди - 7 (плюс-мінус 2) людини. Згідно ідеологам скрам, команди більшого розміру вимагають занадто великих ресурсів на комунікації, в той час як команди меншого розміру підвищують ризики (за рахунок можливої відсутності необхідних навичок) і зменшують розмір роботи, який команда може виконати в одиницю часу.

2.1. Процес Scrum

Основою скрам є Sprint, протягом якого виконується робота над продуктом. По закінченню Sprint повинна бути отримана нова робоча версія продукту. Sprint завжди обмежений в часі (1-4 тижні) і має однакову тривалість протягом всього життя продукту.

Перед початком кожного Sprint складається Sprint Planning, на якому проводиться оцінка вмісту Product Backlog і формування Sprint Backlog, який містить завдання (Story, Bugs, Tasks), які повинні бути виконані у поточному спринті. Кожен спринт повинен мати мету, яка є мотивуючим фактором і досягається за допомогою виконання завдань з Sprint Backlog.

Кожного дня проводиться Daily Scrum, на якому кожний член команди відповідає на запитання «Що я зробив учора?», «Що я планую зробити сьогодні?», «Які перешкоди під час роботи я зустрів?». Завдання Daily Scrum – визначення статусу і прогресу роботи над Sprint, раннє виявлення перешкод, що виникли, вироблення рішень щодо зміни стратегії, необхідних для досягнення цілей Sprint'a.

На рисунку 1 представлена діаграма згорання задач (Burndown Chart).

Діаграма, яка ілюструє кількість зробленої роботи та роботи, яка ще залишилась. Оновлюється щоденно для того, щоб у простій формі показати просування в роботі над спринтом. Графік повинен бути загальнодоступним.

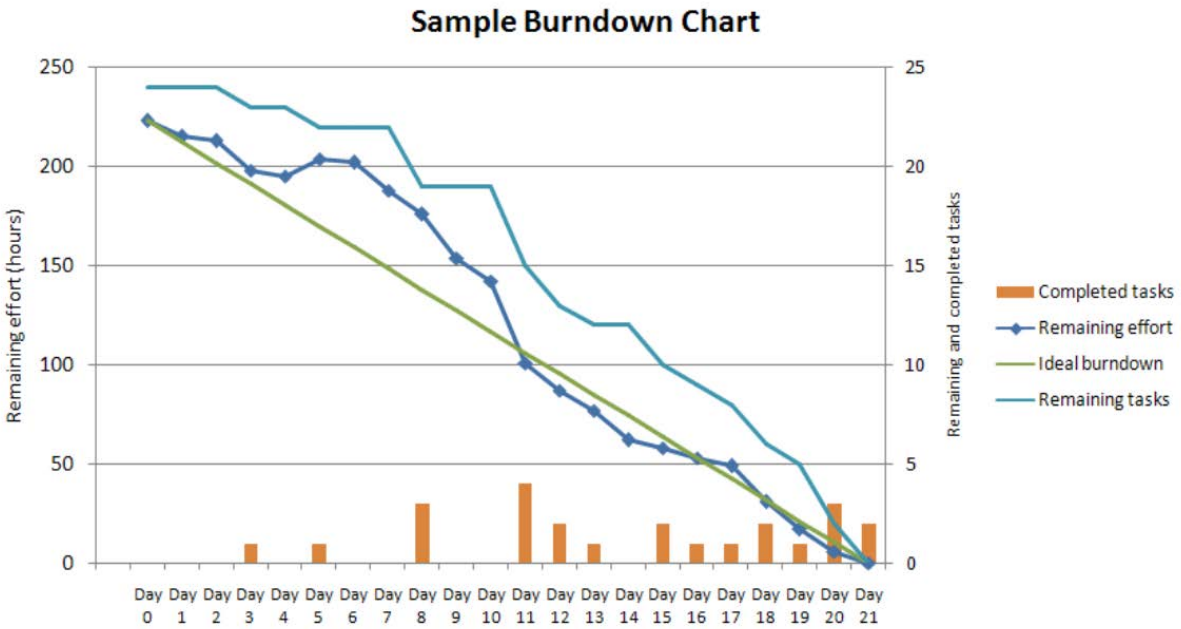

Рисунок 1 – Діаграма згорання задач

Існують різні види діаграми:

- діаграма згорання робіт для спринта – показує, скільки вже задач зроблено і скільки ще залишається зробити в поточному спринті;
- діаграма згорання роботі для випуску проекту – показує, скільки вже задач зроблено і скільки ще залишається зробити до випуску продукту (зазвичай будується на базі декількох спринтів).

По завершенню Sprint'a складається Sprint Review і Sprint Retrospective, завдання яких оцінити ефективність (продуктивність) команди в минулому Sprint'i, спрогнозувати очікувану ефективність (продуктивність) в наступному спринті, виявити наявні проблеми, оцінити ймовірності завершення всіх необхідних робіт по продукту і інше .

2.2. Особливості Scrum

Особливості Scrum, які варто пам’ятати і на які часто скаржаться:

1. *Скрам застосовується невірно або не повністю.*

Згідно авторам Scrum, емпіричний досвід є головним джерелом достовірної інформації. Необхідність повного і точного виконання скрам вказана в The Scrum Guide і обумовлена нетиповою організацією процесу, відсутністю формального лідера і керівника.

2. *Недооцінена важливість роботи по забезпеченню мотивації команди.*

Одним з основних принципів Scrum є самоорганізовані, багатофункціональні команди. Згідно з дослідженнями соціологів, чисельність самовмотивованих співробітників, здатних на самоорганізацію не перевищує 15% від працездатного населення.

Таким чином, лише невелика частина співробітників здатна ефективно працювати в Scrum без істотних змін в ролях Scrum Master і Product Owner, що суперечить ідеології Scrum, і потенційно призводить до невірного або неповного використання Scrum.

3. Scrum застосовується для продукту, вимоги до якого суперечать ідеології Scrum.

Scrum відноситься до сімейства Agile, так Scrum вітає зміни у вимогах в будь-який момент (Product backlog може бути змінений в будь-який момент). Це ускладнює використання Scrum в fixed-cost/fixed-time проектах. Ідеологія Scrum стверджує, що заздалегідь неможливо передбачити усі зміни, таким чином немає сенсу завчасно планувати увесь проект, обмежившись лише just-in-time плануванням, тобто планувати тільки ту роботу, яка повинна бути виконана в поточному Sprint.

2.3. Переваги і недоліки Scrum

Переваги:

1. Скрам орієнтований на клієнта, адаптивний, дає клієнтові можливість робити зміни у вимогах у будь-який момент часу (але не гарантує того, що ці зміни будуть виконані). Можливість зміни вимог приваблива для багатьох замовників програмного забезпечення.

2. Scrum досить простий у вивченні, дозволяє економити час, за рахунок усунення не критичних активностей. Scrum дозволяє отримати потенційно робочий продукт в кінці кожного Sprint'у.

3. Скрам робить акцент на самоорганізованій, багатофункціональній команді, здатній вирішити необхідні завдання з мінімальною координацією. Це особливо привабливо для малих компаній і стартапів, оскільки позбавляє від необхідності наймати або навчати спеціалізованого персоналу керівників.

Недоліки:

1. Скрам відноситься до сімейства Agile, в Scrum не прийнято, наприклад, створення плану комунікацій та реагування на ризики. Таким чином, роблячи складною

або неможливою формальну (юридична або адміністративна) протидію порушенням правил Scrum.

2. Іншою слабкою особливістю скрам є акцент на самоорганізовану, багатофункціональну команду. При, так би мовити зниженні витрат на координацію команди, це призводить до підвищення витрат на відбір персоналу, його мотивацію, навчання. При певних умовах ринку праці, формування повноцінної, ефективної Scrum команди може бути неможливим.

3. Kanban

Цей термін прийшов з Японії завдяки широко відомої у вузьких колах виробничої системи Тойота. Хотілося б, щоб якомога більше людей прочитало про цю систему і основні її принципи - бережливе виробництво, постійний розвиток, орієнтацію на клієнта і т.д. Всі ці принципи описані в книзі Тайіті Оно «Виробнича система Тойоти».

Термін **Канбан** має дослівний переклад: "Кан" означає видимий, візуальний, і "бан" означає картка або дошка.

На заводах Тойота картки Канбан використовуються повсюди для того, щоб не захаращувати склади і робочі місця заздалегідь створеними запчастинами. Наприклад, уявіть, що ви ставите двері на Тойоти Королли. У вас біля робочого місця знаходиться пачка з 10 дверей. Ви їх ставите одну за одною на нові машини і, коли в пачці залишається 5 дверей, то ви знаєте, що пора замовити нові двері. Ви берете картку Канбан, пишете на ній замовлення на 10 дверей і відносите її тому, хто робить двері. Ви знаєте, що він їх зробить якраз до того моменту, як у вас закінчаться решта 5 дверей. І саме так і відбувається – коли ви ставите останні двері, прибуває пачка з 10 нових дверей. І так постійно – ви замовляєте нові двері тільки тоді, коли вони вам потрібні.

А тепер уявіть, що така система діє на всьому заводі. Ніде немає складів, де запчастини лежать тижнями і місяцями. Всі працюють тільки за запитом і виробляють саме стільки запчастин, скільки запрошено. Якщо раптом замовлень стало більше або менше – система сама легко підлаштовується під зміни.

Основне завдання карт Канбан в цій системі - це зменшувати кількість «роботи, яка виконується в даний момент» (work in progress).

Наприклад, на всю виробничу лінію може бути виділено рівно 10 карток для дверей. Це означає, що в кожен момент часу на лінії не буде більше 10 готових дверей. Коли замовляти нові двері і скільки – це завдання для того, хто їх встановлює. Тільки він знає свої потреби, і тільки він може розміщувати замовлення виробнику дверей, але він завжди обмежений числом 10.

Цей метод *Бережливого виробництва* (Lean manufacturing) був придуманий в Тойоті і зараз багато виробничих компаній по всьому світу його впроваджують або вже впровадили.

Відмінності між Канбан і SCRUM:

- у Канбан немає таймбоксів ні на що (ні на завдання, ні на спринти);
- у Канбан завдання більше і їх менше;
- у Канбан оцінки термінів на завдання опційні або взагалі їх немає;
- у Канбан «швидкість роботи команди» відсутня і вважається тільки середній час на повну реалізацію завдання.

Канбан розробка відрізняється від SCRUM в першу чергу орієнтацією на завдання. Якщо в SCRUM основна орієнтація команди - це успішне виконання спринтів (треба визнати, що це так), то в Канбан на першому місці завдання.

Спринтів ніяких немає, команда працює над завданням з самого початку і до завершення. Деплоймент завдання робиться тоді, коли воно готове. Презентація виконаної роботи - теж. Команда не повинна оцінювати час на виконання завдання, бо це має мало сенсу і майже завжди помилково спочатку.

Якщо менеджер вірить команді, то навіщо мати оцінку часу? Завдання менеджера - це створити пріоритезувати пул завдань, а завдання команди - виконати якомога більше завдань з цього пулу. Усе. Ніякого контролю не потрібно. Все, що потрібно від менеджера, – це додавати завдання в цей пул або міняти їм пріоритет. Саме так він керує проектом.

Команда для роботи використовує Канбан-дошку. Наприклад, вона може виглядати так (див. рис. 2).

Рисунок 2 – Приклад Канбан-дошки

Стовпці зліва направо:

Цілі проекту:

Необов'язковий, але корисний стовпець. Сюди можна помістити високорівневі цілі проекту, щоб команда їх бачила і все про них знала. Наприклад, «Збільшити швидкість роботи на 20%» або «Додати підтримку Linux».

Черга завдань:

Тут зберігаються завдання, які готові до того, щоб почати їх виконувати. Завжди для виконання береться верхня, пріоритетна задача і її картка переміщується в наступний стовпець.

Опрацювання дизайну: (цей і інші стовпці до «Закінчено» можуть змінюватися, тому що саме команда вирішує, які кроки проходить завдання до стану «Закінчено»)

Наприклад, в цьому стовпці можуть перебувати завдання, для яких дизайн коду або інтерфейсу ще не зрозумілий і обговорюється. Коли обговорення закінчені, завдання пересувається в наступний стовпець.

Розробка:

Тут завдання висить до тих пір, доки розробка фічі не завершена. Після завершення воно пересувається в наступний стовпець.

Або, якщо архітектура не вірна або не точна - завдання можна повернути в попередній стовпець.

Тестування:

У цьому стовпці завдання знаходиться, доки воно тестується. Якщо знайдені помилки - повертається в Розробку. Якщо немає - пересувається далі.

Деплоймент:

У всіх проектів свій деплоймент. У когось це значить викласти нову версію продукту на сервер, а у когось - просто закомітити код в репозиторій.

Завершено:

Сюди завдання попадає тільки тоді, коли всі роботи по задачі завершені повністю.

В будь-якій роботі трапляються **термінові** завдання. Заплановані чи ні, але такі, які треба зробити прямо зараз. Для таких можна виділити спеціальне місце (на зображенні відзначено, як «Expedite»). У Expedite можна помістити одне термінове завдання і команда повинна почати його виконувати негайно і завершити якомога швидше. Але може бути тільки одна така задача! Якщо з'являється ще одна - вона повинна бути додана в «Чергу завдань».

А тепер найважливіше. Бачите цифри під кожним стовпцем? Це число завдань, які можуть бути одночасно в цих стовпцях. Цифри підбираються експериментально, але вважається, що **вони повинні залежати від числа розробників в команді.**

Наприклад, якщо ви маєте 8 програмістів в команді, то в рядок «Розробка» ви можете помістити цифру 4. Це означає, що одночасно програмісти будуть робити не більше 4-х завдань, а значить у них буде багато причин для спілкування і обміну досвідом. Якщо ви поставите туди цифру 2, то 8 програмістів, які займаються двома завданнями, можуть занудьгувати або втрачати занадто багато часу на обговореннях. Якщо поставити 8, то кожен буде займатися своїм завданням і деякі завдання будуть затримуватися на дошці надовго, але ж головне завдання Канбан - це зменшення часу проходження завдання від початку до стадії готовності.

Ніхто не дасть точної відповіді, які повинні бути ці ліміти, але спробуйте для початку розділити число розробників на 2 і подивитися, як це працює в вашій команді. Потім ці числа можна підігнати під вашу команду.

Під «розробниками» я розумію не тільки програмістів, але й інших фахівців. Наприклад, для стовпця «Тестування» розробники - це тестери, тому що тестування - це їхній обов'язок.

Що нового і корисного дає така дошка з лімітами?

По-перше, **зменшення числа паралельно виконуваних завдань сильно зменшує час виконання кожної окремої задачі**. Немає потреби перемикати контекст між завданнями, відстежувати різні сутності, планувати їх і т.д. – робиться тільки те, що потрібно. Немає потреби влаштовувати спринт пленінги та 5% воркшопи, тому що планування вже зроблено в стовпці «Черга завдань», а детальне опрацювання завдання починається тільки тоді, коли завдання починає виконуватися.

По-друге, **одразу видно заминки.** Наприклад, якщо тестери не справляються з тестуванням, то вони дуже скоро заповнять весь свій стовпець і програмісти, які закінчили нову задачу, вже не зможуть перемістити її в стовпець тестування, тому що він заповнений. Що робити? Тут час згадати, що «ми - команда» і вирішити цю проблему. Наприклад, програмісти можуть допомогти тестерам завершити одну з задач тестування і тільки тоді пересунути нову задачу на місце, що звільнилося. Це дозволить виконати обидва завдання швидше.

По-третє, **можна обчислити час на виконання усередненої задачі.** Ми можемо позначати на картці дату, коли вона потрапила в чергу завдань, потім дату, коли її взяли в роботу і дату, коли її завершили. За цим трьом точкам для хоча б 10 завдань можна вже порахувати середній час очікування в черзі завдань і середній час виконання завдання. А з цих цифр менеджер або Product Owner може вже розраховувати все, що йому завгодно.

Весь Канбан можна описати всього **трьома основними правилами:**

1. Візуалізуйте виробництво

- розділіть роботу на завдання, кожне завдання напишіть на картці і помістіть на стіну або дошку;
- використовуйте названі стовпці, щоб показати положення завдання у виробництві.

2. Обмежуйте WIP (work in progress або роботу, виконувану одночасно) на кожному етапі виробництва.

3. Виміряйте час циклу (середній час на виконання одного завдання) і оптимізуйте постійно процес, щоб зменшити цей час.

4. Issue Tracking System

Баг трекер – рахується різновидом системи управління задачами (task management system).Класичними зразками task management system являються Trello та Google Task Manager.

Баг трекер - це прикладна браузерна чи десктопна програма, розроблена з метою допомогти усім причетним до розробки програмного продукту: програмістам, тестерам програмного забезпечення, керівникам проектів - рахувати та контролювати помилки, знайдені в програмах, побажання користувачів, а також стежити за процесом усунення цих помилок розробниками і виконанням або невиконанням побажань.

Дуже важливо щоб сучасний bug tracker tool мав фічу Issue Tracking System (ITS). Не слід плутати Issue з багами! По-суті Issue це питання пов'язані з багами та розробкою. Issue присвоюються різним відповідальним особам, які контролюють їх обробку. У ITS трекається скільки часу вони витрачають, щоб забезпечити відповідність внутрішнім робочим процесам, виконується статистичний аналіз. На вигляд Issue Tracking System і по логіці схожа на Канбан дошку. Issue Tracking System – атрибут Agile.

Отже, Багтрекерів є багато, але найпопулярніші багтрекери, що є у всіх на слуху це **JIRA, Redmine, Bugzilla, Asana, YouTrack** тощо.

4.1 JIRA

Jira – платна програма, яка дозволяє управляти не тільки помилками й тікетами, але також і проектами в цілому. Jira підтримує технології Agile методології. За допомогою інтерактивної дошки можна стежити за процесом переміщення тасків, таким чином регулюючи загальну тенденцію роботи по проекту. Jira розроблена компанією Atlassian Software Systems. Назва системи отримана шляхом усічення слова «Gojira» у відповідь на не менш популярний в минулому баг трекер Bugzilla – його оглянемо і про нього ми поговоримо пізніше. Баг трекер Jira використовується у більш аніж 15 000

компаній по всьому світу. Серед її користувачів: Microsoft, BBC, Nokia, Boeing та ін. відомі компанії.

У даної програми надзвичайно широкий функціонал. Як чогось із функціоналу у Jira не вистачає, можна його доставити за рахунок плагінів. Зупинимося на безпосередньому призначенні Jira, як багтрекеру (див.рис.3).

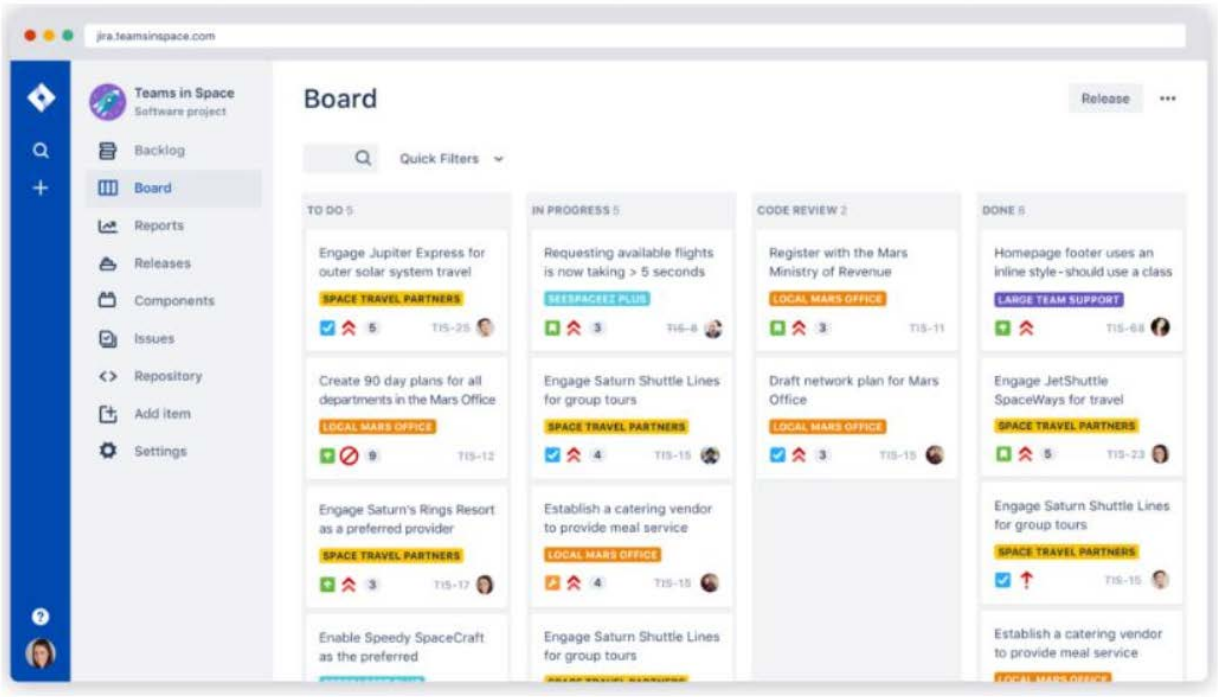

Рисунок 3 – Робоча область бак трекера Jira

Jira заповнюється задачами (від англ. tickets або issues) (див. рис. 4).

Задача містить наступні компоненти:

– назва проекту
– тайтл
– тип
– пріоритет
– версії
– компоненти
– підкомпоненти
– статус
– резолюція
– зміст
– додатки (фото, відео, документ)

– коментарі

– саб-таски (якщо є).

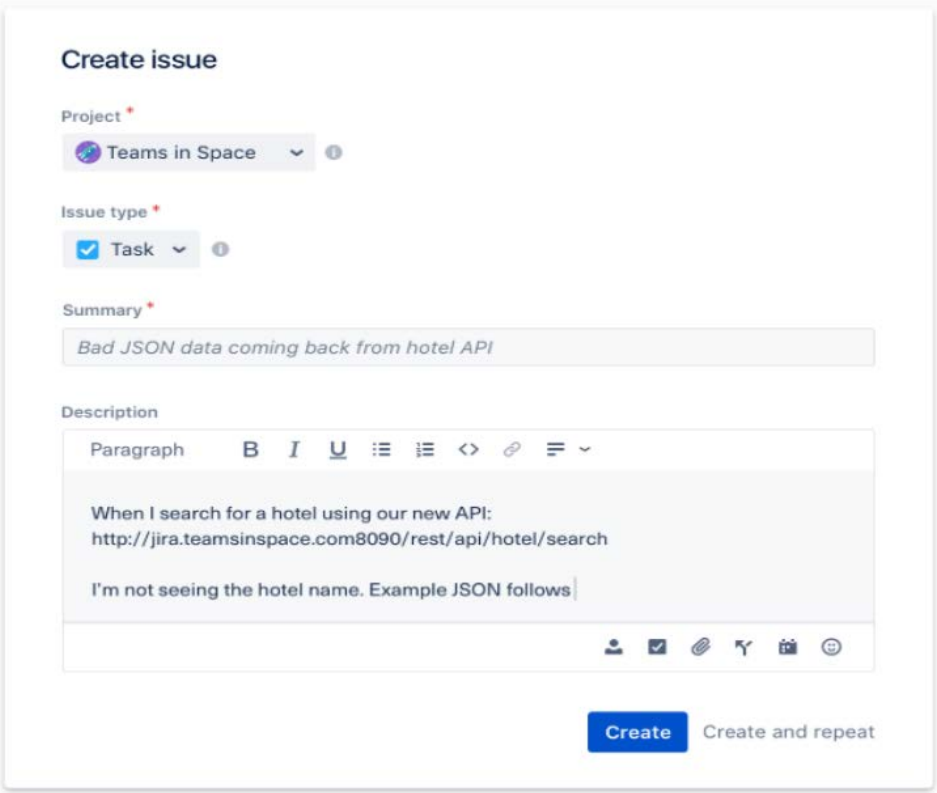

Рисунок 4 – Створення завдання в Jira

Компоненти завдання можуть бути розширені додатковими полями або обмежені налаштуваннями меню програми Jira. Завдання може редагуватися або просто змінювати статус, наприклад, з «відкритий» на «закритий». Які переходи між станами можливі, визначається робочим процесом (бізнес-процесом) (workflow). Загалом за допомогою Jira можна управляти робочим процесом на проекті, визначати ролі і т.д. Будь-які зміни в задачі записуються у журнал.

Jira має велику кількість можливостей конфігурації: для кожної програми може бути визначений окремий тип задач з власним workflow, набором статусів, одним або декількома видами представлення (англ. screens). Детально фічі за посиланням шматка документації, де розписані усі можливості Jira у якості системи відстеження помилок. Крім того, за допомогою так званих «схем» можна визначити для кожного індивідуального Jira-проекту власні права доступу, поведінку, видимість полів і багато іншого.

4.2. Bugzilla

Bugzilla – опенсорсна система відслідковування багів від Mozilla.org. Це безкоштовний bugtracker tool. Bugzilla являється базою даних обліку багів і запитів щодо поліпшення Firefox, Thunderbird, SeaMonkey, Camino та інших проектів mozilla.org.

Скажімо так, Багзілла, хороший баг трекер для початківців. Даний баг трекер найпростіший з усіх перерахованих і володіє найменшим функціоналом, що одночасно і добре, і погано. З одного боку, Bugzilla досить простий, з іншого боку, там є все потрібне для типового проекту. Головне завдання bug tracking systems – виконує на відмінно! Головний мінус Bugzilla – інтерфейс. Він, легко кажучи, так собі не дуже User-Friendly, у порівнянні з конкурентами. Також неможливо регулювати workflow. І багтрекер Bugzilla не вийде використовувати для великих складних проектів, але для навчання, для малих і простих проектів – цілком.

Таск менеджер у даному баг трекері виглядає наступним чином, як це проілюстровано на рисунку 5 , і з 2015 року він практично не змінився.

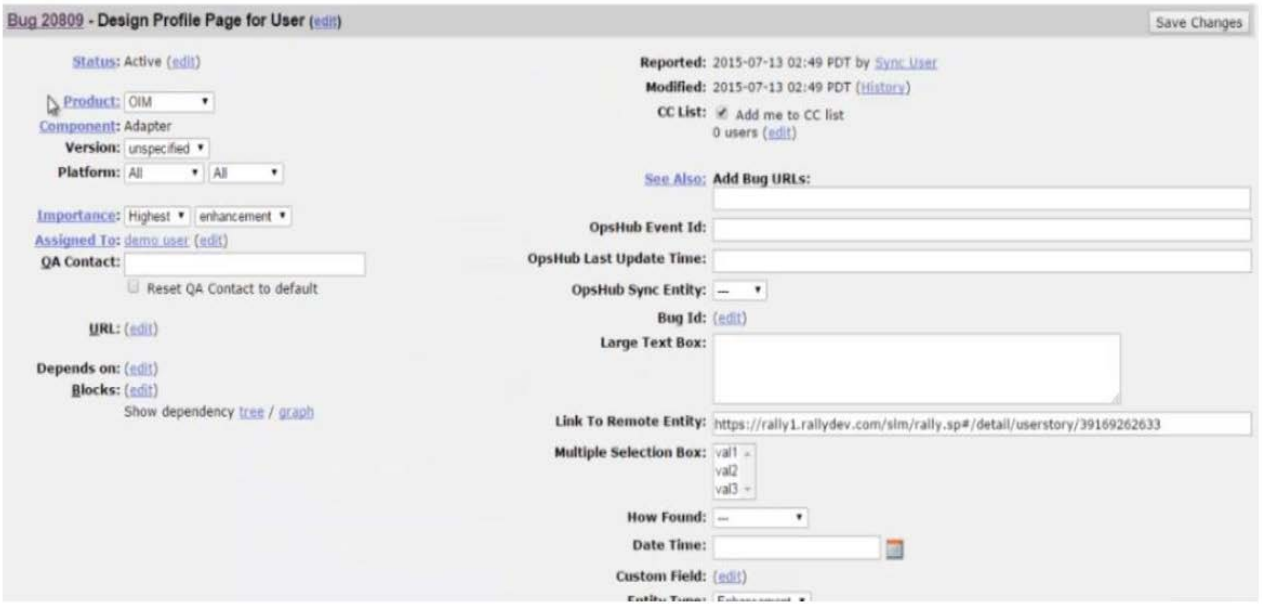

Рисунок 5 – Робоча область бак трекеру Bugzilla

Основними пунктами у Bugzilla є:

- тайтл;
- статус;
- severity / priority;
- ключові слова;

– посилання на ресурс;
– attachment;
– кому призначений;
– оточення.

4.3. Redmine

Redmine – це безкоштовний веб-додаток на основі відомого веб- фреймворку Ruby on Rails з платними розширеннями плагінами (69$ -999$).

Redmine – це не тільки баг трекер, але і безкоштовний хмарний (SAAS) веб-додаток для ефективного управління проектами для малого та великого бізнесу. А можливості як баг трекеру Redmine одні з найбільш прогресивних, хоча у ньому не має наворотів, як от у попередньо розглянутій Jira. У використанні Redmine досить-таки простий і зрозумілий. Якщо звикнути користуватися Redmine, він не знадобиться у якості системи відстеження помилок, то може легко знадобитися для якихось інших задач у бізнесі.

Redmine надає такі можливості:

– робота з декількома проектами;
– облік часових витрат;
– діаграми Ганта;
– система доступу по ролях;
– для кожного проекту є форуми обговорення;
– календар;
– система Redmine для відстеження помилок;
– інтеграція з іншими системами управління.

Головна перевага системи, яку найбільше позиціонує Redmine, окрім відсутності оплати, – багатомовний інтерфейс.

Особливостями Redmine можна назвати використання діаграми Ганта, опція створення форумів для кожного існуючого проекту, самостійна реєстрація нових користувачів, обмеження доступу до певних завдань проекту, підтримка Agile технологій. Із мінусів, у деяких користувачів по правах відсутнє оповіщення по пошті про те, що в задачі зроблені зміни.

На рисунку 6 проілюстровано приклад завдання в баг трекері Redmine.

Рисунок 6 – Приклад завдання в Redmine

Складові задачі у Redmine наступні:

- трекер (визначає вид завдання);
- тема;
- опис;
- статус завання;
- пріоритет;
- категорія (до чого відноситься завдання);
- версія;
- додаток.

4.4 YouTrack

YouTrack – відверто кажучи цей баг трекер більше для розробників програмістів, які перейшли в управління проектами (див.рис.7). YouTrack початково повноцінно розроблений під Agile. Чеська компанія JetBrains відома перш за все своїми IDE (Інтегровані Середовища розробки): IntelliJ IDEA – IDE для мови програмування Java, PyCharm – IDE для Python, PhpStorm – IDE для PHP, RubyMine – IDE для Ruby і Ruby on Rails та інші.

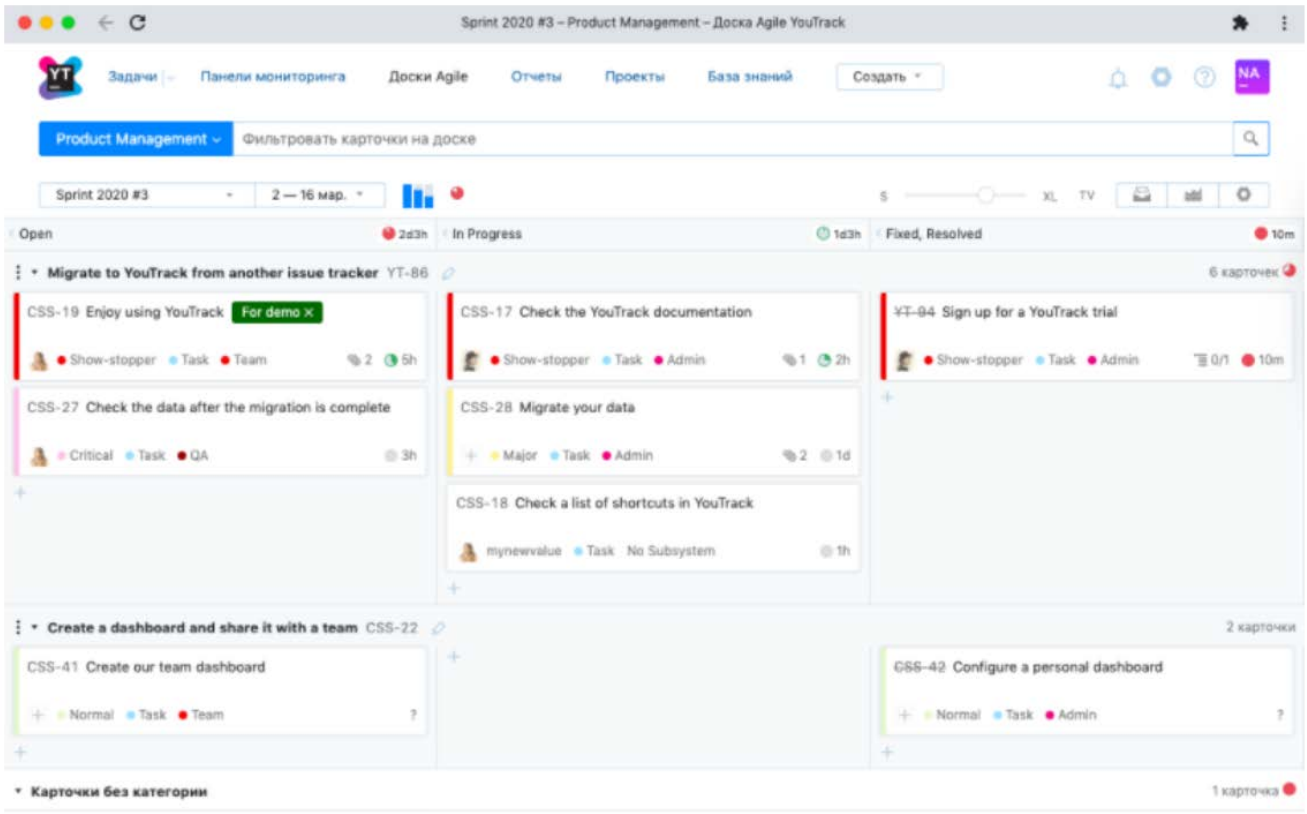

Рисунок 7 – Робоча область баг трекера YouTrack

Безкоштовний YouTrack відносно. JetBrains надає свій YouTrack для безкоштовного використання розробникам відкритих проектів і для навчання. YouTrack доступний також у вигляді сервісу (SaaS), під назвою YouTrack InCloud, безкоштовно в базовій конфігурації.

Обмеження безкоштовної версії YouTrack:

– Обмеження до десяти користувачів;

– Хмарна версія має обмеження на зберігання 5 ГБ;

– Хмарна версія не дозволяє створювати приватні проекти;

– Вартість оновлення: найнижча платна версія рішення SaaS від YouTrack складає 200 доларів США на рік для максимум 15 користувачів.