МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

Інститут Прикладного системного аналізу

Кафедра Системного проектування

**Лабораторна робота № 3.**

З дисципліни

«Проектування інформаційних систем»

На тему

***«****Розробка життєвого циклу проекту та системи багтрекінгу.*

*Розробка поетапного плану проекту.****»***

Виконала

студентка групи ДА-61

Шкульова Катерина

КИЇВ

2019

Частина 1. Розробка життєвого циклу проекту та системи багтрекінгу.

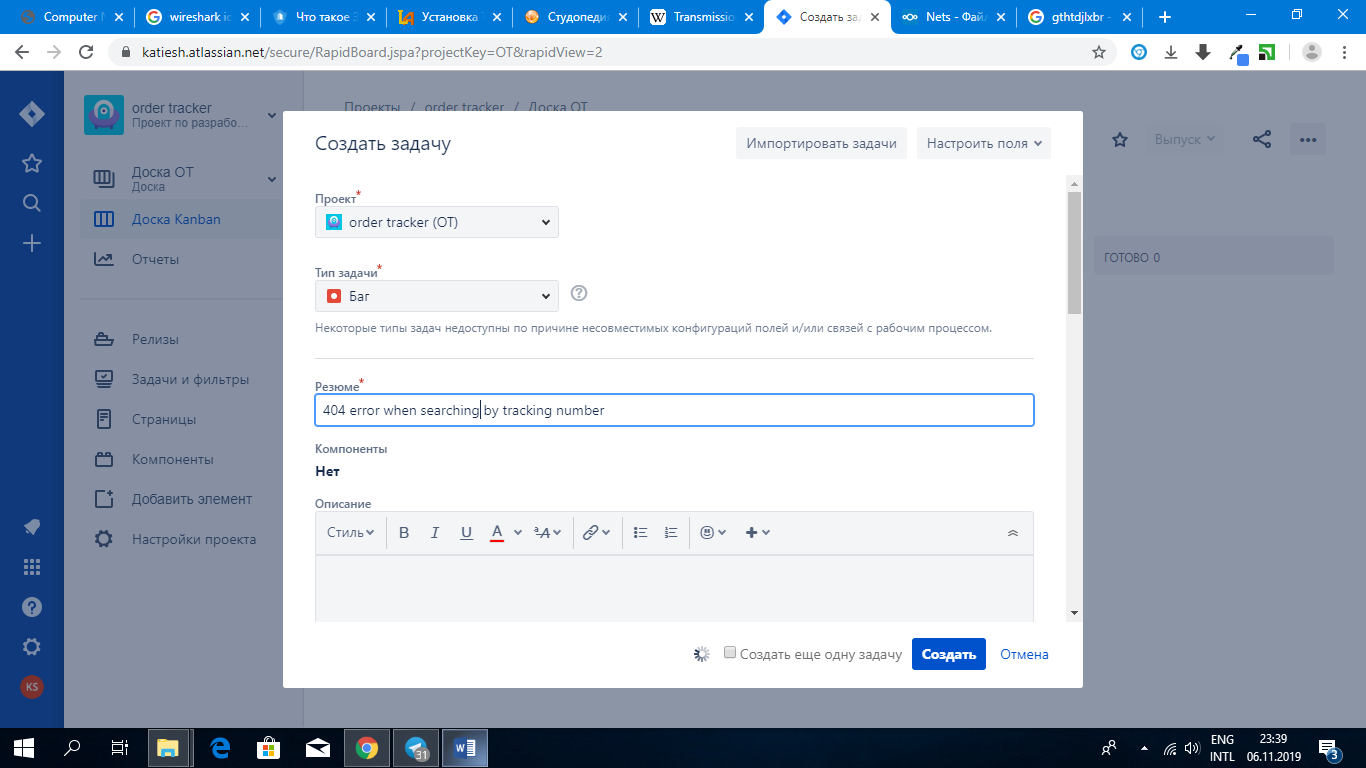
Мета роботи: Вивчити типові життєві цикли розробки програмного забезпечення. Усвідомити методику роботи з багтрекінгом, навчитися створювати питання, задавати їх статус, а також навчитися оперувати статистикою багтрекінга.

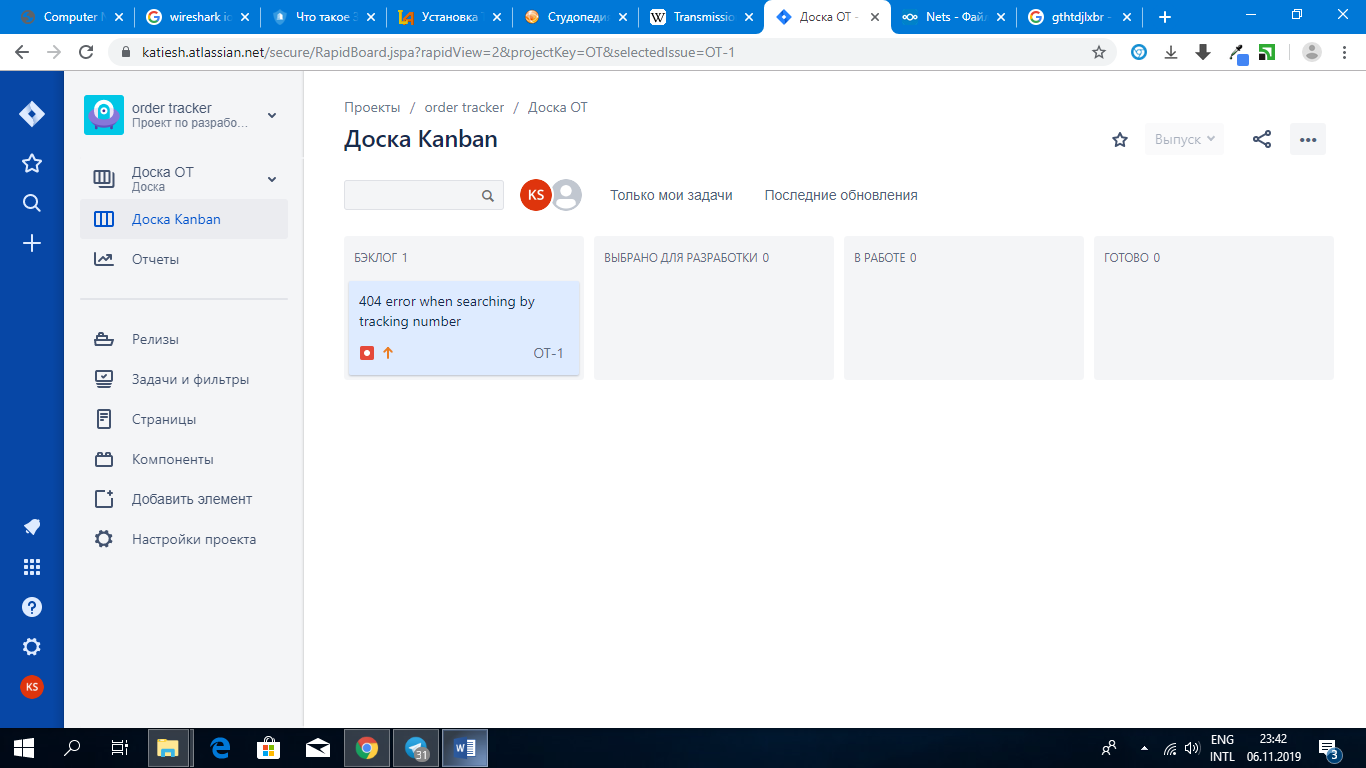
Задача: Вивчити зміст прикладу інформаційної системи та створити опис життєвого циклу розробки відповідного програмного забезпечення. Використовуючи систему багтрекінга MantisBT або подібну необхідно ознайомитися веденням журналу для повного циклу розробки та тестування, відтворити статуси для питання (ticket) та ролі у процесі розробки програмного забезпечення.

Завдання

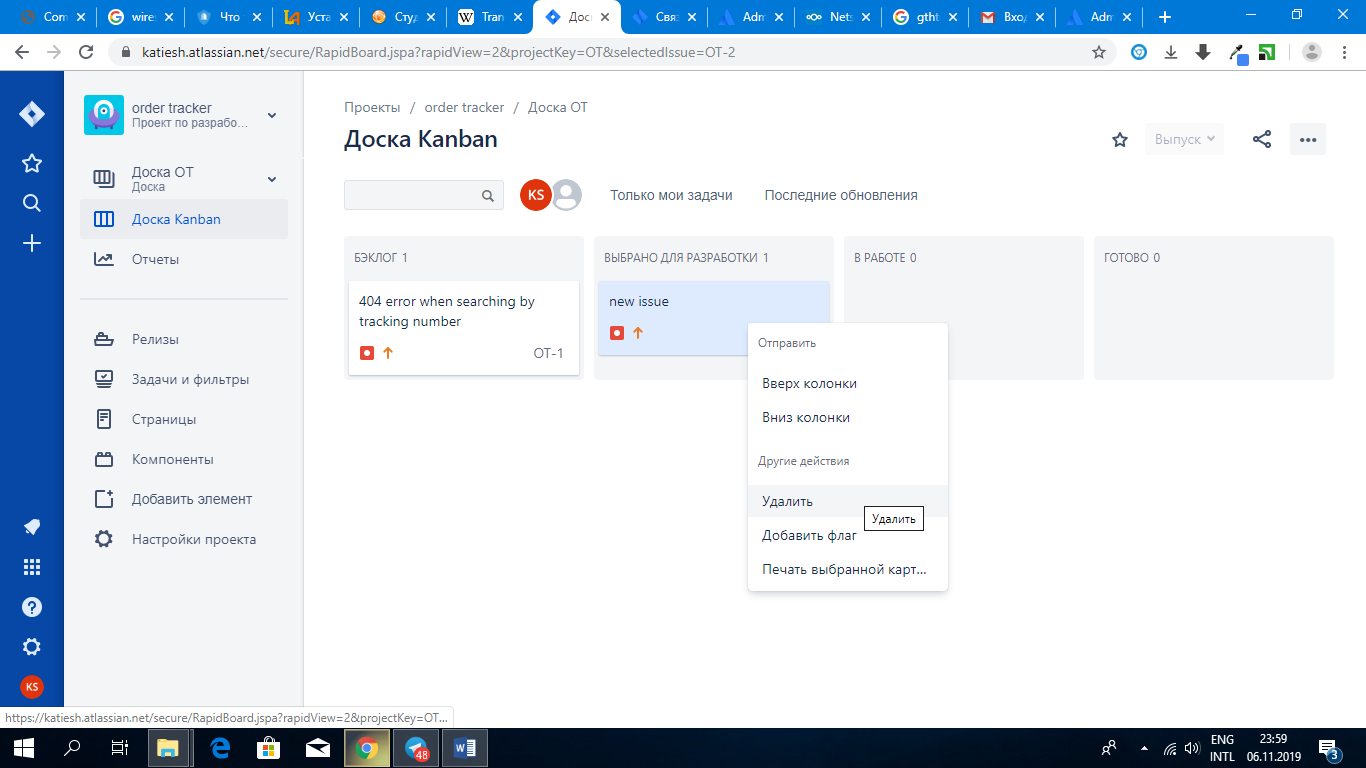
Використовуючи систему багтрекінга MantisBT або подібну необхідно виконати наступні дії:

- створити питання (ticket)

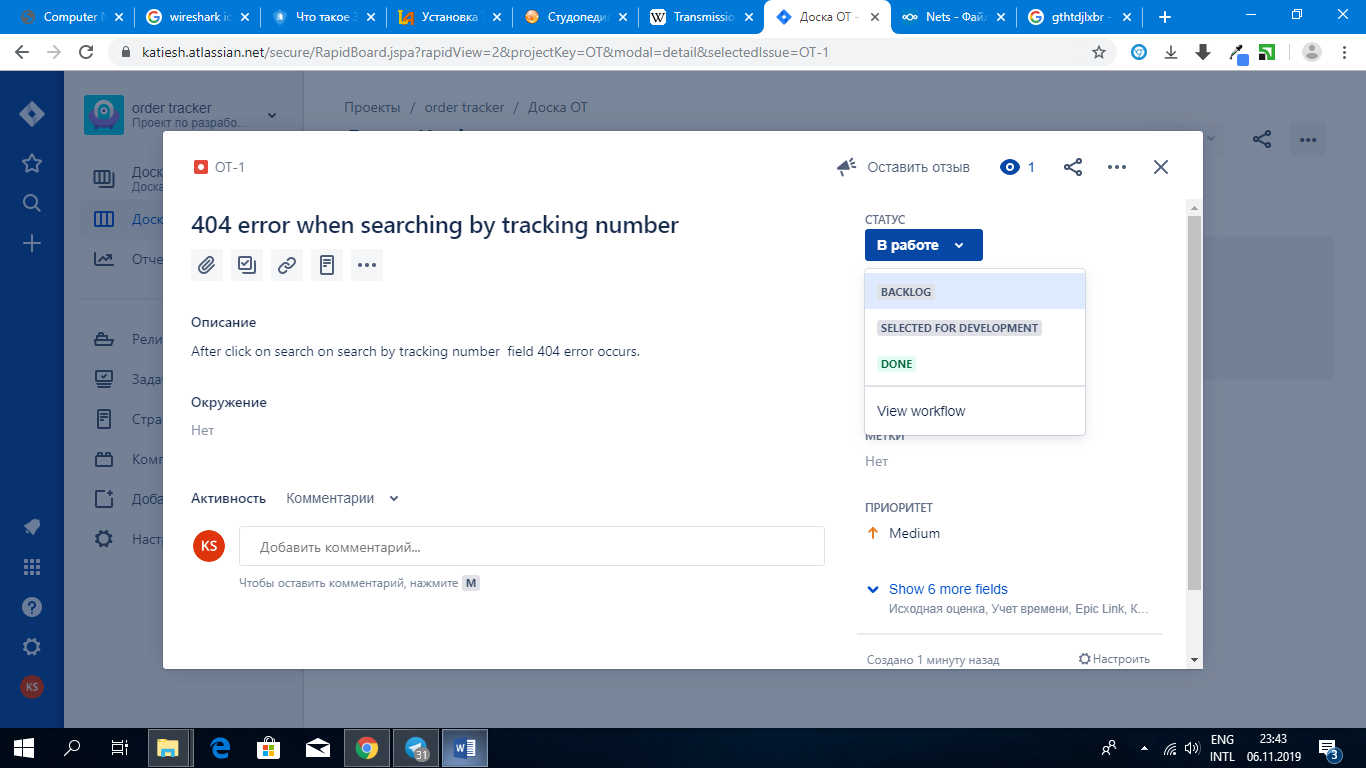




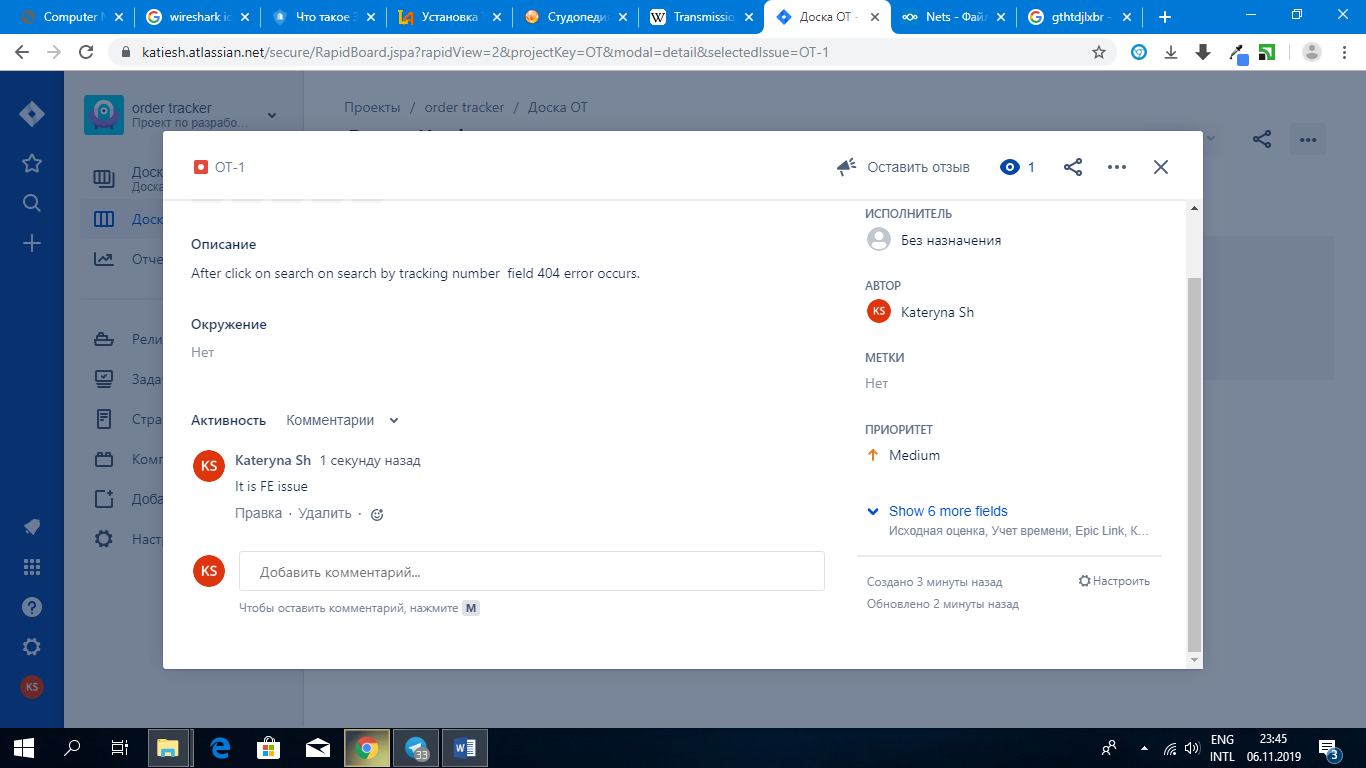
- видалити питання



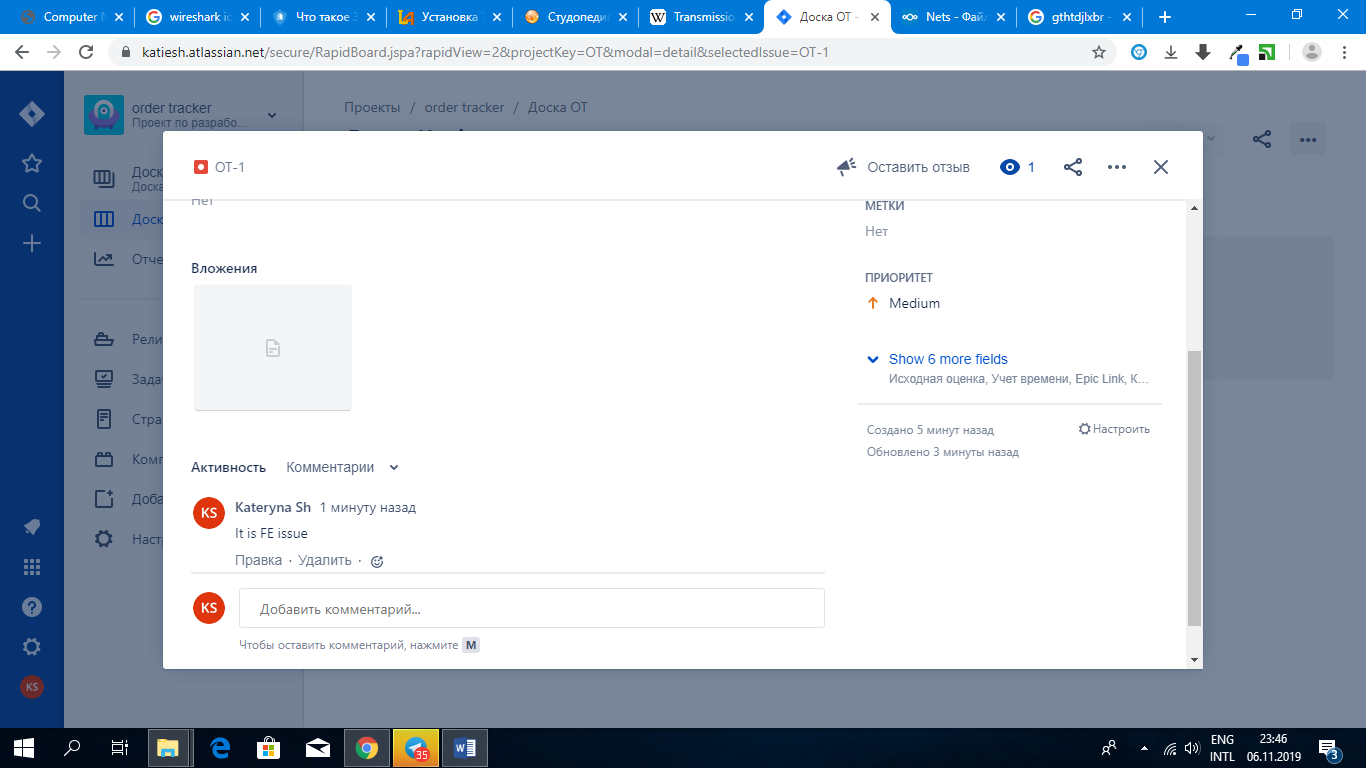
- змінити статус



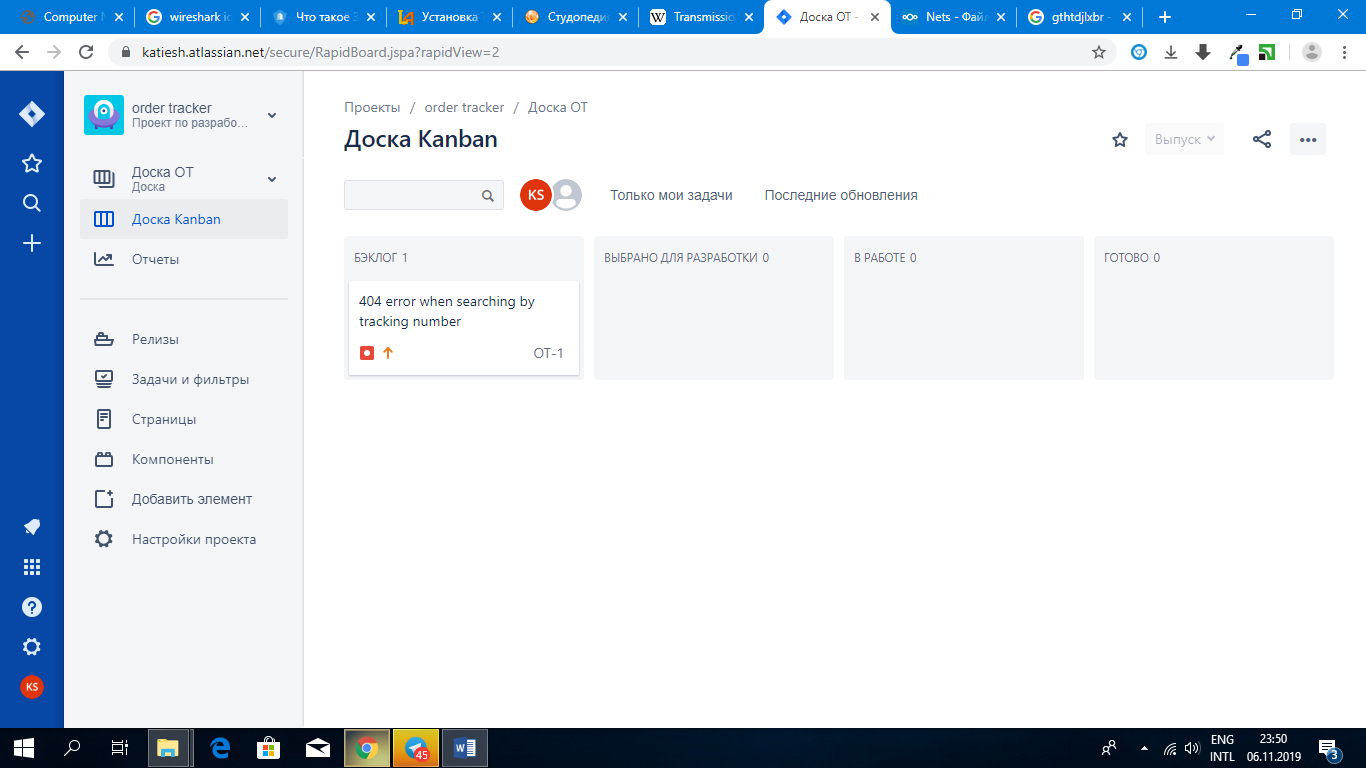
- додати коментар

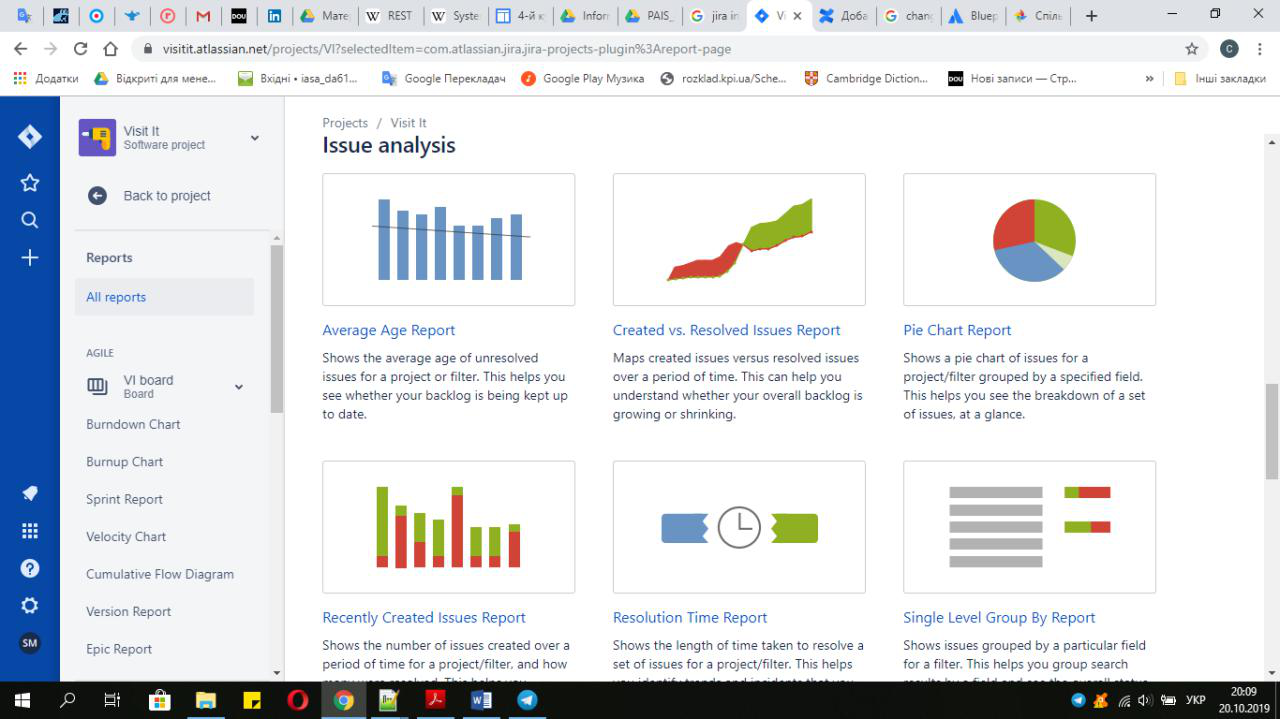


- прикріпити файл



- ознайомитися з журналом змін



- ознайомитися з виведенням статистики.

**Створити опис життєвого циклу розробки програмного забезпечення інформаційної системи.**

Розробка ітераціями відображає об'єктивно існуючий спіральний цикл створення системи. Неповне завершення робіт на кожному етапі дозволяє переходити на наступний етап, не чекаючи повного завершення роботи на поточному. При ітеративному способі розробки відсутню роботу можна буде виконати на наступній ітерації. Головне ж завдання - щонайшвидше показати користувачам системи працездатний продукт, тим самим активізуючи процес уточнення і доповнення вимог.

Виходячи з можливості внесення змін, як в процес, так і в проміжний продукт було створено спіральну модель ЖЦ.

Внесення змін орієнтоване на задоволення потреби користувачів одразу, як тільки буде встановлено, що створені артефакти або елементи документації не відповідають дійсному стану розробки.

Дана модель ЖЦ допускає аналіз продукту на витку розробки, його перевірку, оцінку правильності та прийняття рішення про перехід на наступний виток або повернення на попередній виток для доопрацювання на ньому проміжного продукту.

Відмінність цієї моделі від каскадної полягає в можливості багато разів повертатися до процесу формулювання вимог і до повторної розробки версії системи з будь-якого процесу моделі.

1. Інженерія вимог

#### Проектування програмного забезпечення

#### Конструювання програмного забезпечення

#### Тестування програмного забезпечення

#### Супровід програмного забезпечення

#### Вивчити маршрут проходження питання від моменту створення до моменту закриття.

#### Вивчити ролі керівника проекту, архітектора, програміста і тестувальника в маршруті проходження питання.

#### Результат пошуку зображень за запитом bug tracking issue life cycle

#### Результат пошуку зображень за запитом bug tracking issue life cycle

#### Частина 2. Розробка поетапного плану проекту.

#### Мета роботи: Скласти і описати поетапний план проекту.

#### Завдання

#### 1) Опис об'єкту взяти з SRS в лабораторній роботі No 1.

#### 2) Використовувати методологію створення програмного забезпечення: upper/lower design methods.

In the [software development process](https://en.wikipedia.org/wiki/Software_development_process), the **top-down** and **bottom-up** approaches play a key role.

Top-down approaches emphasize planning and a complete understanding of the system. It is inherent that no coding can begin until a sufficient level of detail has been reached in the design of at least some part of the system. Top-down approaches are implemented by attaching the stubs in place of the module. This, however, delays testing of the ultimate functional units of a system until significant design is complete. Bottom-up emphasizes coding and early testing, which can begin as soon as the first module has been specified. This approach, however, runs the risk that modules may be coded without having a clear idea of how they link to other parts of the system, and that such linking may not be as easy as first thought. [Re-usability of code](https://en.wikipedia.org/wiki/Code_reuse) is one of the main benefits of the bottom-up approach.

Top-down design was promoted in the 1970s by [IBM](https://en.wikipedia.org/wiki/IBM) researchers [Harlan Mills](https://en.wikipedia.org/wiki/Harlan_Mills) and [Niklaus Wirth](https://en.wikipedia.org/wiki/Niklaus_Wirth" \o "Niklaus Wirth). Mills developed [structured programming](https://en.wikipedia.org/wiki/Structured_programming) concepts for practical use and tested them in a 1969 project to automate the [*New York Times*](https://en.wikipedia.org/wiki/The_New_York_Times) morgue index. The engineering and management success of this project led to the spread of the top-down approach through IBM and the rest of the computer industry. Among other achievements, Niklaus Wirth, the developer of [Pascal programming language](https://en.wikipedia.org/wiki/Pascal_programming_language), wrote the influential paper *Program Development by****Stepwise Refinement***. Since Niklaus Wirth went on to develop languages such as [Modula](https://en.wikipedia.org/wiki/Modula) and [Oberon](https://en.wikipedia.org/wiki/Oberon_(programming_language)) (where one could define a module before knowing about the entire program specification), one can infer that top-down programming was not strictly what he promoted. Top-down methods were favored in [software engineering](https://en.wikipedia.org/wiki/Software_engineering) until the late 1980s, and [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) assisted in demonstrating the idea that both aspects of top-down and bottom-up programming could be utilized.

Modern software design approaches usually combine both top-down and bottom-up approaches. Although an understanding of the complete system is usually considered necessary for good design, leading theoretically to a top-down approach, most software projects attempt to make use of existing code to some degree. Pre-existing modules give designs a bottom-up flavor. Some design approaches also use an approach where a partially functional system is designed and coded to completion, and this system is then expanded to fulfill all the requirements for the project

#### 3) Створити поетапний план для об'єкта проектування за допомогою OpenProj або аналогічного програмного забезпечення.

#### 4) Оформити діаграму Ганта для етапів проекту.

#### 

#### 

#### 5) Оформити для кожного етапу проекту список ризиків, їх вплив на проект та ймовірність виникнення.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *Ризик* | Причини | Етап | Вплив на проект | Ймовірність виникнення |
| *Ризики поганої взаємодії між замовником і виконавцем* | це ризики, пов'язані з відсутністю комунікації між виконавцем і замовником або їх представниками. Недостатнє обговорення завдань або архітектури може негативно позначитися на ПЗ що розробляється. | 2 | Значний | Середня |
| *Ризики управління проектом* | це ризики, пов'язані з відсутністю навичок проектного менеджменту у менеджера проекту, а також з відсутністю інтересу або мотивації у нього. | 2-6 | Значний | Середня |
| *Ризики, пов'язані з недостатньою обізнаністю керуючого проектом про точний стан проекту* | це вид ризиків, пов'язаних з відсутністю зворотного зв'язку. Він виникає, коли проектний менеджер не вибудував робочий процес таким чином, щоб контролювати хід виконання проекту на всіх його етапах. | 3-6 | Незначний | Низька |
| *Ризики планування* | це ризики, які можуть бути пов'язані з відсутністю навичок планування за проектом як менеджером, так і виконавцями, якщо вони готують інформацію про терміни виконання робіт. | 2-6 | Середній | Середня |
| *Ризики відсутності системи контролю* | обумовлені великою кількістю аспектів у області проектного менеджменту при розробці ПЗ, коли складно врахувати всі можливі ситуації. | 3-5 | Середній | Висока |
| *Ризик появи нових вимог* | виникає в процесі розробки ПЗ, коли з'являються все нові і нові [вимоги](https://uk.wikipedia.org/wiki/%D0%92%D0%B8%D0%BC%D0%BE%D0%B3%D0%B8_%D0%B4%D0%BE_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE_%D0%B7%D0%B0%D0%B1%D0%B5%D0%B7%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%BD%D1%8F" \o "Вимоги до програмного забезпечення), які відсувають терміни і оцінку конкретних завдань. | 2 | Середній | Висока |
| *Ризик суперечливості у вимогах (декомпозиція специфікації)* | це ризики, пов'язані з виявленням суперечливості в вимогах замовника на етапі програмування або інтеграції проекту. | 1-2 | Середній | Низька |
| *Ризики неправильно сформульованих певних системних вимог* | це ризики, коли на самому початку проекту були некоректно сформульовані характеристики цільової системи, для якої розробляється програмне забезпечення: програмне оточення (операційна система, встановлені компоненти, сервіси і т.п.) або вимоги до апаратної частини (частота процесора, об'єм жорсткого диска, обсяг оперативної пам'яті і т.п.). | 1-2 | Значний | Висока |
| *Ризики використання нестабільних технологій* | це ризики, пов'язані з використанням нових технологій, які ще не пройшли апробацію у виробництві або інших проектах. | 3-5 | Значний | Низька |
| *Ризики, пов'язані з нездатністю впоратися зі складністю проекту* | іноді проект може бути настільки складним, що команда просто може з ним не упоратися. | 1-4 | Значний | Низька |
| *Ризик низької продуктивності* | обумовлений тривалістю реалізації проекту. Це на самому початку проекту створює велику втрату часу, яку складно буде надолужити. При цьому доводиться або переносити терміни, або працювати в більш динамічному режимі на більш пізніх етапах проекту. | 3-5 | Середній | Середня |
| *Ризик зміни співробітників* | коли проект покидають ключові співробітники, які максимально володіють інформацією. | 4-5 | Середній | Середня |
| *Ризики розкрадання вихідного коду* | виникають, коли розробники, йдучи з компанії, забирають з собою проект, що розробляється ними і, трохи модифікувавши вихідний код, можуть продати його або використовувати в інших проектах, наприклад, у конкурентів. | 4-5 | Значний | Низька |
| *Ризики порушення закону про авторське право* | можуть виникнути при використанні розробниками без відома проектного менеджера чужого вихідного коду, алгоритму або бібліотеки, які захищені законом про авторське право, але не придбано або їх використання не узгоджене з автором. | 4-5 | Значний | Низька |
| * *Ризики фінансових обмежень* | можуть виникнути як з вини менеджера, який планував бюджет проекту, так і з інших причин. | 1-6 | Значний | Середня |
| * *Ризики зміни кон'юнктури ринку* | обумовлені зміною економічної ситуації, яка складалася на ринку при плануванні. При цьому могли закладатися фактори, актуальні на момент планування, а їх зміна не була врахована. | 1-6 | Значний | Висока |
| * *Валютні ризики* | це ризики, пов'язані з можливим виникненням збитків або додаткових доходів внаслідок несприятливої або сприятливої зміни курсів іноземних валют | 1-6 | Значний | Низька |

#### 6) Для кожного ризику розробити список анти-ризикових заходів.

|  |  |
| --- | --- |
| *Ризик* | *Протиризиковий захід* |
| *Ризики поганої взаємодії між замовником і виконавцем* | *Ретельний підбір виконавця і узгодження ключових питань заздалегіть* |
| *Ризики управління проектом* | *Співпраця з відповідальним і досвідченим проджект менеджером* |
| *Ризики, пов'язані з недостатньою обізнаністю керуючого проектом про точний стан проекту* |
| *Ризики планування* | *Правильне та грамотне формулювання технічної документації вимог* |
| *Ризик суперечливості у вимогах (декомпозиція специфікації)* |
| *Ризики неправильно сформульованих певних системних вимог* |
| *Ризик появи нових вимог* |
| *Ризики використання нестабільних технологій* | *Обговорення і використання стабільних технологій, узгодження всіх технологій з архітектором чи іншим досвідченим співробітником* |
| *Ризики відсутності системи контролю* | *Ефективний менеджмент проекту і конкурентна спроможність на фоні ринку праці* |
| *Ризики, пов'язані з нездатністю впоратися зі складністю проекту* |
| *Ризик низької продуктивності* |
| *Ризик зміни співробітників* |

#### Висновки: в ході виконання лабораторної роботи було розроблено поетапний план проекту, створено діаграму Ганта, розбито проект на етапи, розроблено відповідну документацію. Для кожного етапу розраховано ризики та розроблено список анти-ризикових заходів. А також вивчено типові життєві цикли розробки програмного забезпечення. Ми усвідомили методику роботи з багтрекінгом, навчилися створювати питання, задавати їх статус, а також навчитися оперувати статистикою багтрекінга.