МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ  
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»  
(ФГБОУ ВО «ВГУ»)

Факультет прикладной математики, информатики и механики

Кафедра программного обеспечения и администрирования  
информационных систем

**Разработка расширения для Jira для планирования и контроля работ**

Бакалаврская работа

Направление 02.03.03 Математическое обеспечение и администрирование информационных систем

Профиль Информационные системы и базы данных

|  |  |  |  |
| --- | --- | --- | --- |
| Зав. кафедрой | \_\_\_\_\_\_\_\_\_\_ | д. ф.-м. н., проф. | М. А. Артемов \_\_\_.\_\_\_ 2019 г. |
| Обучающийся | \_\_\_\_\_\_\_\_\_\_ |  | Л. В. Цаплина |
| Руководитель | \_\_\_\_\_\_\_\_\_\_ | ст. преп. | В. М. Мельников |

Воронеж 2019**Аннотация**

Работа посвящена разработке расширения Plan для системы отслеживания выполнения задач Jira. Данное расширение позволяет планировать работу на длительный период времени, отображать задачи плана в виде деревьев и отслеживать отклонения от плана.

Целью работы является программная реализация набора функций: JQL-функция для извлечения задач плана, создание и сравнение снимков деревьев задач, реорганизация задач посредством перемещения задач внутри дерева, автоматическая корректировка оценок, поиск последних измененных задач плана, выявление и вывод ошибок плана.

Программа была создана на базе Jira 7.2.4 с использованием языка программирования Java 8, инструмента для разработки плагинов Atlassian SDK и построена на архитектуре Model-View-Controller (MVC). Для работы с базой данных использовалась библиотека Active Objects, основанная на ORM-технологии.

**Содержание**

[Введение 5](#_Toc10318532)

[1. Постановка задачи 5](#_Toc10318533)

[2. Анализ задачи 7](#_Toc10318534)

[2.1. Анализ существующих решений 7](#_Toc10318535)

[2.2. Анализ реализованной части приложения 11](#_Toc10318536)

[2.3. Анализ поставленной задачи 14](#_Toc10318537)

[3. Средства реализации 19](#_Toc10318538)

[4. Требования к аппаратному и программному обеспечению 20](#_Toc10318539)

[5. Интерфейс пользователя 21](#_Toc10318540)

[5.1. Создание снимков задач плана 21](#_Toc10318541)

[5.2. Сравнение снимков версии плана 22](#_Toc10318542)

[5.3. Отображение последних измененных задач 24](#_Toc10318543)

[5.4. Использование JQL-функции поиска задач версии плана 25](#_Toc10318544)

[5.5. Панель для автоматической корректировки оценки родительской задачи 26](#_Toc10318545)

[5.6. Ошибки плана 28](#_Toc10318546)

[5.7. Перемещение задач 29](#_Toc10318547)

[6. Реализация 32](#_Toc10318548)

[6.1. Структура проекта 32](#_Toc10318549)

[6.2. Снимки деревьев задач плана 34](#_Toc10318550)

[6.3. Система ошибок плана 38](#_Toc10318551)

[6.4. JQL-функция planIssues() 39](#_Toc10318552)

[6.5. Автоматическая корректировка оценки 40](#_Toc10318553)

[7. План тестирования 41](#_Toc10318554)

[Заключение 49](#_Toc10318555)

[Литература 50](#_Toc10318556)

[Приложение 1. Листинг класса «SnapshotsComparator» 52](#_Toc10318557)

# Введение

На сегодняшний день система отслеживания ошибок и управления проектами Jira остается наиболее популярной. Однако она не располагает средством для удобного контроля большого потока создаваемых задач при выполнении работы, запланированной на некоторый длительный период, например, на полгода. Для руководителей проектов является актуальной задача разработки плагина, позволяющего определить набор основных направлений работы (статей) и соответствующие им задачи, задать для статей предварительные плановые оценки и контролировать процесс выполнения плана, изменения числа подзадач и объема работ в течение длительного периода. Предполагается, что статьи плана в процессе работы будут детализироваться с помощью подзадач на многих уровнях, и руководитель может рассматривать план только на верхних уровнях, оценивая общий масштаб и результаты и не вдаваясь в детали работы. Особенно данная функциональность полезна для руководителей длительных проектов с большой командой исполнителей. В связи с этим, возникла идея разработать расширение для Jira, позволяющее:

1. Создавать планы, задавая для них название, администратора и наблюдателей, тип связи задач для создания многоуровневой структуры, JQL-фильтр выбора корневых задач или, по-другому, статей.
2. Создавать версии планов, имеющих период и статус.
3. Задавать для статей плановые оценки.
4. Проверять соответствие имеющихся ресурсов команды исполнителей запланированному объему работ.
5. Просматривать все задачи плана на одной странице в иерархическом виде (в виде деревьев).
6. Контролировать отклонения от плана по различным критериям. Например, выявлять превышение плановых оценок фактическими трудозатратами и такие ошибки, как возникновение циклов в дереве задач или появление отчетов исполнителей в задачах, не входящих в план.
7. Рассылать уведомления об ошибках плана по электронной почте заинтересованным лицам.

Было создано расширение, предоставляющее первые пять пунктов из приведенного выше списка. Данная работа включает разработку функции отслеживания отклонений от плана. Однако, при использовании расширения была выявлена проблема, заключающаяся в невозможности определить, в какой момент и из-за каких конкретно задач суммарная оценка статьи или фактически затраченное время, включающее трудозатраты подзадач, стало превышать время, запланированное на статью в начале работы. Поэтому требуется разработать решение этой проблемы с помощью инструмента для создания снимков деревьев задач, сравнения их между собой, а также сравнения текущего состояния деревьев со снимком и автоматическое выявление таким образом изменений, вызвавших отклонения от плана. Кроме того, еще одним вспомогательным средством выявления последних изменений будет функция определения последних измененных задач в дереве или поддереве.

Другая проблема заключается в том, что для того, чтобы избежать возникновения превышения плановых оценок статей суммами оценок дочерних задач, следует уменьшать оценку родительской задачи при добавлении в план новых оцененных дочерних задач или увеличении оценок существующих. Однако, вручную это делать неудобно. Поэтому возникла идея разработки функции автоматической корректировки родительской оценки.

Для корректной работы необходимым является обнаружение различных ошибок. Это могут быть превышения плановых оценок статей или ошибки пользователей, такие как появление отчетов исполнителей в задачах вне плана, появление циклов в дереве задач.

Чтобы применять стандартные возможности расширенного поиска Jira, такие как фильтрация, сортировка, суммирование и другие, необходима JQL-функция получения задач плана.

Для того, чтобы пользователь не был вынужден запоминать, как он хочет изменить связи между задачами, переходить на страницы задач и вручную удалять и добавлять их, нужно создать возможность легко и наглядно изменять связи между задачами с помощью перетаскивания задач мышью на странице плана.

Таким образом, создание расширения с описанными функциями является актуальной задачей, а ее решение – необходимым для сферы управления большими проектами.

# Постановка задачи

Реализовать модули Jira–расширения, позволяющие автоматизировать выявление изменений, вызвавших отклонения от плана, извлекать задачи плана специальным фильтром, перемещать задачи внутри деревьев, автоматически корректировать оценки родительских задач, выявлять и выводить ошибки плана. Для этого необходимо разработать набор инструментальных и алгоритмических средств, позволяющих:

* проверять план на возникновение ошибок:
  + превышение плановой оценки суммой оценок задач поддерева;
  + превышение плановой оценки суммой фактических трудозатрат задач поддерева;
  + возникновение циклов в дереве;
  + появление отчетов исполнителей в задачах, не входящих в план;
  + изменение названия статьи плана;
  + изменение состава статей;
* автоматически уменьшать оценку родительской задачи на столько же, на сколько увеличилась оценка задачи в ее поддереве;
* создавать снимок деревьев задач плана по желанию пользователя;
* автоматически создавать снимки и удалять устаревшие;
* выбирать снимки для сравнения и критерии сравнения;
* выводить отличия снимков по заданным критериям;
* выводить для любой не терминальной задачи список последних измененных задач внутри поддерева;
* перетаскивать задачи мышью для изменения связей между ними;
* при использовании расширенного поиска Jira находить задачи заданного плана или его части.

Разрабатываемое расширение должно поддерживать локализацию интерфейса на русском и английском языке и позволять разработчику расширять набор доступных языков без модифицирования исходного кода.

# Анализ задачи

## 2.1. Анализ существующих решений

В результате исследовательской работы был изучен рынок существующих решений, схожих с плагином Plan. Рассмотрим найденные решения по критерию наличия интересующих в рамках решаемой задачи функций.

### 2.1.1. Simplified Planner for Jira

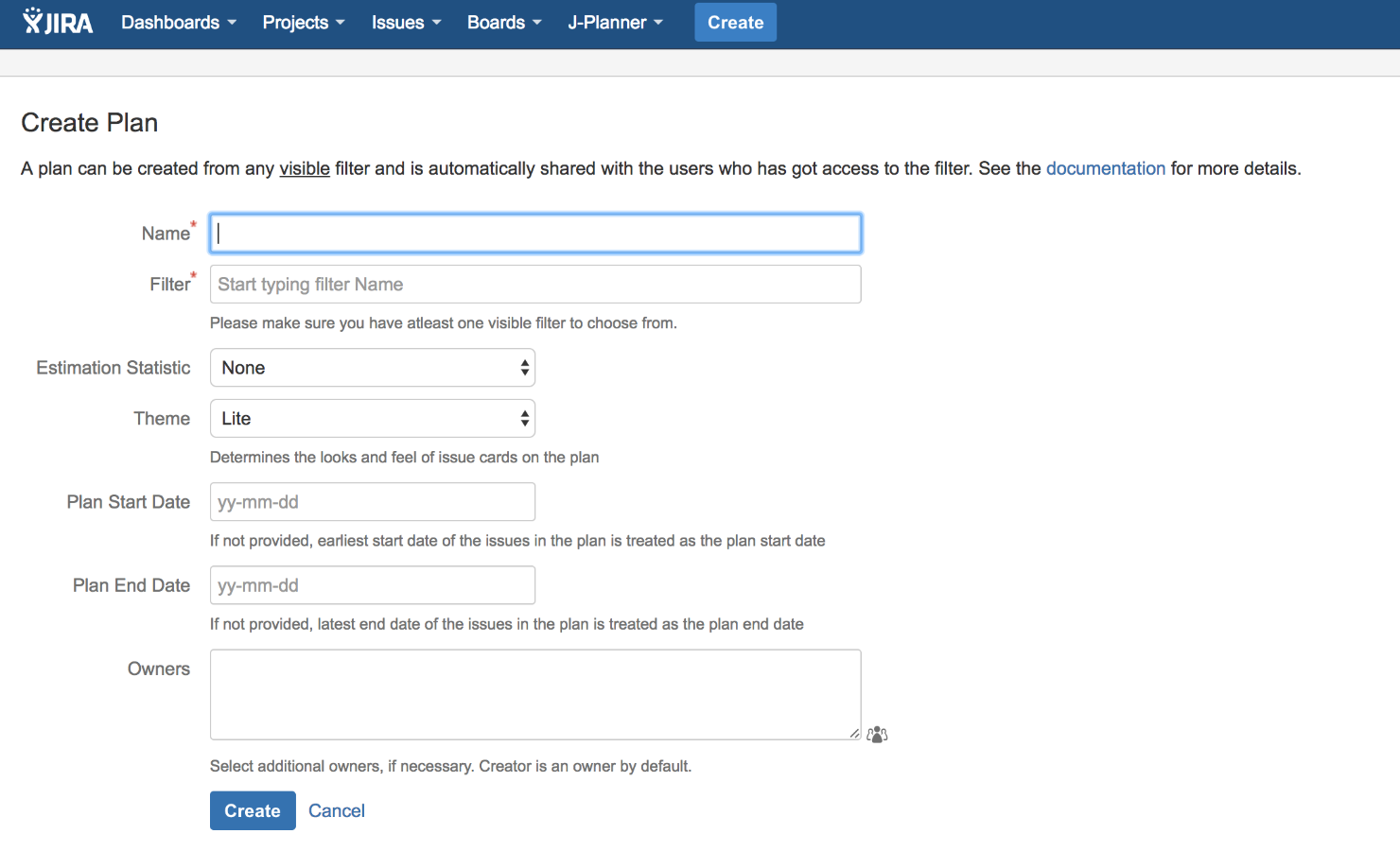
На рис. 2.1 показана страница плагина «Simplified Planner for Jira».

Рис. 2.1. Страница создания плана плагина «Simplified Planner for Jira»

Плагин обладает возможностью создавать планы с заданными датами начала и конца и фильтром для выбора задач плана. Есть функция просмотра задач в рамках времени с отметкой важных дат и текущего времени, отслеживания нарушения сроков выполнения задач. Есть функция перемещения задач мышкой для изменения их дат начала и конца, исполнителя.

Возможность связать задачи плана в многоуровневую структуру и просматривать в иерархическом виде отсутствует. Поэтому основную проблему данный плагин не решает и дальнейший анализ не имеет смысла.

### 2.1.2. Plangle

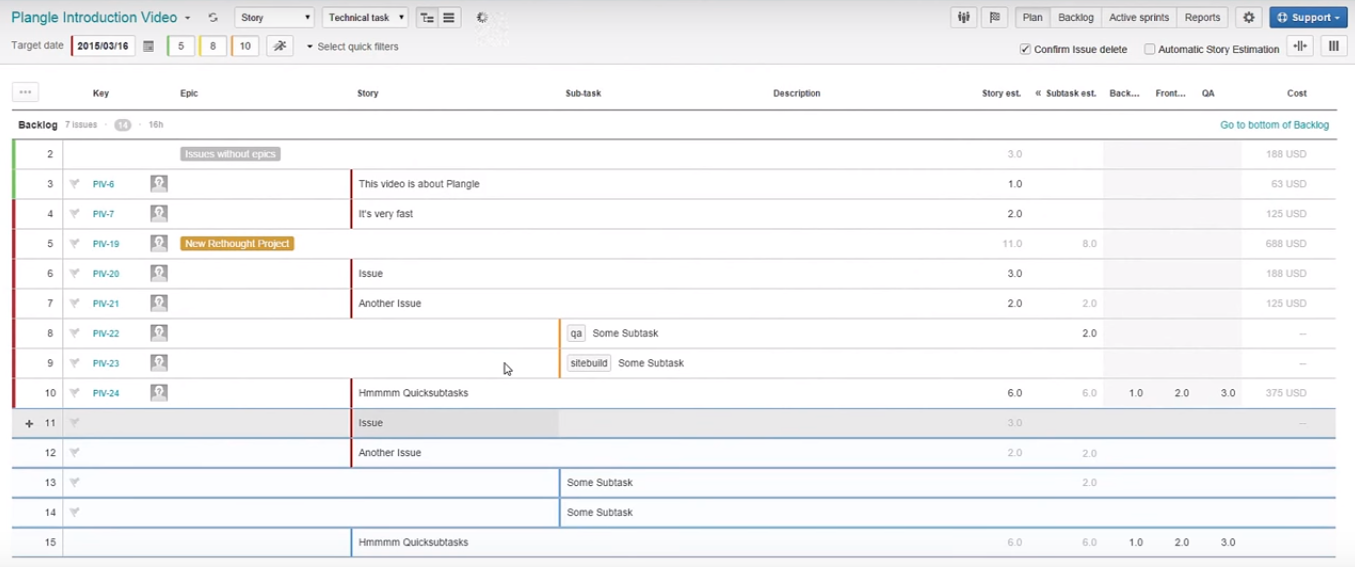
На рис. 2.2 представлена главная страница расширения «Plangle».

Рис. 2.2. Главная страница расширения «Plangle»

«Plangle» имеет отображение задач в виде иерархии с возможностью перетаскивания задач мышкой для их реорганизации и сортировки. Есть подсчет сумм оценок дочерних задач для родительских. Имеется возможность отслеживания выполнения спринтов, имеющих даты начала и конца, однако спринты обычно рассчитаны на короткий период времени. Кроме того, иерархия имеет только три уровня: уровень эпика и уровни задач и подзадач, как встроенных типов задач Jira. Jira не позволяет создавать подзадачи для подзадач. Поэтому данный плагин не может позволять многоуровневую структуру без ограничения по количеству уровней и дальнейший анализ не имеет смысла.

### 2.1.3. Easy Issue Organizer for Jira

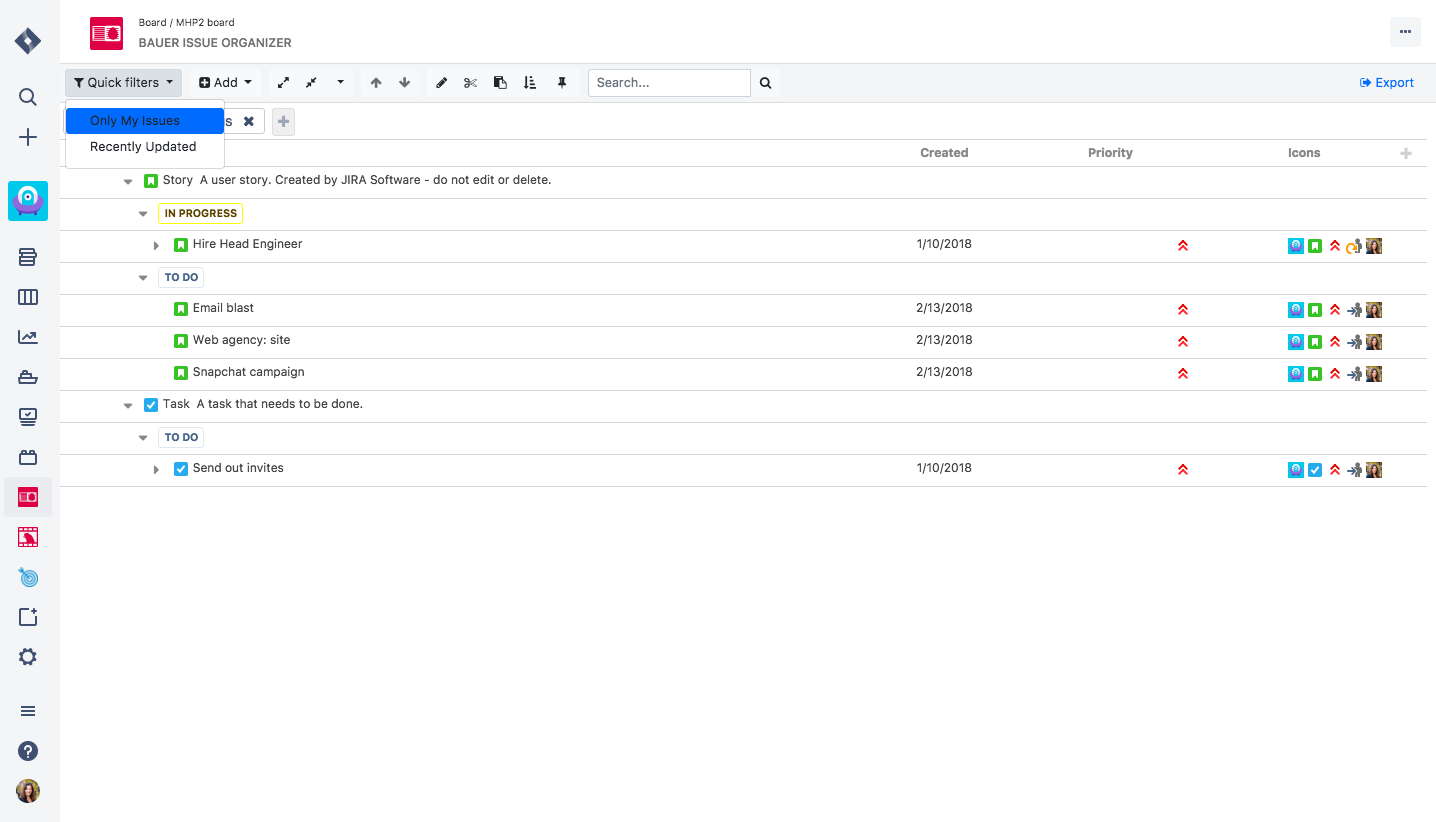
На рис. 2.3 показана страница расширения «Easy Issue Organizer for Jira».

Рис. 2.3. Страница расширения «Easy Issue Organizer for Jira»

Данное расширение позволяет выводить двухуровневую иерархию задач и подзадач, напротив каждой есть визуальное отображение прогресса выполнения. Имеется функция перетаскивания задач для реорганизации структуры.

### Issue Archiver for Jira

На рис. 2.4 представлена страница расширения «Issue Archiver for Jira».

Рассматриваемый плагин позволяет архивировать задачи, созданные в задаваемый период. Эта функция схожа с решаемой задачей по созданию снимков задач плана. Однако расширение не предполагает функцию автоматического сравнения снимков.

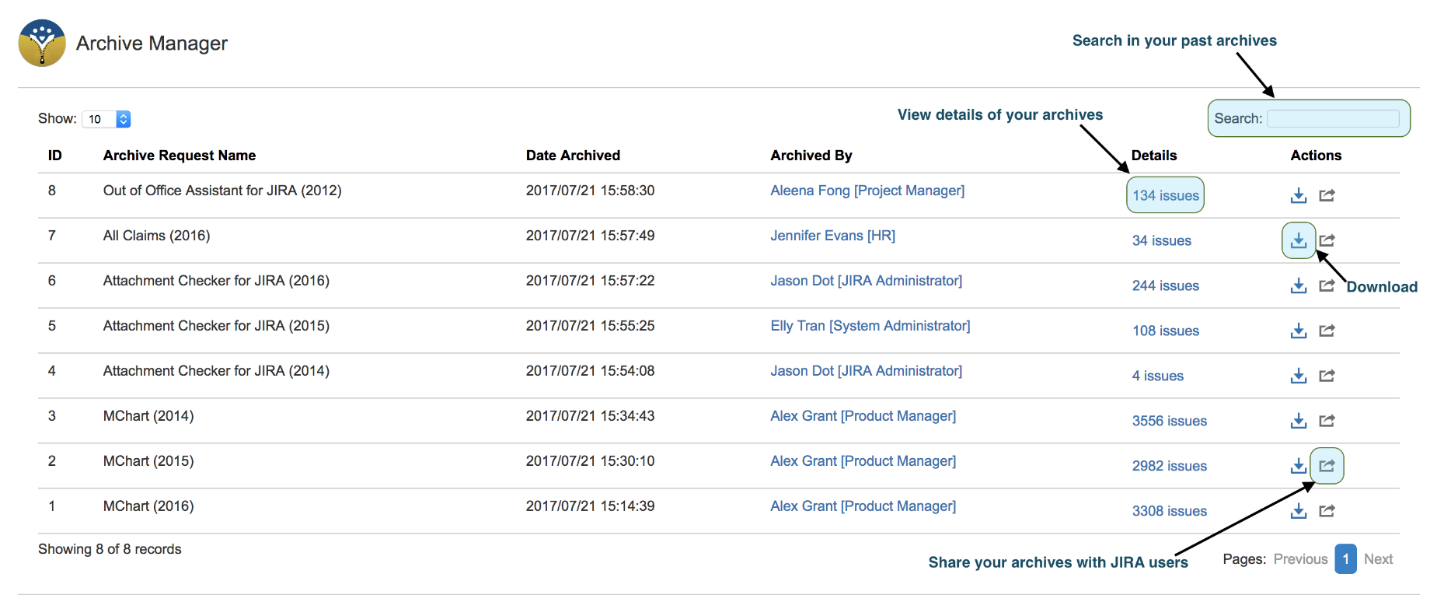


Рис. 2.4. Страница расширения «Issue Archiver for Jira»

### Links Hierarchy for Agile Cloud

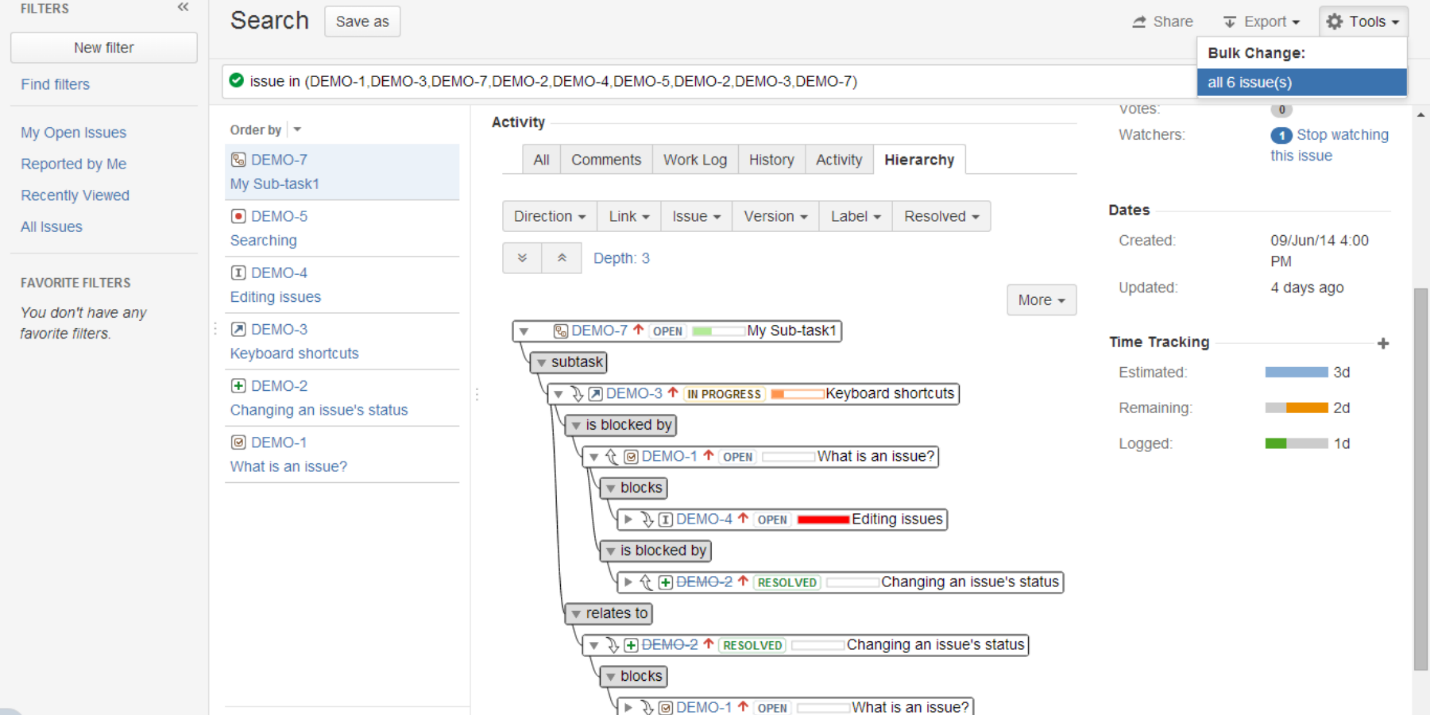
На рис. 2.5. показана страница расширения «Links Hierarchy for Agile Cloud».

Рис. 2.5. Страница расширения «Links Hierarchy for Agile Cloud»

Рассматриваемое расширение позволяет отобразить задачи, выбранные JQL-фильтром, в иерархическом виде со всеми связями между ними. Есть экспорт данных в формате Excel файла. С помощью данного плагина можно наблюдать структуру задач, но нельзя вместе с тем решать проблемы планирования, а делать это с помощью отдельного инструмента неудобно.

### Результат анализа существующих решений

В результате анализа рынка существующих расширений для Jira было выявлено, что ни одно из них не позволяет в полной мере решить проблему планирования работы на длительный период времени и контроля за выполнением плана на разных уровнях детализации. Кроме того, все выше перечисленные расширения являются платными. Можно отметить популярность функции перетаскивания задач для реорганизации их структуры.

## 2.2. Анализ реализованной части приложения

Поставленная задача должна решаться в рамках уже созданного приложения.

Приложение построено с использованием архитектурного стиля REST (Representational State Transfer) и состоит из модулей: клиент, сервер и база данных. На рис. 2.6 представлена схема, отображающая взаимодействие модулей приложения.

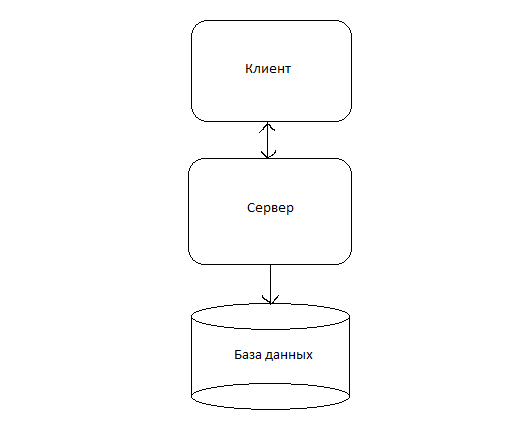


Рис. 2.6. Взаимодействие модулей приложения

Приложение также реализует шаблон проектирования MVC (Model-View-Controller), который подразумевает наличие трех компонентов:

* Модель, которая отвечает за связь с базой данных и реализацию методов объекта, получаемого из базы данных;
* Представление, отвечающее за отображение пользовательского интерфейса;
* Контроллер, являющийся связующим звеном между моделью и представлениями, управляющий бизнес-логикой веб-приложения.

Структура шаблона MVC представлена на рис. 2.7.

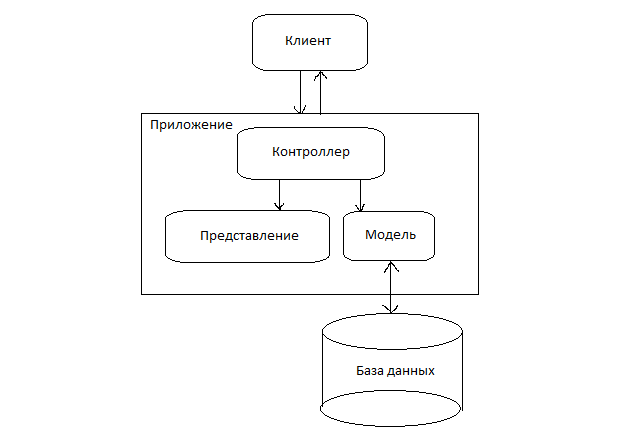


Рис. 2.7. Структура MVC

Клиентская часть приложения использует фреймворк Backbone.js, позволяющий определить модели и связать с ними представления, взаимодействующие с моделями с помощью событий. На рис. 2.8. представлена UML-диаграмма артефактов компонента клиентской части приложения, содержащего JavaScript-файлы, отвечающие за инициализацию элементов интерфейса трех страниц, и используемые ими JavaScript-файлы, определяющие и реализующие backbone-модели и представления.

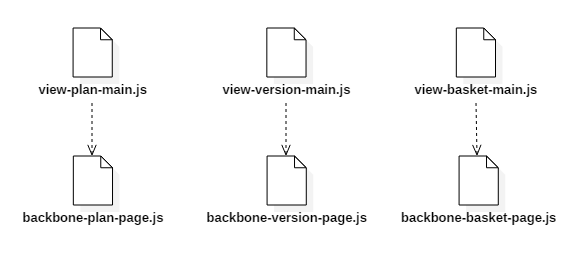


Рис. 2.8. Диаграмма артефактов компонента клиентской части приложения, содержащего JavaScript-файлы

На рис. 2.9. представлена UML-диаграмма (Unified Model Language) пакетов серверной части приложения.



Рис. 2.9. UML-диаграмма пакетов серверной части приложения

Пакет «ao» содержит подпакеты «entity», включающий классы моделей, и «service», содержащий классы, отвечающие за бизнес-логику и обращающиеся к базе данных. В результате анализа структуры серверной части было решено провести рефакторинг, отделив логику работы с базой данных от бизнес-логики приложения путем выделения кода по работе с БД (база данных) из сервисов в специальные DAO-классы (Data Access Object). Также, т. к. дальнейшая работа потребует создания новых DTO-объектов (Data Transfer Object), требуется переместить их в собственный пакет. Кроме того, классы сервисов и DAO-классы должны реализовывать интерфейсы, используемые вместо конкретных реализаций для большей гибкости кода.

## 2.3. Анализ поставленной задачи

Для реализации поставленной задачи необходимо решить следующие подзадачи:

1. спроектировать и создать таблицы базы данных, содержащие информацию о снимках и задачах снимков;
2. предоставить пользовательский интерфейс, позволяющий включать/отключать функцию автоматического создания снимков, создавать снимки вручную, выводить отличия снимков;
3. реализовать вывод заданного количества последних измененных задач поддерева любой задачи;
4. реализовать сохранение и удаление снимков;
5. разработать алгоритм сравнения снимков деревьев задач плана, а также сравнения текущего состояния дерева со снимком;
6. реализовать проверку плана на наличие ошибок, вывод ошибок в интерфейсе и рассылку информации об ошибках по электронной почте;
7. создать модуль JQL-функции и разработать алгоритм поиска задач заданного плана или поддерева дерева задач заданного плана;
8. предоставить возможность перетаскивания задач внутри дерева плана мышью для изменения связей между задачами;
9. разработать алгоритм автоматической корректировки оценки родительской задачи при добавлении или увеличении оценки дочерней задачи.

### Снимки деревьев задач

Для создания снимков вручную предполагается добавление специальной кнопки в панель управления на странице задач плана. Следует сохранить информацию о дате и времени создания снимка. Также требуется возможность добавить пользовательское название или комментарий к снимку, но данную информацию можно сделать необязательной. На серверной части приложения нужно реализовать создание и хранение снимков и задач в базе данных.

Автоматическое создание снимков потребует реализации на сервере функции, вызываемой по расписанию ежедневно. Кроме того, в пользовательский интерфейс стоит поместить настройку, позволяющую включить или отключить данную опцию. Кроме создания, снимки должны автоматически удаляться для того, чтобы избежать слишком большого объема занимаемой ими памяти. Так как для пользователя важность снимков уменьшается с течением времени, удалять снимки нужно таким образом, чтобы всегда имелись ежедневные снимки за последнюю неделю, еженедельные за последний месяц и ежемесячные за остальной период плана.

Задача сравнения снимков делится на реализацию пользовательского интерфейса и алгоритма сравнения на серверной части приложения. В интерфейсе потребуются элементы для выбора двух снимков, множественного выбора критериев сравнения и кнопки сравнения. После получения результата сравнения следует отобразить отличия. Критериев сравнения на данный момент требуется четыре: состав дочерних задач, изменение названия задачи, изменение плановой оценки и изменение суммарной оценки, где плановая оценка – это оценка статьи (т. е. корневой задачи), задаваемая пользователем, а суммарная оценка – это сумма первоначальных оценок самой задачи и всех задач из ее поддерева. Саму процедуру сравнения деревьев можно реализовать с помощью рекурсивного алгоритма сравнения задач каждого уровня.

Для задачи сравнения текущего состояния со снимком можно использовать тот же интерфейс, что и для сравнения снимков между собой. Для этого добавим в список для выбора первого снимка элемент «Текущее состояние» и сделаем его выбранным по умолчанию. Реализация серверной части может быть выполнены двумя способами:

* вызвать код создания снимка, не сохраняя его в базу данных, а затем воспользоваться функцией сравнения снимков;
* вызвать построение дерева задач, передав обработчик построения очередного узла, выполняющий сравнение с соответствующей задачей снимка.

Преимущество первого варианта в переиспользовании функции сравнения снимков и экономии времени на разработку новой функции. Преимущество второго варианта в скорости работы алгоритма. Так как план может включать достаточно большое число задач, скорость обработки запроса на сравнение является существенным критерием, поэтому был сделан выбор в пользу второго варианта реализации.

#### Структура данных

Все необходимые для работы функции сравнения снимков данные можно разделить на два блока:

1. Данные о снимках.
2. Данные о задачах снимков.

Первый блок должен включать следующую информацию: идентификатор версии плана, для которой сделан снимок, идентификатор пользователя, создавшего снимок, дату и время создания, название.

Второй блок должен содержать следующие данные: идентификатор снимка, к которому принадлежит задача, идентификатор задачи, плановая оценка, суммарная оценка, сумма фактических трудозатрат, название задачи, идентификатор родительской задачи.

На основе данного анализа можно сделать вывод, что потребуется создание двух таблиц реляционной базы данных, связанных отношением один-ко-многим.

### Анализ подзадачи автоматической корректировки оценки

Данная подзадача предполагает автоматическое уменьшение оценки родительской задачи при увеличении оценки дочерней задачи на ту же величину. Для этого потребуется создать обработчик событий добавления задач в план и обновления задачи. Jira 7.2.4 предоставляет API для реализации обработчиков создания и обновления задач. Однако в данной версии Jira отсутствует возможность написать обработчик добавления связи между задачами, благодаря чему задача так же появится в плане. Поэтому эта часть функции не может быть реализована до использования более новой версии Jira. Таким образом, полностью автоматическая корректировка оценки может выполняться не всегда. Для пользователя может быть неудобно помнить о том, для каких случаев функция работает, а для каких — нет. Поэтому было решено создать панель на странице каждой задачи с кнопкой, по нажатию которой можно выбрать задачу из списка родительских задач для автоматического вычитания из ее первоначальной оценки и остатка оценки остатка текущей задачи.

# Средства реализации

Для реализации поставленной задачи использовались следующие средства:

* Atlassian Plugin SDK – набор программных инструментов для создания, установки и сборки плагина.
* H2 – встроенная реляционная система управления базами данных.
* Java 8 – язык программирования для серверной части приложения.
* IntelliJ IDEA 2018.1.1. Данная среда предоставляет удобный интерфейс для разработки приложений на Java.
* JavaScript – язык программирования для реализации клиентской части приложения.
* jQuery – библиотека для упрощения программирования клиентской части.
* Backbone.js – фреймворк для удобной разработки клиентской части приложения.
* Active Objects – Java-библиотека, основанная на ORM-технологии. Осуществляет связь Java-интерфейсов с таблицами базы данных, предоставляет средства для автоматического построения запросов для извлечения данных.
* Maven – инструмент для сборки Java-проекта, интегрированный со средой разработки IntelliJ IDEA. Maven также позволяет управлять зависимостями, разрешать конфликты версий и легко переходить на новые версии библиотек.

Выбор средств продиктован спецификой сферы разработки.

# Требования к аппаратному и программному обеспечению

Созданное приложение предназначено для работы на сервере приложения Jira 7.1.9 или более поздней версии. Требования к браузеру для работы клиентской части приложения совпадают с требованиями для использования Jira.

Размер необходимого пространства зависит от размера хранимых данных. Размер самого приложения составляет 610 Кб.

# Интерфейс пользователя

## 5.1. Создание снимков задач плана

Интерфейс для создания снимка вручную содержит кнопку «Сделать снимок». На рис. 5.1 представлен фрагмент страницы задач плана, содержащий кнопку «Сделать снимок».

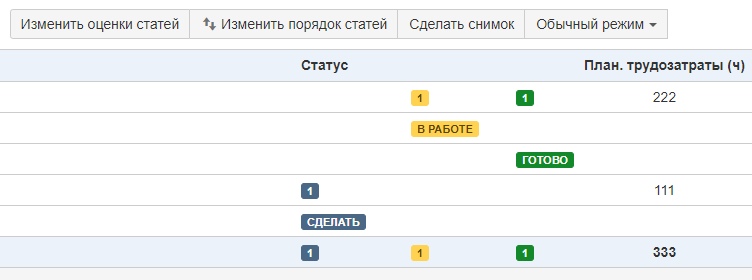


Рис.5.1. Фрагмент страницы задач плана, содержащий кнопку «Сделать снимок»

После нажатия на рассмотренную кнопку появляется диалоговое окно создания снимка. Внешний вид данного окна представлен на рис. 5.2.

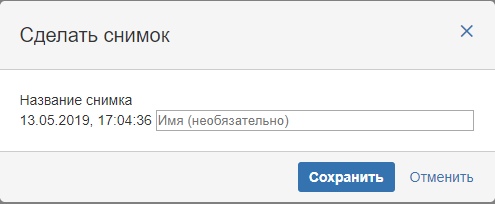


Рис. 5.2. Окно создания снимка

Окно создания снимка содержит информацию о том, что название снимка будет автоматически включать дату и время создания. Также на нем располагается поле для ввода необязательного названия/комментария пользователя и кнопки «Сохранить» и «Отмена».

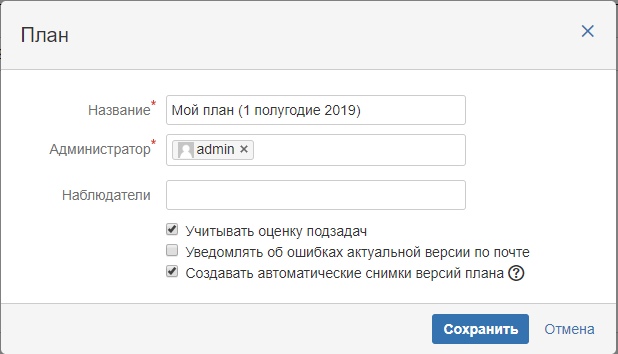
Интерфейс функции автоматического создания снимков содержит флажок «Автоматически создавать снимки плана» в окне настроек плана с иконкой вопроса, при наведении на которую появляется всплывающая подсказка с информацией о режиме удаления устаревших снимков. Окно настроек плана с описанным флажком показано на рис. 5.3.

Рис. 5.3. Окно настроек плана

## 5.2. Сравнение снимков версии плана

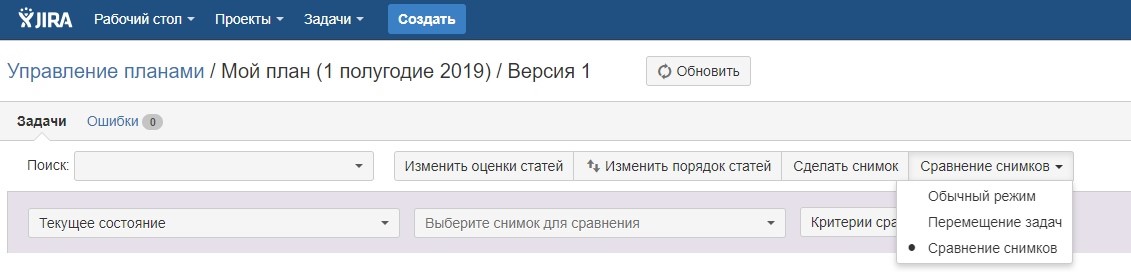
Для сравнения снимков нужно в выпадающем списке режимов отображения таблицы задач версии плана выбрать режим «Сравнение снимков», как показано на рис. 5.4.

Рис.5.4. Выбор режима «Сравнение снимков»

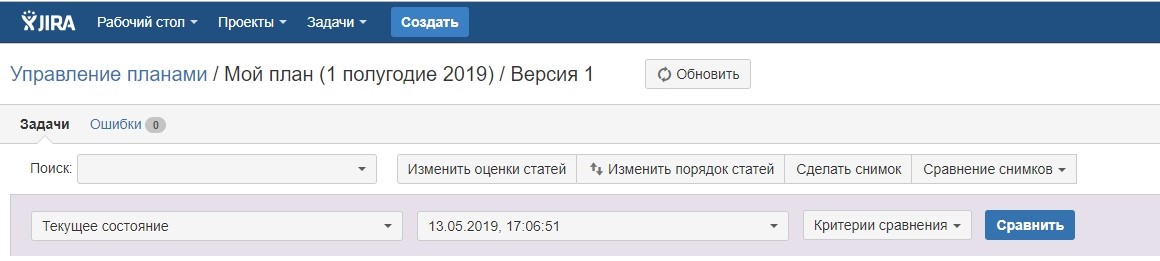
После переключения в режим сравнения снимков страница приобретает вид, изображенный на рис. 5.3.

Рис.5.5. Страница задач после перехода в режим сравнения

Для сравнения и вывода результатов пользователю следует выбрать снимки (или пункт «Текущее состояние» для сравнения текущего состояния со снимком) и критерии сравнения из выпадающего списка критериев с возможностью множественного выбора, а затем нажать кнопку «Сравнить», вызывающую выполнение процесса сравнения и отображение результатов. Результаты сравнения представлены на рис. 5.6.

 Рис.5.6. Результаты сравнения снимков

Для отображения результатов сравнения выводится иерархическая структура задач версии плана. При выбранном критерии «Совокупность задач», новые задачи отмечены зеленым цветом названия, а удаленные – красным. Если был выбран критерий сравнения «Названия задач», то оранжевый цвет названия задачи означает, что оно было изменено. При наведении на него появится всплывающая подсказка со старым названием. Справа от названия каждой задачи есть значки с двойной стрелкой в круге, означающие наличие добавлений, удалений или изменений в поддереве данной задачи. Изменившиеся плановые или суммарные оценки так же отмечены оранжевым цветом и имеют всплывающую подсказку со старым значением.

## Отображение последних измененных задач

Количество последних измененных задач задается в окне настроек для каждой версии плана. По умолчанию устанавливается значение «5». На рис. 5.7. показано окно настроек с полем для количества последних измененных задач.

Рис. 5.7. Поле для ввода количества последних измененных задач

На странице версии плана для задач, имеющих превышение суммы оценок трудозатратами и имеющие дочерние задачи, отображается вплывающая иконка в виде часов, при наведении на которую осуществляется поиск и вывод последних измененных подзадач. Их вывод показан на рис. 5.8.

Рис. 5.8. Вывод списка последних измененных задач

## Использование JQL-функции поиска задач версии плана

На рис. 5.9 показано использовано JQL-функции поиска задач версии плана. Для поиска нужно в строку расширенного поиска ввести «issues in planIssues(<параметры>)», где параметры следующие:

1. Название плана (строка, обязательный параметр).
2. Номер версии плана (число, необязательный параметр). Если номер версии не задан, то поиск будет осуществляться в актуальной версии, о чем будет отображено предупреждение на странице поиска.
3. Ключ задачи плана (строка, необязательный параметр). Если данный параметр задан, то результат поиска будет содержать задачи поддерева задачи с данным ключом.

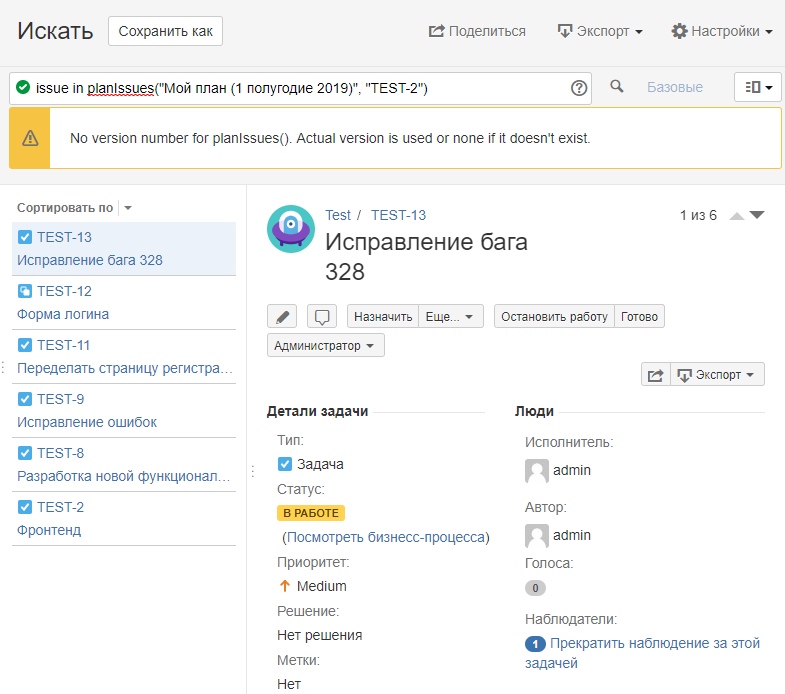
Второй и третий параметры могут быть переданы в любом порядке. При неверном числе или типе параметров выводится сообщение об ошибке.

Рис. 5.9. Результат поиска с использованием JQL-функции planIssues()

## Панель для автоматической корректировки оценки родительской задачи

На рис. 5.10 представлена страница задачи актуальной версии плана, содержащая выделенную зеленой рамкой панель с информацией о принадлежности задачи актуальной версии плана и кнопкой «Корректировать оценку родительской задачи».

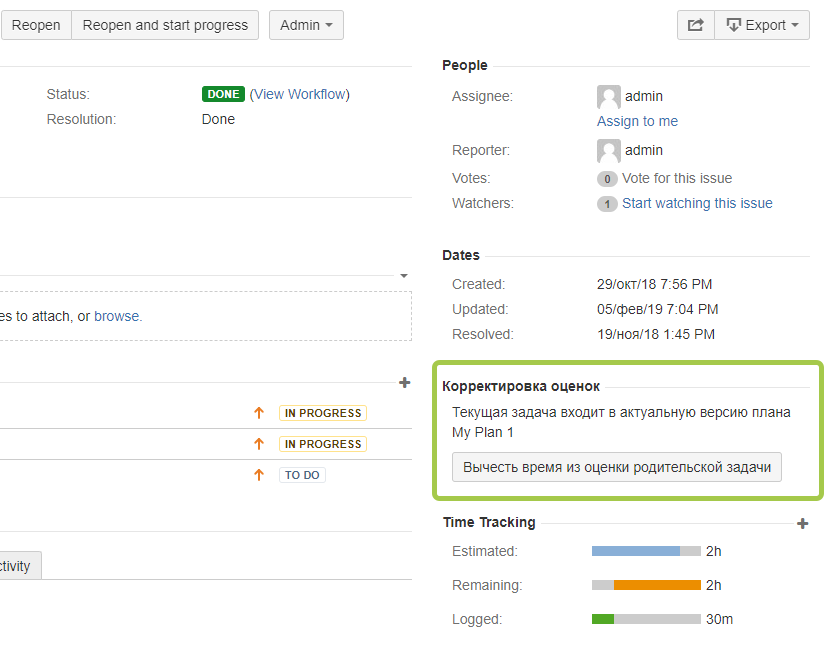


Рис. 5.10. Панель плагина на странице задачи

По нажатию на эту кнопку открывается диалоговое окно со списком родительских задач, представленное на рис. 5.11. Для выбора доступны только задачи, имеющие оценку не менее остатка времени текущей задачи. При нажатии на выбранную родительскую задачу ее оценка будет уменьшена на величину, равную остатку времени текущей задачи.

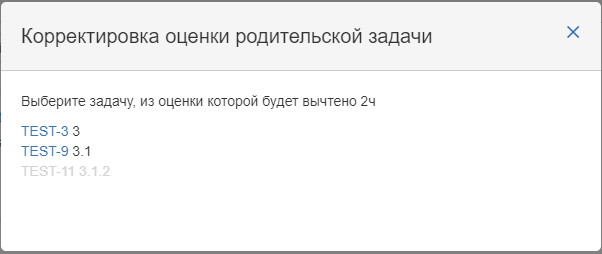


Рис. 5.11. Окно выбора родительской задачи

## 5.6. Ошибки плана

Ошибки плана выводятся на вкладке «Ошибки», расположенной на странице версии плана и представленной на рис. 5.12. Ошибки появляются здесь при загрузке страницы.

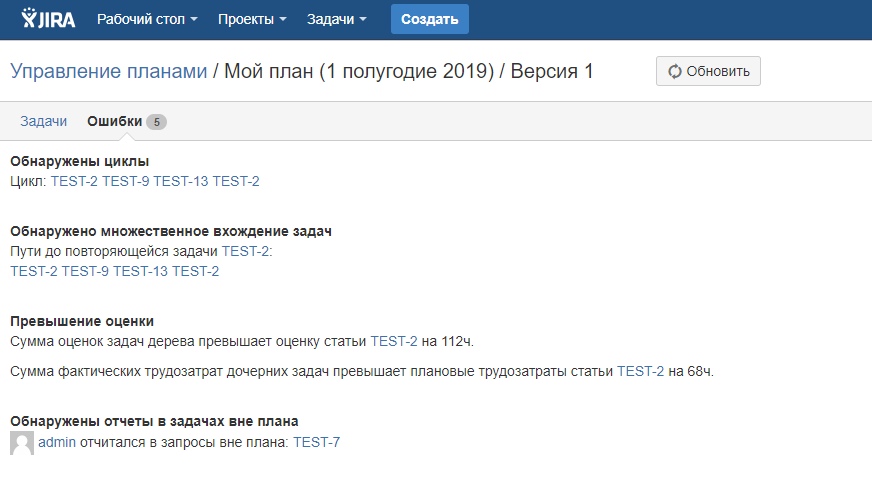


Рис. 5.12. Вкладка «Ошибки»

Кроме того, уведомление о наличии ошибок в плане присылается по электронной почте. Образец электронного письма представлен на рис. 5.13. Письмо содержит информацию об ошибках и ссылку на версию плана.

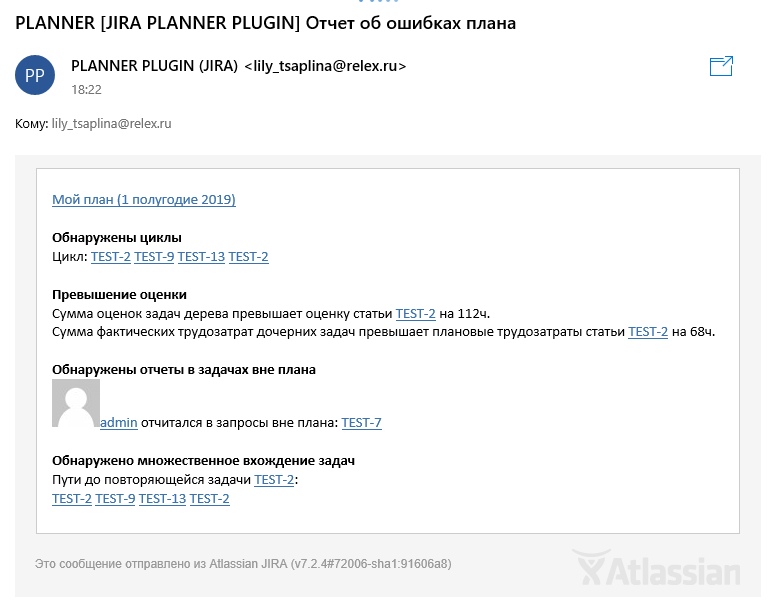


Рис. 5.13. Образец электронного письма с уведомлением об ошибках

## 5.7. Перемещение задач

Для перемещения задачи требуется переключить режим отображения задач плана на «Перемещение задач», как показано на рис. 5.14.

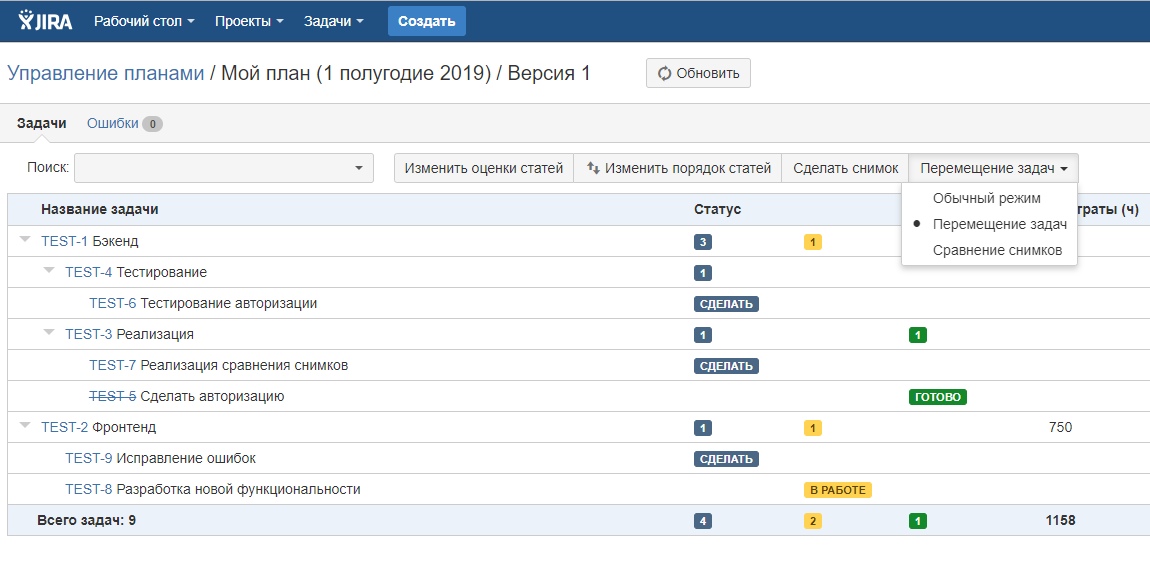


Рис. 5.14. Переключение в режим «Перемещение задач»

В результате появится возможность мышкой перетаскивать задачи. Процесс перетаскивания задачи показан на рис. 5.15.

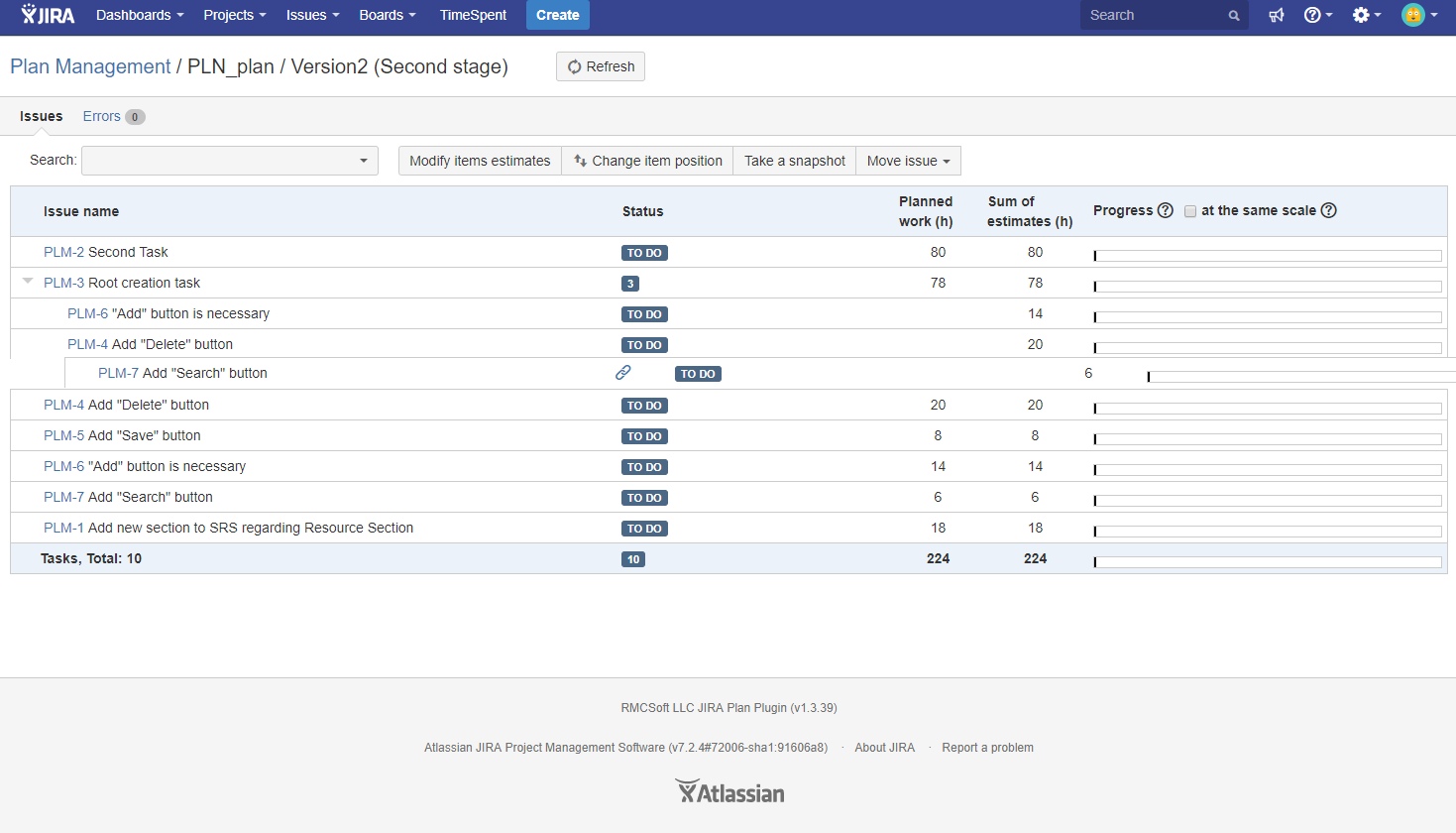


Рис. 5.15. Процесс перетаскивания задачи «PLM-7»

При отпускании задачи производится определение новой родительской задачи, удаление связи с прежней родительской задачей и добавление связи с новой родительской задачей. Новый уровень перетаскиваемой задачи определяется, исходя из совпадения левого отступа задачи и отступа остальных задач.

# Реализация

## 6.1. Структура проекта

Рассмотрим реализацию задач в рамках проекта в целом. На рис. 6.1 представлена диаграмма пакетов приложения.

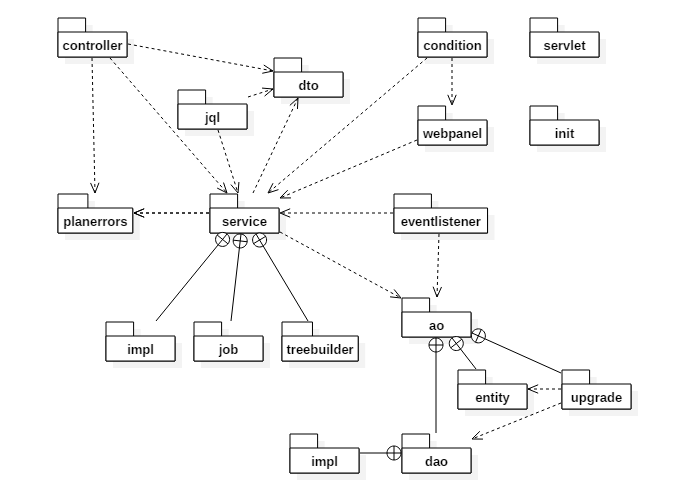


Рис. 6.1. Диаграмма пакетов проекта

Пакет контроллер содержит классы, отвечающие за обработку HTTP-запросов, распределяемым по нужным обработчикам сервлетом фреймворка Spring DispatcherServlet. Собственные сервлеты находят в пакете «servlet». Пакет «service» теперь не входит в пакет «ao», так как в классах сервиса осталась только бизнес-логика, а логика по работе с БД вынесена в классы пакеты «dao». Также были созданы новые пакеты:

* «upgrade» – содержит классы, отвечающие за обновление структуры или данных в БД.
* «job» – содержит классы, вызывающие функции по расписанию. Здесь размещены классы для ежедневной проверки актуальных планов на наличие ошибок и рассылки результатов по электронной почте и для автоматического создания снимков деревьев задач.
* «treebuilder» – содержит логику по построению дерева задач и некоторые связанные вспомогательные функции.
* «service::impl» – содержит реализацию интерфейсов сервисов.
* «dao::impl» – содержит реализацию интерфейсов DAO-классов.
* «eventlistener» – содержит классы, обрабатывающие события Jira. А нем размещается класс для автоматической корректировки оценки родительской задачи при добавлении или обновлении задач плана.
* «planerrors» – содержит классы, представляющие собой ошибки плана.
* «webpanel» – содержит класс для работы с веб-панелями. В нем находится класс для веб-панели на странице задачи, созданной для функции автоматической корректировки оценки родительской задачи.
* «dto» – в этот пакет были перенесены и добавлены классы, представляющие собой объекты передаваемые между клиентом и сервером.
* «jql» – данный пакет содержит классы для работы с собственными JQL-функциями. В нем размещен класс для работы JQL-функции planIssues().

Для реализации функции перемещения задач по иерархии задач плана использовались существующие классы контроллера и сервиса, в которые были добавлены метода для удаления старой связи задачи и создания новой. Основная сложность легла на сторону фронтенд-части приложения, где было решено использовать библиотеку JQueryUI. JavaScript код реализации приведен в приложении.

## 6.2. Снимки деревьев задач плана

### 6.2.1. Модель данных

Расширенная модель сущностей-связей (EER) в нотации IDEF1X, описывающая структуру части базы данных, относящейся к хранению снимков, создана с помощью MySQL Workbench и представлена на рис. 6.2.

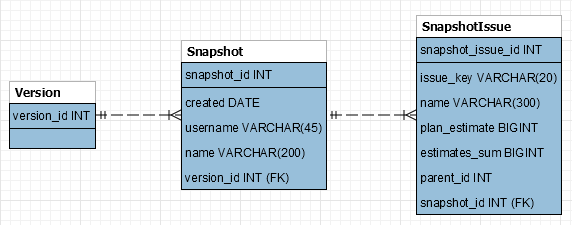


Рис.6.2. EER-модель структуры части базы данных

Проиллюстрированная часть базы данных содержит описанные ниже таблицы.

* Version – версия плана. Приведена без полей, для понимания связи описанных ниже таблиц с остальной базой данных плагина.
* Snapshot – снимок дерева задач. Поля таблицы:
  + id (целое число) – уникальный идентификатор записи таблицы;
  + created (дата) – дата и время создания снимка;
  + username (строка) – идентификатор пользователя, создавшего снимок;
  + version\_id (целое число) – id версии плана, к дереву задач которого относится снимок.
* SnapshotIssue – задача снимка. Поля таблицы:
  + id (целое число) – уникальный идентификатор записи таблицы;
  + issue\_key (строка) – уникальный идентификатор задачи;
  + name (строка) – название задачи;
  + plan\_estimate – плановая оценка;
  + estimates\_sum – суммарная оценка;
  + parent\_id – идентификатор родительской задачи;
  + snapshot\_id – идентификатор снимка, которому принадлежит задача.

### 6.2.2. Алгоритмы

Создание снимков осуществляется вручную или автоматически. Автоматическое создание снимка вызывается ежедневно в специальном классе «SnapshotJob», листинг которого приведен в приложении 1. В то же время, кроме создания снимка выполняется выборка и удаление некоторых устаревших снимков. Алгоритм создания с удалением устаревших снимков реализован в классе «SnapshotServiceImpl». Листинг данного класса приведен в приложении 2. На рис. 6.3 приведена блок-схема алгоритма.

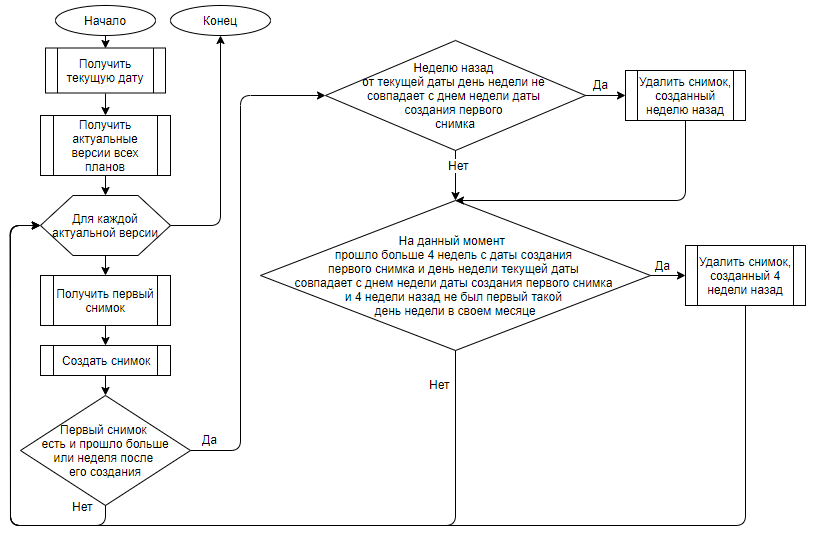


Рис. 6.3. Блок-схема алгоритма создания снимка с удалением устаревших

Алгоритмы сравнения снимков и текущего состояния дерева со снимком реализованы в классе «SnapshotsComparator», листинг которого приведен в приложении 3. На рис. 6.4. представлена диаграмма класса «SnapshotsComparator» с методами и внутренними классами.

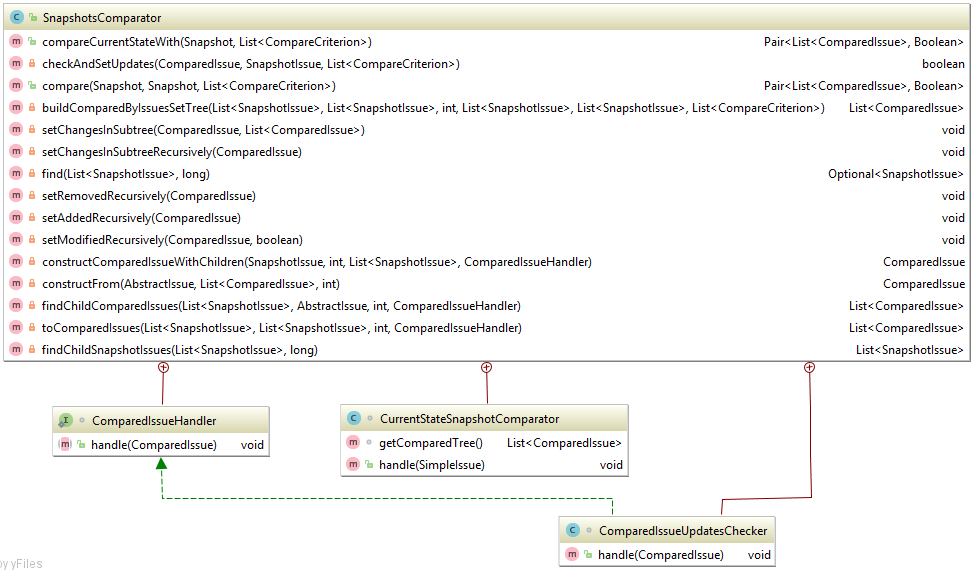


Рис. 6.4. Диаграмма класса «SnapshotsComparator» с внутренними классами

Открытые методы «compareCurrentStateWith» и «compare» выполняют сравнение текущего состояния дерева со снимком и двух снимков соответственно.

Сравнение двух снимков осуществляется рекурсивно от уровня корневых узлов для уровня терминальных, на одной итерации сравниваются списки задач одного уровня. Внутренний класc «ComparedIssueUpdatesChecker» обрабатывает объект задачи после сравнения, сохраняя в нее данные об отличиях.

Для сравнения текущего состояния дерева со снимком выполняется построение текущего дерева, во время которого каждый новый узел обрабатывается методом класса «CurrentStateSnapshotComparator», реализующего функциональный интерфейс, «handle», который выполняет поиск обрабатываемой задачи в снимке и сравнение.

Результат сравнения выглядит как дерево объектов задач, хранящих внутри данные об изменениях: является ли задача добавленной, удаленной или измененной, а также старые значения измененных полей.

## 6.3. Система ошибок плана

На рис. 6.5 представлена диаграмма классов пакета «planErrors».

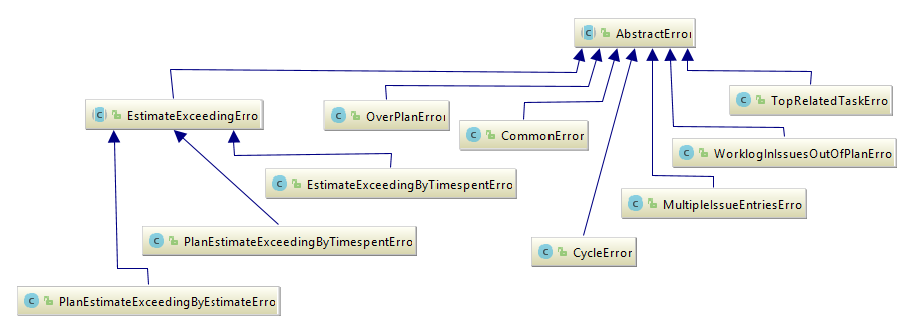


Рис. 6.5. Диаграмма классов пакета «planErrors»

Все классы ошибок являются наследниками абстрактного класса, поэтому есть возможность однотипно работать с коллекцией ошибок плана. Создание экземпляров классов происходит при построении дерева задач плана, во время которого выполняются проверки на наличие ошибок:

* множественное вхождение задач;
* наличие циклов в дереве;
* превышение плановой оценки статьи суммой оценок подзадач;
* превышение плановой оценки статьи суммой трудозатрат подзадач;
* изменение названия статьи (для актуальных версий плана);
* появление новый статей (для актуальных версий плана).

Отдельно выполняется проверка актуальной версии плана на наличие трудозатрат исполнителей в задачах, не входящих в план.

Для рассылки уведомлений об ошибках проверка выполняется ежедневно, вызываемая из класса «SendErrorsJob».

Листинги классов ошибок приведены в приложении.

## 6.4. JQL-функция planIssues()

На рис. 6.6 представлена диаграмма классов пакета «jql», включающая класс «PlanJQLFunction», отвечающий за работу функции, и класс «LightIssue», используемый для облегченных объектов задач, содержащих только идентификатор и список идентификаторов потомков.

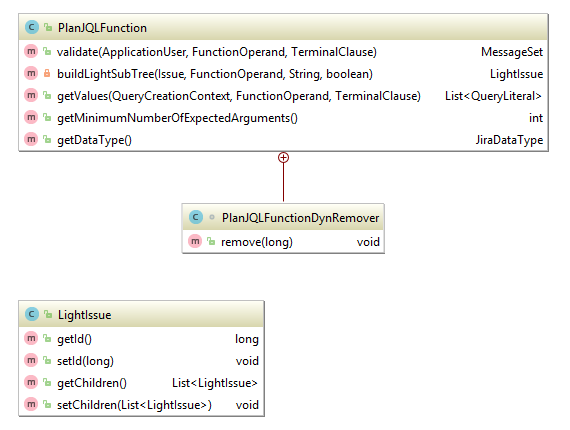


Рис. 6.6. Диаграмма классов пакета «jql»

Метод «validate» выполняет проверку и распознавание параметров функции, введенных пользователем. Метод «buildLightSubTree» осуществляет построение дерева задач, добавляя при этом каждый узел, удовлетворяющий критериям поиска, в коллекцию особого типа результатов поиска для отправки на сервер Jira.

## 6.5. Автоматическая корректировка оценки

Для функции автоматической корректировки был сгенерирован модуль веб-панели, встраиваемой в страницу задач Jira, контекст для отображения которого обеспечивается классом «ParentEstimatesAdjuster». Диаграмма данного класса представлена на рис. 6.7.

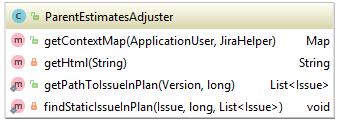


Рис. 6.7. Диаграмма класса «ParentEstimatesAdjuster»

Для работы панели в классе выполняется поиск пути до текущей задачи в дереве задач актуальной версии плана. После выбора пользователя одной из родительских задач для автоматической корректировки выполняется запрос и уменьшение оценки.

# План тестирования

Тестирование включает проверку наличия всех необходимых элементов интерфейса и корректную обработку действий пользователя для каждой разработанной функции, а также поведения системы в случае ошибок. При этом тестирование следует проводить при заданном русском языке интерфейса и английском для проверки локализации.

Подготовительный этап тестирования для всех тестов включает в себя следующие действия:

1. Запустить Jira.
2. Создать проект «Test» и несколько задач, образующих собой иерархию. Задачам верхнего уровня добавить метку «main». Пусть одной из таких задач будет задача с ключом «TEST-1», связанная с дочерними задачами «TEST-2» и «TEST-3».
3. Нажать в меню «Проекты» ссылку «Управление планами».
4. Создать план с помощью кнопки «Добавить план», задав ему имя «План» и JQL-запрос «labels=main».
5. Создать версию плана.

***Тест 1***. Тест функции ручного создания снимка.

***Цель***. Проверка ручного создания снимка.

***Порядок проведения***. Открыть страницу версии плана, нажать кнопку «Создать снимок».

***Результат***. Появится всплывающее окно уведомления об успешном выполнении операции.

***Тест 2***. Тест функции автоматического создания снимков.

***Цель***. Проверка автоматического создания и удаления снимков.

***Порядок проведения***. На странице планов открыть настройки плана и включить автоматическое создание снимков, пометив галочкой соответствующую опцию. Сохранить настройки нажатием кнопки «Сохранить». Нажать «Ввести в работу» напротив версии плана. Проверить список снимков в выпадающем списке на странице введенной в работу версии плана при выбранном режиме «Сравнение снимков» через сутки, через 3 недели и через 3 месяца.

***Результат***. Через сутки в выпадающем списке должен присутствовать автоматически созданный в 16.00 снимок с выведенными датой и временем создания. Через 3 недели в списке должны присутствовать 7 снимков за последние 7 дней (по одному в день), 1 снимок, созданный в промежутке между датой, бывшей 14 дней назад, и датой, бывшей 7 дней назад, и 1 снимок, созданный в промежутке между датой, бывшей 21 день назад, и датой, бывшей 14 дней назад. Через 3 месяца после ввода версии в работу в списке должны присутствовать 7 снимков за последние 7 дней (по одному в день), 3 снимка по одному на промежуток в 7 дней, созданные не ранее, чем 4 недели назад, 2 снимка по одному на промежуток в 4 недели, созданные не позднее, чем 4 недели назад.

***Тест 3***. Тест результатов сравнения текущего состояния со снимком.

***Цель***. Проверка корректности результата сравнения.

***Порядок проведения***. Изменить название и оценку одной из задач дерева плана. Переместить другую задачу к другому родителю, удалив старую связь и создав новую на странице задачи. На странице версии плана выбрать режим «Сравнение снимков». При выбранных элементах «Текущее состояние», последний снимок и отмеченных галочкой всех критериях в выпадающем списке «Критерии» нажать кнопку «Сравнить».

***Результат***. На странице появится таблица задач плана, где желтым цветом будет отмечено новое название задачи, изменившиеся суммы оценок у задачи и ее родителей, а также у всех бывших и новых родителей перемещенной задачи. Перенесенная задача будет красным цветом отмечена на свое прежнем месте и зеленым – на новом. Во второй колонке для родительских задач, в поддереве которых произошли изменения, будут располагаться значки соответствующих цветов с всплывающими подсказками «Есть добавления», «Есть удаления», «Есть изменения». При наведении на изменившиеся значения названия и сумм оценок появляется подсказка со старым значением.

***Тест 4***. Тест сравнения двух снимков.

***Цель***. Проверка корректности выводимого результата.

***Порядок проведения***. Выполнить действия из теста 3. На странице версии плана нажать кнопку «Создать снимок». Изменить в первом списке «Текущее состояние» на последний созданный снимок. Нажать «Сравнить».

***Результат***. Результат должен быть точно такой же, как в тесте 3.

***Тест 5***. Тест JQL-функции planIssues при заданных имени плана и номере версии.

***Цель***. Проверка корректности работы JQL-функции planIssues при заданных имени плана и номере версии.

***Порядок проведения***. Нажать на «Поиск задач» в главном меню Jira. Нажать на «Дополнительно» для перехода в расширенный режим поиска. Ввести в поисковую строку «issue in planIssues(“План”, 1)». Нажать «Найти».

***Результат***. Результатом поиска должен быть список всех задач версии 1 плана «План».

***Тест 6***. Тест JQL-функции planIssues при заданном имени плана.

***Цель***. Проверка корректности работы JQL-функции planIssues при заданном имени плана.

***Порядок проведения***. Перевести версию 1 плана «План» в статус «Актуальная». Нажать на «Поиск задач» в главном меню Jira. Нажать на «Дополнительно» для перехода в расширенный режим поиска. Ввести в поисковую строку «issue in planIssues(“План”)». Нажать «Найти».

***Результат***. Результатом поиска должен быть список всех задач версии 1 плана «План». Также должно отобразиться предупреждающее сообщение о том, что номер версии не задан и поиск произведен для актуальной версии указанного плана.

***Тест 7***. Тест JQL-функции planIssues при заданных имени плана, номере версии и ключе задачи плана.

***Цель***. Проверка корректности работы JQL-функции planIssues при заданных имени плана, номере версии и ключе задачи плана.

***Порядок проведения***. Нажать на «Поиск задач» в главном меню Jira. Нажать на «Дополнительно» для перехода в расширенный режим поиска. Ввести в поисковую строку «issue in planIssues(“План”, 1, “<ключ задачи из версии 1 плана>”)». Нажать «Найти».

***Результат***. Результатом поиска должен быть список, содержащий задачу с заданным ключом и всех дочерних задач ее поддерева версии 1 плана «План».

***Тест 8***. Тест JQL-функции planIssues при некорректно заданных параметрах.

***Цель***. Проверка вывода сообщения об ошибке работы при некорректно заданных параметрах.

***Порядок проведения***. Нажать на «Поиск задач» в главном меню Jira. Нажать на «Дополнительно» для перехода в расширенный режим поиска. Ввести в поисковую строку «issue in planIssues(1, “План”)». Нажать «Найти».

***Результат***. Появится сообщение об ошибке, содержащее информацию о корректном количестве и типе параметров функции.

***Тест 9***. Тест работы режима «Перемещение задач».

***Цель***. Проверка корректности работы функции перемещения задач.

***Порядок проведения***. Выбрать режим «Перемещение задач» в выпадающем списке режимов на странице версии плана. Нажать на задачу и, не отпуская, перетащить и расположить под другой, новой родительской задачей, сдвинуть перетаскиваемую задачу относительно предшествующей вправо и отпустить.

***Результат***. Перемещенная задача должна располагаться среди дочерних задач новой родительской задачи.

***Тест 10***. Тест работы режима «Перемещение задач».

***Цель***. Проверка вывода сообщения при попытке установить задачу на уровень статей.

***Порядок проведения***. Выбрать режим «Перемещение задач» в выпадающем списке режимов на странице версии плана. Нажать на задачу и, не отпуская, перетащить, расположить, выровняв ее на уровне статей, и отпустить.

***Результат***. Перемещенная задача должна вернуться на свое прежнее место, а в верхнем правом углу странице появится всплывающее окно с сообщением о невозможности установить задачу на уровень статьи.

***Тест 11***. Тест работы функции последних измененных задач.

***Цель***. Проверка вывода списка последних измененных задач.

***Порядок проведения***. Выбрать задачу, в которой сумма трудозатрат превышает суммарную оценку и есть более 3 дочерних задач. В настройках версии плана в поле «Количество последних измененных задач» ввести 3. Изменить оценки трех или более задач, входящих в поддерево выбранной вначале задачи. Навести мышку на иконку часов справа от названия выбранной задачи.

***Результат***. Появляется список ключей последних 3 измененных задач, где каждый ключ является ссылкой на страницу задачи.

***Тест 12***. Тест наличия панели «Корректировка оценок».

***Цель***. Проверка отображения панели «Корректировка оценок».

***Порядок проведения***. Перейти на страницу задачи, входящей в актуальную версию плана. Установить для нее оценку больше нуля.

***Результат***. В правой части страницы отображается панель «Корректировка оценок». На ней выводится информация о том, что задача входит в актуальную версию плана и о плане. Ниже расположена кнопка «Вычесть время из оценки родительской задачи».

***Тест 13***. Тест работы функции автоматической корректировки оценки.

***Цель***. Проверка корректности вывода списка родительских задач.

***Порядок проведения***. После проведения теста 12 нажать на кнопку «Вычесть время из оценки родительской задачи».

***Результат***. Открывается окно, содержащее список задач ветви дерева задач плана от статьи до текущей задачи. Задачи, имеющие оценки меньше оценки текущей задачи, недоступны для нажатия.

***Тест 14***. Тест работы функции автоматической корректировки оценки.

***Цель***. Проверка корректного выполнения уменьшения оценки.

***Порядок проведения***. После проведения теста 13 выбрать любую задачу из списка родительских задач текущей задачи и нажать на нее, предварительно запомнив ее первоначальную оценку и оценку оставшегося времени. Перейти на страницу выбранной задачи.

***Результат***. Первоначальная оценка и оценка оставшегося времени были уменьшены на величину, равную оценке оставшегося времени задачи, на странице которой выполнялся выбор родительской задачи.

***Тест 15***. Тест функции проверки наличия ошибок в плане.

***Цель***. Проверка корректного определения и вывода ошибок плана.

***Порядок проведения***. Установить статье «TEST-1» плановую оценку 100 ч. Установить двум дочерним задачам («TEST-2», «TEST-3») оценки 100 ч. Добавить в одну их них трудозатраты на 200 ч. Создать задачу с названием «Задача вне плана», не имеющую метку main и каких-либо связей. Добавить в нее трудозатраты на 1 ч. Создать задачу «Для проверки цикла» и добавить для нее две обратные статические связи с задачей «TEST-2». Изменить название задачи «TEST-1». Создать задачу «Проверка изменения состава статей», добавить ей метку «main». Нажать кнопку «Обновить» над деревом задач.

***Результат***. На вкладке «Ошибки» отображаются сообщения о превышении плановой оценки статьи «TEST-1» на 100 ч. суммой оценок дочерних задач и суммой трудозатрат дочерних задач, об отчете пользователя в задачу вне плана, о наличия цикла в дереве с выводом задач, входящих в цикл («TEST-1», «TEST-2», «TEST-1»), о появлении новой статьи, об изменении имени статьи «TEST-1».

***Тест 16***. Тест рассылки ошибок плана.

***Цель***. Проверка функции уведомления администратора плана о возникших ошибках.

***Порядок проведения***. Порядок проведения такой же, как для теста 15.

***Результат***. В 15.00 по электронной почте, указанной в профиле администратора плана, отправлено письмо со списком ошибок плана, аналогичным отображающемуся на вкладке «Ошибки».

# Заключение

В результате проделанной работы в рамках Jira-расширения был разработан ряд инструментов, позволяющих удобно работать с задачами плана, отслеживать отклонения и ошибки, помогающих находить их причины. Реализованная функциональность предоставляет следующие возможности:

* включение/отключение автоматического создания снимков с удалением устаревших;
* создание снимка вручную;
* сравнение двух снимков;
* сравнение текущего состояния дерева задач со снимком;
* поиск последних измененных задач;
* использование возможностей расширенного поиска Jira для задач плана;
* проверка наличия в плане ошибок и рассылка уведомлений о них;
* автоматическая корректировка оценки родительской задачи;
* менять для задач дерево и/или уровень вложенности перетаскиванием мышью.

Разработанная программа протестирована на Jira 7.2.4 и Jira 7.1.9 и работает корректно. Расширение было внедрено в работу на предприятии. Планируется его публикация в магазине Atlassian Marketplace.

# Литература

1. Backbone.js (1.4.0) [Электронный ресурс] // Jeremy Ashkenas and DocumentCloud, 2010-2018. URL: [https://backbonejs.org](https://backbonejs.org/docs/backbone.html) (дата обращения 05.11.2018).
2. Atlassian SDK Developer [Электронный ресурс] // Atlassian, 2018. URL: <https://developer.atlassian.com/server/framework/atlassian-sdk/> (дата обращения 01.11.2018).
3. Jobin Kuruvilla JIRA Development Cookbook / Jobin Kuruvilla – 3-е издание – Birmingham: Packt Publishing Ltd., 2016 – 560 c.
4. Atlassian JIRA 7.1.9 API [Электронный ресурс] // Atlassian, 2002-2016. URL: <https://docs.atlassian.com/software/jira/docs/api/7.1.9/> (дата обращения 2.02.2019).
5. Atlassian JIRA 7.2.4 API [Электронный ресурс] // Atlassian, 2002-2016. URL: <https://docs.atlassian.com/software/jira/docs/api/7.2.4/> (дата обращения 2.02.2019).
6. AUI 5.7.31 Documentation [Электронный ресурс] // Atlassian, 2009-2013. URL: <https://docs.atlassian.com/aui/5.7.31/docs/applicationHeader.html> (дата обращения 5.02.2019).
7. Кей Н. Хорстманн, Корнелл Г. Java. Библиотека профессионала, том 1. Основы / Кей Н. Хорстманн, Г. Корнелл; пер. с англ. И. В. Берштейна. – 10-е издание – М.: ООО «И. Д. Вильямс», 2016 – 864 с.
8. Java™ Platform, Standard Edition & Java Development Kit Version 8 API Specification [Электронный ресурс] // ORACLE Help Center, 1993-2018. URL: https://docs.oracle.com/javase/8/docs/api/ (дата обращения 1.10.2018).
9. Maven Documentation [Электронный ресурс] // Apache Maven Project, 2002-2018. URL: https://maven.apache.org/guides/ (дата обращения 10.01.2019).
10. JavaScript [Электронный ресурс] // Mozilla Developer Network and individual contributors, 2005-2018. URL: <https://devdocs.io/javascript/> (дата обращения 25.10.2018).
11. jQuery API [Электронный ресурс] // The jQuery Foundation, 2019. URL: <https://api.jquery.com/> (дата обращения 2.03.2019).

# Приложение 1. Листинг класса «SnapshotsComparator»

package ru.relex.jira.plugins.planner.service;

import com.atlassian.jira.component.ComponentAccessor;

import com.atlassian.jira.issue.Issue;

import com.atlassian.jira.issue.IssueManager;

import com.atlassian.jira.util.lang.Pair;

import ru.relex.jira.plugins.planner.Constants;

import ru.relex.jira.plugins.planner.ao.entity.Snapshot;

import ru.relex.jira.plugins.planner.ao.entity.SnapshotIssue;

import ru.relex.jira.plugins.planner.dto.AbstractIssue;

import ru.relex.jira.plugins.planner.dto.ComparedIssue;

import ru.relex.jira.plugins.planner.dto.SimpleIssue;

import ru.relex.jira.plugins.planner.service.treebuilder.DynRemover;

import ru.relex.jira.plugins.planner.service.treebuilder.NodeHandler;

import ru.relex.jira.plugins.planner.service.treebuilder.TreeHelper;

import javax.annotation.Nonnull;

import javax.annotation.Nullable;

import java.util.ArrayList;

import java.util.Arrays;

import java.util.List;

import java.util.Optional;

import java.util.stream.Collectors;

/\*\*

\* Class for comparing snapshots of issues trees state.

\*/

public class SnapshotsComparator {

private IssueManager issueManager = ComponentAccessor.getIssueManager();

private boolean foundDifferences;

/\*\*

\* Compares current state with snapshot (current state + changes)

\*

\* @param snapshot snapshot

\* @param criteria criteria

\* @return list of trees of ComparedIssue objects and flag of existing of differences

\*/

public Pair<List<ComparedIssue>, Boolean> compareCurrentStateWith(Snapshot snapshot, List<Constants.CompareCriterion> criteria) {

foundDifferences = false;

TreeHelper treeHelper = TreeHelper.getInstance(true);

treeHelper.setVersion(snapshot.getVersion());

List<SnapshotIssue> snapshotIssues = Arrays.stream(snapshot.getIssues()).collect(Collectors.toList());

CurrentStateSnapshotComparator nodeHandler = new CurrentStateSnapshotComparator(snapshotIssues, criteria);

DynRemover treeDynRemover = treeHelper.getTreeDynRemover();

treeHelper.setTreeDynRemover(new DynRemover() {

@Override

public void remove(long id) {

treeDynRemover.remove(id);

List<ComparedIssue> comparedTree = nodeHandler.getComparedTree();

removeDynamic(id, comparedTree);

}

private void removeDynamic(long id, List<ComparedIssue> comparedIssues) {

for (ComparedIssue comparedIssue : comparedIssues) {

if (comparedIssue.getIssueId() == id && comparedIssue.isDynamic()) {

if (comparedIssue.isAdded()) {

comparedIssues.remove(comparedIssue);

return;

} else {

setModifiedRecursively(comparedIssue, false);

setRemovedRecursively(comparedIssue);

}

} else {

removeDynamic(id, comparedIssue.getChildIssues());

}

}

}

});

treeHelper.getTree(nodeHandler);

return Pair.of(nodeHandler.getComparedTree(), foundDifferences);

}

class CurrentStateSnapshotComparator implements NodeHandler {

private List<ComparedIssue> comparedTree = new ArrayList<>();

private List<Constants.CompareCriterion> criteria;

private List<SnapshotIssue> snapshotIssues;

CurrentStateSnapshotComparator(List<SnapshotIssue> snapshotIssues, List<Constants.CompareCriterion> criteria) {

this.snapshotIssues = snapshotIssues;

this.criteria = criteria;

}

List<ComparedIssue> getComparedTree() {

return comparedTree;

}

@Override

public void handle(SimpleIssue simpleIssue) {

Optional<SnapshotIssue> snapshotIssueOptional = find(snapshotIssues, simpleIssue.getIssueId());

ComparedIssue comparedIssue = constructFrom(simpleIssue, new ArrayList<>(), simpleIssue.getDeep());

if (snapshotIssueOptional.isPresent()) {

SnapshotIssue snapshotIssue = snapshotIssueOptional.get();

if (snapshotIssue.getParentId() != simpleIssue.getParentId()) {

if (criteria.contains(Constants.CompareCriterion.ISSUES\_SET)) {

comparedIssue.setAdded(true);

foundDifferences = true;

int childrenDeep = snapshotIssue.getDeep() + 1;

List<SnapshotIssue> snapshotChildren = findChildSnapshotIssues(snapshotIssues, snapshotIssue.getIssueId());

snapshotIssues.removeAll(snapshotChildren);

List<ComparedIssue> children = toComparedIssues(snapshotIssues, snapshotChildren, childrenDeep,

(compIss) -> compIss.setRemoved(true));

ComparedIssue comparedIssueRemoved = constructFrom(snapshotIssue, children, snapshotIssue.getDeep());

comparedIssueRemoved.setParentId(snapshotIssue.getParentId());

if (children.size() > 0) {

comparedIssueRemoved.setDeletionsInSubtree(true);

}

setRemovedRecursively(comparedIssueRemoved);

Optional<ComparedIssue> parent = comparedTree.stream()

.filter(compIss -> compIss.getId() == comparedIssueRemoved.getParentId())

.findFirst();

parent.ifPresent(p -> p.addChild(comparedIssueRemoved));

if (!parent.isPresent()) {

comparedTree.add(comparedIssueRemoved);

}

snapshotIssues.remove(snapshotIssue);

} else {

checkAndSetUpdates(comparedIssue, snapshotIssue, criteria);

}

} else {

checkAndSetUpdates(comparedIssue, snapshotIssue, criteria);

}

} else {

if (criteria.contains(Constants.CompareCriterion.ISSUES\_SET)) {

comparedIssue.setAdded(true);

foundDifferences = true;

}

}

List<ComparedIssue> children = comparedTree.stream()

.filter(compIss -> compIss.getParentId() == comparedIssue.getId())

.peek(compIss -> {

if (comparedIssue.isAdded()) {

compIss.setAdded(true);

compIss.setModified(false);

}

})

.collect(Collectors.toList());

comparedIssue.addAllChildren(children);

setChangesInSubtree(comparedIssue, comparedIssue.getChildIssues());

comparedTree.removeAll(children);

int childrenDeep = comparedIssue.getDeep() + 1;

List<SnapshotIssue> removedChildren = findChildSnapshotIssues(snapshotIssues, comparedIssue.getId());

if (criteria.contains(Constants.CompareCriterion.ISSUES\_SET)) {

List<ComparedIssue> removedComparedChildren = toComparedIssues(snapshotIssues, removedChildren, childrenDeep,

(compIss) -> compIss.setRemoved(true));

List<Long> existingChildIssuesIds = comparedIssue.getChildIssues().stream().map(ComparedIssue::getIssueId).collect(Collectors.toList());

removedComparedChildren = removedComparedChildren.stream()

.filter(iss -> !existingChildIssuesIds.contains(iss.getIssueId()))

.collect(Collectors.toList());

if (removedComparedChildren.size() > 0) {

foundDifferences = true;

comparedIssue.addAllChildren(removedComparedChildren);

setChangesInSubtreeRecursively(comparedIssue);

}

}

snapshotIssues.removeAll(removedChildren);

comparedTree.add(comparedIssue);

}

}

private boolean checkAndSetUpdates(ComparedIssue comparedIssue, SnapshotIssue snapshotIssue, List<Constants.CompareCriterion> criteria) {

for (Constants.CompareCriterion criterion : criteria) {

switch (criterion) {

case NAMES: {

String oldName = snapshotIssue.getIssueName();

if (!comparedIssue.getName().equals(oldName)) {

comparedIssue.setOldName(oldName);

}

break;

}

case PLAN\_ESTIMATES: {

Long oldEstimate = snapshotIssue.getPlanEstimate();

if (!comparedIssue.getPlanEstimate().equals(oldEstimate)) {

comparedIssue.setOldPlanEstimate(oldEstimate);

}

break;

}

case ESTIMATES\_SUM: {

long oldSum = snapshotIssue.getEstimatesSum();

if (comparedIssue.getEstimatesSum() != oldSum) {

comparedIssue.setOldEstimatesSum(oldSum);

}

break;

}

}

}

boolean modified = comparedIssue.getOldName() != null || comparedIssue.getOldPlanEstimate() != null || comparedIssue.getOldEstimatesSum() != null;

comparedIssue.setModified(modified);

if (modified) {

foundDifferences = true;

}

return modified;

}

/\*\*

\* Compares two snapshots by given criteria.

\*

\* @param firstSnapshot first snapsnot

\* @param secondSnapshot second snapshot

\* @param criteria criteria on which it compares

\* @return list of trees of ComparedIssue objects and flag of existing of differences

\*/

public Pair<List<ComparedIssue>, Boolean> compare(Snapshot secondSnapshot, Snapshot firstSnapshot, List<Constants.CompareCriterion> criteria) {

foundDifferences = false;

List<SnapshotIssue> firstSnapshotIssues = Arrays.stream(firstSnapshot.getIssues()).collect(Collectors.toList());

List<SnapshotIssue> secondSnapshotIssues = Arrays.stream(secondSnapshot.getIssues()).collect(Collectors.toList());

List<ComparedIssue> comparedTree;

List<SnapshotIssue> secondRootIssues = findChildSnapshotIssues(secondSnapshotIssues, Constants.ARTICLE\_PARENT\_ID);

if (criteria.contains(Constants.CompareCriterion.ISSUES\_SET)) {

criteria.remove(Constants.CompareCriterion.ISSUES\_SET);

List<SnapshotIssue> firstRootIssues = findChildSnapshotIssues(firstSnapshotIssues, Constants.ARTICLE\_PARENT\_ID);

comparedTree = buildComparedByIssuesSetTree(firstRootIssues, secondRootIssues, 0, firstSnapshotIssues, secondSnapshotIssues, criteria);

} else {

comparedTree = new ArrayList<>();

for (SnapshotIssue secondRootIssue : secondRootIssues) {

comparedTree.add(constructComparedIssueWithChildren(secondRootIssue, 0, secondSnapshotIssues,

new ComparedIssueUpdatesChecker(firstSnapshotIssues, criteria)));

}

}

return Pair.of(comparedTree, foundDifferences);

}

/\*\*

\* Recursively handles nodes level by level and builds a list of trees with added, modified and removed nodes.

\*

\* @param firstListToCompare first list of issues to compare

\* @param secondListToCompare second list of issues to compare

\* @param deep deep means the number of current level (they are started from 0 for root nodes)

\* @param allFirst all issues of the first snapshot

\* @param allSecond all issues of the second snapshot

\* @param criteria criteria for comparison

\* @return a list of trees with added, modified and removed nodes

\*/

private List<ComparedIssue> buildComparedByIssuesSetTree(List<SnapshotIssue> firstListToCompare,

List<SnapshotIssue> secondListToCompare,

int deep,

List<SnapshotIssue> allFirst,

List<SnapshotIssue> allSecond, List<Constants.CompareCriterion> criteria) {

List<ComparedIssue> resultList = new ArrayList<>();

List<Long> viewedIds = new ArrayList<>();

for (SnapshotIssue firstSnapshotIssue : firstListToCompare) {

ComparedIssue comparedIssue;

Optional<SnapshotIssue> secondSnapshotIssue = find(secondListToCompare, firstSnapshotIssue.getIssueId());

if (!(secondSnapshotIssue).isPresent()) {

comparedIssue = constructComparedIssueWithChildren(firstSnapshotIssue, firstSnapshotIssue.getDeep(), allFirst, null);

setRemovedRecursively(comparedIssue);

foundDifferences = true;

setChangesInSubtreeRecursively(comparedIssue);

} else {

List<SnapshotIssue> childrenForFirstIssue = findChildSnapshotIssues(allFirst, firstSnapshotIssue.getIssueId());

List<SnapshotIssue> childrenForSecondIssue = findChildSnapshotIssues(allSecond, secondSnapshotIssue.get().getIssueId());

List<ComparedIssue> childrenForPermanentIssue = buildComparedByIssuesSetTree(childrenForFirstIssue,

childrenForSecondIssue, ++deep, allFirst, allSecond, criteria);

comparedIssue = constructFrom(secondSnapshotIssue.get(), childrenForPermanentIssue, secondSnapshotIssue.get().getDeep());

Optional<SnapshotIssue> oldIssue = find(allFirst, comparedIssue.getId());

oldIssue.ifPresent(oldIss -> checkAndSetUpdates(comparedIssue, oldIss, criteria));

setChangesInSubtree(comparedIssue, childrenForPermanentIssue);

}

resultList.add(comparedIssue);

viewedIds.add(firstSnapshotIssue.getIssueId());

}

for (SnapshotIssue secondSnapshotIssue : secondListToCompare) {

if (!viewedIds.contains(secondSnapshotIssue.getIssueId())) {

ComparedIssue comparedIssue = constructComparedIssueWithChildren(secondSnapshotIssue, secondSnapshotIssue.getDeep(), allSecond, null);

setAddedRecursively(comparedIssue);

foundDifferences = true;

setChangesInSubtreeRecursively(comparedIssue);

resultList.add(comparedIssue);

}

}

return resultList;

}

private void setChangesInSubtree(ComparedIssue comparedIssue, List<ComparedIssue> children) {

for (ComparedIssue child : children) {

if (child.isModified()) {

comparedIssue.setChangesInSubtree(true);

}

if (child.isAdded()) {

comparedIssue.setInsertionsInSubtree(true);

}

if (child.isRemoved()) {

comparedIssue.setDeletionsInSubtree(true);

}

if (child.isChangesInSubtree()) {

comparedIssue.setChangesInSubtree(true);

}

if (child.isInsertionsInSubtree()) {

comparedIssue.setInsertionsInSubtree(true);

}

if (child.isDeletionsInSubtree()) {

comparedIssue.setDeletionsInSubtree(true);

}

if (comparedIssue.isChangesInSubtree() && comparedIssue.isInsertionsInSubtree() && comparedIssue.isDeletionsInSubtree()) {

break;

}

}

}

private void setChangesInSubtreeRecursively(ComparedIssue comparedIssue) {

List<ComparedIssue> childIssues = comparedIssue.getChildIssues();

setChangesInSubtree(comparedIssue, childIssues);

childIssues.forEach(child -> setChangesInSubtreeRecursively(child));

}

@FunctionalInterface

interface ComparedIssueHandler {

void handle(ComparedIssue comparedIssue);

}

class ComparedIssueUpdatesChecker implements ComparedIssueHandler {

private List<SnapshotIssue> oldList;

private List<Constants.CompareCriterion> criteria;

ComparedIssueUpdatesChecker(List<SnapshotIssue> oldList, List<Constants.CompareCriterion> criteria) {

this.oldList = oldList;

this.criteria = criteria;

}

@Override

public void handle(ComparedIssue comparedIssue) {

Optional<SnapshotIssue> oldIssue = find(oldList, comparedIssue.getId());

oldIssue.ifPresent(oldIss -> checkAndSetUpdates(comparedIssue, oldIss, criteria));

setChangesInSubtree(comparedIssue, comparedIssue.getChildIssues());

}

}

private Optional<SnapshotIssue> find(List<SnapshotIssue> list, long issueToSearchId) {

return list.stream().filter(issue -> issue.getIssueId() == issueToSearchId).findFirst();

}

private void setRemovedRecursively(@Nonnull ComparedIssue compIssue) {

compIssue.setRemoved(true);

for (ComparedIssue childIssue : compIssue.getChildIssues()) {

setRemovedRecursively(childIssue);

}

}

private void setAddedRecursively(@Nonnull ComparedIssue compIssue) {

compIssue.setAdded(true);

for (ComparedIssue childIssue : compIssue.getChildIssues()) {

setAddedRecursively(childIssue);

}

}

private void setModifiedRecursively(@Nonnull ComparedIssue compIssue, boolean isModified) {

compIssue.setModified(isModified);

if (!isModified) {

compIssue.setChangesInSubtree(false);

compIssue.setOldEstimatesSum(null);

compIssue.setOldPlanEstimate(null);

compIssue.setOldName(null);

}

for (ComparedIssue childIssue : compIssue.getChildIssues()) {

setModifiedRecursively(childIssue, isModified);

}

}

private ComparedIssue constructComparedIssueWithChildren(SnapshotIssue snapshotIssue, int deep, List<SnapshotIssue> allIssues, @Nullable ComparedIssueHandler handler) {

Issue firstIssue = issueManager.getIssueObject(snapshotIssue.getIssueId());

ComparedIssue comparedIssue = new ComparedIssue(firstIssue.getId(), firstIssue.getKey(),

firstIssue.getSummary(), snapshotIssue.getPlanEstimate(), snapshotIssue.getEstimatesSum(), snapshotIssue.getTimeSpent(),

findChildComparedIssues(allIssues, snapshotIssue, deep, handler), snapshotIssue.getParentId(), snapshotIssue.isDynamic(), deep);

if (handler != null) {

handler.handle(comparedIssue);

}

return comparedIssue;

}

private ComparedIssue constructFrom(AbstractIssue abstractIssue, List<ComparedIssue> childIssues, int deep) {

Issue realIssue = issueManager.getIssueObject(abstractIssue.getIssueId());

return new ComparedIssue(realIssue.getId(), realIssue.getKey(),

realIssue.getSummary(), abstractIssue.getPlanEstimate(), abstractIssue.getEstimatesSum(),

abstractIssue.getTimeSpent(), childIssues, abstractIssue.getParentId(), abstractIssue.isDynamic(), deep);

}

private List<ComparedIssue> findChildComparedIssues(List<SnapshotIssue> all, AbstractIssue parent, int parentDeep, @Nullable ComparedIssueHandler handler) {

int childrenDeep = parentDeep + 1;

return toComparedIssues(all, findChildSnapshotIssues(all, parent.getIssueId()), childrenDeep, handler);

}

private List<ComparedIssue> toComparedIssues(List<SnapshotIssue> all, List<SnapshotIssue> foundSnapshotIssues, int childrenDeep, @Nullable ComparedIssueHandler handler) {

return foundSnapshotIssues.stream()

.map(snapshotIssue -> constructComparedIssueWithChildren(snapshotIssue, childrenDeep, all, handler))

.collect(Collectors.toList());

}

private List<SnapshotIssue> findChildSnapshotIssues(List<SnapshotIssue> all, long parentId) {

return all.stream()

.filter(snapshotIssue -> snapshotIssue.getParentId() == parentId)

.collect(Collectors.toList());

}

}