УДК 004.9

WEB-ПРИЛОЖЕНИЕ ДЛЯ СОВМЕСТНОЙ РАЗРАБОТКИ КОМПЬЮТЕРНЫХ ИГР

М. В. Селиверстова, А. Н. Даниленко

В настоящей работе представлена реализация системы совместной разработки компьютерных игр, которая позволяет увеличить эффективность труда команды за счет использования одной системы для организации труда специалистов всех направлений представленных в разработке игр. Произведен обзор предметной области, в частности организации процесса разработки, по результатам которого были составлены требования к системе. Рассмотрены архитектурные решения, которые были приняты в процессе разработки системы. Система была разработана как клиент-серверное приложение, в ней используется модель асинхронного выполнения task asynchronous programming model. Для обеспечения ограничений доступа была создана гибкая система, которая настраивается с помощью файлов конфигурации. Пользователи имеют широкие возможности по отслеживанию жизненного цикла задач, составлению документации и общению. Рассмотрены возможные пути улучшения системы в будущем.

Ключевые слова: отслеживание версий, документирование, отслеживание ошибок, заявка, ресурс, ошибка, web-приложение, асинхронное выполнение.

В современном мире наблюдается тенденция к переходу на удалённый тип взаимодействия. При этом одним из важнейших параметров эффективности команд является совместное знание, то есть знание, выработанное совместно для дальнейшего использования в конкретном эпизоде [1]. Для решения этой и многих других проблем совместной работы создано множество различных систем, таких как Atlassian Jira, Trello или GitLab, однако не всегда такие системы отвечают требованиям всех членов команды. Например, системы Jira и Trello обладают широкими возможностями для организаторов, но не имеют всех функций, необходимых разработчикам, при этом GitLab предоставляет как возможности для организации, так и для разработки, однако имеет ограниченные возможности по работе с файлами особых форматов и высокий порог вхождения.

Это приводит к одновременному использованию нескольких систем совместной работы, а значит к фрагментации команды и соответственно меньшему взаимодействию. меньшему совместному знанию, у специалистов разных направлений. Web-приложение, описание которого представлено в данной работе, предоставляет возможность проводить работу по всем направлениям, которые как правило представлены в области разработки игр, в рамках одной системы. Такой подход позволяет увеличить взаимодействие всех членов команды и дает общую базу знаний по проекту.

Материалы и методы

Представленная система автоматизирует процесс организации разработки, который можно разбить на несколько этапов:

1. Планирование.
2. Разработка.
3. Контроль качества.
4. Сбор обратной связи.

При этом процесс может быть разделен на части и данные этапы могут повторяться неограниченное количество раз до получения конечного результата. Соответственно можно выделить следующие объекты предметной области: заявка, ресурс и ошибка, с которыми производится взаимодействие (рис. 1).

Селиверстова Мария Викторовна (seliver.maria@gmail.com),
студент IV курса факультета информатики;
Даниленко Александра Николаевна (danilenko.al@gmail.com),
доцент кафедры программных систем Самарского университета,
443086, Россия, г. Самара, Московское шоссе, 34.

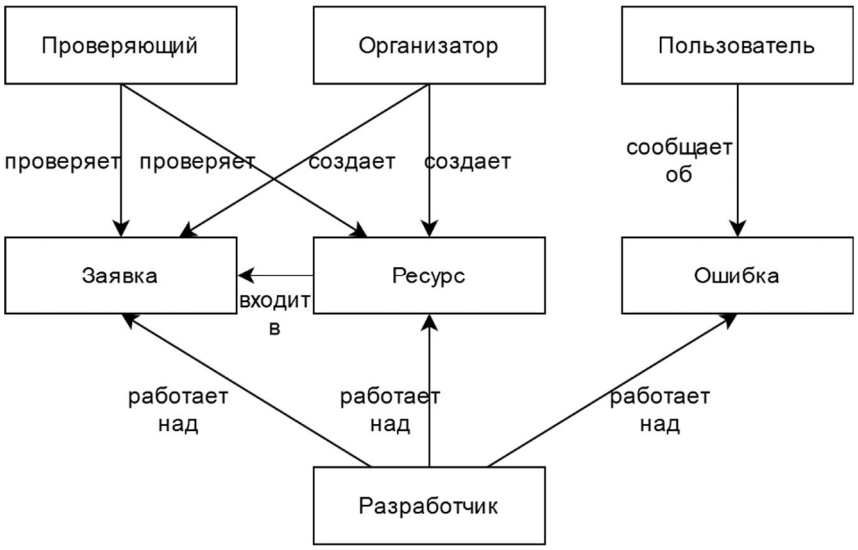

Рис. 1. Процесс разработки игры

При этом под заявкой понимается оформленная задача на разработку элементов, напрямую связанных с игровым процессом, а под ресурсом задача на разработку используемых в процессе работы над заявками ресурсов, таких как модели или звуковые эффекты.

Для организаторов важно предоставить возможность гибкого, но четкого определения ролей участников команды, постановки и отслеживания степени выполнения конкретных задач.

Разработчикам важна возможность четко видеть требования к задаче, найти все задачи, соотносящиеся с ней, иметь доступ к справочной информации, а также взаимодействовать с другими разработчиками. При этом следует понимать, что саму группу разработчиков можно разбить на дополнительные подгруппы в зависимости от их различных потребностей.

В данной системе такое разделение присутствует и производится на основании основного инструмента с которым работает разработчик. Разработчики, которые работают напрямую с игровыми модулями, предназначенными для использования игровым движком выделяются в отдельную категорию, так как у них присутствуют дополнительные потребности, связанные с необходимостью отслеживать иерархию модулей, которые могут зависеть друг от друга.

Специалистам контроля качества важна возможность легко видеть историю изменения файлов и любую дополнительную информацию о задаче, которую хотел оставить разработчик. Необходима также возможность связи с разработчиком для определения необходимых поправок к задаче, которая выполнена с ошибками.

Сбор обратной связи заключается в сборе отчетов об ошибках или пожеланий от пользователей конечного продукта. Желательно отделять такие отчеты и пожелания от задач, которые прошли процесс планирования и готовы к работе.

Архитектура системы

При разработке системы было принято решение воспользоваться клиент-серверной архитектурой с тонким клиентом. Это связано с необходимостью в обеспечении возможности многопользовательского доступа и автоматической синхронизации результатов работы. Технология тонкого клиента выбрана, так как она налагает минимальные ограничения на техническое и программное обеспечение клиентской части, что важно, поскольку из-за различных потребностей разработчиков их рабочие места могут быть оборудованы различным образом.

При формировании ответа сервера можно выделить несколько точек, на которых процесс можно прервать, поэтому было целесообразным воспользоваться шаблоном проектирования «цепочка обязанностей». В частности, он используется для контроля доступа пользователя к конкретному ресурсу. Например, редактирование заявки может производиться только организатором или разработчиком и проверяющим, которые работают над ней. На разных этапах обработки запроса система обладает ограниченным набором

информации. В первую очередь происходит проверка правильности самого запроса, если он сформирован корректно, система производит аутентификацию пользователя. Контроллер при регистрации заявляет, что для доступа к нему необходимо наличие у пользователя роли организатора, разработчика или проверяющего, поэтому перед обращением к нему система проверяет наличие данных ролей у пользователя. Контроллер получает необходимые данные из базы данных и производит последнюю проверку, если пользователь не является организатором, был ли он назначен на исполнение выбранной заявки.

Если на любом из этих этапов необходимые условия не были выполнены, выполнения дальнейших этапов не происходит, и пользователь перенаправляется на страницу авторизации или ошибки, что позволяет экономить ресурсы системы.

Асинхронное выполнение

Поскольку для системы была выбрана клиент-серверная архитектура, при разработке системы важно было учесть возможность одновременного взаимодействия с системой множества пользователей. Очевидным решением является выполнение каждого запроса в отдельном потоке. Частое создание и закрытие потоков приводит к уменьшению производительности, поскольку является дорогой операцией, но производительность можно увеличить, использовав шаблон проектирования «пул потоков» [2]. При таком подходе приложение хранит заранее подготовленный набор потоков (пул потоков), который при необходимости извлекаются из пула потоков для выполнения задачи, а по окончанию выполнения возвращаются обратно и могут быть использованы снова.

Однако можно заметить ещё одну особенность системы, большая часть запросов к системе приводит к обращению к базе данных. Поскольку такое обращение требует ожидания, часть потоков будет простаивать и не может быть задействована для обработки запросов. Это может привести к исчерпанию пула потоков при большом количестве запросов, так как его размер ограничен доступными системе ресурсами. Язык C# предлагает модель task asynchronous programming model (TAP), которая предоставляет расширенные возможности для написания асинхронных процедур, позволяя при этом сохранить логическую структуру программы [3]. В данной ситуации интересны ключевые слова async и await. По достижению точки отмеченной ключевым словом await, выполнение текущего метода приостанавливается и контроль передается вызывающему методу, при этом текущий поток не блокируется и может быть использован для других целей [4]. Это позволяет уменьшить вероятность исчерпания пула потоков, поэтому модель TAP была использована для всех моментов, где необходимо было ожидание окончания выполнения внешней по отношению к выполняемой процедуре задачи.

Система проверки доступа

Поскольку конкретные моменты процесса разработки могут меняться, должна быть обеспечена гибкость системы ролей пользователей. Если логика проверки доступа прописана во множестве мест в зависимости от того где она применяется, любое изменение в системе ролей с точки зрения процесса разработки приводит к необходимости поиска, изменения и тестирования программного кода во множестве не связанных мест. Для решения данной проблемы в рассматриваемой системе выделены объекты роли, которые хранятся в базе данных. Для определения уровня доступа объекты системы обращаются к менеджеру доступа, который использует хранимые данные о ролях, предоставленные данные о пользователе и данные об объекте, к которому производится доступ. Такой подход позволяет производить настройку прав доступа с помощью изменения конфигурации менеджера ролей без необходимости изменения кода программы.

Конфигурация правил доступа хранится как файлы формата JSON, где каждому допустимому действию может быть сопоставлен один файл. Структура данных в этих файлах такова: на внешнем уровне находится массив объектов правил, объект правила может хранить массив строк названий ролей пользователей, а также массив объектов сравнений, которые необходимо провести (рис. 2).

```
[
  {
    "roles": [
      "Admin",
      "Reviewer"
    ]
  },
  {
    "roles": [
      "Developer"
    ],
    "fields": [
      {
        "operator": "=",
        "value1": "<User>.Id",
        "value2": "<Object>.ClaimantId"
      },
      {
            "operator": "<",
            "value1": "<Object>.Progress",
            "value2": "<int>.100"
      }
    ]
  }
]
```

Рис. 2. Пример набора правил доступа

Если хоть одно правило было выполнено, доступ к объекту предоставляется. Правило считается выполненным если все условия, содержащиеся в нем, выполнены. Данный метод требует регистрации каждого нового типа взаимодействия или объекта в менеджере доступа, однако поскольку такие изменения в любом случае требуют внесения изменений в код программы, этот недостаток был сочтен несущественным.

Для оформления сравнений был разработан особый синтаксис, необходимо указать оператор, которым может быть одна из стандартных операций сравнения («=», «!=», «>», «<», «<=», «>=»), а также операции «IN» и «NOT IN», которые проверяют элемент на присутствие или отсутствие в массиве значений соответственно. Для указания полей с которыми необходимо провести операцию необходимо указать одно из зарезервированных значений «<User>» или «<Object>», которые представляют собой текущего пользователя или объект, к которому производится доступ, соответственно, а затем через точку указать название поля, к которому должно производиться обращение. Также можно указывать значения-константы, для этого в угловых скобках указывается название одного из типов языка C#, а затем через точку представление константы в строковом виде. Совпадение обозначения объекта, к которому производится доступ, с обозначением встроенного типа «object» допустимо, поскольку использование встроенного типа «object» не имеет смысла в рамках поставленной задачи.

Отслеживание жизненного цикла

Одной из основных задач системы является отслеживание жизненного цикла задач, для этих целей в системе присутствуют три отображения: обозреватель заявок, обозревать ресурсов и обозреватель ошибок. Они предоставляют возможность просмотра списка всех задач в системе, при этом пользователи имеют возможность отфильтровать задачи по названию, разработчику, статусу или типу. Такой набор фильтров был выбран не случайно, фильтрация по названию позволяет найти конкретную задачу, что актуально для всех пользователей системы. Фильтрация по разработчику актуальна для организаторов, для оценки уровня занятости разработчика, или производительности его труда. Возможность фильтровать задачи по статусу позволяет оценить объем произведенных и необходимых работ, при этом дополнительная фильтрация по типу позволяет выделить конкретное направление для оценки.

Для того чтобы информация в обозревателях была всегда актуальна, важно убедиться, что правила перехода между состояниями адекватны для поставленных задач. Рассмотрим этот процесс на примере ресурса.

При создании, ресурс находится в стадии «Предложен» и находится в нем до подтверждения организаторами необходимости работы над ним и корректности информации. После такого подтверждения, ресурс переходит в стадию «Необходима работа», только в этот момент у разработчика появляется возможность запросить ресурс в работу. Если такой запрос прошел и был одобрен организатором, ресурс переходит в статус «В разработке» при этом по мере прогресса работы разработчик загружает файлы, созданные им, и обновляет прогресс. Когда разработчик подтверждает, что прогресс работы достиг 100%, он имеет возможность отправить работу на проверку. Такая работа переходит в статус «Необходима проверка» и проверяющие получают возможность взять такую работу на проверку. В процессе проверки, работа может быть принята или возвращена разработчику на доработку. Если работа принята, она переходит в стадию «Готова к использованию» на этом этапе изменения в работе недопустимы.

Такая система правил покрывает большинство возможных случаев, при этом любая дополнительная информация может быть оставлена в виде комментариев к задаче. Для более широких обсуждений или для обсуждений, не связанных с конкретной задачей, пользователи могут воспользоваться встроенным форумом или личными сообщениями.

Обмен знаниями

Поскольку совместное знание является ценным ресурсом для команды, важно предоставить возможность для его формирования и сохранения. Форум, который предоставляет система дает пользователям возможность обсуждения и совместного принятия решений. Также в системе присутствует блог для организаторов, в котором они могут освещать новости проекта или выделять наиболее выдающиеся идеи. Для обмена рекомендациями, инструкциями или правилами в системе присутствует встроенная справочная система, которая может пополняться организаторами.

Заключение

В работе были рассмотрены основные этапы процесса совместной разработки компьютерных игр, а также возможности повышения эффективности работы команд с помощью программных систем на примере конкретного web-приложения для совместной разработки. Исследованы архитектурные решения, которые были приняты в процессе разработки системы, их преимущества и недостатки. Было рассмотрено как система выполняет поставленную задачу облегчения организации разработки и увеличения совместного знания разработчиков.

Рассматриваемая система была апробирована на реальных задачах. Пользователи восприняли систему положительно, но поступили также предложения по улучшению системы, такие как возможность динамического задания правил перехода между состояниями по аналогии с правилами доступа и графический интерфейс для создания правил, а также отдельный обозреватель для концепт-дизайна с дополнительными мультимедийными возможностями. В настоящий момент производится работа в данных направлениях.

Литература

1. Salas E., Cooke N. J., Rosen M. A. On Teams, Teamwork, and Team Performance: Discoveries and Developments // Human Factors. 2008. Vol. 50(3). P. 540-547.

2. Holub A. Taming Java Threads. New York: Apress. 2000. 209 p.

3. The Task Asynchronous Programming Model (TAP) with async and await (C#) | Microsoft Docs [Электронный ресурс]. // URL: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/task-asynchronous-programming-model (дата обращения: 28.04.2020)

4. Asynchronous programming in C# | Microsoft Docs [Электронный ресурс]. // URL: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/async/ (дата обращения: 05.05.2020).

WEB-APPLICATION FOR COLLABORATIVE VIDEO GAME DEVELOPMENT

M. V. Seliverstova, A. N. Danilenko

This work presents an implementation of a collaborative video game development system that allows users to improve team efficiency by using one system to manage the work process of professionals in all fields commonly found in game development. An overview of the domain knowledge was made, in particular the management of the development process, the results of which were used to gather the requirements for the system. Architectural choices that were made in the process of developing the system were examined. The system was developed as a client-server application; it uses the task asynchronous programming model. In order to provide access restriction, a flexible system that can be configured with configuration files was used. Users have many ways to track the lifecycle of the tasks, write documentation and communicate. Potential ways to improve the system in the future were also examined.

Key words: versioning, documentation, issue tracking, claim, resource, issue, web-application, asynchronous execution.

Статья поступила в редакцию 08.07.2020 г.

Seliverstova Maria Victorovna (seliver.maria@gmail.com),
student IV course of the faculty of informatics;
Danilenko Aleksandra Nikolaevna (danilenko.al@gmail.com),
associate professor of the Department of software systems of the Samara University,
443086, Russia, Samara, Moskovskoye Shosse, 34.