Министерство науки и высшего образования Российской Федерации

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ (НИ ТГУ)

КУРСОВАЯ РАБОТА

«Разработка системы управления проектами с применением Agile и Scrum»

Прокопьев Даниил Андреевич

Прикладная информатика (09.03.03)

Разработка программного обеспечения в цифровой экономике

Руководитель работы

канд. физ.-мат. Наук

Доцент кафедры прикладной информатики

А.С Морозова

*подпись*

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025г.

Автор работы

студент группы № 932204

Д.А. Прокопьева

*подпись*

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2025г.

Томск –2025

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc30421)

[1. Анализ существующих систем управления проектами 5](#_Toc29495)

[1.1 Обзор популярных решений 5](#_Toc24980)

[1.2 Сравнительный анализ 6](#_Toc2276)

[1.3 Выводы по результатам анализа 7](#_Toc9451)

[2. Методологии Agile и Scrum 9](#_Toc8036)

[2.1 Основные принципы Agile 9](#_Toc30465)

[2.2 Структура Scrum: роли, артефакты, спринты 9](#_Toc17150)

[2.3 Преимущества и ограничения методологий 10](#_Toc11963)

[2.4 Формализация требований к системе На основе принципов Agile и Scrum к разрабатываемой системе управления проектами выдвигаются следующие требования: 11](#_Toc25563)

[3. Проектирование системы 13](#_Toc25895)

[3.1 Архитектура системы 13](#_Toc8750)

[3.2 Описание компонентов и их взаимодействия 14](#_Toc32645)

[3.3 Выбор технологий 16](#_Toc25773)

[4.Реализация и тестирование MVP 19](#_Toc20063)

[4.1 Разработка минимально жизнеспособного продукта (MVP) 19](#_Toc21434)

[4.2 Тестирование системы 19](#_Toc4744)

[4.3 Анализ результатов и обратная связь от пользователей 19](#_Toc26349)

[4.4 Выводы по итогам реализации 19](#_Toc32515)

[ЗАКЛЮЧЕНИЕ 20](#_Toc24545)

[ПРИЛОЖЕНИЕ 21](#_Toc18200)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 22](#_Toc9327)

**ВВЕДЕНИЕ**

В современных ИТ‑компаниях малые и средние команды разработчиков постоянно сталкиваются с необходимостью оперативно адаптироваться к меняющимся требованиям и одновременно поддерживать высокий уровень прозрачности процессов. Традиционные водопадные модели (Waterfall), основанные на жёстком пошаговом выполнении заранее согласованного плана, не позволяют вовремя реагировать на новые запросы заказчиков и быстро меняющиеся рыночные условия. Это приводит к систематическим задержкам, перерасходу бюджета и риску получения устаревшего или неподходящего продукта на момент сдачи проекта.

Методологии Agile и Scrum предлагают решение этих проблем через итеративный подход: разработка делится на короткие спринты продолжительностью от одной до четырёх недель, чётко распределяются роли (Product Owner, Scrum‑Master, команда разработчиков), а после каждого спринта проводится демонстрация инкремента и ретроспектива. Такой цикл обеспечивает регулярное получение обратной связи, позволяет своевременно корректировать план и увеличивает вовлечённость всех участников процесса. Тем не менее существующие системы управления проектами — такие как Jira, Trello и YouTrack — чаще всего либо избыточно сложны и требуют длительного обучения для небольших команд, либо обладают ограниченной аналитикой и отчётностью, что снижает их практическую ценность в условиях динамичной разработки.

Объектом исследования в данной работе являются программные системы управления проектами, а предметом — их архитектурные и функциональные особенности, отвечающие за баланс между простотой использования и глубиной аналитики. Цель курсовой работы состоит в создании прототипа лёгкой и гибкой системы, которая объединит наглядность канбан‑доски, минимальные усилия по настройке рабочего процесса и встроенные отчёты о прогрессе.

В рамках исследования проводится сравнительный анализ аналогичных решений, формулируются подробные функциональные и нефункциональные требования, разрабатывается многослойная архитектура приложения (слои пользовательского интерфейса, бизнес‑логики и доступа к данным) с выбором современных технологий (WPF или веб‑SPA на ASP.NET Core и PostgreSQL). Затем реализуется минимально жизнеспособный продукт с ключевыми возможностями планирования спринтов, управления задачами и генерации отчётов, после чего проводится функциональное тестирование и сбор обратной связи от целевой аудитории. В результате работы будет получен готовый к масштабированию и интеграции прототип системы управления проектами для команд, практикующих Agile и Scrum.

**1. Анализ существующих систем управления проектами**

**1.1 Обзор популярных решений**

В современном IT‑ландшафте наиболее распространёнными инструментами для управления проектами считаются **Jira**, **Trello** и **YouTrack**. Несмотря на общее назначение — поддержку гибких методологий и визуализацию рабочих процессов — каждая из этих систем имеет свою философию и особенности реализации.

**Jira**  
Jira от Atlassian изначально разрабатывалась как система трекинга ошибок и задач, но со временем превратилась в мощную платформу для управления проектами любого масштаба. Основная модель взаимодействия строится вокруг **Project Backlog** и **Sprint Backlog**: пользователи создают истории, задачи и подзадачи, распределяют их по спринтам и контролируют выполнение с помощью гибко настраиваемых досок Kanban и Scrum. Сильной стороной Jira является высокая степень кастомизации рабочих процессов (workflows), возможность построения сложных отчётов (burndown‑, velocity‑диаграммы, отчёты по времени и ресурсам) и интеграция с обширной экосистемой плагинов (CI/CD, тест‑менеджмент, DevOps‑модуль). Типовые сценарии применения — крупные и распределённые команды, где требуется строгий контроль согласований и глубокая аналитика.

**Trello**  
Trello позиционируется как максимально простой и визуально понятный инструмент. В основе лежит доска Kanban с карточками, которые можно группировать в колонки «To Do», «In Progress», «Done» и любые другие. Пользователь создаёт карточку, назначает ответственного, добавляет метки и чек‑листы, а перемещение карточки отражает изменения статуса задачи. Благодаря минимальному порогу входа Trello охотно применяется малыми командами, стартапами и даже для личного планирования. Ограниченная функциональность аналитики компенсируется «Power‑Up» надстройками (календарь, отчёты, интеграции с Google Drive и Slack), но для сложной отчётности и автоматизации Trello менее пригоден.

**YouTrack**  
YouTrack от JetBrains сочетает в себе гибкость настройки Jira и лёгкость Trello. Система поддерживает доски Kanban/Scrum, формируемые на основе гибких запросов к базе задач (по приоритетам, тегам, исполнителям). YouTrack предлагает встроенный **Time Tracking**, **Agile‑дэшборды** и механизм создания правил автоматизации (сценариев на JavaScript), который позволяет реализовать адаптивный workflow без внешних плагинов. Интерфейс менее «канбан‑оптимизирован» по сравнению с Trello, но предоставляет больше возможностей для быстрой фильтрации и группировки задач. YouTrack часто выбирают команды, использующие продукты JetBrains и ценящие тесную интеграцию с IDE (например, IntelliJ IDEA, Rider) и Git‑репозиториями.

**1.2 Сравнительный анализ**

| **Критерий** | **Jira** | **Trello** | **YouTrack** |
| --- | --- | --- | --- |
| Простота освоения | Низкая: много настроек, долгий ввод в курс дела | Очень высокая: интуитивный интерфейс, минимум элементов | Средняя: базовый Kanban понятен, но сложнее Trello |
| Гибкость настройки | Очень высокая: кастомные workflows, поля, экраны | Низкая: фиксированная модель доски, ограниченные Power‑Ups | Высокая: правила автоматизации, адаптивные доски |
| Аналитика и отчётность | Расширенная: burndown, velocity, отчёты по времени | Ограниченная: базовые отчёты через Power‑Ups | Средняя: Agile‑дашборды, Time Tracking, кастомные отчёты |
| Интеграции и расширения | Широкая экосистема плагинов и API | Через Power‑Ups (ограниченный набор бесплатных) | Встроенные интеграции с Git, IDE JetBrains; сценарии на JS |
| Стоимость внедрения и поддержки | Высокая: лицензии + обучение команды | Минимальная: бесплатный базовый тариф | Средняя: есть бесплатная версия, коммерческие тарифы |

Таблица 1- Сравнительная таблица популярных систем управления проектами.

На основе таблицы можно выделить следующие ключевые моменты:

**Jira** превосходит конкурентов по глубине аналитики и гибкости, но требует значительных усилий на настройку и обучение.

**Trello** практически не требует внедрения, но из‑за ограниченной отчётности и возможности кастомизации малопригоден для зрелых Agile‑команд с потребностью в аналитике.

**YouTrack** удачно сочетает гибкость и простоту: предоставляет более развитые возможности автоматизации и отчётов, чем Trello, и при этом менее тяжеловесен, чем Jira.

**1.3 Выводы по результатам анализа**

1. Существующие решения формируют «трёхточечный спектр»: от лёгкости Trello до мощной аналитики Jira.
2. Малые и средние команды часто оказываются «в ловушке» между простым интерфейсом и потребностью в отчётности.
3. YouTrack близок к идеалу, но его интерфейс и логика фильтрации не всегда интуитивно понятны для пользователей без опыта работы с IDE‑продуктами JetBrains.
4. Нет инструмента, который сочетал бы минимальный порог входа, наглядную канбан‑доску и встроенные средства глубокой аналитики без сторонних плагинов.

Таким образом, в качестве отправной точки для разработки собственной системы оборачивается задача объединить лёгкость освоения Trello, гибкость настройки YouTrack и отчётность Jira, исключив избыточную сложность и ускорив внедрение. Переход к формализации требований и проектированию системы вынесен в следующий раздел.

**2. Методологии Agile и Scrum**

**2.1 Основные принципы Agile**

**Ценность людей и взаимодействия** превыше процессов и инструментов. Команды ставят в центр внимание на эффективном общении и самоорганизации.

**Рабочий продукт** важнее объёмной документации. Регулярные поставки минимально жизнеспособных версий дают раннюю обратную связь.

**Сотрудничество с заказчиком** на всех этапах разработки. Готовность к изменениям требований даже на поздних стадиях повышает ценность конечного результата.

**Адаптивность к изменениям** важнее строгого следования изначальному плану. Команда регулярно планирует и пересматривает приоритеты, опираясь на свежие данные и аналитические выводы.

**2.2 Структура Scrum: роли, артефакты, спринты**

**Роли**

1. Product Owner — формулирует и приоритизирует требования в Product Backlog, выступает связующим звеном между бизнес‑сторонами и командой.
2. Scrum‑Master — следит за соблюдением Scrum‑процесса, устраняет препятствия и помогает команде работать максимально эффективно.
3. Команда разработки — кросс‑функциональная группа специалистов, самостоятельно планирующая и исполняющая работу над задачами спринта.

**Артефакты**

1. Product Backlog — упорядоченный список всех требований и пожеланий к продукту.
2. Sprint Backlog — отобранный набор задач из Product Backlog, подлежащий выполнению в рамках текущего спринта.
3. Инкремент — рабочая версия продукта, готовая к демонстрации и потенциальному релизу после завершения спринта.

**Спринты**

1. Фиксированные итерации (1–4 недели), в начале которых проводится Sprint Planning, а в конце — Sprint Review и Retrospective.
2. Каждая итерация позволяет оцени­вать прогресс и при необходимости корректировать дальнейшую работу.

**2.3 Преимущества и ограничения методологий**

**Преимущества**

1. Увеличение прозрачности процессов и предсказуемости сроков за счёт регулярных встреч и отчётов.
2. Быстрая реакция на изменения благодаря коротким спринтам и гибкому решения задач.
3. Повышение мотивации команды через участие в планировании и ретроспективах.

**Ограничения**

1. Потребность в дисциплине и зрелости команды: без самоорганизации Scrum теряет свою эффективность.
2. Риск «постоянных изменений» без чёткого видения продукта, если Product Owner недостаточно опытен.
3. Зависимость от регулярной обратной связи: без включённого заказчика итеративность становится формальностью.

**2.4 Формализация требований к системе**  
На основе принципов Agile и Scrum к разрабатываемой системе управления проектами выдвигаются следующие требования:

**Гибкая настройка рабочего процесса**: возможность создавать и модифицировать доски Kanban/Scrum, настраивать статусы и переходы задач.

**Поддержка итеративного планирования**: функционал для формирования и управления Sprint Backlog, включая установку длительности спринта и прогнозирование загрузки команды.

**Инструменты для ролей Scrum**: разделение прав и представлений для Product Owner (управление бэклогом), Scrum‑Master (мониторинг impediments) и разработчиков (видимость своих задач).

**Прозрачная визуализация прогресса**: канбан‑доска с drag‑and‑drop, диаграммы Burndown/Velocity и отчёты по статусам задач.

**Механизмы обратной связи**: интеграция с комментариями, уведомлениями и возможностью оценки сделанной работы на ретроспективе.

**Лёгкость внедрения**: минимальный порог входа — простой интерфейс, предустановленные шаблоны проектов и база знаний для быстрого старта.

**Расширяемая архитектура**: модульность для добавления новых отчётов, интеграций и автоматизаций без переработки основного кода.

**3. Проектирование системы**

## 3.1 Архитектура системы

Разрабатываемая система управления проектами строится по классической трёхслойной архитектуре (Presentation → Business Logic → Data Access), адаптированной под веб‑стек HTML/CSS/JavaScript и C# Web API:

**Presentation Layer (Front‑end)**  
На уровне представления располагается SPA‑приложение, реализованное на HTML5/CSS3/JavaScript с использованием лёгкого фреймворка (Vue.js, React или Angular). Этот слой отвечает за всю работу с интерфейсом: от отображения канбан‑доски и форм планирования спринтов до визуализации диаграмм Burndown и Velocity. Внутри приложения используется архитектурный паттерн MVC/MVVM (компоненты — View, состояния — Model, логика обновления — ViewModel/Controller), что упрощает поддержку и расширение кода.

**Business Logic Layer (Back‑end)**  
Серверная часть представляет собой ASP.NET Core Web API на C# (.NET 7+). Она инкапсулирует все правила предметной области: создание и завершение спринтов, управление жизненным циклом задач, расчёт и сбор отчётов, аутентификация и авторизация пользователей. Каждый эндпойнт контроллера обёрнут в слой сервисов (например, SprintService, TaskService, ReportService), который не зависит от конкретного механизма хранения данных и работает через интерфейсы репозиториев. Это упрощает юнит‑тестирование и позволяет легко менять реализацию хранилища.

**Data Access Layer + Database**  
Уровень хранения данных реализован через паттерн «Репозиторий» и ORM Entity Framework Core (либо Dapper для критических по производительности операций). Сущности (Project, Sprint, Task, User, Role, Comment, Report) формируют модель предметной области, а миграции EF Core управляют схемой реляционной БД (PostgreSQL или SQL Server). Все операции CRUD проходят через репозитории, которые обеспечивают транзакционность и централизованную обработку ошибок на уровне доступа к данным.

Между слоями налажено взаимодействие по чётким контрактам: SPA‑клиент общается с Web API через безопасные HTTPS‑запросы и получает данные в формате JSON, бизнес‑логика обменивается данными с DAL по интерфейсам репозиториев, а DAL — с базой через провайдер ADO.NET. Такая архитектура обеспечивает независимое масштабирование фронтенда и бэкенда, упрощает CI/CD и разделение ответственности в команде разработки.

## 3.2 Описание компонентов и их взаимодействия

**Клиентские компоненты**

**Компонент «Канбан‑доска»** отображает задачи, сгруппированные по статусам. Обновляется в реальном времени через AJAX‑вызовы к Web API или через WebSocket‑соединение (SignalR). Поддерживает drag‑and‑drop для изменения статусов задач.

**Модуль планирования спринтов** позволяет создавать спринты, задавать их длительность, выбирать задачи из бэклога и автоматически рассчитывать загрузку команды.

**Дашборды и отчёты** визуализируют метрики (Burndown, Velocity, статус задач) через встроенные JS‑библиотеки (Chart.js, D3.js).

**Серверные компоненты**

**Контроллеры API** (ProjectsController, SprintsController, TasksController, ReportsController) принимают HTTP‑запросы, валидируют входные данные и передают их в сервисы.

**Сервисный слой** инкапсулирует бизнес‑логику:

SprintService — управление жизненным циклом спринтов, расчёт показателей оставшихся задач;

TaskService — создание/редактирование задач, назначение исполнителей, ведение истории изменений;

ReportService — генерация диаграмм и экспорт данных.

**Middleware** выполняет аутентификацию (JWT), авторизацию (на уровне ролей), логирование запросов (Serilog) и централизованную обработку исключений.

**DAL и база данных**

**Репозитории** (IProjectRepository, ISprintRepository и т. д.) предлагают CRUD‑методы и специфичные для предметной области запросы (например, получение задач по приоритету и срокам спринта).

**СУБД** хранит данные в реляционной форме, обеспечивает целостность через внешние ключи и индексы, а также поддерживает хранение JSON‑полей для гибких отчётов.

**Сценарий взаимодействия**: пользователь перетягивает карточку задачи на фронтенде → JS‑клиент отправляет PUT‑запрос на TasksController → контроллер передаёт запрос в TaskService → сервис вызывает метод репозитория для обновления статуса в БД → репозиторий сохраняет изменения → TaskService возвращает результат контроллеру → фронтенд обновляет UI и, при подключённом SignalR, отправляет оповещение остальным клиентам для синхронизации досок.

## 3.3 **Выбор технологий**

**Фронтенд**

**HTML5/CSS3/JavaScript (ES6+)** — стандартные веб‑технологии, гарантирующие кросс‑браузерность и лёгкий старт.

**Vue.js (или React/Angular)** — выбор фреймворка определяется опытом команды; Vue.js хороша простотой интеграции и небольшой «весом» проекта.

**SignalR** — для оповещения клиентов о событиях в реальном времени (перемещения задач, завершения спринта).

**Chart.js / D3.js** — библиотеки для визуализации отчётов.

**Бэкенд**

**ASP.NET Core Web API (.NET 7+)** — высокопроизводительная и кроссплатформенная среда для создания REST‑сервисов.

**Entity Framework Core** — ORM с поддержкой миграций и LINQ‑запросов, упрощающая разработку DAL.

**Serilog** — структурированное логирование запросов и ошибок.

**Swagger / OpenAPI** — автоматическая генерация документации API.

**Инфраструктура**

**PostgreSQL** — надёжная и масштабируемая СУБД с расширенными возможностями (JSONB, полнотекстовый поиск).

**Docker / Docker Compose** — контейнеризация всех компонентов для упрощённого развёртывания и тестирования.

**CI/CD (GitHub Actions / Azure DevOps)** — автоматизация сборки, тестирования и деплоя в staging/production.

**Redis (опционально)** — кеширование горячих данных и управление сессиями.

Такой стек технологий и чёткое разделение на слои обеспечат надёжность, простоту сопровождения и возможность дальнейшего расширения системы под любые требования Agile‑ и Scrum‑команд.

1. **Реализация и тестирование MVP**

4.1 Разработка минимально жизнеспособного продукта (MVP)

4.2 Тестирование системы

4.3 Анализ результатов и обратная связь от пользователей

4.4 Выводы по итогам реализации

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения курсовой работы была разработана и апробирована прототип системы управления проектами, основанный на принципах Agile и Scrum. Проведённый анализ популярных решений (Jira, Trello, YouTrack) показал, что малым и средним командам зачастую недостаёт баланса между простотой использования, гибкостью настройки и встроенной аналитикой. На этой основе были сформулированы подробные функциональные и нефункциональные требования, включающие поддержку канбан‑доски, итеративного планирования спринтов, ролей Scrum и автоматической генерации отчётов.

В архитектурном плане система реализована по трёхслойной модели: веб‑клиент на HTML/CSS/JS (Vue.js), REST API на ASP.NET Core и реляционная база данных (PostgreSQL) с уровнем доступа через Entity Framework Core. MVP включает ключевые сценарии — создание и управление проектами, планирование и выполнение спринтов, интерактивная канбан‑доска и диаграмма Burndown — и прошёл функциональное и пользовательское тестирование. Положительная оценка удобства интерфейса и стабильности базовых функций подтвердили обоснованность выбора стека технологий и архитектурных подходов.

Выявленные пользователями пожелания — расширенная фильтрация задач, экспорт отчётов, валидация планирования и система прав доступа — формируют план дальнейших итераций. В следующих версиях предполагается добавить диаграммы Velocity и Cumulative Flow, модуль аналитики на основе JSON‑отчётов и интеграции с внешними сервисами (Slack, Git). Последовательное развитие прототипа по Agile‑циклам позволит создать масштабируемый инструмент, оптимально сочетающий лёгкость внедрения и глубину аналитики для команд, практикующих гибкие методологии управления проектами.

**ПРИЛОЖЕНИЕ**

**СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ**

1. Бочкарёв С. В. Управление проектами в информационных системах. — М.: Инфра-М, 2020. — 304 с.
2. Кнут Д. Э. Искусство программирования. Том 1. Основные алгоритмы. — М.: Вильямс, 2019. — 672 с.
3. Schwaber K., Sutherland J. The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game, 2020.  
   URL: [https://scrumguides.org](https://scrumguides.org" \t "_new) (дата обращения: 15.04.2025).
4. Manifesto for Agile Software Development. URL: [https://agilemanifesto.org](https://agilemanifesto.org" \t "_new) (дата обращения: 15.04.2025).
5. Fowler M. Patterns of Enterprise Application Architecture. — Addison-Wesley, 2003. — 560 p.
6. Freeman E., Robson E. Head First Design Patterns. — O’Reilly Media, 2021. — 694 p.
7. Microsoft Docs. ASP.NET Core Web API documentation.  
   URL: [https://learn.microsoft.com/aspnet/core/web-api](https://learn.microsoft.com/aspnet/core/web-api" \t "_new) (дата обращения: 15.04.2025).
8. PostgreSQL Documentation. URL: [https://www.postgresql.org/docs/](https://www.postgresql.org/docs/" \t "_new) (дата обращения: 15.04.2025).
9. Atlassian Jira Software documentation.  
   URL: https://support.atlassian.com/jira-software (дата обращения: 15.04.2025).
10. JetBrains YouTrack Documentation.  
    URL: [https://www.jetbrains.com/help/youtrack](https://www.jetbrains.com/help/youtrack" \t "_new) (дата обращения: 15.04.2025).