НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО

Факультет Программной Инженерии и Компьютерной Техники

# Экономика программной инженерии

Лабораторная работа № 1

Выполнили студенты:

Гречкина Юлия, Султанов Халил

Группа № P34232

Преподаватель: Машина Екатерина Алексеевна

г. Санкт-Петербург

2024

Для выданного веб-проекта https://godbolt.org:

1. Сформировать набор функциональных требований для разработки проекта.
2. Оценить трудоемкость разработки проекта наивным методом.
3. Оценить трудоемкость разработки проекта методом PERT (Project Evaluation and Review Technique**).** Нарисовать сетевую диаграмму взаимосвязи работ и методом критического пути рассчитать минимальную продолжительность разработки. Предложить оптимальное количество разработчиков и оценить срок выполнения проекта.
4. Оценить размер проекта методом функциональных точек, затем, исходя из предположения, что собранной статистики по завершенным проектам нет, рассчитать трудоемкость методом COCOMO II (Обновленная таблица количества строк на точку для разных языков программирования)
5. Оценить размер проекта методом оценки вариантов использования (Use Case Points). Для расчета фактора продуктивности PF использовать любой свой завершенный проект с известными временными трудозатратами, оценив его размер методом UCP.
6. Сравнить полученные результаты и сделать выводы.

**Выполнение работы.**

Godbolt, также известный как Compiler Explorer, — это веб-приложение, которое позволяет пользователям писать, компилировать и просматривать ассемблерный код для различных языков программирования и компиляторов в реальном времени.

**Структура сайта https://godbolt.org:**

1. Основные компоненты:
    - Веб-приложение
    - База данных компиляторов и исходного кода
    - Поддержка мобильных устройств
2. Предполагаются следующие разделы для веб-приложения:
    - Главная страница (Основная функциональность)
        - Описание сайта и его возможностей
        - Основные инструменты (компиляция, отладка кода)
    - Редактор кода
        - Полнофункциональный редактор кода
        - Подсветка синтаксиса
        - Возможность выбора языка программирования
        - Поддержка нескольких вкладок
    - Компиляторы и их конфигурации
        - Список доступных компиляторов
        - Выбор версии компилятора
        - Настройки компиляции (опции компилятора)
    - Визуализация результата компиляции
        - Текстовый вывод (ошибки/предупреждения)
        - Ассемблерный код
        - Анализ производительности (оптимизации, использование памяти)
    - Профилирование кода
        - Опции для анализа производительности
        - Инструменты визуализации (графики)

    - Примеры кода и готовые шаблоны
        - Список часто используемых шаблонов программ
    - Форма обратной связи
        - Отправка баг-репортов и предложений
    - О проекте
        - Информация о разработчиках
        - Документация и ссылки на GitHub
3. Предполагаются следующие разделы меню:
    - Главная
    - Редактор кода
    - Компиляторы
    - Профилирование
    - Примеры кода
    - Обратная связь
    - О проекте

Общие блоки:

- Динамическое верхнее меню: позволяет выбирать язык программирования, компилятор, настройки компиляции
- Статический блок с контактами: ссылки на GitHub проекта, соцсети
- Статический блок с юридической информацией: лицензии, права на использование
- Динамический блок с результатами компиляции: отображение ассемблерного кода и ошибок компиляции

1. Список функциональных требований:
    1. Пользователь должен иметь возможность писать и редактировать код в онлайн-редакторе с поддержкой подсветки синтаксиса.
    2. Пользователь должен иметь возможность выбрать компилятор для компиляции кода.
    3. Пользователь должен иметь возможность запускать компиляцию кода и видеть результат (вывод ассемблера, ошибки).
    4. Пользователь должен иметь возможность выбирать различные версии компиляторов и изменять их настройки.
    5. Пользователь должен иметь возможность визуализировать результат компиляции, включая ассемблерный код и предупреждения/ошибки.
    6. Пользователь должен иметь возможность сохранять и загружать свои проекты.
    7. Пользователь должен иметь возможность использовать встроенные примеры кода для тестирования.
    8. Пользователь должен иметь возможность анализировать производительность кода, используя профилирование и визуализацию.
    9. Пользователь должен иметь возможность делиться кодом с другими пользователями через ссылки.
    10. Администратор должен иметь возможность добавлять и обновлять список доступных компиляторов.
    11. Администратор должен иметь возможность управлять конфигурациями компиляторов и их параметрами.
    12. Пользователь должен иметь возможность просматривать историю компиляций и их результаты.
    13. Пользователь должен иметь возможность менять настройки отображения кода (темы, размер шрифта).
    14. Пользователь должен иметь возможность отправлять обратную связь или репортить баги через форму обратной связи.
    15. Система должна обеспечивать быструю компиляцию кода с минимальными задержками.

Система должна предоставлять пользователю интуитивно понятную навигационную панель, которая включает доступ ко всем ключевым функциям сайта. Основные элементы панели включают:

### 1.1.1 Логотип сайта.

Нажатие на логотип возвращает пользователя на главную страницу сайта для удобного возврата к основному интерфейсу.

### 1.1.2 Кнопка Add.

Система должна предоставлять пользователю возможность добавлять элементы в рабочее пространство, такие как:

- Source Editor: добавляет текстовый редактор для написания и редактирования кода. Пользователь может закрывать, расширять окно, изменять название файла.
- Diff view: визуализирует различия между двумя версиями кода, облегчая их сравнение. Окно можно закрыть, расширить или переименовать.
- Tree (IDE mode): включает режим древовидного представления проекта, что помогает легко ориентироваться в структуре файлов. Окно можно закрыть, расширить или переименовать.

### 1.1.3 Кнопка More.

Система предоставляет доступ к расширенным настройкам и дополнительным функциям, таким как:

- Settings: пользователь может изменять параметры компиляции и настройки интерфейса.
- Reset code and UI layout: сбрасывает код и макет интерфейса к исходному состоянию.
- Open new tab: открывает новую вкладку для работы над другим проектом или вариантом кода.
- History: предоставляет доступ к истории изменений кода, что позволяет вернуться к предыдущим состояниям проекта.
- Apply Default Font Scale: восстанавливает стандартный размер шрифта, если пользователь изменял его.

### 1.1.4 Кнопка Templates.

Система должна предоставлять готовые шаблоны кода, чтобы ускорить создание нового проекта или примера.

### 1.1.5 Share.

Система должна включать возможность поделиться проектом через:

- Short Link: система генерирует короткую ссылку для удобной передачи.
- Full Link: генерирует полную ссылку на проект, которую можно скопировать и передать.

- Embed in iframe: система должна генерировать код для встраивания проекта на другие веб-сайты через iframe.

1.1.6 Policies.

Система должна предоставлять пользователю доступ к юридической информации, такой как:

- Cookie policy: система отображает информацию о политике использования cookie с опциями согласия и отказа.
- Privacy policy: система должна отображать политику конфиденциальности данных.

1.1.7 Other.

Кнопка "Other" должна предоставлять доступ к следующим внешним ресурсам и опциям:

- Become a Patron, Sponsor on GitHub, Donate via PayPal: система должна предоставлять возможность поддержать проект через разные платформы.
- Source on GitHub: система должна предоставлять ссылку на исходный код проекта на GitHub.
- Mailing List: возможность подписаться на новости и обновления.
- Report an issue: система должна позволять отправлять отчеты об ошибках разработчикам.
- How it works: описание функциональности сайта.
- Statistics, Changelog, Version tree: доступ к статистике использования сайта, истории изменений и древу версий проекта.
- Installed libraries: система должна предоставлять пользователю доступ к списку установленных библиотек, которые можно использовать для работы с кодом. Этот список облегчает выбор нужных библиотек для проекта.
- Wiki: система должна предоставлять ссылку на вики проекта, где пользователи могут найти документацию, статьи и инструкции по использованию различных функций сайта и работы с проектом.
- Contact the author: система должна предоставлять возможность пользователю напрямую связаться с автором проекта для обратной связи или обсуждения любых вопросов, связанных с проектом.
- Tweet, Share on Reddit: система должна включать кнопку для быстрой публикации ссылки на проект в Twitter, Reddit.

1.2 Окно редактора кода.

Система должна предоставлять пользователю удобный и настраиваемый редактор кода, который поддерживает множество функций для эффективной работы с проектами. Редактор должен быть гибким, адаптированным под нужды пользователя, и обеспечивать комфортное взаимодействие с кодом. В окне редактора должны быть реализованы следующие функции:

### 1.2.1 Имя темплейта.

Система должна позволять пользователю задать имя для своего шаблона кода, чтобы легко идентифицировать и повторно использовать его в дальнейшем. Это улучшает организацию проектов и упрощает навигацию по созданным шаблонам.

### 1.2.2 Размер шрифта.

Система должна предоставлять возможность выбора размера шрифта в редакторе кода. Доступные размеры должны варьироваться от 8 до 30, чтобы пользователи могли настроить отображение текста в зависимости от своих предпочтений и удобства работы.

### 1.2.3 Сохранение/Загрузка

- Examples
- Browser-local Storage
    - Save to browser-local storage: Сохранение проекта локально в браузере для дальнейшего использования.
- Browser-local History: система должна предоставлять доступ к истории сохранений проекта в локальном хранилище браузера, позволяя пользователю восстанавливать предыдущие версии.
- File-System
    - Load from a local file: система должна позволять загружать проект из локального файла.
    - Save to file: система должна предоставлять возможность сохранения проекта в файл на локальный диск.

### 1.2.4 Добавление новых элементов (Add new)

Система должна предоставлять возможность добавлять следующие элементы в рабочее пространство:

- Compiler: Добавление нового компилятора для работы с кодом.
- Execution only: Запуск кода без отображения исходного текста в окне редактора.
- Conformance View: Просмотр результатов компиляции с точки зрения соответствия стандартам.
- Source Editor: возможность добавления еще одного редактора исходного кода для одновременной работы с несколькими файлами.

### 1.2.5 Vim

Система должна предоставлять возможность включения функциональности редактора Vim для пользователей, которые предпочитают использовать клавиатурные команды для редактирования кода. Это расширяет функциональные возможности редактора и удовлетворяет потребности опытных разработчиков.

### 1.2.6 AppInsights

При выборе определенных языков программирования система должна предоставлять возможность отображения аналитики по проекту с помощью AppInsights. Это позволяет пользователю отслеживать производительность и использование ресурсов в процессе разработки.

### 1.2.7 Quick-bench

Система должна включать функцию быстрого выполнения кода и замера времени выполнения с помощью Quick-bench. Эта функция доступна при выборе определенных языков программирования и полезна для оптимизации производительности.

### 1.2.8 Выбор языков программирования

Система должна предоставлять пользователю возможность выбора языка программирования для работы с проектом. Этот выбор должен быть гибким, с поддержкой популярных языков программирования для различных типов задач.

### 1.2.9 Стандартные функции редактора

Редактор должен поддерживать базовые функции для работы с кодом, такие как:

- Подсветка синтаксиса для различных языков программирования
- Автодополнение кода
- Проверка синтаксиса и отображение ошибок в режиме реального времени.
- Интеграция с системой сборки и тестирования
- Поддержка клавиатурных сокращений (hotkeys) для выполнения стандартных действий (копирование, вставка, отмена, возврат).
- Форматирование кода для улучшения читаемости.

## 1.3 Окно компиляции

Система должна предоставлять пользователю окно компиляции, которое может быть расширено, минимизировано или закрыто, обеспечивая гибкость настройки рабочей среды. Окно компиляции должно быть адаптированным и функционально насыщенным, чтобы пользователи могли максимально эффективно анализировать и работать с результатами компиляции кода. Окно компиляции должно включать следующие функции:

### 1.3.1 Размер шрифта (Fontsize)

Система должна предоставлять возможность выбора размера шрифта для отображения вывода компиляции. Доступные размеры должны варьироваться от 8 до 30, чтобы пользователи могли настроить отображение результатов в соответствии с их предпочтениями.

### 1.3.2 Опции вывода компилятора (Compiler Output Options)

Отображает результаты компиляции кода. Включает такие опции:

- Compile to binary object: Компиляция кода в бинарный объект, который можно использовать в других проектах или библиотеках.
- Link to Library: Привязка скомпилированного кода к библиотеке.
- Execute the code: Выполнение кода с выводом результатов.
- Intel asm syntax: Вывод результата компиляции в формате ассемблерного кода Intel.
- Demangle identifiers: Преобразование сложных имен в упрощенные.

### 1.3.3 Фильтры для результата компиляции (Filter)

Система должна предоставлять пользователю возможность применения фильтров к результатам компиляции для упрощения анализа:

- Unused labels: Исключение из вывод неиспользуемых меток.
- Library functions: Исключение стандартных функций библиотек для более точного анализа пользовательского кода..
- Directives: Исключение директив компилятора.
- Comments: Исключение комментариев.
- Horizontal whitespace: Исключение пробелов.
- Debug intrinsics: Исключение отладочной информации.

### 1.3.4 Поиск и выбор библиотек (Libraries)

- Поиск текста: система должна предоставлять поле для поиска библиотек, чтобы пользователь мог быстро находить необходимые зависимости для проекта.
- Выбор библиотек: система должна поддерживать возможность выбора библиотек для работы с кодом, основываясь на параметрах, заданных пользователем.

### 1.3.5 Переопределение настроек компилятора (Overrides)

Система должна поддерживать функцию переопределения настроек компилятора через ввод параметров в формате KEY=VALUE. Это позволяет пользователям настраивать компилятор в зависимости от их специфических потребностей и целей.

### 1.3.6 Добавление новых элементов (Add new)

Система должна поддерживать возможность добавления новых инструментов и функций для работы с кодом:

- Clone Compiler: клонирование текущего компилятора для работы с его дубликатом.
- Executor From This: создание нового исполнителя на основе существующего компилятора для выполнения кода.
- Stack Usage: система должна отображать информацию об использовании стека программой, что полезно для отладки и оптимизации.
- Preprocessor: Добавление препроцессора для анализа кода.
- GCC Tree/RTL: Просмотр внутреннего представления GCC кода, что полезно для углубленного анализа и оптимизации.
- Control Flow Graph: система должна позволять пользователю просматривать граф управления потоком программы, чтобы лучше понимать структуру и логику кода.

### 1.3.7 Add tool

Система должна позволять пользователю выбирать и добавлять инструменты в зависимости от их потребностей, обеспечивая гибкость в анализе и оптимизации кода:

- Форматирование и анализ: clang-format, clang-tidy, clang-query, include-what-you-use.
- Производительность и архитектура: llvm-mca, OSACA, Sonar.
- Анализ бинарных файлов: readelf, nm, pahole, strings, llvm-dwarfdump, bloaty.
- Дополнительные: ldd, x86-to-6502.

### 1.3.8 Выбор компилятора

Система должна поддерживать гибкий выбор компиляторов, включая добавление их в избранное для быстрого доступа. Пользователь должен иметь возможность:

- Расширить окно выбора компиляторов (Compiler picker popup): система должна предоставлять возможность расширить интерфейс для более удобного просмотра всех доступных компиляторов.
- Изменить заголовок окна: система должна поддерживать возможность смены заголовка окна выбора компиляторов для персонализации интерфейса.
- Показ всех опций компилятора: система должна предоставлять кнопку для отображения всех доступных опций компилятора.
- Поле ввода опций компилятора: система должна включать поле для ввода дополнительных опций компилятора.

### 1.3.9 Дополнительные функции

- Кнопка перезагрузки компилятора: система должна предоставлять возможность перезагрузки выбранного компилятора для применения изменений.
- Output кнопка: система должна поддерживать возможность открытия вывода компилятора в новом окне для более удобного просмотра результатов.
- Кнопка информации о компиляторе: система должна позволять переключаться между краткой и полной версией отображаемой информации о компиляторе.
- Кнопка временных показателей ping: система должна отображать информацию о времени отклика сервера для текущего проекта, что полезно для оценки производительности.
- Лицензия компилятора: система должна предоставлять пользователю информацию о лицензионных условиях использования выбранного компилятора.

### 1.3.10 Поддержка компиляции

Система должна поддерживать компиляцию исходного кода для различных языков программирования. Пользователь может выбрать целевой язык программирования для компиляции. После компиляции система должна предоставить возможность выполнения скомпилированного кода и отображения его вывода в окне компиляции."

1.4 Окно вывода

Система должна предоставлять пользователю окно вывода, где отображаются результаты выполнения кода или компиляции. Окно вывода должно быть функциональным и адаптированным для удобного просмотра и анализа данных. В окне вывода должны быть реализованы следующие дополнительные функции:

- Система должна предоставлять возможность пользователю переименовать окно вывода
- Выбор шрифта: Система должна предоставлять возможность изменения шрифта в окне вывода
- Wrap lines галочка: Система должна предоставлять функцию переноса строк (через галочку "Wrap lines"), что позволяет отображать длинные строки полностью на экране без горизонтальной прокрутки..
- Select All: Система должна включать кнопку для выделения всего текста в окне вывода.

1.5 Движение окон и гибкость интерфейса

1.5.1 Система должна позволять пользователю двигать все окна интерфейса (редактор, окно компиляции, окно вывода и другие) по рабочей области. Это обеспечит гибкость настройки рабочей среды, чтобы пользователи могли организовать окна так, как им удобно, в зависимости от текущих задач и предпочтений.

1.5.2 Система должна поддерживать адаптивный дизайн, позволяющий использовать функционал на других устройствах.

## 2. Оценка трудоемкости разработки проекта наивным методом

| № | Раздел | Оценка минимальная (чел.часы) | Оценка максимальная (чел.часы) |
|---|---|---|---|
| 1 | Дизайн | 320 | 640 |
| 2 | Разработка функциональности | 920 | 1800 |
| 3 | Верстка и программирование | 480 | 960 |
| 4 | Контент | 40 | 80 |
| 5 | Адаптивность и респонсивный дизайн | 20 | 40 |
| 6 | Тестирование и оптимизация | 30 | 50 |
| 7 | Управление проектом | 30 | 50 |
| 8 | Внедрение и запуск | 15 | 40 |

**Дизайн** включает в себя время, потраченное на создание дизайна сайта, включая логотип, цветовую схему, компоновку страниц и элементы интерфейса

**Разработка функциональности** включает в себя время, потраченное на необходимые функции сайта визитки, такие как навигация, контактная форма, отображение информации о компании или человеке, загрузка изображений и другие возможности

**Верстка и программирование** включает в себя время, потраченное на верстку и программирование, включая создание HTML-разметки, CSS-стилей и программного кода для реализации функциональности сайта.

**Контент** - время, требуемое для написания и форматирования текстового контента, создания графических изображений и подготовки других материалов, которые должны быть размещены на сайте

**Адаптивность и респонсивный дизайн** - время, потраченное на создание адаптивного дизайна, который будет корректно отображаться на различных устройствах, таких как мобильные телефоны и планшеты.

**Тестирование и оптимизация** - время, необходимое для тестирования функциональности сайта, обнаружения и устранения ошибок, а также оптимизации производительности и SEO

**Управление проектом** - время, затраченное на координацию с клиентом, общение, планирование и другие административные задачи, связанные с разработкой проекта

**Внедрение и запуск** - временные затраты на финальную подготовку сайта к запуску, его размещение на хостинге и настройку домена.

### 3. Оценка проекта методом PERT

| № | Раздел | Наиболее вероятностная оценка трудозатрат, Mi | Оптимистичная оценка, Oi | Пессимистичная оценка, Pi | Оценка средней трудоемкости проекта | Среднеквадратичное отклонение |
|---|---|---|---|---|---|---|
| 1 | Дизайн | 40 | 30 | 50 | 40 | 3,33 |
| 2 | Разработка функциональности | 45 | 30 | 60 | 45 | 5,00 |
| 3 | Верстка и программирование | 115 | 80 | 150 | 115 | 11,67 |
| 4 | Контент | 60 | 40 | 80 | 60 | 6,67 |
| 5 | Адаптивность и респонсивный дизайн | 30 | 20 | 40 | 30 | 3,33 |
| 6 | Тестирование и оптимизация | 35 | 20 | 50 | 35 | 5,00 |
| 7 | Управление проектом | 40 | 30 | 50 | 40 | 3,33 |
| 8 | Внедрение и запуск | 27,5 | 15 | 40 | 27,5 | 4,17 |

$E_i = (P_i + 4M_i + O_i)/6$ – оценка средней трудоемкости проекта

$СКО_i = (P_i - O_i) / 6$ – среднеквадратичное отклонение

$E = \sum Ei$ – общая оценка трудоемкости = 392.5

$СКО = \sqrt{\Sigma СКО^2 i}$ – среднеквадратичное отклонение для оценки суммарной трудоемкости = 16.77

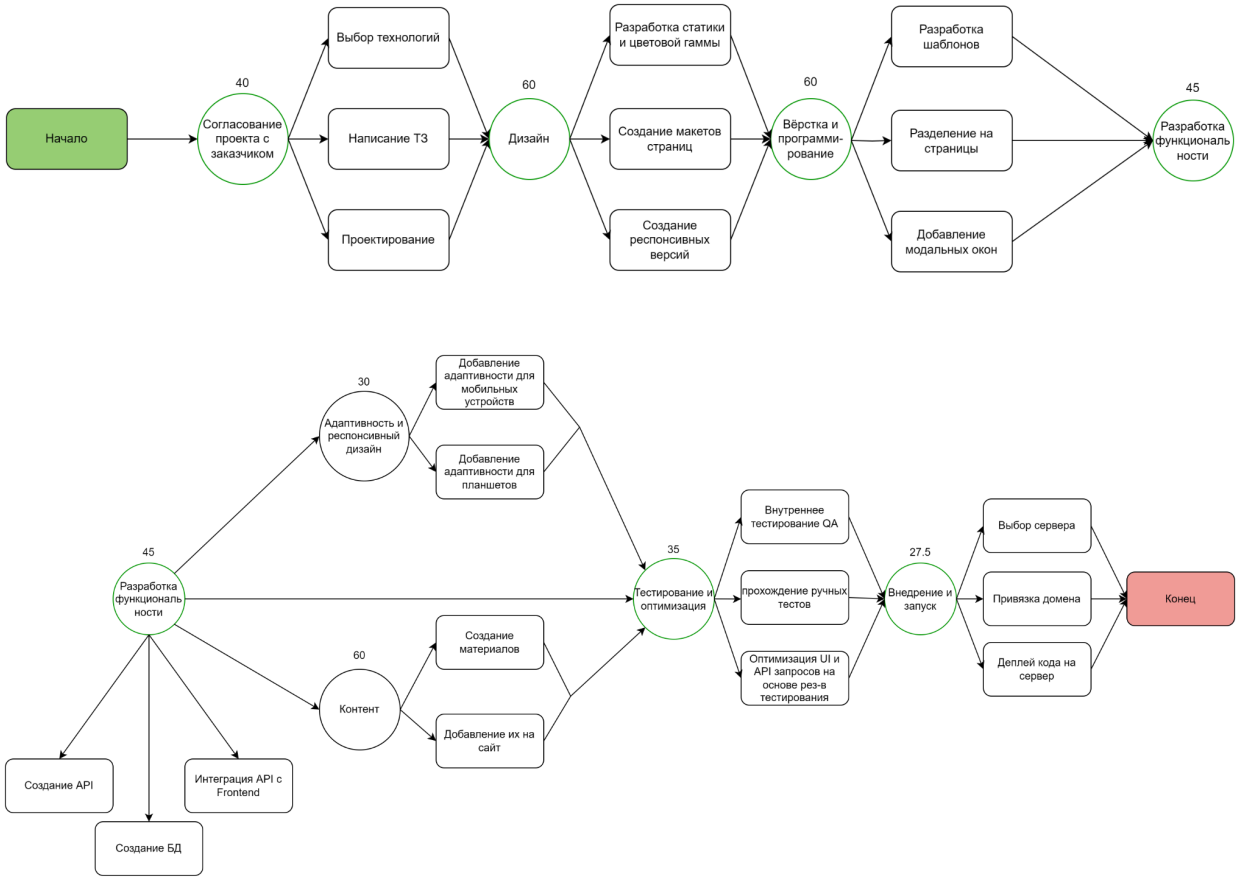
Оптимистичная оценка суммарной трудоемкости проекта = 265

Пессимистичная оценка суммарной трудоемкости проекта = 520

Наиболее вероятностная оценка суммарной трудоемкости проекта = 392.5

E = E + 2 * СКО – суммарная трудоемкость проекта с вероятностью 95% процентов = 426.04

**Сетевая диаграмма взаимосвязи работ и метод критического пути:**




Минимальная продолжительность разработки по методу критического пути - 267.5 часов.

1. Согласование проекта с заказчиком (40 часов):
    - Встречи с заказчиком для уточнения требований.
    - Специалисты: 1 менеджер по проекту.
    - Зависимости: Отсутствуют, начальная задача.
2. Дизайн (60 часов):
    - Разработка графического дизайна макетов.
    - Специалисты: 2 графических дизайнера.
    - Зависимости: Задача начинается после завершения согласования.
3. Верстка и программирование (60 часов):
- Адаптация дизайна в код, базовая разработка.
- Специалисты: 2 разработчика.
- Зависимости: Задача начинается после утверждения дизайна.

4. Разработка функциональности (45 часов):
- Реализация основных функций сайта/приложения.
- Специалисты: 2 разработчика.
- Зависимости: Начинается после завершения верстки.

5. Адаптивность и респонсивный дизайн (30 часов):
- Оптимизация дизайна для мобильных устройств.
- Специалисты: 1 дизайнер, 1 разработчик.
- Зависимости: Параллельно с разработкой функциональности.

6. Контент (60 часов):
- Подготовка и загрузка текстов, изображений, видео.
- Специалисты: 1 контент-менеджер.
- Зависимости: Независимо от других задач.

7. Тестирование и оптимизация (35 часов):
- Проверка и исправление ошибок.
- Специалисты: 2 тестировщика.
- Зависимости: После завершения разработки функциональности.

8. Внедрение и запуск (27.5 часов):
- Размещение на сервере, подготовка к запуску.
- Специалисты: 1 разработчик.
- Зависимости: После завершения тестирования.

**4. Метод функциональных точек**

Метод функциональных точек (Function Points, FP) — это структурированный способ оценки размера проекта. Он основывается на анализе количества функций, которые система должна выполнять. В рамках этого метода функции делятся на следующие категории:

1. Ввод данных (External Inputs, EI) — количество экранов или интерфейсов для ввода данных.
2. Вывод данных (External Outputs, EO) — количество отчетов, экранов, которые предоставляют информацию пользователю.
3. Запросы к данным (External Inquiries, EQ) — запросы пользователя на получение данных, которые не изменяют состояние системы.
4. Логические группы данных (Internal Logical Files, ILF) — внутренние логические базы данных системы.
5. Внешние интерфейсы (External Interface Files, EIF) — взаимодействие с внешними системами и базами данных.

Оценка функциональных точек

Для проекта Godbolt.org, функциональные точки распределяются следующим образом:

| Категория | Количество | Сложность | Вес |
|---|---|---|---|
| Ввод данных | 5 | Средняя | 4 |
| Вывод данных | 6 | Средняя | 4 |
| Логические файлы | 5 | Средняя | 4 |
| Запросы к данным | 7 | Высокая | 5 |
| Внешние интерфейсы | 5 | Высокая | 5 |
| Категория | Количество | Сложность | Вес |

Общее количество функциональных точек:

FP=20+24+20+35+25=124

Перевод FP в строки кода (KLOC)

Для перевода функциональных точек в количество строк кода используется коэффициент, зависящий от языка программирования. Предположим, что для реализации системы используется Java, для которой в среднем приходится 46 строк кода на одну функциональную точку. Тогда количество строк кода (KLOC) будет:

KLOC=124×46=5704 строк кода

Оценка трудоемкости по модели COCOMO II

COCOMO II (Constructive Cost Model) — это модель для оценки трудоемкости разработки программного обеспечения на основе количества строк кода. Формула модели:

$$E = A \times (KLOC)^{B}$$

Где:

A — константа, зависящая от сложности проекта. Для средних проектов A=2.94

B — показатель сложности. Для средних проектов B=1.12

Подставляем значения:

$$E = 2.94 \times (5.704)^{1.12} \approx 18.4 \text{ человеко-месяцев}$$

| **Язык программирования** | **Оценка количества строк** | | |
|---|---|---|---|
| | **Наиболее вероятная** | **Оптимистичная** | **Пессимистичная** |
| Assembler | 172 | 86 | 320 |
| C | 148 | 9 | 704 |
| C++ | 60 | 29 | 178 |
| C# | 59 | 51 | 66 |
| J2EE | 61 | 50 | 100 |
| JavaScript | 56 | 44 | 65 |
| Java | 46 | 14 | 134 |

**5. Оценка проекта методом вариантов использования (Use Case Points)**

UCP = UUCP * TCF * ECF * PF

***<u>Оценка текущего проекта:</u>***

**UUCP (Unadjusted UCP) = UUCW (невыровненный вес прецедента) + UAW (невыровненный вес эктора)**

1) Неадаптированный вес участника

$$\text{UAW} = \sum_{i=1}^{3} = AW\,i \times N\,i = \mathbf{6}$$

| *Сложность* | *Вес (AWi)* | *Количество участников Ni* | * |
|---|---|---|---|
| Простая | 1 | 0 | 0 |
| Средняя | 2 | 0 | 0 |
| Высокая | 3 | 2 | 6 |

2) Нескорректированный вес варианта использования

$$\text{UUCW} = \sum_{i=1}^{3} = UCW\,i \times N\,i = \mathbf{145}$$

| *Сложность* | *Вес (UCWi)* | *Количество транзакций Ni* | * |
|---|---|---|---|
| Простая | *5* | *8* | *40* |
| Средняя | *10* | *6* | *60* |
| Высокая | *15* | *3* | *45* |

**TCF (Technical Complexity Factor) - факторы технической сложности**

| *TF* | *Описание* | *Вес (Wi)* | *влн. Fi* | * |
|---|---|---|---|---|
| E1 | Уверенное использование UML | 0.5 | 2 | 1 |
| E2 | Количество работников на неполный рабочий день | -1 | 0 | 0 |

| | | | | |
|---|---|---|---|---|
| E3 | Опыт работы с приложениями | 0.5 | 3 | 1.5 |
| E4 | Опыт внедряющей разработки | 1.5 | 1 | 1.5 |
| E5 | Мотивация | 2 | 4 | 6 |
| E6 | Сложный язык разработки | -1 | 0 | 0 |
| E7 | Повторное использование кода | 1 | 1 | 1 |
| E8 | Переносимость на другие платформы | 4 | 4 | 16 |
| E9 | Безопасность | 5 | 1 | 5 |
| E10 | Время отклика | 2 | 1 | 2 |
| | | | | Сумма = 33 |

$$TCF = C_1 + C_2 \sum_{i=1}^{10} W i \times F i = \mathbf{0.6 + 0.1*33 = 0.93}$$

TCF от 0.6 до 1.3
$C_1 = 0.6$; $C_2 = 0.01$;
Wi - вес фактора
Fi - субъективная сложность, выбирается группой разработчиков на основании своего опыта и восприятия сложности проекта от 1 до 5

**ECF (Environment Complexity Factor) - факторы сложности окружения (определение веса факторов окружения)**

| *TF* | *Описание* | *Вес (Wi)* | *влн. Fi* | * |
|---|---|---|---|---|
| E1 | Объектно-ориентированный опыт команды | 1 | 2 | 2 |
| E2 | Количество работников на неполный рабочий день | -1 | 1 | -1 |
| E3 | Сложный язык программирования | -1.5 | 2 | -3 |
| E4 | Опыт внедряющей разработки | 1.5 | 2 | 3 |

| | | | | |
|---|---|---|---|---|
| E5 | Мотивация команды | 2 | 4 | 8 |
| E6 | Сложность используемого процесса разработки | 1.5 | 2 | 3 |
| E7 | Стабильность требований | 1 | 2 | 2 |
| | | | | Сумма = 14 |

ECF = $C_1 + C_2 \sum_{i=1}^{7} W i \times F i$ = **1.4 - 0.03*14 = 0.98**

ECF от 0.425 до 1.4
$C_1$ = 1.4; $C_2$ = -0.03;
Wi - вес фактора
Fi - субъективная сложность, выбирается группой разработчиков на основании своего опыта и восприятия сложности проекта от 1 до 5

UCP = UUCP * TCF * ECF = (6 + 145) *** 0.93 * 0,98 = 137,6**

**Итого, точки варианта использования UCP без учета PF = 137,6**

***<u>Оценка стороннего (завершенного) проекта:</u>***

**Список прецедентов проекта:**

| № | Описание |
|---|---|
| 1 | Аутентификация пользователя |
| 2 | Создание и редактирование данных профиля |
| 3 | Выполнение кода |
| 4 | Просмотр результатов выполнения |
| 5 | Сравнение фрагментов кода |
| 6 | Поддержка различных языков программирования |
| 7 | Удаление сохраненных фрагментов кода |

**UUCP (Unadjusted UCP) = UUCW (невыровненный вес прецедента) + UAW (невыровненный вес эктора)**

1) Неадаптированный вес участника

$$\text{UAW} = \sum_{i=1}^{3} = AW\,i \times N\,i = \mathbf{6}$$

| *Сложность* | *Вес (AWi)* | *Количество участников Ni* | * |
|---|---|---|---|
| Простая | 1 | 0 | 0 |
| Средняя | 2 | 1 | 2 |
| Высокая | 3 | 2 | 6 |

2) Нескорректированный вес варианта использования

$$\text{UUCW} = \sum_{i=1}^{3} = UCW\,i \times N\,i = \mathbf{75}$$

| *Сложность* | *Вес (UCWi)* | *Количество транзакций Ni* | * |
|---|---|---|---|
| Простая | *5* | *2* | *10* |
| Средняя | *10* | *2* | *20* |
| Высокая | *15* | *3* | *45* |

**TCF (Technical Complexity Factor) - факторы технической сложности**

| *TF* | *Описание* | *Вес (Wi)* | *влн. Fi* | * |
|---|---|---|---|---|
| E1 | Уверенное использование UML | 0.5 | 3 | 1.5 |
| E2 | Количество работников на неполный рабочий день | -1 | 2 | -2 |
| E3 | Опыт работы с приложениями | 0.5 | 4 | 2 |
| E4 | Опыт внедряющей разработки | 1.5 | 2 | 3 |
| E5 | Мотивация | 2 | 4 | 8 |

| | | | | |
|---|---|---|---|---|
| E6 | Сложный язык разработки | -1 | 2 | -2 |
| E7 | Повторное использование кода | 1 | 2 | 2 |
| E8 | Переносимость на другие платформы | 4 | 1 | 4 |
| E9 | Безопасность | 5 | 2 | 10 |
| E10 | Время отклика | 2 | 1 | 2 |
| | | | | Сумма = 28.5 |

$$TCF = C_1 + C_2 \sum_{i=1}^{10} W_i \times F_i = \mathbf{0.6 + 28.5 * 0.01 = 0.89}$$

TCF от 0.6 до 1.3
$C_1 = 0.6$; $C_2 = 0.01$;
Wi - вес фактора
Fi - субъективная сложность, выбирается группой разработчиков на основании своего опыта и восприятия сложности проекта от 1 до 5

**ECF (Environment Complexity Factor) - факторы сложности окружения (определение веса факторов окружения)**

| *TF* | *Описание* | *Вес (Wi)* | *влн. Fi* | * |
|---|---|---|---|---|
| E1 | Объектно-ориентированный опыт команды | 1 | 4 | 4 |
| E2 | Количество работников на неполный рабочий день | -1 | 2 | -2 |
| E3 | Сложный язык программирования | -1.5 | 2 | -3 |
| E4 | Опыт внедряющей разработки | 1.5 | 2 | 3 |
| E5 | Мотивация команды | 2 | 4 | 8 |
| E6 | Сложность используемого процесса разработки | 1.5 | 2 | 3 |
| E7 | Стабильность требований | 1 | 1 | 1 |

| | |
|---|---|
| | Сумма = 14 |

ECF = $C_1 + C_2 \sum_{i=1}^{7} W i \times F i$ = **1.4 - 0.03*14 = 0.98**

ECF от 0.425 до 1.4
$C_1$ = 1.4; $C_2$ = -0.03;
Wi - вес фактора
Fi - субъективная сложность, выбирается группой разработчиков на основании своего опыта и восприятия сложности проекта от 1 до 5

UCP = UUCP * TCF * ECF = (6 + 75) *** 0.89 * 0,98 = 70,6**

**Итого, точки варианта использования UCP без учета PF = 70,6**

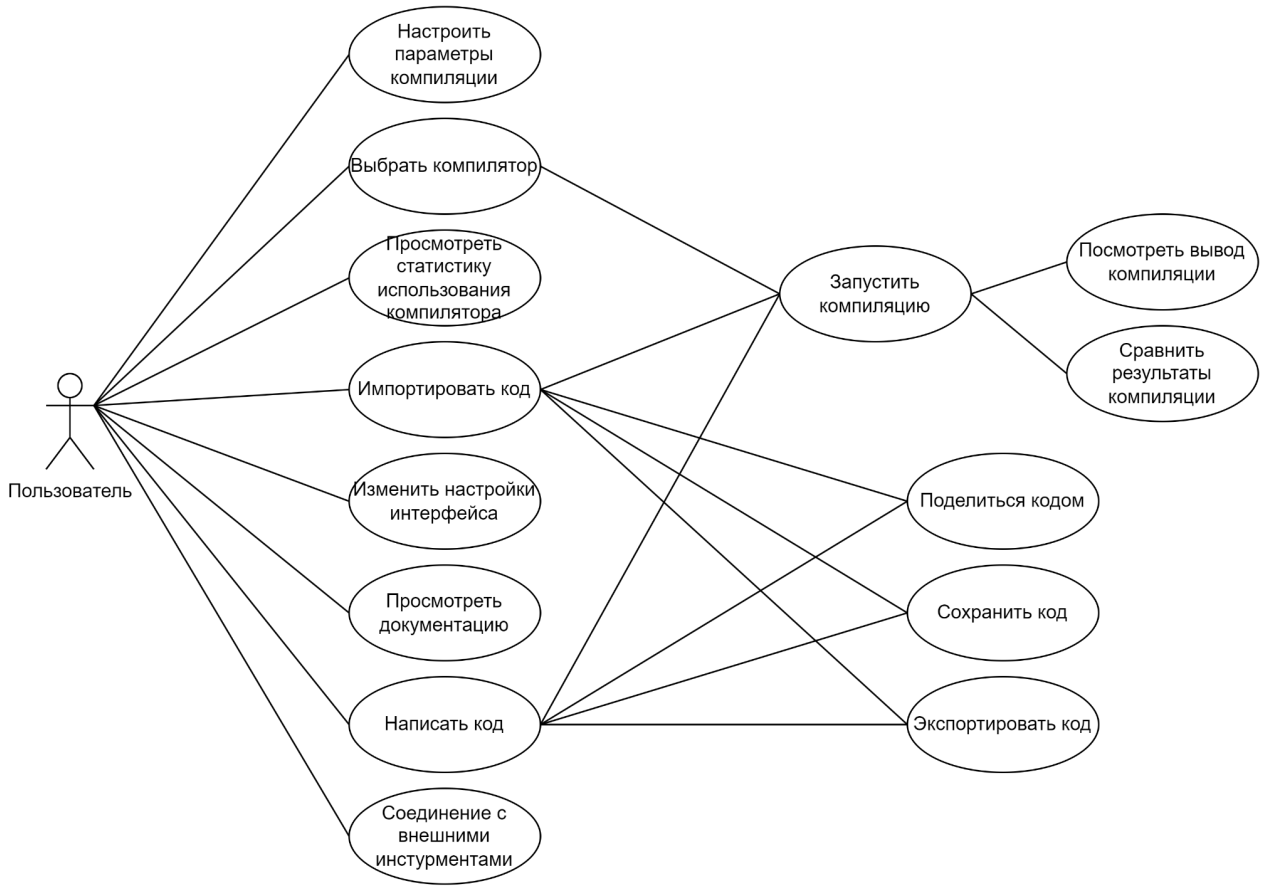
**Расчет итогового UCP**

Время разработки проекта в часах = 160

Фактор продуктивности PF = (Время разработки / UCP') = 160 / 70,6 = 2,27

**СКОРЕЛЛИРОВАННОЕ UCP данного проекта = UCP * PF = 137,6 * 2,27 = 312**

**Диаграмма Use Case Points для user:**




## 6. Сравнение использованных методов

Итоговая трудоемкость различными методами:

- Наивный метод: 392,5 чел.ч.
- PERT: 426 чел.ч.
- COCOMO II: 18,4 чел.м.
- UCP: 312 чел.ч.

По итогам наших расчетов менее формальные методы оценки выдали большее количество человеко-часов, это может быть обусловлено отсутствием надлежащего опыта у участников команды.
Также стоит отметить, что при оценке наивным методом мы выделили пункты важные для разработки всего проекта, что увеличило конечный результат.

Метод PERT это более формальный метод, но строится он на наивной оценке. Результат этого метода можно считать более правдоподобным, но только при условии, что у команды есть большой опыт и хорошее понимание проекта.

Наименьшую оценку показал метод COCOMO II, и на это могло повлиять несколько факторов. Во-первых, этот метод основывается на оценке функциональных точек и не берет во внимание другие важные аспекты, не связанные с написанием кода и его сложностью. Во-вторых, наш проект довольно простой, а этот метод лучше подойдет для оценки более крупных проектов.

Метод UCP более энергозатратный, но его результат показался нам наиболее правдоподобным, так как при оценке учитывает наш опыт написания похожего проекта, что помогает правильнее оценить затраты времени не те или иные задачи.

**Вывод**

В ходе выполнения данной работы мы изучили такие методы оценки трудоемкости проектов как PERT, COCOMO II, UCP.