Санкт-Петербургский государственный университет

Системного программирования

Группа 24.М71-мм

Разработка фреймворка анализа повторов в документации в рамках проекта DocLine

Макаров Павел Михайлович

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
доцент кафедры системного программирования, к.ф.-м.н. Луцив Д.В.

Санкт-Петербург
2026

Оглавление

Введение

Разработка документации является важным этапом создания современного программного обеспечения. С развитием ПО технические документации становятся всё более запутанными и большими в объёме текста, из-за чего кратно увеличивается вероятность ошибок и неточностей. В будущем, например при найме новых разработчиков, процесс понимация работы и функционала программы может быть затруднён из-за некачественного ведения документации.

Одним из решений данной проблемы является использование специального программного обеспечения для автоматизации процесса создания документации проекта. Но в таком случае возникают повторы текста, что может и облегчить понимание функционирования программного обеспечения, так и усложнить повторяющимся одинаковым текстом в больших по объему документациях.

Примером проекта разработки техник повторного использования документации является проект DocLine, созданный на кафедре Системного программирования СПбГУ. Целью проекта являются создание метода повторного использования документации и подхода к поиску и анализу повторов в документации. В рамках DocLine был создан программный инструмент Duplicate Finder, призванный помочь при анализе повторов.

Программный инструмент Duplicate Finder предназначается для поиска и анализа нечётких повторов в документациях программного обеспечения. Его разработка ведётся минимум с 2007 года [12], использовались несколько языков программирования для реализации различного функционала. На основе последнего утверждения, что инструмент написан на нескольких языках программирования, возникают проблемы долгой разработки ПО без заинтересованности фирм, которые способны профинансировать разработку, а именно сложности использования. Для запуска приложения необходимо установить несколько сторонних программ (JRE для Heuristic Finder, .NET Framework для Fuzzy Finder) и выполнить другие условия (Компьютер архитектуры x86_64 для за-

пуска Clone Miner, имя компьютера в сети только из ASCII и т.п.) — данные проблемы усложняют использование и дальнейшую поддержку программного инструмента.

Из-за просчётов разработки и недостатков инструмента стало ясно, что необходимо изменить тип продукта и сосредоточиться на чётко ограниченных инструментах разработки ПО. Появилась необходимость в создании нового программного продукта для поиска нечётких повторов.

1. Постановка задачи

Целью данной учебной практики является разработка инструмента для анализа нечётких повторов. Для достижения этой цели в рамках работы были сформулированы следующие задачи:

1. анализ предметной области;
2. изучение алгоритма работы Duplicate Finder;
3. разработка архитектуры инструмента;
4. реализация обновлённого программного инструмента.

2. Обзор

Технология распознавания повторов в тексте активно начала развиваться с начала 90-х годов [14]. Результатом этого стали различные программные инструменты, упрощающие, автоматизирующие процесс поиска повторов.

2.0.1. Алгоритм шинглов

Одним из примеров алгоритмов поиска нечётких повторов является **алгоритм шинглов** (w-shingling) [7]. Алгоритм сравнивает случайную выборку контрольных сумм шинглов (подпоследовательностей) двух текстов между собой, что позволяет определить их сходство или различие. [6]

Основное использование алгоритма шинглов - системы антиплагиата для выявления оригинальности текстов. Алгоритм позволяет сравнивать проверяемую работу с другими текстами из базы данных различных исследований, журналов, статей и т.д для выявления совпадающих фрагментов.

Вычисление шинглов происходит следующим образом: gусть документ $D = (t_1, t_2, \ldots, t_n)$ — последовательность токенов (символов или слов). Множество **k-шинглов** документа определяется как:

$$S(D, k) = \{(t_i, t_{i+1}, \ldots, t_{i+k-1}) \mid i = 1, 2, \ldots, n - k + 1\}.$$

Для двух документов A и B их сходство вычисляется через коэффициент Жаккара для множеств шинглов:

$$J(A, B) = \frac{|S(A, k) \cap S(B, k)|}{|S(A, k) \cup S(B, k)|}.$$

2.0.2. Сходство Жаккара

Сходство Жаккара — метрика сходства двух последовательностей данных. Математически сходство Жаккара определяется как отношение мощности пересечения двух множеств к мощности их объеди-

нения:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

, где A и B — два множества, $|A \cap B|$ — мощность пересечения множеств, а $|A \cup B|$ — мощность их объединения [9].

Основное использование сходства Жаккара - алгоритмы кластеризации и задачи кваллификации данных, рекомендательные системы и поиск дубликатов в наборе данных. Также часто используется в экологии (сравнение видового состава), геномике (сравнение генетических последовательностей) и обработке естественного языка (выявление релевантных предложений).

2.0.3. Сходство Левенштейна

Сходство Левенштейна — это метрика, измеряющая количество операций, необходимых для преобразования одной строки в другую. Операции включают вставку, удаление и замену символов. Формула для вычисления расстояния Левенштейна между двумя строками a и b длиной $|a|$ и $|b|$ соответственно определяется рекурсивно следующим образом:

$$d(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0 \\ \min \begin{cases} d(i-1,j)+1 \\ d(i,j-1)+1 \\ d(i-1,j-1)+cost \end{cases} & \text{otherwise} \end{cases}$$

где $cost$ равен 0, если символы $a[i-1]$ и $b[j-1]$ совпадают, и 1 в противном случае [15].

2.1. Обзор существующих технических решений

2.1.1. dupeGuru

dupeGuru - это инструмент для поиска дублирующих файлов на компьютере, с возможностью сканирования как названий файлов, так и

их содержимого с помощью алгоритма нечёткого сопоставления (fuzzy matching) [11], то есть программа будет находить повторы, даже если они не совсем схожи. Приложение работает на Windows, Linux, Mac OS X.

Несмотря на то, что основное предназначение программы - поиск дубликатов, ищет повторы программа в различных типах файлов, а не только в тексте. Это хоть и универсально, но размывает основной функционал в целях унификации с разными видамипрограммных файлов. Для работы используются другие алгоритмы, которые могут пригодиться в работе, но отводят нас на более глубокий уровень данных, а не на текстовый поиск повторов

2.1.2. Антиплагиат

Антиплагиат - это российская система проверки текста на заимствования [13]. Система нацелена на проверку плагиата в научных текстах (дипломы, статьи и т.д.).

Несмотря на то, что проект нацелен на поиск повторов, основное отличие в том, что поиск дубликатов проходит в других, уже существующих текстах, а не в самом анализируемом тексте. Кроме того, у системы есть зарегистрированный "Модуль поиска заимствованных изображений", что является дополнением к уже существующему текстовому поиску дубликатов.

2.1.3. Trados Studio

Trados Studio - это программа, среда перевода на базе искуственного интелекта, разработанная для переводчиков [10]. ПО в режиме анализа проекта может экспортировать список часто повторяющихся фраз.

Данная программа является профессиональным инструментов для переводчиков. Причём среди функционала есть функция поиска повторов, причём в отличие от двух предыдущих решений (Антиплагиат, Text.ru) ищущая плагиат именно в анализируемом тексте. Тем не ме-

нее, данный инструмент имеет лишь крайне ограниченный функционал поиска повтора, без возможности изменения настроек и алгоритма поиска повторов.

2.1.4. Lingo4G

Lingo4G - это движок для исследования и визуализации огромного количества документов и текста. Система позволяет объединить компоненты обработки Lingo4G в аналитические запросы для выполнения различных задач интеллектуального анализа текста, начиная от простого поиска документов на основе запросов, кластеризации и 2d-отображения и заканчивая выявлением дублирующихся или похожих по содержанию документов [5].

Система имеет огромные возможности для визуализации текстовых файлов и поиска повторов. Тем не менее, возможности анализа повторов ограничены лишь поиском схожил или одинаковых документов. Алгоритм не выявляет схожести одного анализируемого текста, а ищет совпадения между текстами в разных текстовых файлах.

2.1.5. AllDup

AllDup - это бесплатная программа для поиска и удаления дубликатов файлов на вашем компьютере, сетевых ресурсах или внешних носителях. Быстрый алгоритм поиска находит дубликаты файлов любого типа, например, текстовых документов, изображений, музыки или фильмов. Мощная поисковая система позволяет находить дубликаты по комбинации следующих критериев: Имя файла, Расширение файла, Размер файла, Содержание файла, Даты файла и Атрибуты файла. Кроме того, есть возможность поиска схожих имен файлов, почти одинаковых фотографий, почти одинаковых музыкальных файлов, видео- и аудиофайлов с одинаковой или почти одинаковой длиной звука, а также поиска на диске жёстких ссылок. [1].

Аналогично некоторым примерам выше, данный инструмент ищет дубликаты файлов, но не зацикливается на текстовых файлов.

2.1.6. Duplicate Finder

Duplicate Finder - это приложение с открытым исходным кодом для поиска похожих фрагментов текста в одном или нескольких файлах. Его можно использовать для обнаружения одинаковых фрагментов текста документации, а также контента, который схож, но не идентичен. Инструмент совместим с разными форматами, включая plaintext, Markdown и XML.

Из плюсов стоит отметить наличие нескольких алгоритмов поиска повторов: Fuzzy Finder, Heuristic Finder и Clone Miner — каждый из них имеет свои особенности и подходит для разных сценариев поиска повторов. Кроме того, программа имеет графический интерфейс, делая доступным её использование для пользователей без опыта работы с командной строкой. Также стоит отметить, что Duplicate Finder является проектом с открытым исходным кодом, с соответствующими плюсами и минусами разработки.

Из основных недостатков — сложные (для такого маленького проекта) внешние зависимости (JRE для Heuristic Finder, .NET для Fuzzy Finder, Python для NGram Duplicate Finder) и вытекающая из этого строгая платформенная зависимость (разные режимы написаны на разных языках программирования: .NET для Fuzzy Finder, для UNIX-подобных систем есть аналоги; Wine для запуска Clone Miner на Unix-подобных системах; использование только ASCII-символы в сетевом имени из-за python-пакета bottle).

2.2. Выводы

Проанализировав существующие решения, становится очевидно, что абсолютное большинство решений сосредоточено на поиске плагиата для научных целей или для поиска одинаковых файлов на жёстких дисках компьютеров. Кроме того, в этих решениях не хватает простоты использования, в них не хватает инструмента, который можно использовать в коде без лишних проблем.

3. Реализация

3.1. Обзор используемых технологий

После анализа уже готовых решений, с их преимуществами и недостатками, были определены основные технологии, которые будут использоваться для реализации фреймворка.

3.1.1. Go

Go (golang) - язык программирования, разработанный внутри компании Google. Язык отличается ориентацией на строгую типизацию, сборку мусора и явную поддержку параллельного программирования. Синтаксис является компактным и простым для анализа, что позволяет легко выполнять анализ с помощью автоматических инструментов, таких как интегрированные среды разработки [3].

Использование языка Go обусловлено его упором на выразительность, высокоэффективность как при компиляции, так и при выполнении программ с возможностью легко и просто писать надёжные высокоинтеллектуальные программы

3.1.2. Pandoc

Pandoc - универсальная утилита для преобразования различных текстовых документов в другой текстовый формат [8].

Использование Pandoc обусловлено его возможностью конвертации различных текстовых документов в подходящий для анализа формат, не разрабатывая дополнительную утилиту, сосредоточившись на основных задачах.

3.1.3. go-webui

go-webui - это библиотека для создания графических интерфейсов на языке Go с использованием WebView [4], сохраняя при этом все пре-

имущества Go в части многопоточности и работы с системными ресурсами.

Использование go-webui обусловлено его возможностью создавать кроссплатформенные графические интерфейсы с минимальными усилиями, сохраняя при этом высокую производительность и надёжность приложений на Go. Графический интерфейс является дополнением, упрощая взаимодействие фреймворка с пользователем. Использование интерфейсов (графического и командной строки) необязательно для полноценного функционирования.

3.2. Архитектура предыдущего инструмента

В рамках проекта DocLine, для поиска нечётких повторяющихся фрагментов в документациях был разработан инструмент Documentation Refactoring Toolkit [2]

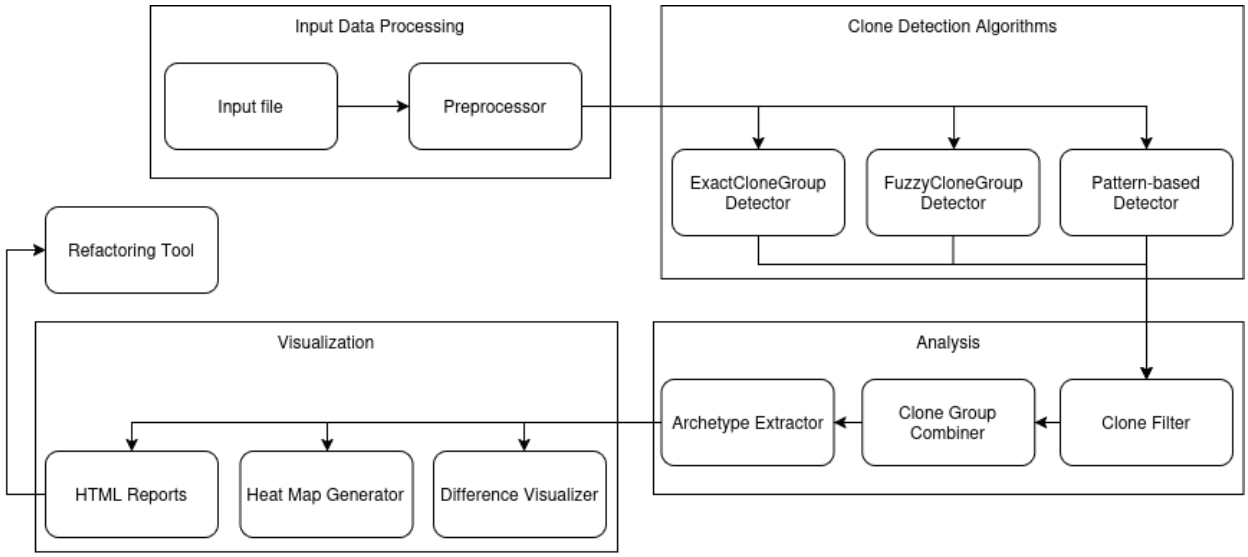

Рис. 1: Общая схема архитектуры Documentation Refactoring Toolkit

На Рис.1 представлена архитектура инструмента Documentation Refactoring Toolkit. Его алгоритм: после загрузки файла происходит его предобработка (Preprocessor), после файл анализируется одним из трёх алгоритмов: ExactCloneGroup (обнаружение идентичных текстовых фрагментов), FuzzyCloneGroup (обнаружение похожих, но не идентичных фрагментов), Pattern-Near-Duplicate Search (обнаружение по

образцу с настраиваемым порогом сходиства). Далее результаты фильтруются с помощью Clone Filter, объединяют связанные группы через Clone Group Combiner и с помощью Archetype Extractor извлекают общие паттерны из схожих клонов. Визуализация возможна тремя способами: созданием HTML отчёта, генерацией тепловой карты (heat map) и визуализацией различий с помощью графического интерфейса на основе PyQt, и других элементов для просмотра и настройки поиска клонов. При необходимости разбора HTML отчётов в проекте есть инструменты рефакторинга для создания информационных элементов и словарных записей.

3.3. Архитектура новой разработки

Разработка представляет собой фреймворк с алгоритмами в виде плагинов, дополненный приложениями с графическим веб-интерфейсом и интерфейсом командной строки (CLI).

На Рис.2 представлена архитектура новой разработки: разработанным фреймворком с системой плагинов и дополнительным слоем приложений для реализации интерфейсов.

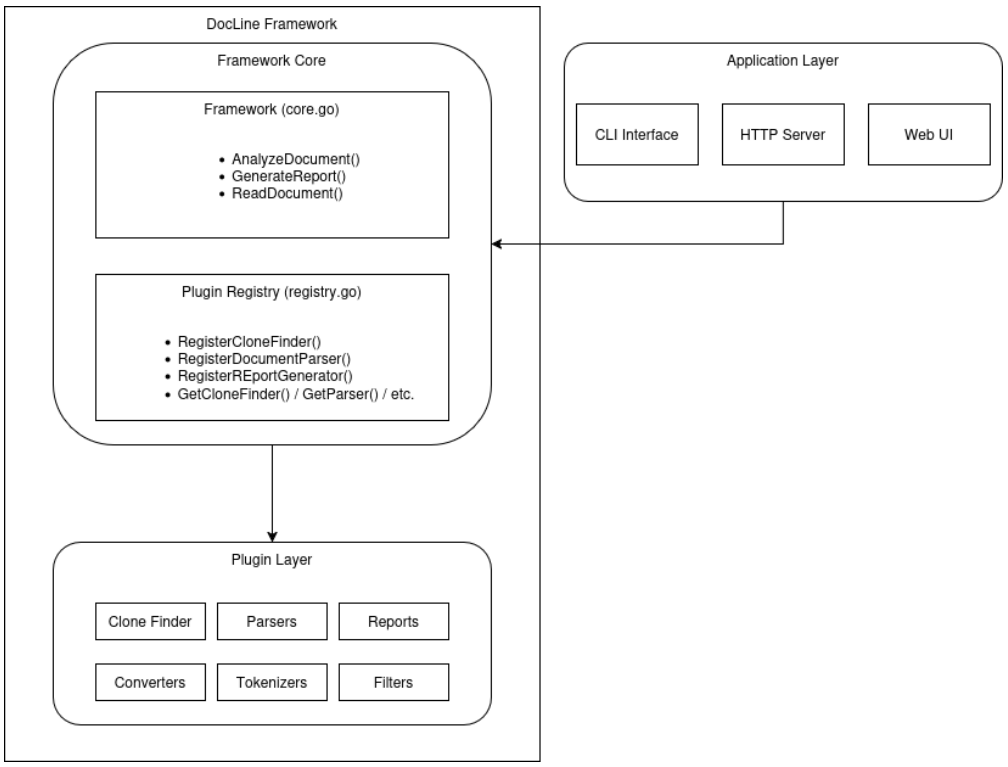

Рис. 2: Архитектура проекта

3.4. Особенности реализации

В процессе планирования и создания инструмента был сделан упор на производительность, параллелизм, обработку данных и управление файлами, что обеспечивает язык программирования Go.

Вся система реализована как монолитный модуль, при этом функциональные подсистемы в виде алгоритмов поиска клонов, парсинга и конвертации текста, создания отчёта и интерфейсы(графический и интерфейс командной строки) находятся на отдельных этапах для лучшей оптимизации процесса взаимодействия с инструментом и удобства работы с кодом.

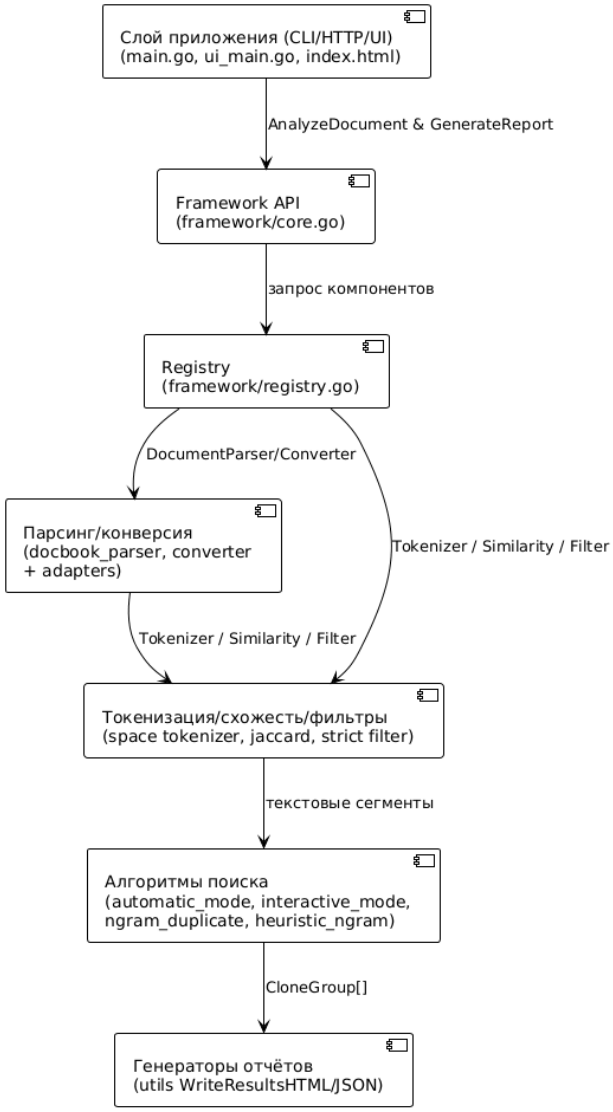

Рис. 3: Диаграмма компонентов проекта

3.4.1. Режимы

Все режимы анализа опираются на токенизацию текста, рассматривая текст как последовательность лексем, различными алгоритмами: скользящее окно для автоматического, двухпроходный для интерактивного,

В автоматическом режиме работы опциональная функция конвертации в DRL (Deterministic Representation Language) - текст приводится к нижнему регистру с удалением пунктуации через регулярные выражения и нормализация пробелов. Данная функция упрощает поиск дубликатов, отличающихся только регистром или пунктуацией.

В интерактивном режиме работы перед сбором позиций выполняется фильтрация по минимальной мощности группы: создаётся карта, куда попадают окна не ниже определенного порога (minGroupPower). Позиции собираются только после фильтрации для повышения производительности, без создания больших структур данных для окон. С включённой функцией расчёта архетипов (Archetype Calculation) выбирается фрагмент с наибольшим средним коэффициентом Жаккара к остальным фрагментам группы. Для каждого кандидата вычисляется средняя схожесть со всеми остальными фрагментами, где определяется фрагмент с максимальным значением схожести.

Коэффициент Жаккара (Jaccard index, Jaccard similarity) используется на уровне токенов для объединения похожих групп. Для создания группы токены преобразуются в множества, вычисляются пересечение и объединение. При превышении коэффициентом определённого порога, установленного для режима, группы объединяются, позволяя находить дубликаты с небольшими вариациями.

Для двух режимов работы (автоматический и интерактивный) реализован двухпроходный анализ: на первом проходе подсчитываются частоты окон фиксированного размера для определения кандидатов без дополнительного хранения из позиций, на втором проходе собираются позиции самых частовстречающихся окон. Это снижает потребление памяти: в автоматическом режиме работы сохраняются только те окна, которые встречаются минимум два раза, а в интерактивном режиме работы это соответствует минимальной мощности группы, что особенно важно для работы с большими документами.

N-gram режим использует попарное сравнение частей текста — текст разбивается на части (n-граммы), затем для каждой части строится карта n-грамм, где размер части задается параметром минимальной длины клона. На первом этапе для каждой части создается частотная карта, которая хранит количество вхождений каждой уникальной n-граммы, позволяя компактно представить текстовый фрагмент без хранения всех позиций. На втором этапе выполняется попарное сравнение всех частей текста с использованием сходства Жаккара, вычисляемого

на основе пересечения и объединения множеств n-грамм. Если коэффициент сходства превышает порог, заданный параметром максимального нечеткого хеш-расстояния (нормализованный к диапазону 0-1), части считаются дубликатами и группируются.

Эвристический режим применяет фильтрацию на основе правил для выявления потенциальных точек расширения (extension points) в документе. Алгоритм работает в два этапа: сначала генерируются все возможные n-граммы заданного размера из текста, что создает полный набор кандидатов для анализа. После этого происходит фильтрация результатов: отбрасываются n-граммы с цифрами и дубликаты. Результатом является отфильтрованный список потенциальных точек расширения, которые могут быть полезны для дальнейшего анализа документации (где текст, в зависимости от контекста, может быть семантически расширен, дополнен).

3.4.2. Обработка документов

Обработка документов состоит из нескольких компонентов, выполняющихся последовательно. Сначала файл валидируется на соответствие поддерживаемым форматам (если требуется конвертация, то документ преобразуется в DocBook XML с помощью Pandoc). Затем DocBook XML парсится для извлечения текстового содержимого, или, если это обычный текстовый файл, содержимое читается напрямую. Извлеченный текст разбивается на части (размер определяется пользователем) для более детального анализа. Выбранный алгоритм анализа (automatic, interactive, n-gram или heuristic) обрабатывает текст и находит повторяющиеся фрагменты, группируя их. Схема поддерживает разные форматы и конвертирует в DocBook XML для унификации парсинга.

Группирование данных (binning) при генерации тепловой карты (heatmap) и отчёта о плотности дубликатов (density reports) для ограничения размера окон проходит следующим образом: токены разбиваются на бины, для каждого бина вычисляется средняя плотность. Данный способ группировки позволяет визуализировать большие докумен-

ты без создания тысяч DOM-элементов.

Парсер DocBook использует рекурсивный обход XML-дерева с настраиваемым набором элементов для извлечения, при необходимости адаптируя парсер под разные структуры DocBook-документов. Для парсинга XML используется модуль encoding/xml с параметром decoder.Strict = false для обработки некорректно сформированных XML-документов, повышая устойчивость парсера к ошибкам в структуре входных данных.

Для обработки пограниченых случаев проверяется, что позиции не выходят за границы массива токенов при вычислении плотности: при биннинге проверяется деление на ноль; при расчёте схожести проверяется пустое объединение множеств — данные действия повышают надёжность при обработке нестандартных типов данных.

При парсинге DocBook HTML-значения обрабатываются через встроенную в XML-декодер карту для корректного преобразования возможные конфликтуемые значения в символы для успешного выполнения последующего анализа.

Встроенная в язык Go функция Builder из модуля strings используется для эффективности по памяти и производительности при больших объемах данных вместо конкатенации строк: Builder позволяет сохранять данные в буфер и формирует строку в один проход.

3.4.3. Интерфейсы пользователя

Интерфейс пользователя — это интерфейс, обеспечивающий взаимодействие между компьютерной системой и человеком. В данном проекте интерфейс необязателен, но для удобства использования он был реализован в виде графического интерфейса и интерфейса командной строки.

Графический интерфейс реализован с помощью стороннего открытого модуля go-webui [4], представляющий собой встраиваемое веб-приложение, позволяющее запускать специально веб-страницы как интерфейс программы.

Веб-приложение работает на HTTP-сервере, запускаемый на отдельной горутине. Сервер регистрирует маршруты через функцию RegisterRoutes, которая добавляет обработчики для различных режимов анализа: /upload для загрузки файлов, /heuristic для эвристического анализа, /ngram для n-граммного анализа, /automatic для автоматического режима и /interactive для интерактивного режима.

Интерфейс веб-приложения реализован через HTML (index.html) в виде формы с выбором методов анализа и соответствующими параметрами. Поведением интерфейса управляет код в файле main.js.При отправке формы JavaScript собирает все настройки в объект, создает объект FormData с файлом и настройками, и отправляет POST-запрос на соответствующий endpoint (один из режимов анализа).

Для создания интерфейса командной строки (CLI) используется встроенная в Go библиотека flag, позволяя запускать все алгоритмы анализа через командные флаги, обеспечивая управление фреймворком через терминал и позволяя при необходимости реализовать, например, автоматизированные скрипты. При использовании CLI-режима веб-интерфейс не запускается, анализ выполняется напрямую, результаты сохраняются в файлы. Для интерактивного режима дополнительно запускается сервер для отображения тепловой карты (heatmap).

3.4.4. Фреймворк

Фреймворк — это программная платформа, предоставляющая универсальный функционал для многократного использования, которую разработчики могут изменять при создании комплексных решений. Из-за удобства использования такого формата, фреймворк был выбран как основа новой разработки.

Ядро фреймворка (core.go) представляет собой высокоуровневый API для оркестрации процессов анализа документов: чтение, парсинг, токенизация, поиск дубликатов и генерация отчёта.

В виде плагинов реализованы алгоритмы поиска дубликатов. При изменении одного алгоритма, другие не будут затронуты. При необходимости, состав алгоритмов можно изменить.

Реестр (registry.go) написан для централизированного потокобезопасного управления различными плагинами. Код реализует регистрацию, извлечение, управление плагинами с особенностями в виде расширяемости, изоляции, гибкости и стандартизации.

В результате фреймворк состоит из трёх основных частей: ядра, управляющий оркестрацией процессов; плагинов, реализующие алгоритмы и прочую логику; реестр, связывающий остальные компоненты, в дальнейшем позволяя расширить функционал фреймворка путём разработки дополнительных плагинов с безопасным внедрением их в логику фреймворка.

Заключение

Результатом работы стали:

- проведён анализ предметной области;
- изучена предыдущая реализация Duplicate Finder;
- разработана архитектура и реализован обновлённый инструмент.

Исходный код можно найти в репозитории PavelMkr/docline-new.

Список литературы

[1] AllDup. — URL: https://alldup.ru/ (дата обращения: 19 октября 2025 г.).

[2] Documentation Refactoring Toolkit. — URL: https://github.com/spbu-se/pldoctoolkit/blob/master/doc-clone-miner/README_ru.md (дата обращения: 15 октября 2025 г.).

[3] The Go Programming Language Specification. — URL: https://go.dev/ref/spec (дата обращения: 15 октября 2025 г.).

[4] Go WebUI. — URL: https://github.com/webui-dev/go-webui (дата обращения: 15 октября 2025 г.).

[5] Lingo4G. — URL: https://carrotsearch.com/lingo4g/ (дата обращения: 19 октября 2025 г.).

[6] Manber Udi. FINDING SIMILAR FILES IN A LARGE FILE SYSTEM // Department of Computer Science, University of Arizona. — 1994. — URL: https://www.cs.princeton.edu/courses/archive/spr05/cos598E/bib/manber94finding.pdf (дата обращения: 1 ноября 2025 г.).

[7] Near-duplicates and shingling. — URL: https://nlp.stanford.edu/IR-book/html/htmledition/near-duplicates-and-shingling-1.html (дата обращения: 19 октября 2025 г.).

[8] Pandoc. — URL: https://pandoc.org/index.html (дата обращения: 15 октября 2025 г.).

[9] Т.Т. Tanimoto. Двоичные коды с исправлением выпадений, вставок и замещений символов // IBM Internal Report. — 1967.

[10] Trados Studio. — URL: https://www.trados.com/product/studio/ (дата обращения: 19 октября 2025 г.).

[11] dupeGuru. — URL: https://dupeguru.voltaicideas.net/ (дата обращения: 15 октября 2025 г.).

[12] pldoctoolkit repository. — URL: https://github.com/spbu-se/pldoctoolkit/tree/8e371c9d886e43da2fc50ed1493226c40d94b3b3 (дата обращения: 15 октября 2025 г.).

[13] Антиплагиат.ру. — URL: https://antiplagiat.ru/about/ (дата обращения: 19 октября 2025 г.).

[14] Гращенко Л. А. Романишин Г. В. Опыт автоматизированного анализа повторов в научных текстах // Новые информационные технологии в автоматизированных системах. — 2015. — Vol. 18. — P. 582–590.

[15] И. Левенштейн В. Двоичные коды с исправлением выпадений, вставок и замещений символов // Доклады Академий Наук СССР. — 1965. — P. 845–848.