ПРОГРАММИРОВАНИЕ, 2016, № 4, с. 39–49

ПЕРСПЕКТИВЫ СИСТЕМ ИНФОРМАТИКИ

УДК 004.45

ЗАДАЧА ПОИСКА НЕЧЕТКИХ ПОВТОРОВ ПРИ ОРГАНИЗАЦИИ ПОВТОРНОГО ИСПОЛЬЗОВАНИЯ ДОКУМЕНТАЦИИ

© 2016 г. **Д.В. Луцив[1], Д.В. Кознов[1], Х.А. Басит[2], А.Н. Терехов[1]**

[1] *Санкт-Петербургский государственный университет*
199034 Санкт-Петербург, Университетская наб., 7/9
[2] *Лахорский университет менеджмента*
Opposite Sector U, DHA., Lahore 54792, Пакистан
E-mail: d.lutsiv@spbu.ru, d.koznov@spbu.ru, a.terekhov@spbu.ru, hamidb@lums.edu.pk

Поступила в редакцию 12.02.2016

В связи с повышающейся сложностью документации программного обеспечения возрастают требования к качеству процесса сопровождения документации. Одним из способов решения этой задачи может быть повторное использование с последующей реструктуризацией документации. В работе предлагается метод поиска нечётких повторов в документации на основе поиска точных клонов. Результаты поиска используются при рефакторинге документации. В работе представлен инструмент Documentation Refactoring Toolkit, реализующий данный метод и интегрированный со средствами рефакторинга технологии DocLine. Подход апробирован на ряде пакетов документации для открытых проектов: ядро ОС Linux, PHP-фреймворк Zend, Subversion, DocBook.

1. ВВЕДЕНИЕ

Документация является важной составной частью современного программного обеспечения. Существует два вида документации: техническая документация (требования, проектные спецификации, тестовые планы и отчёты, API-документация [1] и т.д.) и пользовательская документация. Техническая документация помогает разработчикам ПО понимать и осмыслять созданные ими продукты, а также проводить их разработку и модификацию более успешно [2]. При этом техническая документация может иметь значительные размеры и сложную структуру. Подобно самому программному обеспечению документация также постоянно меняется в процессе разработки и сопровождении. Качество технической документации является известной проблемой, которая не решена за последние десятилетия [3, 4].

Одна из причин неудовлетворительного качества документации — это большое количество неконтролируемых повторов, которые создают трудности при сопровождении документации, поскольку новую информацию нужно вводить не в одно место документа, а сразу в несколько. Последнее часто не делают — из-за недостатка времени и технических средств — и в документации накапливаются ошибки. Ситуация является более сложной, поскольку очень часто речь идёт не о строгих текстуальных повторах — одни и те же свойства программного обеспечения могут быть описаны в документе с разных точек зрения, с разной степенью детальности. Также в документации описывается большое количество однотипных объектов — функций, прерываний, классов, сообщений — и соответствующие фрагменты текста имеют как сходства так и различия. Однако хорошим стилем является единообразие документации, то есть описание однотипных объектов имеет общую струк-

туру и совпадает в тех частях, где описывается одна и та же информация.

Существуют различные техники повторного использования программного кода [5, 6]. Одной из них является метод адаптивного повторного использования Бассета-Ерзабека [7, 8]. Эти идеи были применены в разработке технологии создания документации DocLine [9]. В рамках данной технологии был введён процесс рефакторинга XML-документации (изменение XML-представления документа и сохранение его внешнего вида) [10]. Однако возникла проблема автоматизированного поиска повторяющихся фрагментов. В работах [11, 12] было предложено использовать для поиска повторов подход к поиску клонов (software clone detection), основываясь на инструменте Clone Miner [13]. Однако были рассмотрены только чёткие повторы.

Данная работа продолжает исследование [9] и предоставляет возможности для поиска нечётких повторов. Последние конструируются из чётких повторов, если соответствующие группы клонов располагаются в тексте близко друг от друга. Созданный алгоритм реализован в виде Document Refactoring Toolkit [14] и апробирован на документации для ряда открытых программных проектов — Linux Kernel [15], Zend PHP Framework [16], Subversion [17], DocBook [18].

2. ОБЗОР

Обзор области разработки технической документации — проблем и методов их решения — можно найти в работе [19]. Ниже будут рассмотрены некоторые исследования, которые обеспечивают автоматический анализ и обработку документации.

В работе [20] предложен подход к автоматической генерации спецификации ресурсов ПО на основе API-документации (Application Programming Interface). Подход адресован следующей проблеме: разработчики плохо читают документацию и делают много ошибок при разработке ресурсов, в то же время описание ресурсов полностью изложено в документации и может быть использовано как вход для автоматической генерации.

В [21] предлагается подход к обнаружению ошибок в документации путём сравнения сущностей кода, упоминаемых в документации (типы данных, процедуры, переменные, классы и пр.), с примерами кода, также содержащимися в документации.

В [2] предлагается анализировать качество проектной документации при разработке и сопровождении, основываясь на мнении экспертов, выявленном с помощью опросов.

В [22, 23] предлагаются метрики для измерения качества документации. Авторы адаптировали инструмент VizzAnalyzer [24], предназначенный для поиска клонов, к поиску повторов в документации.

При разработке технической документации широко используются XML-языки. Самыми известными являются DocBook [25] и DITA [26], оба языка поддерживают модульный подход и позволяют создавать повторно используемые модули текста. Однако языки слабо поддерживают параметрическое повторное использования, а также не имеют соответствующих средств планирования. В технологии DocLine [12] техника адаптивного повторного использования Бассета-Ерзабека [7, 8] была применена к документации, в Docline были реализованы возможности параметризации и планирования повторного использования. Однако следует отметить, что все эти подходы подразумевают, что документация разрабатывается в виде повторно используемых модулей с самого начала, и не предлагают средств для поиска повторов.

Можно сделать следующие выводы. Поиск повторов в технической документации используется лишь в [27, 28] и [22, 23]. Однако в работах [22, 23] найденные повторы используются лишь для определения качеств документации и не предназначены для трансформации документации. В [28, 29] на базе повторов выполняется автоматизированный рефакторинг документации, согласно методу рефакторинга, изложенному в [10]. Однако во всех этих работах ищутся лишь чёткие повторы.

3. ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ И ПРОЕКТ DOCLINE

3.1. DocBook

DocBook [25] является набором стандартов и инструментов для создания технических текстов

в виде XML[1]. DocBook отделяет представление текста (полужирный шрифт, курсив, выравнивание и т.д.) от содержания документов, что позволяет реализовать идею single source, то есть один и тот же документ может иметь представления в HTML, PDF и т.д. DocBook легко расширяем, в частности, путём добавления дополнительных конструкций и препроцессирования.

3.2. *DocLine*

Технология DocLine [12] создана для разработки и поддержки сложной документации в программных проектах на основе адаптивного повторного использования [7, 8], позволяя конфигурировать фрагменты текста в зависимости от того, в каком контексте они используются. DocLine расширяет язык DocDook, предлагая язык DRL, содержит модель процесса разработки документации и программные инструменты на базе Eclipse. DRL предлагает два механизма адаптивного повторного использования: настраиваемые информационные элементы и многовидовые каталоги.

Информационные элементы. Рассмотрим пример. Пусть у нас имеется новостной агрегатор, который загружает ленты новостей из различных источников. Ниже представлен фрагмент его документации — возможность загрузки новостей в формате RSS и Atom:

```
When module instance receives refresh_news
call, it updates its data from RSS and
Atom feeds it is configured to listen
to and pushes new articles to the main
storage.
```
(1)

Помимо этого агрегатор может загружать новости из микроблогов Twitter:

```
When module instance receives refresh_news
call, it updates its data from Twitter
feeds it is subscribed to and pushes new
articles to the main storage.
```
(2)

Для того, чтобы повторно использовать повторяющийся текст из (1) и (2), мы можем создать следующий информационный элемент:

```
<infelement id="refresh_news≫ When
module instance receives refresh_news
call, it updates its data from <nest
id="SourceType≫</nest> and pushes new
articles to the main storage.</infelement>
```
(3)

С помощью тега `<nest/>` обозначена точка расширения — то есть возможность подстановки параметров. При использовании информационного элемента в определённом контексте точка расширения может быть удалена, замещена или дополнена соответствующим текстом. Ниже показано, как использовать (3) для формирования текста (2):

```
<infelemref infelemid="refresh_news≫
<replace-nest nestid="SourceType≫Twitter
feeds it is subscribed to </replace-nest>
</infelemref>
```
(4)

В (4) мы имеем ссылку на определённый в (3) информационный элемент (`<infelemref/>`) с замещением его точки расширения новым фрагментом текста (`<replace-nest/>`).

Многовидовые каталоги. В документации большинства программных продуктов встречаются описания сущностей, подразумевающих один и тот же функционал, но по-разному представленный. Частным случаем каталога является словарь, который содержит описания терминов. Словари полезны для создания глоссариев и для унификации используемых в документации названий. Подробнее многовидовые каталоги описаны в [10], [12].

3.3. *Рефакторинг документации*

Рефакторинг — это процесс изменения программного обеспечения, улучшающий его внутреннюю структуру, но не меняющий его функциональность [29]. В [10] понятие рефакторинга было введено для XML-документации как изменение внутреннего представления XML-документов с сохранением конечных представлений (например, PDF). Был разработан следующий набор операций рефакторинга.

1. Операции выделения общих сущностей, в частности, для перехода от неформатированного текста и DocBook к DRL.
2. Операции настройки ключевых сущностей.
3. Операции “мелкозернистого” повторного использования с применением словарей и многовидовых каталогов.
4. Операции переименования различных структурных элементов.

[1] О достоинствах и недостатках XML-средств при разработке документов см. [26].

3.4. *Выявление клонов в программном обеспечении и Clone Miner*

В программной инженерии активно развиваются методы поиска клонов (Software Clone Detection), существует большое количество готовых инструментов [30, 31]. В [11, 12] было предложено использовать данный метод и инструменты для поиска повторов в текстах. Был использован инструмент Clone Miner [13]. Он обладает интерфейсом командной строки и является лексическим детектором клонов, преобразуя программный код в список токенов (лексем). Алгоритм поиска повторов реализован на основе суффиксных массивов [32]. Clone Miner позволяет настраивать минимальную длину клонов (количество токенов). Например, фрагмент текста “FM registers” состоит из двух токенов. Для наших целей Clone Miner был доработан с целью поддержки Unicode, что позволяет экспериментировать с текстами на разных языках.

4. ПРОЦЕСС ПОИСКА ПОВТОРОВ И РЕФАКТОРИНГ ДОКУМЕНТАЦИИ

Общая схема процесса показана на рис. 1. Сначала выполняется подготовка файла с документацией, потом для него запускается Clone Miner, его результаты фильтруются, и пользователь может выполнить автоматический рефакторинг любой из найденных групп. В ходе рефакторинга создаётся повторно используемый информационный элемент или элемент словаря.

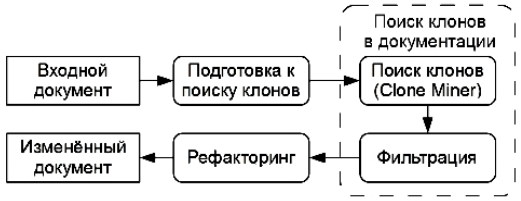

Рис. 1. Общая схема процесса.

4.1. *Подготовка к поиску клонов*

DocLine может работать только с конструкциями DRL, в частности, поиск клонов выполняется для информационного элемента. Если исходный файл является “плоским” текстом, то DocLine превращает его в один информационный элемент (операция рефакторинга “Переход к DRL”). После этого возможна дальнейшая обработка документа.

4.2. *Поиск клонов*

Clone Miner производит поиск по неформатированному тексту, т.к. найденные повторы могут нарушать синтаксис XML. Ниже приведён пример, в котором курсивом выделен найденный клон, включающий открывающий тег, но не закрывающий. В процессе рефакторинга нарушенная разметка клонов и окружающего их текста восстанавливаются.

```
<section id="file-tree-isa-directory≫
<title>Reviving incoming calls </title>
<para>
Once you receive an incoming call,
the phone gets CallerID information
and reads it out. But if...</para>
</section> (5)
```

4.3. *Фильтрация*

Окончательное решение о том, к каким из найденных клонов следует применить рефакторинге, принимает технический писатель. Однако список найденных Clone Miner оказывается очень большим (тысячи групп клонов), и его нужно фильтровать, отбрасывая мусор. Предлагаются следующие шаги фильтрации.

1. Отбрасываются группы клонов, содержащие менее 5 печатных символов (например, “is a” — три символа); как правило, эти клоны не имеют самостоятельного смысла, но оказываются многочисленными.

2. Отбрасываются группы, состоящие только из XML-разметки и не содержащие текста.

3. Исключаются группы, чьи клоны состоят из стандартных речевых оборотов — “that is” или “there is a”. В ходе анализа документов был составлен словарь речевых оборотов, и каждая группа клонов сравнивается на вхождение её членов в этот словарь. Если находится совпадение, то группа исключается.

4.4. *Рефакторинг*

После завершения предыдущих шагов мы получаем набор групп клонов. Наша конечная цель — использование клонов для получения повторно используемых элементов. Мы можем её достичь, следуя описанному ниже (см. рис. 2)

процессу. Часть действий процесса выполняется автоматически, но выбор кандидатов для рефакторинга и применение определённой операции рефакторинга осуществляется пользователем.

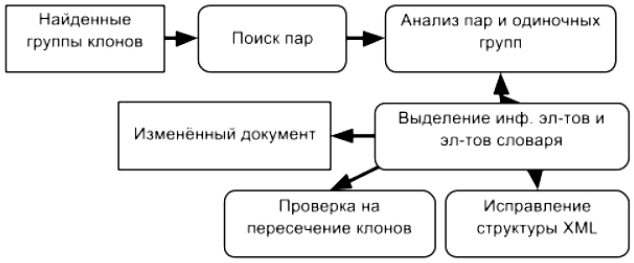

Рис. 2. Процесс рефакторинга.

Поиск пар клонов. Множество групп клонов, найденных Clone Miner, обозначим $SetG$. Для рефакторинга выбираются пары групп клонов из $SetG$, клоны которых в тексте расположены близко друг к другу. Например, следующая фраза в тексте встречается в пяти разных вариантах с разными номерами портов: “inet daemon can listen on ... port and then transfer the connection to appropriate handler”. Данный пример получается из двух групп по пяти клонов в каждой: первая содержит клоны вида “inet daemon can listen on”, вторая — клоны “port and then transfer the connection to appropriate handler”. Можно скомпоновать из этих двух групп один информационный элемент с единственной точкой расширения, которая может принимать значения различных номеров портов.

Предлагается алгоритм, который ищет соответствующие пары групп клонов и который может быть легко расширен и для работы с n группами.

Определим расстояние между клонами как количество символов в тексте между ними (перекрывающиеся клоны не рассматриваются). Тогда мы можем ввести расстояние между группами клонов G_1 и G_2 в случае, если выполняются следующие условия:

1. Количество клонов в группах одинаково: $\#G_1 = \#G_2$.

2. Мы упорядочиваем клоны каждой группы по вхождению в документ, и назначаем каждому клону порядковый номер. В результате мы получаем множество пар клонов, в каждой паре первый клон принадлежит первой группе, а второй клон — второй группе. В каждой паре клоны не перекрываются: $\forall k \in [1, \#G_1] g_1^k \bigcap g_2^k = \varnothing$, где g_1^k и g_2^k — клоны групп G_1 и G_2 соответственно.

3. Для каждой пары клонов из одной группы в тексте встречаются всегда перед клонами из другой:

$$\forall k \in [1, \#G_1] Before(g_1^k, g_2^k) \bigvee \forall k \in [1, \#G_1] Before(g_2^k, g_1^k),$$

где g_1^k и g_2^k — клоны групп G_1 и G_2 соответственно.

Расстояние между группами G_1 и G_2 определим как $dist(G_1, G_2) = \max_{1 \le k \le G_1} dist(g_1^k, g_2^k)$, где $dist$ является расстоянием в символах между клонами g_1^k и g_2^k. Данное определение позволяет, когда выбрана одна группа клонов, легко сравнить расстояния от неё до других групп с тем, чтобы выбрать ближайшую к ней. Кроме того, мы отсеиваем пары, в которых расстояния между соответствующими клонами из первой и второй групп сильно варьируются. Например, если расстояния между клонами первой пары 1 символ, а между клонами второй — 10000 символов, то шансов, что они как-то связаны по смыслу, практически нет. Экспериментируя, мы пришли к выводу, что для пар групп дисперсию расстояний между соответствующими их клонами следует ограничить сверху 2000: $Var(\{dist(g_1^k, g_2^k) \bigvee k \in [1, \#G_1], g_1^k \in G_1, g_2^k \in G_2\}) \le 2000$. Пары, дающие бóльшую дисперсию, мы не рассматриваем.

Алгоритм поиска пар рассматривает все группы из $SetG$, и для каждой находит ближайшую к ней. Если это удаётся сделать, в набор $PairG$ добавляется новая пара групп, и обе группы более не рассматриваются.

Анализ пар и одиночных групп. На данном шаге пары групп клонов и одиночные группы объединяются в список L:

$$L = PairG \bigcup \{G | G \in SetG \bigwedge \nexists P \in PairG : G = left(P) \bigvee G = right(P)\}.$$

Данный список предоставляется пользователю. Последний может выбрать те варианты, которые он желает выделить в повторно используемые информационные элементы или в элементы

словаря (далее будем называть элементы списка кандидатами на рефакторинг, или коротко — кандидатами). Важно, чтобы повторно используемые фрагменты текста были осмысленными, являясь частью описания функции, прерывания и т.д. Повторное использование, которое опирается лишь на синтаксис, а семантику игнорирует, неэффективно. Однако автоматически проанализировать осмысленность кандидатов не представляется возможным, поэтому мы предлагаем пользователю браузер для просмотра и вызова операций рефакторинга.

Представляя список L пользователю, мы упорядочиваем его по убыванию длины кандидата в символах. При этом считается, что длина клона — это количество его символов, а длина группы — сумма длин клонов этой группы: $\forall G \in \in Llength(G) = \#G \cdot length(g)$, $g \in G$ (клоны группы повторяют друг друга, поэтому их длины берутся равными). Длина пары групп вычисляется как сумма длин групп, образующих эту пару: $length(Pair(G_1, G_2)) = length(G_1) + +length(G_2)$. В начале списка L попадают кандидаты, которые содержат наибольшее количество текста документа и, таким образом, являются наиболее предпочтительными кандидатами для повторного использования.

Выделение информационных элементов и пополнение словаря. Для выбранного кандидата пользователь может выполнить следующие операции рефакторинга (см. раздел 3.3): выделение информационного элемента, выделение вариативного информационного элемента, выделение элемента словаря.

Перед тем, как выполнять данные операции, наш алгоритм проверяет, что клоны кандидата не перекрываются с теми клонами, которых уже были задействованы при рефакторинге. Clone Miner позволяет найденным группам клонов перекрываться, т.к. он не участвует в дальнейшей работе с клонами. В нашем же случае, такое перекрытие может привести к ошибкам при рефакторинге. Пересекающиеся с использованными клоны исключаются из кандидатов.

Далее, алгоритм корректирует синтаксические конструкции XML кандидата и окружающего их текста в документе. Как уже было сказано, Clone Miner выдаёт некорректные с точки зрения XML результаты, тогда как DocLine может работать только с корректными документами DRL и DocBook. Наш инструмент балансирует нарушенную XML-разметку (открывает и закрывает все недостающие теги). Отметим, что полноценная реализация такой балансировки достаточно сложна. Так, например, если мы лишний раз откроем и закроем тег `<para>` (задает параграф), то в результирующем документе DocBook получим два параграфа вместо одного. Обработка таких ситуаций — одно из дальнейших направлений работы.

После того, как операция рефакторинга для выбранного кандидата выполнена, координаты вхождений в документ всех оставшихся кандидатов пересчитываются в соответствии с внесёнными в текст изменениями. Пользователь же может вновь вернуться к анализу пар и групп (кандидатов).

5. ИНСТРУМЕНТ

Для реализации описанного выше процесса предложен программный инструмент Documentation Refactoring Toolkit [14], который является частью пакета DocLine. Инструмент реализован на Python и может быть использован отдельно от Eclipse и DocLine. Он предоставляет возможность синхронной навигации по найденным кандидатам и исходному тексту документации, а также позволяет выбрать кандидатов и выполнить для них рефакторинг.

На рис. 3 показан браузер кандидатов, реализованный предложенным инструментом. Строки таблицы “Refactoring candidates” соответствуют отдельным кандидатам. Контекстное меню позволяет пользователю выбрать операцию рефакторинга для выделенного кандидата — выделение информационного элемента или элемента словаря. Для одиночной группы чередующимися цветами подсвечены расположенные под текстом кандидата ссылки на её клоны (см. цифры в фигурных скобках после текста клона для первого кандидата на рис. 3), для пары клонов таким образом подсвечиваются различные варианты текста в точке расширения (см. информацию о втором кандидате на рис. 3). Секция “Source text” содержит исходный текст документа. При щелчке в таблице по варианту в этой секции подсвечивается фрагмент исходного текста из двух

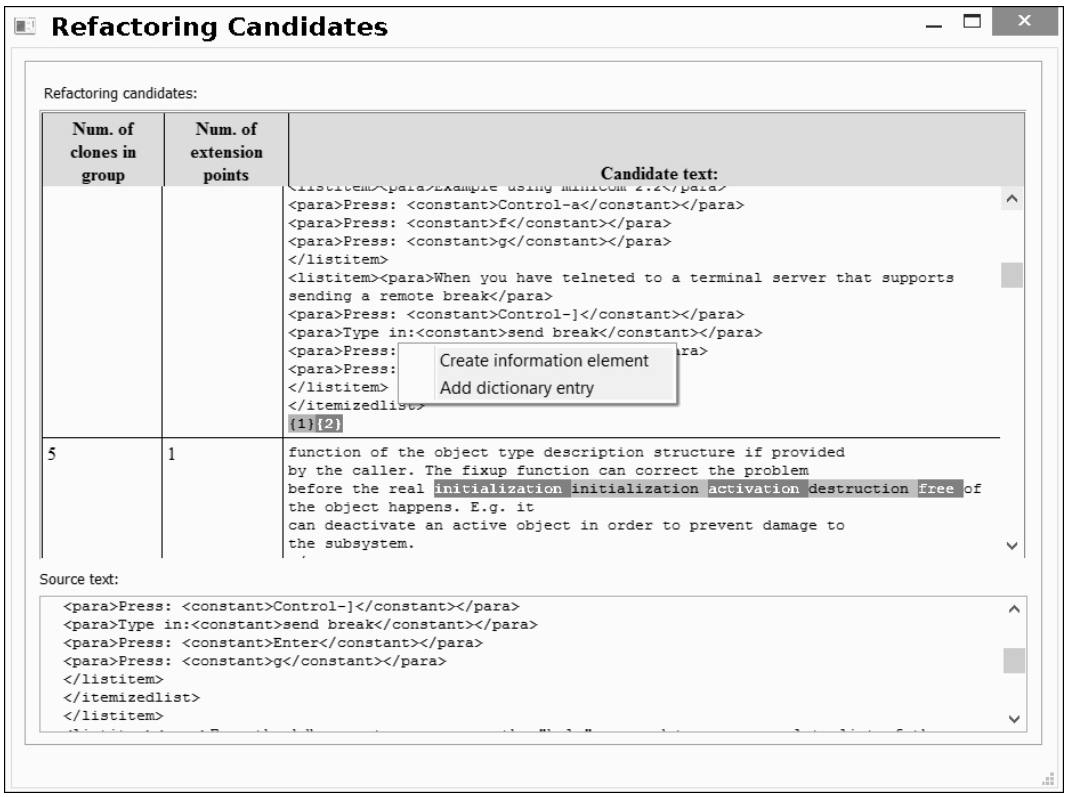

Рис. 3. Documentation Refactoring Toolkit.

клонов и выбранного варианта текста между ними. При щелчке в таблице по ссылке на клон в секции “Source text” подсвечивается соответствующий клон. Подсветка позволяет пользователю видеть не только выделенный фрагмент текста, но и его контекст.

6. ЭКСПЕРИМЕНТЫ

Для определения эффективности подхода был проведён ряд экспериментов — как на специально подготовленных тестах, так и на DockBook-документации проектов с открытым исходным кодом (список проектов/документов см. в таблице 1).

Мы использовали подход GQM (Goal-Question-Metric) [33] и сформулировали 3 вопроса, в соответствии с которыми было организовано три серии экспериментов:

- вопрос 1: качество поиска клонов в документах;
- вопрос 2: эффективность фильтрации клонов;
- вопрос 3: возможности рефакторинга.

Первая серия экспериментов касалась использования Clone Miner. Были создан небольшой набор документы с повторами (10 штук). Мы обнаружили, что Clone Miner допускал ошибки, которые сводились к игнорированию последнего повторяющегося токена в клонах. После внесения исправлений наш инструмент успешно обнаружил все повторы в данных тестах.

Эксперименты, посвящённые ответу на второй вопрос, основывались на документации из таблицы 1. В качестве метрики эффективности фильтрации мы использовали количество клонов, отфильтрованных способами, описанными в разделе 4.4. Результаты можно увидеть в таблице 2. Отметим, что в среднем фильтрация уменьшила количество клонов на 13,2%.

Количество кандидатов после фильтрации представлено в таблице 3 для двух случаев — с минимальной длиной клонов при поиске (параметр Clone Miner), равной 1 и 5 токенов. Для 5 токенов кандидатов получается значительно меньше. Однако короткие клоны, которые при этом отбрасываются, могут быть полезны при формировании пар, а также для формирования словаря и в некоторых других важных случаях.

Таблица 1. Документы, использованные в экспериментах

Проект	Документация	Сокращен.	Размер
Linux Kernel — открытое ядро ОС, основа ОС Linux	“Linux Kernel Documentation” — руководство системного программиста Linux [15]	LKD	892 KB
Zend Framework — открытый фреймворк для разработки сайтов и сервисов на PHP	“Zend PHP Framework documentation” — руководство программиста [16]	Zend	2924KB
Subversion — централизованная система контроля версий	“Version Control with Subversion For Subversion 1.7” — описание инструментов для пользователей и администраторов [17]	SVN	1810 KB
DocBook — язык и набор инструментов разработки документации	“DocBook 4 Definitive Guide” — официальная документация для тех. писателей и разработчиков [18]	DocBook	686KB

Таблица 2. Результаты фильтрации

Метод фильтрации	LKD,%	Zend,%	SVN,%	DocBook,%	В среднем,%
Rejecting clones under 5 symbols in length	7,3	4,8	4,4	7,2	5,9
Rejecting pure XML markup clone groups	3,3	5,8	2,4	6,0	4,4
Rejecting common language phrases	3,2	2,2	2,9	3,4	2,9
Total	13,8	12,8	9,7	16,6	13,2

В свете этого мы рекомендуем техническому писателю работать с клонами длиной от 1 токена. Упорядочивание списка кандидатов поместит большинство коротких кандидатов в конец.

При проведении экспериментов, посвящённых ответу на вопрос 3, была введена метрика “степень переиспользования” [34]. Она вычислялась как отношение объёма всего переиспользуемого текста к общему объёму документации:

$$\frac{c \in \sum_{allcandidates} length(C)}{length(T)},$$

где $length(C)$ — это количество символов в клоне или в паре клонов, умноженное на количество клонов в соответствующей группе(ах) (см. раздел 4.5), а $length(T)$ — длина всего текста в символах.

Размеры документации измерялись в символах. В среднем для всех тестовых документов (таблица 1) степень переиспользования для клонов длиной от 1 токена находится в пределах от 48% до 53%, а для клонов от 5 токенов — от 21% до 28%. Данные результаты показывают, что при автоматическом рефакторинге повторное использование может изменить документы весьма значительно. Однако действительную степень переиспользования оценить сложно, т.к. целесообразно выделять лишь семантически значимые повторы. Для получения более точных оценок необходимо поставить больше экспериментов с документами.

7. ЗАКЛЮЧЕНИЕ

Предложенный в работе подход может найти применение в работе с документацией линеек программных продуктов. Он позволит выделить повторно используемые фрагменты текста для разных продуктов и соответственно реструктурировать документацию, что упростит её сопровождение и повысит качество. Также подход может быт использован в контексте управления вариативностью [35, 36].

При проведении экспериментов мы пришли и к выводу о том, что в целом наш инструмент должен позволять более гибко конструировать

Таблица 3. Количество кандидатов для клонов от 1 и от 5 токенов

Количество кандидатов	LKD	Zend	SVN	DocBook
Одиночные	12819 / 1034	33400 / 5213	27847 / 3119	8228 / 870
Пары	351 / 108	1400 / 613	616 / 249	232 / 50
Всего	13170 / 1254	34800 / 5826	28463 / 3368	8460 / 920

информационные элементы, а его интерфейс — быть дружественнее.

Результаты наших экспериментов показывают, что даже после фильтрации мы имеем дело с большим количеством "мусорных" клонов. Точность фильтрации — одно из основных направлений наших будущих исследований. Также необходимо поддержать более мощное адаптивное повторное использование, предлагая кандидаты с более, чем одной точкой расширения.

Поддержка семантически-ориентированного повторного использования также может позволить совместить наш подход с различными техниками software traceability [37, 38, 39, 40] и обеспечить связи документации с прочими артефактами: программным кодом, требованиями, моделями и т.д. В данном случае повторное использование может повысить качество этих связей, а семантически-ориентированное повторное использование — их точность.

Кроме программной инженерии, предложенный подход также может быть использован в таких областях, как управление знаниями и онтологический инжиниринг [41, 42, 43, 44], а также моделирование архитектуры предприятий [45, 46]: в инструментах, создаваемых и используемых в рамках этих подходов, модели часто хранятся в XML, в них часто встречаются повторы, т.к. при их составлении аналитики работают с большим объёмом информации, которая в значительной степени не структурирована (документы, комментарии к моделям, логические имена сущностей и т.д.).

СПИСОК ЛИТЕРАТУРЫ

1. *Watson R.* Developing best practices for API reference documentation: Creating a platform to study how programmers learn new APIs // Proceedings of IPCC'12. 2012. P. 1–9.
2. *Garousi G., Garousi V., Moussavi M., Ruhe G., Smith B.* Evaluating usage and quality of technical software documentation: an empirical study // Proceedings of EASE'13. 2013. P. 24–35.
3. *Parnas D.L.* Precise Documentation: The Key To Better Software: The Future of Software Engineering, S. Nanz, Ed.: Springer, 2011.
4. *Шалыто А.А.* Новая инициатива в программировании. Движение за открытую проектную документацию / А.А. Шалыто // PC Week / RE. 2003. № 40. С. 38–42.
5. *Holmes R., Walker R.J.* Systematizing Pragmatic Software Reuse // ACM Transactions on Software Engineering and Methodology. 2013. V. 21. No. 4. 44 p.
6. *Czarnecki K.* Software Reuse and Evolution with Generative Techniques // Proceedings of the IEEE/ACM International Conference on Automated Software Engineering, 2007. P. 575.
7. *Bassett P.* The Theory and Practice of Adaptive Reuse. SIGSOFT Software Engineering Notes. 1997. V. 22. No. 3. P. 2–9.
8. *Jarzabek S., Bassett P., Zhang H., Zhang W.* XVCL: XML-based Variant Configuration Language. ICSE 2003. P. 810–811.
9. *Koznov D., Romanovsky K.* DocLine: A Method for Software Product Lines Documentation Development // Programming and Computer Software. 2008. V. 34. No. 4. P. 216–224.
10. *Romanovsky K., Koznov D., Minchin L.* Refactoring the Documentation of Software Product Lines // CEE-SET 2008, Brno (Czech Republic), October 13–15, 2008, LNCS. V. 4980. Springer 2011. P. 158–170.
11. *Кознов Д.В.* Поиск клонов при рефакторинге технической документации / Д.В. Кознов, А.В. Шутак, М.Н. Смирнов, М.А. Смажевский // Компьютерные инструменты в образовании. 2012. № 4. С. 30–40.
12. *Луцив Д.В., Кознов Д.В., Басит Х.А., Ли О.Е., Смирнов М.Н., Романовский К.Ю.* Метод поиска повторяющихся фрагментов текста в технической документации // Научно-технический вестник информационных технологий, механики и оптики. 2014. Т. 4. № 92. С. 106–114.
13. *Basit H.A., Smyth W.F., Puglisi S.J., Turpin A., Jarzabek S.* Efficient Token Based Clone Detection

with Flexible Tokenization // Proceedings of ACM SIGSOFT International Symposium on the Foundations of Software Engineering, ACM Press, 2007. P. 513–516.

14. Document Refactoring Toolkit, http://www.math.spbu.ru/user/kromanovsky/ /docline/index_en.html

15. Linux Kernel Documentation, snapshot on Dec. 11, 2013 (2013), https://github.com/torvalds/linux/tree/master/ /Documentation/DocBook

16. Zend PHP Framework documentation, snapshot on Apr 24, 2015 (2015), https://github.com/zendframework/zf1/tree/ /master/documentation

17. SVN Book, snapshot on Apr 24, 2015 (2015), http://sourceforge.net/p/svnbook/source/HEAD/ /tree/trunk/en/book

18. DocBook Definitive Guide, snapshot on Apr 24, 2015 (2015), http://sourceforge.net/p/docbook/code/HEAD/ /tree/trunk/defguide/en

19. *Zhi J., Garousi V., Sun B., Garousi G., Shahnewaz S., Ruhe G.* Cost, Benefits and Quality of Technical Software Documentation: A Systematic Mapping // J. of Systems and Software, Under Review, 2012. P. 1–24.

20. *Zhong H., Zhang L., Xie T., Mei H.* Inferring source specifications from natural language API documentation // Proceedings of 24th ASE, 2009. P. 307–318.

21. *Zhong H., Su Z.* Detecting API documentation errors // Proceedings of SPASH/OOPSLA, 2013. P. 803–816.

22. *Wingkvist A., Lowe W., Ericsson M., Lincke R.* Analysis and visualization of information quality of technical documentation // Proceedings of the 4th European Conference on Information Management and Evaluation, 2010. P. 388–396.

23. *Wingkvist A., Ericsson M., Lowe W.* A Visualization-based Approach to Present and Assess Technical Documentation Quality // Electronic Journal of Information Systems Evaluation/ 2011. V. 14. No. 1. P. 150–159.

24. VizzAnalyzer Clone Detection Tool http://www.arisa.se/vizz_analyzer.php

25. *Walsh N., Muellner L.* DocBook: The Definitive Guide. O'Reilly, 1999, 644 p.

26. Darwin Information Typing Architecture (DITA) Version 1.2 Specification (2012) http://docs.oasis-open.org/dita/v1.2/os/spec/ /DITA1.2-spec.pdf

27. *Кознов Д.В.* Поиск клонов при рефакторинге технической документации / Д.В. Кознов, А.В. Шутак, М.Н. Смирнов, М.А. Смажевский // Компьютерные инструменты в образовании. 2012. № 4. С. 30–40.

28. *Кознов Д.В.* Метод поиска повторяющихся фрагментов текста в технической документации / Д.В. Луцив, Д.В. Кознов, Х.А. Басит, О.Е. Ли, М.Н. Смирнов, К.Ю. Романовский // Научно-технический вестник информационных технологий, механики и оптики. 2014. № 4. С. 106–114.

29. *Fowler M., Beck K., Brant J., Opdyke W., Roberts D.* Refactoring: Improving the Design of Existing Code. Addison-Wesley, 1999, 431 p.

30. *Rattan D., Bhatia R.K., Singh M.* Software Clone Detection: A Systematic Review. Information & Software Technology (INFSOF). 2013. V. 55. No. 7. P. 1165–1199.

31. *Akhin M., Itsykson V.* Clone Detection: Why, What and How? // Proceedings of CEE-SECR'10, 2010. P. 36–42.

32. *Abouelhoda M.I., Kurtz S., Ohlebusch E.* Replacing suffix trees with enhanced suffix arrays // Journal of Discrete Algorithms 2. 2004. V. 53.

33. *Basili V.R., Caldeira G., Rombach H.D.* The Goal Question Metric Approach. John Wiley & Sons, Inc., 1994. V. 1. P. 528–532.

34. *Frakes W., Terry C.* Software reuse: metrics and models // ACM Comput. Surv. 1996. V. 28. No. 2. P. 415–435.

35. *Krueger C.W.* Variation Management for Software Product Lines // Proceedings of SPL'02, San Diego, CA, USA, 2002. P. 37–48.

36. *Кознов Д.В.* Инструменты для управления вариативностью — готовность к промышленному применению / Д.В. Кознов, И.А. Новицкий, М.Н. Смирнов // Труды СПИИРАН. 2013. № 3. С. 297–331.

37. *Abadi A., Nisenson M., Simionovici Y.* A Traceability Technique for Specifications // Proceedings of ICPC'08, 2008. P. 103–112.

38. *Terekhov A.N., Sokolov V.V.* Document Implementation of the conformation of MSC and SDL diagrams in the REAL technology //

Programming and Computer Software. 2007. V. 33. No. 1. P. 24–33.

39. *Кознов Д.В.* WebMLDoc: подход к автоматизированному отслеживанию изменений в пользовательской документации Web-приложений / Д.В. Кознов, М.Н. Смирнов, В.А. Дорохов, К.Ю. Романовский // Вестник Санкт-Петербургского университета. Серия 10: Прикладная математика. Информатика. Процессы управления. 2011. № 3. С. 112–126.

40. *Кознов Д.В.* Программная среда WebMLDoc для автоматизированного отслеживания изменений в пользовательской документации Web-приложений / М.Н. Смирнов, Д.В. Кознов, В.А. Дорохов, К.Ю. Романовский // Системное программирование. 2010. V. 5. No 1. С. 32–51.

41. *Gavrilova T.A.* Ontological engineering for practical knowledge work // 11th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, KES 2007, LNCS V. 4693. P. 1154–1161.

42. *Гаврилова Т.А.* Логико-лингвистическое управление знаниями // Новости искусственного интеллекта. 2002. № 6. С. 45–60.

43. *Bolotnikova E.S., Gavrilova T.A., Gorovoy V.A.* To a method of evaluating ontologies // Journal of Computer and Systems Sciences International. 2011. V. 50. № 3. P. 448–461.

44. *Гаврилова Т.А., Лещева И.А., Лещев Д.В.* Использование онтологий в качестве дидактического средства. Искусственный интеллект. 2000. № 3. С. 34–39.

45. *Grigoriev L., Kudryavtsev D.* ORG-Master: Combining Classifications, Matrices and Diagrams in the Enterprise Architecture Modeling Tool // Communications in Computer and Information Science (CCIS) Series, Springer, 2013. P. 250–258.

46. *Кознов Д.В.* Особенности проектов в области разработки корпоративной архитектуры предприятий / Д.В. Кознов, М.Ю. Арзуманян, Ю.В. Орлов, М.А. Деревянко, К.Ю. Романовский, А.А. Сидорина // Бизнес-информатика. 2015. № 4. С. 15–26.