Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»
Кафедра информационно-аналитических систем

Клочкова Екатерина Владимировна

Семантический подход к повторному использованию фрагментов технической и нормативной документации

Бакалаврская работа

Допущена к защите.
Зав. кафедрой:
д. ф.-м. н., профессор Новиков Б. А.

Научный руководитель:
старший преподаватель Луцив Д.В.

Рецензент:
аспирант Дзендзик Д.А.

Санкт-Петербург
2015

SAINT-PETERSBURG STATE UNIVERSITY

Sub-Department of Analytical Information Systems

Ekaterina Klochkova

Semantic approach to reuse of technical and regulatory documentation fragments

Graduation Thesis

Admitted for defence.
Head of the chair:
Professor Boris Novikov

Scientific adviser
Senior Lecturer Dmitry Luciv

Reviewer:
Daria Dzendzik

Saint-Petersburg
2015

Оглавление

1. Введение

В наши дни для программного обеспечения (ПО) создаётся широкий спектр самой разнообразной документации. Как правило, документация объёмна и имеет сложную структуру. Известно, что в документации нередко встречаются повторы: одинаковая информация содержится в разных частях документации, предназначенных для разных групп пользователей; она может встречаться в одном и том же документе с разной степенью детализации; для разных версий продукта; в качестве описания одинаковой функциональности, для разных программных продуктов одного семейства и т.д. Таким образом сопровождение документации – а это важный процесс поддержания документации в актуальном состоянии – очень трудоёмкий процесс, когда при внесении изменений приходится находить все повторы и вносить изменения согласованно. Типичные средства для обработки текста здесь не вполне уместны, так как зачастую информация повторяется не буквально: это могут быть похожие разделы в пользовательской документации (например, описание одинакового действия для разных компонент), описания одной функциональности с разной степенью детальности или даже по логике одинаковых разделы, но оба написанные «вручную», а потому неточные.

Очевидна необходимость технологии для управления многочисленными вариативными повторами и облегчения сопровождения документации. В качестве решения была предложена технология DocLine[1]. Она работает с документами в формате DocBook, который является популярным для написания документации, и с его расширением DRL, которое предназначено для документации линеек продуктов, но может также быть использовано для повторного использования фрагментов. Был предложен метод поиска повторяющихся фрагментов текста в технической документации [2,3] на основе Clone Miner [4, 6].

Однако эксперименты показали, что необходимо повышать точность алгоритма, а также настроить семантический подход в повторном использовании повторяющихся фрагментов. В контексте данной работы семантический подход означает, что не следует повторно использовать бессмысленный тест: случайные повторы, устойчивые выражения и так далее. Также ожидается подход, который опирается на смысл синтаксических конструкций языка документации, а не формально ищет синтаксически одинаковые фрагменты.

Целью этой работы является тестирование имеющегося инструмента, добавление предварительной фильтрации документации, а также улучшение восстановления xml-разметки.

Тестирование было проведено на 4 различных реальных комплектах документации в формате DocBook по открыто лицензируемым продуктам.

Фильтрация документации была достигнута с помощью библиотеки NLTK для языка python. Входящий текст разбивается на токены, размечается частями речи, а

потом из него убираются некоторые экспериментально подобранные паттерны со служебными частями речи. Подробнее об этом в разделе 5.1.
Xml-разметка балансируется в зависимости от тегов, в соответствии с нотацией DocBook: структурные просто исключаем; внутренние, от которых не зависит структура документа, а которые меняют, например, шрифт текста, дополняем. Подробнее об этом в разделе 5.2.

2. Постановка задачи

Провести детальное исследование существующего решения. Протестировать имеющийся инструмент, обосновать актуальность проблемы, выяснить, что такое семантика, и как в этом контексте можно улучшить имеющийся инструмент.

1. Изучить существующее решение и связанные технологии: DocBook, DRL, DocLine, CloneMiner, Documentation Refactoring Toolkit
2. Протестировать имеющийся инструмент, оценить его эффективность.
3. Предложить улучшения существующей технологии.
4. Реализовать и апробировать внесенные улучшения.
5. Интегрировать улучшения в исходный код Documentation Refactoring Toolkit

3. Обзор

3.1. Контекст работы

DocLine – метод разработки и сопровождения документации семейств программных продуктов и большой документации с вариативным повторным использованием повторяющихся фрагментов. Для представления документации в DocLine создан новый XML-язык разметки DRL (Documentation Reuse Language) и модель процесса разработки документации. DRL является расширением DocBook, относительно него добавленые две новые конструкции для вариативного повторного использования: информационный элемент и каталог элементов. Технологическое решение используемое в данной работе – легковесный инструмент Documentation Refactoring Toolkit, реализованный на python, имеющий графический интерфейс, позволяющий запускать его из рабочего каталога, не разворачивая DocLine, выбирать файл, в котором необходимо найти клоны, задавать некоторые параметры и производить рефакторинг текста.

DocBook является инструментом, изначально предназначенным для создания большой технической документации со сложной структурой. Он разделяет структуру документа и его стилистическое оформление, что позволяет получать разные выходные форматы из одного изначального документа: PDF, HTML и т.д. Структурно DocBook похож на XML. Существуют различные расширения DocBook, из которых, однако, можно легко получить DocBook для быстрого получения документа в нужном формате с помощью стандартных моделей.

Аналогами DocBook для работы с с документацией, являются DITA от IBM и FrameMaker от Adobe.
В DITA документация представляется в виде независимых топиков (topic) – крупных фрагментов текста, с помощью которых реализуется повторное использование. Механизм повторного использования DITA основан на условном включении фрагментов текста. Существует список переменных, который может дополняться пользователем. Эти переменные можно использовать при написании условно-включаемого текста.
Язык форматирования документов DITA, также как и DocBook, основан на XML.
Документация DITA, также как и DocBook может быть преобразована различные выходные форматы. Но стандартные средства уступают DocBook в области полиграфического оформления текста.
FrameMaker один из самых популярных инструментов для разработки документации, «благодаря стабильности при работе с большими пакетами

документов, а также развитой поддержке полиграфии». Встроенный язык FrameMaker также создан на основе XML, но при этом он позволяет использовать и DocBook, DITA и другие языки XML-разметки.
Его главным недостатком является отсутствие удобных средств адаптивного повторного использования.

Информационный элемент – основная единица языка DRL. Создан для вариативного повторного использования. Повторяющийся фрагмент текста с задаваемыми параметрами. Может включаться в различные места документации с помощью ссылки на него.

Словарь – другая важная конструкция языка DRL. Это структура, состоящая из набора «ключ-значение». Словарь состоит из небольших семантически осмысленных фрагментов текста – номер версии, перечень поддерживаемых платформ и т.д. – *элементов словаря*.

CloneMiner – пакет по поиску клонов. Его основные преимущества: простой программный интерфейс (командная строка), лёгок в интеграции. Clone Miner преобразует исходный текст в последовательность токенов, а затем при помощи алгоритмов поиска, основанных на суффиксных массивах, находит повторяющиеся куски текста. В качестве параметра он принимает минимальную длину искомых слов. В текстовых документах, токен – это слово, отделённое от остальных любым из разделителей `.', `(', `)', и т.д. Невариативные клоны в DRL искал выпускник кафедры информатики 2013 года Дмитрий Копин. [10] Его особенностями является то, что изначально он работает только с плоским текстом и ищет только точные клоны. В нашей ситуации текст представлен в DocBook-разметке, а значит её нужно учитывать, и необходимо уметь искать вариативные клоны, а не только точные.
Неточные (вариативные) клоны – клоны с *точками расширения.* Не абсолютно одинаковые фрагменты текста, а имеющие различия. Например:

<example> This is just a text. </example>	<example> This is just a text *for example*. </example>

В таком случае, если задать минимальную длину клона не более 5, данные фрагменты будут неточными клонами, а '*for example'* – точкой расширения.

3.2. Documentation Refactoring Toolkit

Разработан Documentation Refactoring Toolkit — инструмент поиска клонов и рефакторинга документации в рамках проекта DocLine с графическим интерфейсом на базе пакета по поиску клонов CloneMiner [4]. Это простой в использовании инструмент в рамках проекта DocLine, который можно запускать прямо из рабочего каталога, не разворачивая DocLine.

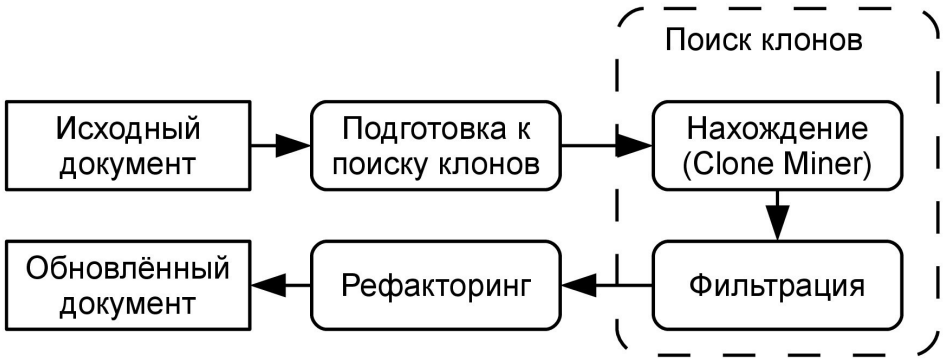

Рисунок 1: Алгоритм работы

Упрощенный алгоритм работы следующий (рис.1):

1. На вход подаётся исходный DRL-файл, а также пользователь может задать следующие параметры:
   - исходный файл для анализа
   - минимальную длину искомых клонов
   - способ проверки и восстановления нарушений XML-разметки
   - дополнительные опции поиска клонов:
2. Производится подготовка к поиску клонов: удаление разметки, токенизация
3. Поиск клонов:
   1. Поиск клонов с помощью CloneMiner
   2. Фильтрация найденных клонов
4. Рефакторинг пользователем
5. На выходе получаем документ после рефакторинга

3.3. Предлагаемые улучшения и их обоснование

CloneMiner сам по себе ищет только синтаксически одинаковые клоны. Программная оболочка Documentation Refactoring Toolkit позволяет искать вариативные клоны с точками расширения. Но для технических писателей, для которых создаётся инструмент, важна семантика клонов. Нужно оперировать не текстом, а информацией.

Были выбраны 2 семантических аспекта клонов: фильтрация полученных

кандидатов на информационные элементы – чтобы технический писатель получал меньше ненужных клонов, но при этом не терялись важные для рефакторинга фрагменты, и корректное восстановление полученной разметки: информационный элемент может содержать непарные теги, при редактировании такого фрагмента текста, пользователь может сломать разметку документа.
На рис. 2 детализированы процессы поиска клонов и рефакторинга.

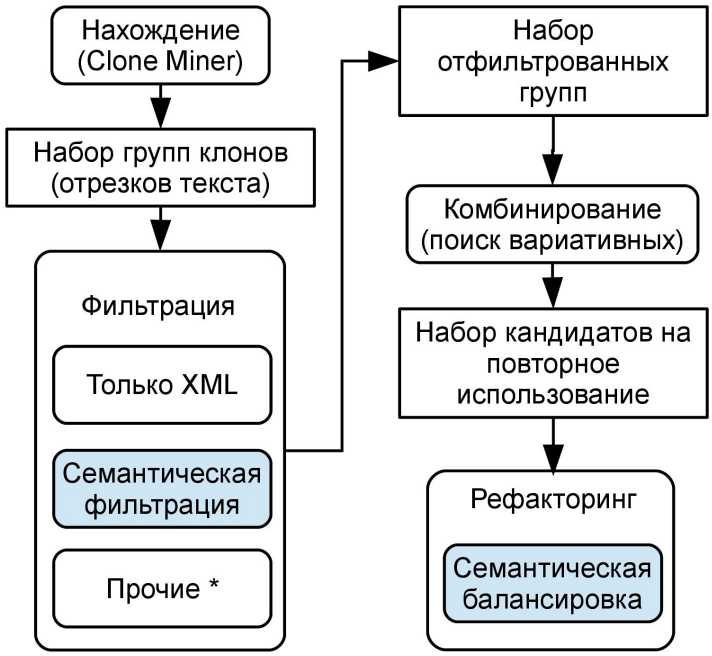

Рисунок 2: Нахождение клонов и рефакторинг

4. Актуальность задачи

Одной из целей этой работы было показать актуальность проблемы. Утверждается, что в документациях существует множество повторяемых фрагментов, что усложняет её написание и сопровождение.
Были взяты 4 реальных документации:

- DocBook Guide
- SVN Guide
- Linux Kernel Documentation
- Zend Manual.

Проведен ряд тестов.

1. Параметры взяты по умолчанию:
   Минимальная длина клона = 10.
   1. Без предварительной фильтрации, пересечения разрешены

Документация	Количество символов в документе	Количество групп клонов	Общий объем клонов
DocBook Guide	720407	247	5260093
Linux Kernel	939645	316	5231219
SVN Book	1898074	771	5484349
Zend Manual	3066801	1842	5789316

Таблица 1: Результат поиска клонов без фильтрации

Большие документы по несколько десятков сотен знаков, количество групп клонов достаточно большое – по несколько сотен, учитывая то, что у нас стоит ограничение на минимальную длину клона. Если её уменьшить, будет значительно больше клонов, на порядок.

2. Те же параметры, но с фильтрацией словосочетаний по словарю, составленному вручную

Документация	Количество символов в документе	Количество групп клонов	Общий объем клонов
DocBook Guide	720407	245	5259766
Linux Kernel	939645	304	5219640
SVN Book	1898074	643	5315970
Zend Manual	3066801	1832	5775211

Таблица 2: Результат поиска клонов с фильтрацией по словарю

Видно заметное улучшение в SVN Book, во всех остальных случаях не так эффективно.
В таблице представлено сравнение объемов найденных клонов без фильтрации и с фильтрацией с помощью словаря.

Документация	Разница числа групп клонов	Разница объемов клонов в символах	% общего объема клонов без фильтр
DocBook Guide	2	327	0.0062166201
Linux Kernel	12	11579	0.2213442029
SVN Book	128	168379	3.0701729595
Zend Manual	10	14105	0.243638454

Таблица 3: Сравнение поиска клонов с фильтрацией по словарю и без

5. Семантический подход

Изначально инструмент CloneMiner оперировал синтаксически одинаковыми фрагментами плоского текста. В нашем случае, это не лучший подход, во-первых, потому что в документации достаточное количество вариативных клонов: похожие фрагменты инструкций или описания функциональности, но с разной степенью детализации или различающиеся для разных групп пользователей и т. д. Для этого был создан язык DRL: для повторного использования *вариативных* информационных элементов. Поэтому ранее была добавлена возможность поиска вариативных клонов с точками расширения — так мы можем находить близкие, но не одинаковые фрагменты документации.

Во-вторых, потому что технический писатель работает в первую очередь с информацией, то есть нужно находить семантически осмысленные и близкие фрагменты, а не просто одинаковые. При поиске вариативных клонов в документациях большого объема — а именно такие и являются объектом рассмотрения – находится большое количество похожих фрагментов, как видно из предыдущего раздела, по несколько сотен. Далее техническому писателю предлагается просмотреть все предложенные варианты и выбрать информационные элементы, с которыми нужно работать (рис.3).

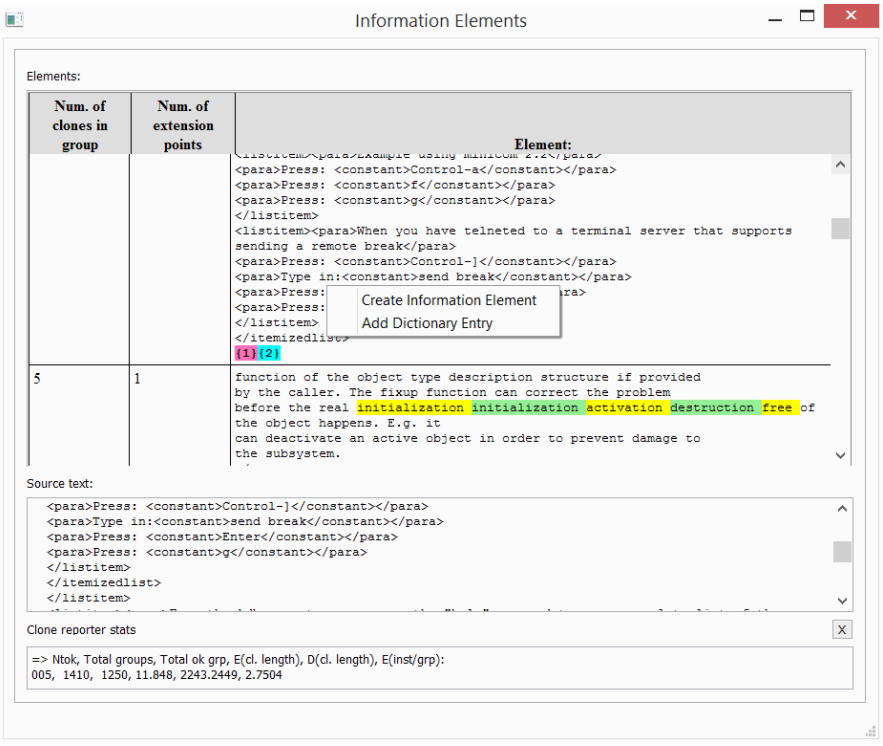

Иллюстрация 1: Результат работы Documentation Refactoring Toolkit

Просмотреть 300 фрагментов текста, длина некоторых из которых тоже достигает

нескольких сотен символов не самая лёгкая задача. Поэтому хочется найденные возможные клоны предварительно фильтровать.
В-третьих, рассматриваемая документация хранится в формате DocBook, то есть это не плоский текст, а текст с разметкой некоторым наборов тегов по некоторым правилам. Перед использованием CloneMiner документация токенизируется и отбрасываются лишние знания о разметке. Но затем разметку необходимо восстановить. Отдельная сложность заключается в случае, если найденный клон разрушает разметку, например, захватывает только закрывающий тег:

<title>It is just title *for example</title>*

Пусть в данном случае клоном оказался только фрагмент for example</title>. Очевидно, в данном случае проще закрывающий тег отбросить и для рефакторинга взять только 'for example', но это только один простой пример для наглядной демонстраци сложностей с разметкой.

5.1. Фильтрация

Рассмотрим способ уменьшения предлагаемых клонов — фильтрацию текста.
В любом большом документе, в частности, в документации, будут устойчивые выражения, которые повторяются множество раз. Есть предположение, что если их отсеять, количество полученных претендентов можно сократить.
Изначально в инструмент была добавлена возможность отсеивания слов и словосочетаний из словаря, составленного вручную. Минус такого подхода, что для каждой новой документации необходимо составить свой словарь, а значит вручную просмотреть огромный текст и выбрать частовстречающие словосочетания. Это довольно трудоемкая задача, а мы хотим облегчить работу человеку, сопровождающему документацию. К тому же, судя по проведённым тестам, результаты были не вполне удовлетворительными. Хотя, это напрямую зависит от качества составленного словаря, но затраченное время на его составление не стоило полученного результата.
Было предложено отсеивать паттерны – некоторые шаблоны словосочетаний. Например, можно отсеивать фразы вида there is/are, this is – предлог + модальный глагол или просто служебные части речи и так далее. Так как Documentation Refactoring Toolkit был написан на python, искались решения и библиотеки для этого языка программирования.
Было решено отсеивать паттерны, состоящие из служебных частей речи, таких как местоимения, артикли, модальные глаголы и так далее (работа велась с англоязычной документацией). Для этого необходимо сначала разметить текст частями речи (part-of-speech tagging) – это этап автоматической обработки текста, целью которого является определение части речи в тексте с приписыванием им

соответствующих тегов.
Был взят инструмент NLTK, являющийся ведущей платформой для работы с естественным языком на Python. Он предоставляет легкий в использовании интерфейс для более чем 50 корпусов и лексических ресурсов, наряду с набором библиотек для обработки текста для классификации, токенизации, разметки частей речи и так далее. К тому же, NLTK имеет хорошее практическое руководство, доступен для всех ОС: Windows, Mac OS X, Linux и является проектом с открытым исходным кодом и свободной лицензией.
В нашем случае выбор корпуса не принципиален, так что был взят стандартный, который идёт в NLTK. Нам нужно лишь выделять служебные части речи.
После разметки, нужно выделить паттерны, которые необходимо фильтровать. Их можно задавать для каждой документации отдельно, с учетом особенностей и её словаря для фильтрации, если таковой имеется.
Здесь представлены результаты тестирования тех же документов с теми же параметрами с фильтрацией по словарю и с помощью двух паттернов: достаточно простой, отсеивающий словосочетания с модальными глаголами и сложнее, составленный на основе имеющегося словаря для фильтрации:

1. if ((fword == 'MD') and (sword == 'VB') or (fword == 'MD') and (sword == 'DT') or (fword == 'MD') and (sword == 'WDT')):
    return True
2. if ((fword == 'MD') and (sword == 'VB') or (fword == 'MD') and (sword == 'DT') or (fword == 'MD') and (sword == 'WDT') or (fword == 'DT') and (sword == 'is') or (fword == 'DT') and (sword == 'DT')or (fword == 'IN') and (sword == 'DT') or (fword == 'WRB') and (sword == 'is') or (fword == 'WRB') and (sword == 'DT') or (fword == 'IN') and (sword == 'CC')):
    return True

Фильтрация с помощью словаря:

Документация	Количество символов в документе	Количество групп клонов	Общий объем клонов
DocBook Guide	720407	245	5259766
Linux Kernel	939645	304	5219640
SVN Book	1898074	643	5315970
Zend Manual	3066801	1832	5775211

Таблица 4: Результат поиска клонов с фильтрацией по словарю

Фильтрация с помощью первого паттерна и NLTK:

Документация	Количество символов в документе	Количество групп клонов	Общий объем клонов
DocBook Guide	720407	236	5254550
Linux Kernel	939645	267	5100957
SVN Book	1898074	718	5443192
Zend Manual	3066801	1692	5718532

Таблица 5: Результат поиска клонов с фильтрацией с помощью NLTK, первый паттерн

Сравнение подходов:

Первый столбец – на сколько групп клонов получилось меньше. Разница объемов клонов в символах – на сколько символов уменьшился общий объем найденных клонов, третий столбец – на сколько процентов уменьшился объем найденных клонов.

Документация	Разница числа групп клонов	Разница объемов клонов в символах	% общего объема клонов без фильтр
DocBook Guide	11	5543	0.105378365
Linux Kernel	49	130262	2.4900888302
SVN Book	53	41157	0.7504445833
Zend Manual	150	70784	1.2226660282

Таблица 6: Сравнение результатов, полученных без фильтрации и с помощью первого паттерна с использованием NLTK

Документация	Разница числа групп клонов	Разница объемов клонов в символах	% общего объема клонов без фильтр
DocBook Guide	9	5216	0.0991617449
Linux Kernel	37	118683	2.2687446272
SVN Book	-75	-127222	-2.3197283761
Zend Manual	140	56679	0.9790275742

Таблица 7: Сравнение результатов, полученных с фильтрацией с помощью словаря и с помощью первого паттерна с использованием NLTK

В 3 из 4 случаев, NLTK дала меньшее количество групп клонов, чем фильтрация

со словарем, в одном — SVN Book – большее, что значит, что словарь был достаточно хорошо подобран для этой документации. В идеальном случае мы вообще можем в словарь внести все ненужные нам клоны и получить «чистый результат» – только те фрагменты текста, которые нам интересны.

Результаты использования второго паттерна:

Документация	Количество символов в документе	Количество групп клонов	Общий объем клонов
DocBook Guide	720407	204	5242277
Linux Kernel	939645	198	579775
SVN Book	1898074	602	5394226
Zend Manual	3066801	1402	5599836

Таблица 8: Результат поиска клонов с фильтрацией с помощью NLTK, второй паттерн

Документация	Разница числа групп клонов	Разница объемов клонов в символах	% общего объема клонов без фильтр
DocBook Guide	43	17816	0.3387012359
Linux Kernel	118	4651444	88.9170191498
SVN Book	169	90123	1.6432761664
Zend Manual	440	189480	3.2729255062

Таблица 9: Сравнение результатов, полученных без фильтрафии и с помощью второго паттерна с использованем NLTK

Документация	Разница числа групп клонов	Разница объемов клонов в символах	% общего объема клонов без фильтр
DocBook Guide	32	12273	0.2333228709
Linux Kernel	69	4521182	86.4269303197
SVN Book	116	48966	0.8928315831
Zend Manual	290	118696	2.050259478

Таблица 10: Сравнение результатов, полученных с помощью разных паттернов и NLTK

Таким образом очевидно, что фильтрация с использованием NLTK имеет смысл, и её можно

гибко настраивать с помощью паттернов. С другой стороны, можно использовать комбинированныц подход, например, в том же паттерне указывать конкретные словосочетания.

5.2. Восстановление xml-разметки

При выделении клона из текста возможно нарушение разметки. Хочется, чтобы при выделении фрагмента в качестве информационного элемента, разметка была максимально корректно восстановлена и сбалансирована.

Можно было бы восстанавливать информационный элемент по документу до минимального объема, при котором все теги входят в него парами. Например, тот же пример, что был выше. Кандидат на информационный элемент выделен курсивом:

<title>It is just title *for example*</title>

Нашли в клоне закрывающийся тег, пошли обратно по документу, нашли открывающийся, получили информационный элемент, соответствующий всему заголовку. У такого подхода есть, как минимум, один значительный минус: пользователю предоставляется много лишней информации: информация о разметке, которая, скорее всего, при рефакторинге не нужна – техническому писателю нужна в первую очередь *информация*, а не голый *текст* документа. Во вторых, не все примеры такие простые, возможно, добавленного текста будет гораздо больше, чем изначально в клоне, например:

```
<sect1 id="routines-module-use-counters">
    <title>
        <function>try_module_get()</function>/
        <function>module_put()</function>
        <filename class="headerfile">include/linux/module.h</filename>
    </title>
    <para>
        These manipulate <....>
     </para>
    <para>
        Most registerable structures have an<structfield>owner</structfield> field,
    such as in the <structname>file_operations</structname> structure. Set this
    field to the macro <symbol> THIS_MODULE</symbol>.
    </para>
 </sect1>
```

Здесь также клон – претендент на информационный элемент выделен курсивом.

Очевидно, что если мы возьмём весь фрагмент текста <sect1 ...> … </sect1> объем фрагмента увеличится в несколько раз, что нежелательно.
При этом не хочется, чтобы при рефакторинге нарушился баланс разметки, поэтому оставлять одиночные незакрытые теги нужно как можно реже. Есть 2 варианта: обрезать незакрытые теги или вставлять дополнительные, чтобы получилось 2 элемента, закрытые тегами. Поясним на примерах.

1. Как обрезать незакрытые теги очевидно:

<title>It is just title *for example</title>*

Информационным элементом возьмём 'for example'. Такой подход плох тем, что мы скрываем от пользователя тег непосредственно относящийся к интересующему информационному элементу.

2. Вставка дополнительных тегов:

<msgtext>It is just msg *for example</msgtext>*

<msgtext>It is just msg</msgtext> *<msgtext> for example</msgtext>*

Информационным элементом будет выступать *'<msgtext> for example </msgtext>'*.

В данном случае мы не скрываем ближайшие теги, но если такой тег отвечает за разметку, есть какая-то вложенность тегов, то мы можем что-то сломать.
Был произведён анализ документации по DocBook [7],[8], теги, входящие в его нотацию. Условно все теги DocBook можно разделить на структурные (Block Elements) и внутренние (Inline Elements).
Таким образом, с внутренними элементами мы можем поступать вторым способом, а структурные теги «обрезать от информационного» элемента. Был выделен 91 внутренний тег DocBook, добавлена сущность inlineElements, и при восстановлении разметки, проверяется, структурный или внутренний тег и выбирается соответствующее поведение: удаление одиночного тега или добавление дополнительных и разделение тегированного элемента на два.

6. Заключение

1. Изучены существующие решения и связанные технологии: DocBook, DRL, DocLine, Clone Miner
    - Произведено практическое знакомство с инструментом: установлен, настроен и запущен на тестовом примере плагин
2. Проведены тесты на реальной документации
3. Предложены:
    - семантическая фильтрация на основе паттернов с помощью библиотеки NLTK python
    - восстановление разметки в соответствие с нотацией DocBook с учётом смысла этой разметки: к какой смысловой – структурной или внутренней – группе относится данный тег
4. Предложенные изменения реализованы
5. Улучшения интегрированы в исходный код Documentation Refactoring Toolkit и протестированы

Список литературы

1. Кознов Д.В., Романовский К.Ю. DocLine: метод разработки документации семейств программных продуктов // Программирование. 2008. Т 34. № 4. С. 41−53.

2. Д.В. Луцив, Д.В. Кознов, Х.А. Басѝ, О.Е. Лид, М.Н. Смирнов, К.Ю. Романовский Метод поиска повторяющихся фрагментов текста в технической документации // Научно-технический вестник информационных технологий, механики и оптики 2014, № 4 (92)

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

4. Basit H.A., Jarzabek S. A Data Mining Approach for Detecting Higher-Level Clones in Software. // IEEE Transactions on Software engineering. Vol. 35, № 4, 2009.

5. Walsh N., Muellner L. DocBook: The Definitive Guide. O'ReillyMedia, 1999. 644 p.

6. NLTK 3.0 documentation. – URL: http://www.nltk.org/ (дата обращения: 20.05.2015)

7. Norman Walsh, Leonard Muellner, Bob Stayton DocBook's homepage – URL: http://www.docbook.org/ (дата обращения: 20.05.2015)

8. Norman Walsh, Richard Hamilton DocBook The Definitive Guide. // XML Press, 16 July, 2014

9. Романовский, К.Ю. (2010). Метод повторного использования документации семейств программных продуктов. СПбГУ, математико-механический факультет, Санкт-Петербург.

10. Копин, Д. В. (2013). Поиск клонов в XML-документации. СПбГУ, математико-механический факультет, кафедра информатики, Санкт-Петербург.