1-2014

An approach for clone detection in documentation reuse

Dmitry V. LUTSIV

Dmitry KOZNOV

Hamid A. BASIT

Eng Lieh OUH
Singapore Management University, elouh@smu.edu.sg

Mikhail N. SMIRNOV

See next page for additional authors

УДК 007:681.512.2

МЕТОД ПОИСКА ПОВТОРЯЮЩИХСЯ ФРАГМЕНТОВ ТЕКСТА В ТЕХНИЧЕСКОЙ ДОКУМЕНТАЦИИ

Д.В. Луцив[a, b], Д.В. Кознов[a], Х.А. Баси́т[c], О.Е. Ли[d], М.Н. Смирнов[a, b], К.Ю. Романовский[a]

[a] Санкт-Петербургский государственный университет, Санкт-Петербург, Россия, dkoznov@yandex.ru
[b] ЗАО «Ланит-Терком», Санкт-Петербург, Россия
[c] Университет менеджмента Лахора, Пакистан
[d] Национальный университет Сингапура, Сингапур

Аннотация. Предложен метод, позволяющий искать повторы в технической документации, выполненной в формате DocBook/DRL или в виде «плоского» текста. Разработан алгоритм, основанный на технике поиска клонов в программном обеспечении (software clone detection). Алгоритм реализует фильтрацию стандартного поиска клонов: отбрасываются клоны, чья длина меньше 5 символов; выполняется устранение пересечения клонов, а также удаление несущественных клонов и клонов, состоящих только из XML-конструкций. Поддерживается поиск по остаткам. После нахождения клонов они устраняются из документации, и поиск повторяется. Доказывается достаточность одного шага. Реализована техника адаптированного повторного использования Бассета–Ерзабека. На основе предложенного алгоритма разработан программный инструмент, поддерживающий параметризацию поиска повторов, а также визуализацию полученных результатов. Инструмент интегрирован со средой разработки повторно используемой документации DocLine и реализует рефакторинг документов на основе найденных клонов. Инструмент использует утилиту Clone Miner для поиска клонов. Представлена апробация метода для документации к ядру операционной системы Linux (29 документов, 25 000 строк). Выделено 5 видов клонов: термины, гиперссылки, лицензии, описание функциональности, примеры кода. Всего найдено 451 содержательных групп клонов. Средняя длина клона – 4,43 токена. Среднее количество клонов в группе – 3,56. Предложенный подход может оказаться полезным в средах работы с документацией семейств программных продуктов.

Ключевые слова: документация программного обеспечения, переиспользование документации, поиск клонов в программном обеспечении, вариативное переиспользование, рефакторинг, DocBook, DocLine, DRL.

Благодарности. Работа выполнена при поддержке гранта РФФИ 12-01-00415-а. Авторы благодарят студентов Артема Шутака, Дмитрия Копина, Михаила Смаржевского и Аделя Кана (Adeel Khan), реализовавших часть прототипов использованных программных инструментов и принимавших активное участие в обсуждении предмета работы.

AN APPROACH FOR CLONE DETECTION IN DOCUMENTATION REUSE

D.V. Lutsiv[a, b], D.V. Koznov[a], H.A. Basit[c], O.E. Lieh[d], M.N. Smirnov[a, b], K.Yu. Romanovsky[a]

[a] Saint Petersburg State University, Saint Petersburg, Russia, dkoznov@yandex.ru
[b] "Lanit-Tercom" Ltd., Saint Petersburg, Russia
[c] Lahore University of Management Sciences, Lakhora, Pakistan
[d] National University of Singapore, Singapore

Abstract. The paper focuses on the searching method for repetitions in DocBook/DRL or plain text documents. An algorithm has been designed based on software clone detection. The algorithm supports filtering results: clones are rejected if clone length in the group is less than 5 symbols, intersection of clone groups is eliminated, meaningfulness clones are removed, the groups containing clones consisting only of XML are eliminated. Remaining search is supported: found clones are extracted from the documentation, and clone search is repeated. One step is proved to be enough. Adaptive reuse technique of Paul Bassett – Stan Jarzabek has been implemented. A software tool has been developed on the basis of the algorithm. The tool supports setting parameters for repetitions detection and visualization of the obtained results. The tool is integrated into DocLine document development environment, and provides refactoring of documents using found clones. The Clone Miner clone detection utility is used for clones search. The method has been evaluated for Linux Kernel Documentation (29 documents, 25000 lines). Five semantic kinds of clones have been selected: terms (abbreviations, one word and two word terms), hyperlinks, license agreements, functionality description, and code examples. 451 meaningful clone groups have been found, average clone length is 4.43 tokens, and average number of clones in a group is 3.56.

Keywords: software documentation, documentation reuse, software clone detection, adaptive reuse, refactoring, DocBook, DocLine, DRL.

Acknowledgements. The work was supported by by RFBR grant 12-01-00415-a. The authors express their gratitude to the students (Artyom Shutak, Dmitry Kopin, Michail Smarzhevsky and Adeel Khan) who implemented a part of soft-based prototypes and took an active part in the discussion about the subject of research.

Введение

Документация является важной составляющей современного программного обеспечения (ПО). При разработке ПО создается много видов документации: руководства пользователя, описания архитектуры, сопроводительные заметки о выпусках и пр. Эти документы обладают сложной структурой и значительным объемом, и, как и само ПО, они постоянно изменяются в течение жизненного цикла. Сопровождение документации оказывается трудоемкой работой, поскольку в документах, как правило, много повторов, например, описаний одной и той же функциональности, присущей разным программным продуктам одного семейства. Если никаких связей между дубликатами не задано, то все они должны редактироваться синхронно «вручную». Большая трудоемкость данного процесса, а также обычная для программных проектов нехватка времени приводят к появлению и накоплению в документации противоре-

чий и ошибок. Ситуация дополнительно усложняется тем, что зачастую информация повторяется не буквально: например, в разных разделах одного и того же документа одна и та же функциональность может быть описана с разной степенью подробности. Это приводит к вариативности повторений, что затрудняет их поиск типичными средствами обработки текстов. Кроме того, в конечном итоге обычно требуется не просто обнаружить дубликаты, но и выполнять над ними различные операции, например, оформлять фрагменты текста как повторно используемые, вынеся повторы в одно место и заменив все их вхождения ссылками на это описание.

Плановое повторное использование широко используется при разработке. Различные техники представлены, например, в [1, 2]. Одна из них – XVCL (XML-Based Variant Configuration Language) [3] – основана на технике вариативного повторного использования Пола Бассета [4]. XVCL и вариативное повторное использование были применены для разработки документации в технологии DocLine [5]. В рамках последней была предложена техника рефакторинга XML-документации [6, 7], однако проблема автоматизированного поиска пригодных для повторного использования фрагментов документов осталась нерешенной.

Первой попыткой решить данную задачу стал подход, изложенный в 8]. Для поиска повторов использовалась техника поиска клонов в программном обеспечении (software clone detection) [7–9], в качестве базового средства поиска клонов использован пакет Clone Miner 10]. Подход работал с документацией в формате DocBook [11] (Linux-сообществе) и DRL [5] (надстройка над DocBook для организации вариативного повторного использования средствами технологии DocLine), а также для «плоского» текста. Однако алгоритм, представленный там, давал до 1/3 ложных срабатываний и уменьшал найденные клоны. В настоящей работе представлены улучшенная версия алгоритма и его программная реализация, интегрированная со средствами рефакторинга документации в технологии DocLine, а также апробация на документации к ядру операционной системы Linux [12].

Контекст работы

DocBook. Технология DocBook [11] является набором стандартов и инструментов, предназначенных для создания технической документации, имеющей большой размер и сложную структуру. Особенностью данной технологии является разделение структуры документа и его стилистического оформления, что позволяет преобразовывать один исходный документ в разные выходные форматы – HTML, PDF и т.д. Технология не является WYSIWYG (What You See Is What You Get), поэтому требует дополнительной подготовки для использования (с дискуссией о применении техническими писателями языков разметки можно познакомиться в [13]). DocBook легко расширяется и позволяет вводить дополнительные конструкции, которые потом могут быть автоматически преобразованы в базовый DocBook, чтобы с документами по-прежнему можно было выполнять стандартные операции (например, генерировать PDF).

DocLine. Технология DocLine [5] создана для разработки и сопровождения сложной документации ПО с использованием вариативного повторного использования [3, 4] повторяющихся фрагментов документов. Вариативность в данном случае означает возможность настройки фрагментов текста для использования в различных контекстах. В рамках DocLine создан новый XML-язык разметки DRL (Documentation Reuse Language) и модель процесса разработки документации. Программные средства DocLine созданы на основе Eclipse. Язык DRL является расширением языка DocBook – добавлены две новые конструкции, предназначенные для вариативного повторного использования: информационный элемент и каталог элементов.

Информационный элемент. Рассмотрим документацию семейства телефонных аппаратов с АОН (Автоматического Определения Номера). Она может включать в себя информационный элемент «Прием входящих звонков», содержащий следующий текст:

```
Когда Вы принимаете входящий звонок, телефонный аппарат
получает информацию CallerID и отображает ее на экране.        (1)
```

Телефонные аппараты могут выдавать информацию о вызывающем абоненте и иными способами, для которых пример (1) будет выглядеть иначе:

```
Когда Вы принимаете входящий звонок, телефонный аппарат
получает информацию CallerID и произносит ее.                  (2)
```

Для того чтобы создать элемент документации, объединяющий в себе оба варианта (1) и (2), на DRL может быть описан следующий информационный элемент:

```
<infelement id="CallerIdent">
Когда Вы принимаете входящий звонок, телефонный аппарат
получает информацию CallerID <nest id="DisplayOptions">
и отображает ее на экране</nest>.
</infelement>
```

Информационный элемент задается тэгом (<infelement>), а внутри этой конструкции описывается точка расширения (тэг <nest/>). После этого данный информационный элемент можно использовать в

разных контекстах с помощью конструкции «Ссылка на информационный элемент», которая задается тэгом <infelemref/>:

```
<infelemref infelemid="CallerIdent">
<replace-nest nestid="DisplayOptions">
и произносит ее</replace-nest>
</infelemref>
```

В ссылке на информационный элемент любая точка расширения может быть удалена, заменена или дополнена специальным содержимым без внесения изменений в исходный информационный элемент.

Каталог элементов. В документации для большинства программных продуктов встречаются описания сходных объектов, которые имеют разную форму, но близкое содержание. Например, одна и та же функция пользовательского интерфейса может быть представлена и как пункт главного меню, и как пункт всплывающего меню, и как кнопка панели инструментов. Если описывать такие объекты, не отслеживая связей, то при внесении изменений в их содержание потребуется вручную изменять все соответствующие элементы. Для решения этой проблемы в DRL введена конструкция «Каталог». Каталог содержит набор элементов, каждый из которых содержит набор шаблонов. Когда технический писатель включает в определенный контекст элемент каталога, он должен указать его идентификатор и необходимый шаблон представления в тексте. Частный случай каталога – словарь, который содержит набор терминов без шаблонов представления. Более детальное описание предназначенных для различных представлений каталогов элементов доступно в [5, 6].

Рефакторинг документации. Рефакторинг ПО – это процесс преобразования его исходного кода, не изменяющий его функциональность, но улучшающий его внутреннюю структуру [14, 15]. В [6] понятие рефакторинга было адаптировано и применено к процессу сопровождения XML-документации. В данном случае под рефакторингом понимается изменение внутренней структуры документа (конструкций разметки XML) с сохранением результирующего представления документа (например, PDF). В соответствии с данной идеей для DocLine был описан ряд операций рефакторинга [6], которые собраны в группы, как указано ниже.

1. Операции выделения общих используемых повторяющихся активов, в частности, для преобразования в DRL из DocBook или «плоского» текста.
2. Операции, помогающие настройке основных активов (расширяющие их настраиваемость).
3. Операции по переименованию различных элементов документации.

Обнаружение клонов в программном обеспечении и Clone Miner. Повторное использование программного кода часто происходит хаотически, методом cut & pasted. Подобные действия создают повторяющийся код (программные клоны, software clones) и могут привести к серьезным трудностям при сопровождении. Методы поиска клонов (software clone detection techniques) предназначены для поиска различных повторений в коде для последующего рефакторинга и организации повторного использования. Систематизированный обзор методов и инструментов обнаружения клонов доступен в [9]. В [16, 17] приводятся интересные обсуждения проблем клонирования кода и задачи обнаружения клонов, существует также значительное количество подходов и методов для поиска и классификации клонов (см., например, [18]) и пр.

Для нашего исследования мы выбрали инструмент Clone Miner [10], так как он обладает простым программным интерфейсом (командная строка), легок для интеграции и поддерживает русский язык (формат Unicode). Clone Miner ищет клоны в терминах лексических токенов: он преобразует полученный на входе текст в последовательность токенов, а затем при помощи алгоритмов поиска, основанных на основанных на суффиксных массивах, находит повторяющиеся фрагменты текста. Инструмент позволяет варьировать минимальную длину искомых клонов (в токенах). Применительно к текстовым документам, токен – это слово, отделенное от остальных слов разделителями `.', `(', `)', и т.д. Так, например, следующий фрагмент текста содержит два токена: ``FM registers".

Процесс

В этом разделе будет описан процесс поиска клонов, реализуемый в нашем методе.

Общее описание. Последовательность действий процесса представлена на рис. 1.

На входе имеется файл с документом, который нужно подвергнуть автоматизированному рефакторингу. С этим файлом выполняется определенная работа по его подготовке к процессу, затем запускается Clone Miner, который выдает список групп клонов и их координаты в тексте. По этому списку пользователь для любой из обнаруженных групп может выполнить автоматизированный рефакторинг. При рефакторинге все экземпляры клона заменяются ссылками на повторно используемое определение информационного элемента.

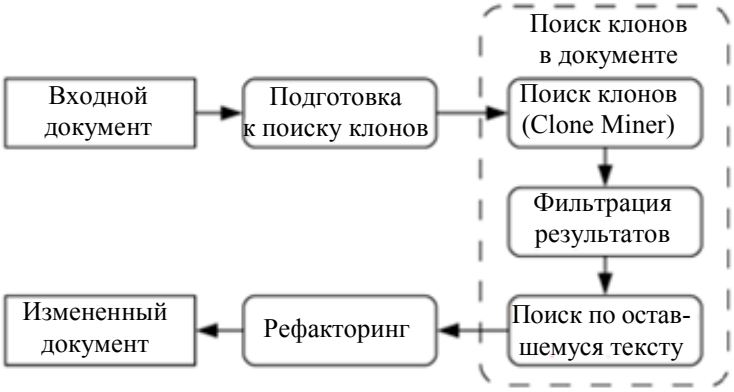


Рис. 1. Процесс рефакторинга документа на основе поиска клонов

Подготовка к поиску клонов. DocLine работает с файлами в формате DRL, а процедура поиска клонов применяется к информационному элементу. Но, поскольку поиск клонов целесообразно применять к «плоскому» тексту (файлы в формате ASCI/Unicode) или к DocBook-файлам, эти файлы нужно преобразовать в DRL, применив к ним операцию рефакторинга «Преобразование в DRL». В результате появляется новый информационный элемент, полностью содержащий исходный текст, и можно двигаться дальше.

Поиск клонов. Мы запускаем Clone Miner в режиме поиска в «плоском» тексте, поскольку основное его предназначение – поиск текстовых клонов в исходном коде программ, написанных на языках с существенно отличающимся от XML синтаксисом, в которых не встречаются такие специфические конструкции, как, например, именованные закрывающие тэги. Кроме того, во многих случаях нам требуется именно поиск в «плоском» тексте, так как нужные нам повторы могут содержаться внутри XML-конструкций. Таким образом, найденные клоны могут существенно нарушать структуру XML-документа. Пример (3) показывает фрагмент текста, в котором полужирным выделен найденный клон. Он включает открывающий тэг, но не включает закрывающего. Клоны делаются XML-корректными при выполнении рефакторинга, вместе с этим делается XML-корректным и то окружение документа, из которого они извлекаются.

```
<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>                                                    (3)
```

Фильтрация результатов. Наш алгоритм производит фильтрацию результатов работы Clone Miner следующим образом.

Шаг 1. Группа клонов отбрасывается, если содержит клоны менее чем из 5 отображаемых символов (например, «is a» содержит 3 отображаемых символа): как правило, такие клоны не имеют значимой для пользователя семантики, хотя и встречаются в большом количестве. При этом возможны потери значимой информации, например, некоторых аббревиатур, но в целом такая фильтрация позволяет заметно снизить количество бесполезных групп клонов. На текст длиной около 900 КБ на английском языке находится в среднем 3000 таких клонов, что значительно затрудняет дальнейшую работу.

Шаг 2. Clone Miner позволяет найденным группам клонов пересекаться, так как он лишь находит клоны, но не обладает информацией о том, для чего выполняется поиск и какие операции над найденными клонами будут производиться. Наш алгоритм исключает пересечения, оставляя лишь те группы, длина клонов в которых наибольшая, так как длинные клоны ожидаемо несут больше информации, чем короткие.

Шаг 3. Мы игнорируем группы, включающие клоны, состоящие исключительно из разметки XML и не включающие какого-либо текста, так как не ставим своей целью организацию повторного использования XML-разметки.

Шаг 4. Мы исключаем группы, состоящие из фраз «that is», «there is a» и др., так как они семантически несодержательны (подробнее в разделе «Апробация»). Для исключения подобных клонов мы разработали словарь и каждую группу проверяем на принадлежность данному словарю.

Поиск по оставшемуся тексту. После выполнения поиска клонов и фильтрации результатов (рис. 1) мы получаем набор клонов A1. Затем мы удаляем из документа все входящие в A1 клоны, вновь запускаем Clone Miner на оставшемся тексте и получаем набор A2. К A2 мы снова применяем шаги 1–4, получаем набор A3. Далее мы объединяем наборы A1 и A3, считая это конечным результатом. Такой повторный поиск может давать существенные результаты, в частности, из-за исключения пересекаю-

щихся групп. Эксперименты (см. раздел «Апробация») показали, что многократный поиск по остаткам не дает значимого улучшения результатов.

Такой повторный поиск может давать существенные результаты, в частности, из-за исключения пересекающихся групп. Эксперименты (см. раздел «Апробация») показали, что многократный поиск по остаткам не дает значимого улучшения результатов.

Рефакторинг. Для каждой группы клонов из полученного результирующего набора групп клонов пользователь может выполнить следующие операции из группы 1 (см. раздел «Рефакторинг документации»): выделение информационного элемента или выделение для включения в словарь. После выполнения любой из этих операций соответствующая группа клонов исключается из набора, а координаты клонов оставшихся групп пересчитываются с учетом произошедших при рефакторинге изменений в тексте документа.

Перед выполнением операций рефакторинга наш алгоритм преобразует выбранные клоны в корректные с точки зрения синтаксиса XML-фрагменты текста. Как было сказано ранее, Clone Miner «не знает» про XML, работая с документом как с «плоским» текстом. Однако DocLine может работать лишь с корректными конструкциями DocBook и DRL. Наш алгоритм восстанавливает балансировку открывающих и закрывающих тегов в клоне и в окружающем тексте, из которого извлекается клон.

Программный инструмент. Представленный процесс поиска клонов реализован в виде программных средств, интегрированных с DocLine и Eclipse. Созданный инструмент позволяет обнаруживать клоны в документах, редактируемых при помощи DocLine: поиск клонов включен как одна из операций рефакторинга в редакторе DRL. Инструмент также позволяет осуществлять навигацию по группам клонов и перемещение по исходному тексту, чтобы наблюдать, в каких контекстах встречается клон. Наконец, есть возможность извлекать все клоны выбранной группы как подлежащие повторному использованию элементы.

На рис. 2 показан процесс навигации по найденным в документе группам клонов и клонам и контекстное меню, предлагающее возможность выделить группу в информационный элемент или в поместить ее в словарь.

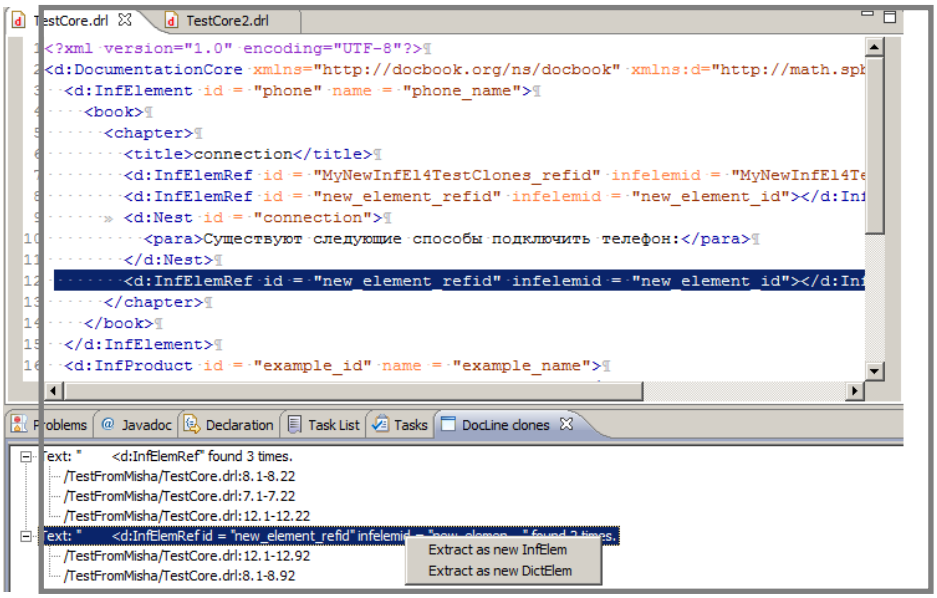
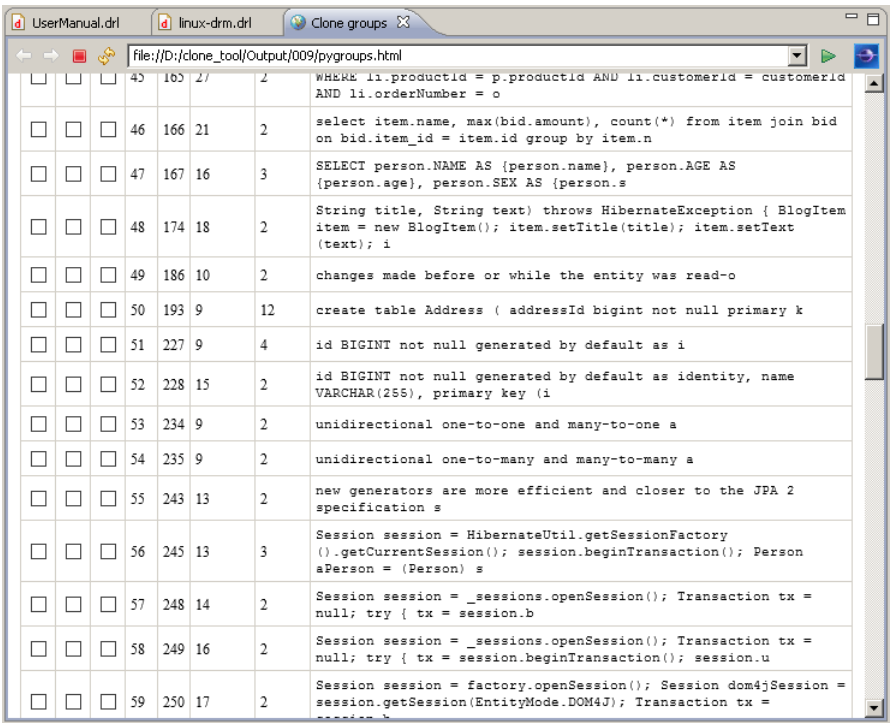


Рис. 2. Навигация по найденным клонам

Кроме инструмента редактирования и навигации, был реализован ряд вспомогательных инструментов, позволяющих собирать статистику по количеству и характеру найденных клонов и выдавать отчеты в табличном виде. Эти вспомогательные инструменты генерируют отчеты в формате HTML, содержание отчетов легко импортируется в офисные пакеты для последующей обработки данных средствами электронных таблиц. Вспомогательные инструменты реализованы на языке Python и интегрированы в среду DocLine. Пример отчета показан на рис. 3.

Апробация

Для апробации предложенного подхода была проведено три серии экспериментов. Первая серия заключалась в запуске инструмента на небольших документах, специально созданных нами для этой цели: в этих документах было точно известно количество клонов. Инструмент обнаружил все клоны.



			45	165	27	2	WHERE li.productId = p.productId AND li.customerId = customerId AND li.orderNumber = o
			46	166	21	2	select item.name, max(bid.amount), count(*) from item join bid on bid.item_id = item.id group by item.n
			47	167	16	3	SELECT person.NAME AS {person.name}, person.AGE AS {person.age}, person.SEX AS {person.s
			48	174	18	2	String title, String text) throws HibernateException { BlogItem item = new BlogItem(); item.setTitle(title); item.setText (text); i
			49	186	10	2	changes made before or while the entity was read-o
			50	193	9	12	create table Address (addressId bigint not null primary k
			51	227	9	4	id BIGINT not null generated by default as i
			52	228	15	2	id BIGINT not null generated by default as identity, name VARCHAR(255), primary key (i
			53	234	9	2	unidirectional one-to-one and many-to-one a
			54	235	9	2	unidirectional one-to-many and many-to-many a
			55	243	13	2	new generators are more efficient and closer to the JPA 2 specification s
			56	245	13	3	Session session = HibernateUtil.getSessionFactory ().getCurrentSession(); session.beginTransaction(); Person aPerson = (Person) s
			57	248	14	2	Session session = _sessions.openSession(); Transaction tx = null; try { tx = session.b
			58	249	16	2	Session session = _sessions.openSession(); Transaction tx = null; try { tx = session.beginTransaction(); session.u
			59	250	17	2	Session session = factory.openSession(); Session dom4jSession = session.getSession(EntityMode.DOM4J); Transaction tx =

Рис. 3. Отчет по найденным группам клонов

Вторая серия экспериментов была посвящена эффективности поиска клонов по оставшемуся тексту (раздел «Поиск по оставшемуся тексту»). Для этого мы использовали фрагменты открытой DocBook-документации для 15 Java-проектов с открытым исходным кодом (15 документов в среднем по 1300 строк). Поиск по оставшемуся тексту увеличил количество групп клонов на 5–10%, однако на 1,5% уменьшил длину клонов. По итогам тестирования мы заключили, что данный прием позволяет, в основном, находить короткие (по 2–5 токенов) клоны и может быть полезен для поиска в документации повторяющихся терминов. Дальнейшее повторение этого шага не дало значимых результатов, так как маленькие группы клонов редко пересекаются, а поиск по оставшемуся тексту восстанавливает, в основном, те клоны, которые были отброшены при фильтрации пересечений.

Третья серия экспериментов была проведена для документации ядра операционной системы Linux [12] (Linux Kernel Documentation, далее – LKD). LKD со-стоит из 29 документов общим объемом 870 КБ (25000 строк), документация написана в формате DocBook. Статистика приведена в табл. 1.

Минимальная длина клона	1	2	3	5	6	7	8
Количество найденных групп клонов	3564	3543	2327	429	231	150	108

Таблица 1. Количество найденных клонов в зависимости от минимальной длины клона

Эксперименты показали, что фильтрация (см. раздел «Поиск по оставшемуся тексту») существенно снижает общее количество клонов, отбрасывая, в основном, малосодержательные клоны длиной 1 и 2 токена. В целом, с увеличением минимальной длины клона количество групп клонов значительно падает, как показано на рис. 4 (данные графика «групп после фильтрации» на рис. 3 соответствуют строке LKD в табл. 1).

Анализируя результаты экспериментов, мы выделили 5 смысловых групп клонов.

1. **Термины** (аббревиатуры, отдельные слова или словосочетания из 2 слов), на основе которых можно составить глоссарий. 58 из найденных в LKD групп клонов были классифицированы как термины (30 из одного токена и 28 из двух). В среднем они повторялись по 11 раз.
2. **Гиперссылки** (ссылки на веб-страницы). Полезно уметь выделять из документации все гиперссылки и собирать их в единый список ссылок. В LKD было найдено 8 групп клонов, содержащих гиперссылки. В ходе второго эксперимента было найдено больше гиперссылок.

3. **Лицензионные соглашения**. Каждый документ LKD содержит лицензионное соглашение, основанное на фрагменте GPL v 2 [19], но варьирующееся. В данном случае важно понимать, какие фрагменты лицензии идентичны, а какие отличаются. Для читателя может оказаться полезной возможность визуализировать эту информацию: так он сможет не читать одинаковые части текста, а сосредоточиться на отличиях. Мы обнаружили 6 групп в среднем по 4,36 токенов и по 2 клона в группе.
4. **Описание функциональности и требования**. Например, «When debugger detects an error, than it calls…». Здесь мы используем адаптивное повторное использование, считая одними и теми же группы клонов, члены которых отличаются мелкими деталями (числами, обозначающими размеры буферов, названиями процедур и т.д.), но данное объединение групп мы проделали вручную. В итоге мы получили 49 групп, в среднем состоящих из 3,27 клонов, включающих в среднем по 3,29 токенов.
5. **Примеры исходного кода.** LKD содержит множество листингов фрагментов исходного кода. Наш инструмент нашел в них 330 групп клонов со средними показателями 5,2 токенов на клон и 2,69 клонов в группе.

Результаты экспериментов по смысловым группам клонов суммируются в табл. 2.

Рис. 4. Количество найденных в LKD групп клонов в зависимости от минимальной длины клона

Показатели	Смысловые группы клонов					
	Термины	Гиперссылки	Лицензионные соглашения	Описание функциональности и требования	Примеры исходного кода	Всего
Количество групп	58	32	6	49	330	451
Токенов/клон	1,48	1*	4,36	3,29	5,2	4,43
Клонов/группу	11	1,25	2	3,27	2,69	3,56

*Каждая гиперссылка состоит из одного токена

Таблица 2. Смысловые типы клонов

В итоге на примере LKD были получены следующие результаты: 451 содержательная группа клонов, где длина клона в среднем равна 4,43 токенов, а количество клонов в группе в среднем составляет 3,65 штук.

Не все найденные клоны удалось распределить по этим смысловым группам. Так, например, фрагмент «This document describes…» в LKD встречается часто, но он лишен семантики (не несет самостоятельного смысла). Для таких случаев полное повторное использование лишь усложнит и удлинит XML-представление документа, в то время как семантическое повторное использование позволит интегрировать наш подход с различными построения соответствия между фрагментами документов и прочими артефактами – кодом, требованиями, сущностями моделей и т.д. (см., например, [20]). В таком случае повторное использование может повысить качество построения этих соответствий, а адаптивное повторное использование позволит фиксировать более «тонкие» связи. Наш подход также может быть использован для управления изменениями при разработке линеек программных продуктов [21].

Заключение

Эксперименты показали удовлетворительное качество работы инструмента. Предложенный подход может оказаться полезным в средах работы с документацией семейств программных продуктов (напри-

мер, [22]) для выделения подлежащих повторному использованию фрагментов документации разных продуктов семейства и создания повторно используемой структуры документации. Эта структура упрощает процесс сопровождения документации и имеет значение в тех случаях, когда сопровождение (элемента семейства программных продуктов и (или) его документации) играет значительную роль. Также могут быть установлены связи между предусматривающей повторное использование структурой документации и моделью функциональных возможностей семейства программных продуктов [23, 24], что полезно, так как позволит, например, автоматически удалить из документации описания функциональности после удаления этой самой функциональности из программного продукта. Из не связанных с программной инженерией областей подход может быть использован, например, при моделировании архитектур предприятий (Enterprise Architecture Management) [25], а также при работе с онтологиями [26, 27]: в данных моделях возможны непредусмотренные повторения с учетом того, что они включают в себя большие объемы информации, значительная часть которой (документы, комментарии, длинные имена сущностей моделей и т.д.) не структурирована. При этом данные модели хранятся в XML-формате, что позволяет применить к ним наш подход с целью выявления повторов и повышения качества моделей.

Эксперименты показывают, что даже после фильтрации у нас остается много не несущих смысла клонов. Некоторые из них, предположительно, достаточно легко вычленить с помощью более мощной фильтрации, но часть все равно придется искать вручную. Повышение точности алгоритма – основной предмет дальнейшей работы. В частности, мы планируем в будущем решить вопрос с поиском терминов короче 5 печатных символов.

Также стало ясно, что инструмент должен обладать большими удобствами для поиска клонов. Требуется также автоматизированная поддержка адаптивного повторного использования. Кроме того, несмотря на многообещающие первые результаты, нам не кажется, что мы предложили исчерпывающую семантическую классификацию для повторного использования в документации. Данная тема подлежит дальнейшим исследованию и разработке.

References

1. Holmes R., Walker R.J. Systematizing pragmatic software reuse. *ACM Transactions on Software Engineering and Methodology*, 2013, vol. 21, no. 4, art. no. 20. doi: 10.1145/2377656.2377657
2. Czarnecki K. Software reuse and evolution with generative techniques. *Proc. of 22nd IEEE/ACM International Conference on Automated Software Engineering*. Atlanta, USA, 2007, p. 575. doi: 10.1145/1321631.1321750
3. Jarzabek S., Bassett P., Zhang H., Zhang W. XVCL: XML-based variant configuration language. *Proc. of International Conference on Software Engineering*. Portland, USA, 2003, pp. 810−811.
4. Bassett P. The theory and practice of adaptive reuse. *Sigsoft Software Engineering Notes*, 1997, vol. 22, no. 3, pp. 2−9. doi: 10.1145/258368.258371
5. Koznov D.V., RomanovskyK.Yu.DocLine: A method for software product lines documentation development. *Programming and Computer Software*, 2008, vol. 34, no. 4, pp. 216–224. doi: 10.1134/S0361768808040051
6. Romanovsky K., Koznov D., Minchin L. Refactoring the documentation of software product lines. *Lecture Notes in Computer Science*, 2011, vol. 4980 LNCS, pp. 158−170. doi: 10.1007/978-3-642-22386-0_12
7. Koznov D.V., Romanovsky K.Yu. Avtomatizirovannyi refactoring dokumentatsii semeistv programmnykh produktov [Automated refactoring of documentation of the software family]. *Sistemnoe Programmirovanie*, 2009, vol. 4. pp. 128–150.
8. Shutak A.V., Smirnov M.N., Smazhevskii M.A., Koznov D.V. Poisk klonov pri refaktoringe tekhnicheskoi dokumentatsii [Search of clones with refactoring of technical documentation]. *Komp'yuternye Instrumenty v Obrazovanii*, 2012, no. 4, pp. 30–40.
9. Rattan D., Bhatia R.K., Singh M. Software clone detection: a systematic review. *Information and Software Technology*, 2013, vol. 55, no. 7, pp. 1165−1199. doi: 10.1016/j.infsof.2013.01.008
10. Basit H.A., Smyth W.F., Puglisi S.J., Turpin A., Jarzabek S. Efficient token based clone detection with flexible tokenization. *Proc. 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2007*. Dubrovnik, Croatia, 2007, pp. 513−516. doi: 10.1145/1287624.1287698
11. Walsh N., Muellner L. *DocBook: The Definitive Guide*. O'Reilly Media, 1999, 644 p.
12. *Linux Kernel Documentation*. Available at: github.com/torvalds/linux/tree/master/Documentation/DocBook (accessed 01.06.2014).
13. Wright C.H.G. Technical writing tools for engineers and scientist. *Computing in Science and Engineering*, 2010, vol. 12, no. 5, pp. 98–103. doi: 10.1109/MCSE.2010.115
14. Fowler M., Beck K., Brant J., Opdyke W., Roberts D. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1999, 464p.
15. Lugovskoi N.L. Podkhod dlya provedeniya refaktoringa «Vydelenie funktsii» v instrumente Klocwork Insight [Approach for refactoring «Select Features» tool in Klocwork Insight]. *Trudy Instituta Sistemnogo Programmirovaniya RAN*, 2012, vol. 23, pp. 107–132.
16. Itsykson V.M., Moiseev M.Yu., Akhin M.Kh., Zakharov A.V., Tsesko V.A. Algoritmy analiza ukazatelei dlya obnaruzheniya defektov v iskhodnom kode programm [Pointer analysis algorithms for detecting defects in the source code of programs]. *Sistemnoe Programmirovanie*, 2009, vol. 5, pp. 5–30.

17. Akhin M., Itsykson V. Obnaruzhenie klonov iskhodnogo koda: teoriya i praktika [Detection of clones at source code: theory and practice]. *Sistemnoe Programmirovanie*, 2010, vol. 5, no. 1, pp. 145–163.
18. Zetser N.G. Poisk povtoryayushchikhsya fragmentov iskhodnogo koda pri avtomaticheskom refaktoringe [Automatic clone detection for refactoring]. *Trudy Instituta Sistemnogo Programmirovaniya RAN*, 2013, vol. 25, pp. 39–50.
19. *GNU General Public License v2.0*. Available at: www.gnu.org/licenses/gpl-2.0.html (accessed 01.06.2014).
20. Abadi A., Nisenson M., Simionovici Y. A traceability technique for specifications. *Proc. 16th IEEE International Conference on Program Comprehension*, 2008, pp. 103−112. doi: 10.1109/ICPC.2008.30
21. Krueger C.W. Variation management for software production lines. *Proc. 2nd Software Product Line Conference*. San Diego, USA, 2002, pp. 37−48.
22. Trung H.D., Jarzabek S. *DME: Documentation Management Environment for Software Product Lines – Tool Demo Proposal*. Available at: www.comp.nus.edu.sg/~stan/DME.pdf (accessed 01.06.2014).
23. Lee J., Muthig D. Feature-oriented variability management in product line engineering. *Communications of the ACM*, 2006, vol. 49, no. 12, pp. 55−59.
24. Mei H., Zhang W., Gu F. A feature oriented approach to modeling and reusing requirements of software product lines. *Proc. 27th Annual International Conference on Computer Software and Applications, COMPSAC'03*. IEEE Computer Society, Washington, USA, 2003, pp. 250–256.
25. Grigoriev L., Kudryavtsev D. ORG-master: combining classifications, matrices and diagrams in the enterprise architecture modeling tool. *Proc. 4th Conference on Knowledge Engineering and Semantic Web, Communications in Computer and Information Science, CCIS*. St. Petersburg, 2013, pp. 250−258.
26. Gavrilova T.A., Kudryavtsev D.V., Gorovoy V.A. Modeli i metody formirovaniya ontologii [Models and methods of ontologies forming]. *Nauchno-Tekhnicheskie Vedomosti SPbSPU*, 2006, no. 46,pp. 21–28.
27. Gavrilova T.A. Ob odnom podkhode k ontologicheskomu inzhiniringu [An approach to the ontological engineering]. *Novosti Iskusstvennogo Intellekta*, 2005, no. 3, p. 25–31.

Луцив Дмитрий Вадимович – старший преподаватель, Санкт-Петербургский государственный университет, Санкт-Петербург, Россия; ведущий инженер-программист, ЗАО «Ланит-Терком», dluciv@math.spbu. ru

Кознов Дмитрий Владимирович – кандидат физико-математических наук, доцент, доцент, Санкт-Петербургский государственный университет, Санкт-Петербург, Россия; dkoznov@yandex.ru

Басит Хамид Абдул – доцент, Университет менеджмента Лахора, Пакистан, hamidb@lums.edu.pk

Ли Оу Енг – преподаватель, Национальный университет Сингапура, issoel@nus.edu.sg

Смирнов Михаил Николаевич – старший преподаватель, Санкт-Петербургский государственный университет, Санкт-Петербург, Россия; директор департамента, ЗАО «Ланит-Терком», m.n.smirnov@math.spbu.ru

Романовский Константин Юрьевич – старший преподаватель, Санкт-Петербургский государственный университет, Санкт-Петербург, Россия; ведущий инженер-программист, ЗАО «Ланит-Терком», kromanovsky@gmail.com

Dmitry V. Lutsiv – senior lecturer, Saint Petersburg State University; leading engineer-programmer, “Lanit-Tercom” Ltd., Saint Petersburg, Russia, dluciv@math.spbu.ru

Dmitry V. Koznov – PhD, Associate professor, Associate professor, Saint Petersburg State University, Saint Petersburg, Russia, dkoznov@yandex.ru

Hamid Abdul Basit – PhD, Associate professor, Lahore University of Management Sciences, Lakhora, Pakistan, hamidb@lums.edu.pk

Ouh Eng Lieh – MS; Associate, National University of Singapore, Singapore, issoel@nus.edu.sg

Mikhail N. Smirnov – senior lecturer, Saint Petersburg State University; Department Director, “Lanit-Tercom” Ltd., Saint Petersburg, Russia, m.n.smirnov@math.spbu.ru

Konstantin Yu. Romanovsky – senior lecturer, Saint Petersburg State University, Saint Petersburg, Russia, kromanovsky@gmail.com

Принято к печати 22.05.14
Accepted 22.05.14