Московский Государственный Университет имени М.В.Ломоносова
Факультет Вычислительной Математики и Кибернетики
Кафедра Системного Программирования

Выпускная квалификационная работа

Методы и инструменты разметки требований в многоверсионных текстовых документах

Выполнил:
студент группы 428
Зинченко Дмитрий Александрович

Научный руководитель:
к.ф-м.н.
Хорошилов Алексей Владимирович

Москва, 2015

Аннотация

Методы и инструменты разметки требований
в многоверсионных текстовых документах

Зинченко Дмитрий Александрович

В данной работе исследуется и проектируется алгоритм поиска соответствий фрагментов текста, выделенных в старой версии документа, фрагментам новой версии того же документа. Проводится оценка работы алгоритма на тестовой базе и сравнение с существующим алгоритмом, использующимся в системе управления требованиями Requality

Abstract

Methods and instruments of requirement marking up
in multiversion documents

Zinchenko Dmitrii

In this work, algorithm of matching text fragments from the old version of the document to the fragments of the new version of the same document is designed. The algorithm is tested on document base and compared to existing method used in requirement management system Requality

Содержание

1 Введение

Выявление и отслеживание требований является важной частью проектирования программных систем, при этом отслеживание выполнения требований остается важным на протяжении всего жизненного цикла системы. Подробнее процессы разработки программного обеспечения описываются в [1]. На рисунке 1 в качестве примера показан цикл итеративной разработки системы.

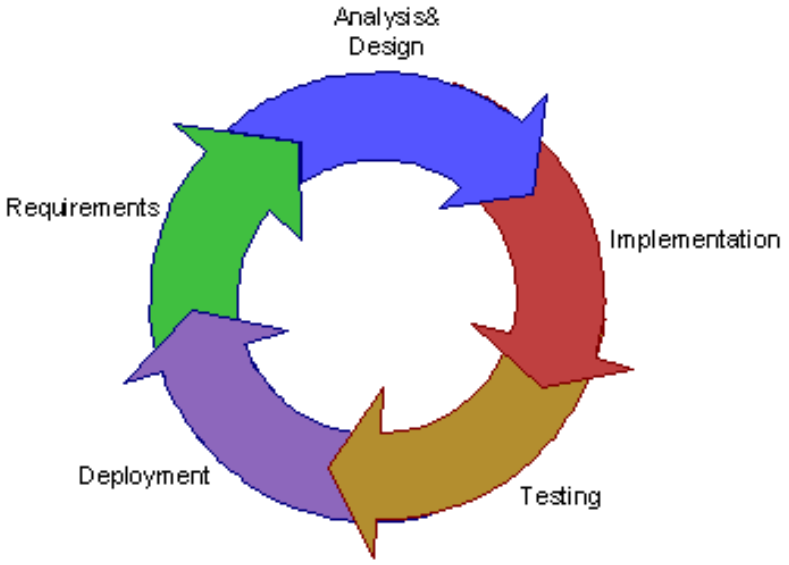


Рис. 1: *Диаграмма цикла итеративной разработки программной системы*

В дальнейшем под понятием требование мы будем понимать свойства, которыми должна обладать разрабатываемая программная система для решения определенной задачи[2], [3]. Документ, составляемый на начальных этапах проектирования системы, называется спецификацией требований.

В процессе разработки ПО важно отслеживать, удовлетворяет ли текущая версия системы требованиям, описанным в спецификации, проверяют ли написанные тесты все требования, которым должен удовлетворять продукт, и уточнять требования на каждой итерации разработки. Для этих целей служат системы управления требованиями, одной из которых является Requality [4], разрабатываемая в Институте Системного Программирования РАН.

Спецификацию требований, как документ, можно рассматривать в качестве одного требования. Однако такой подход обладает существенным недостатком - проверка выполнимости такого требования является очень сложной задачей. Альтернативный подход заключается в выделении утверждений в тексте спецификации, каждое из которых соответствует атомарному (неделимому) требованию к программной системе. При этом логически атомарные требования можно объединить в группы, например, соответствующие требованиям к модулям системы. Каждой такой

группе может соответствовать некоторое структурное требование, не обязательно отраженное в тексте спецификации требований.

Таким образом, совокупность требований к программной системе представляет собой некоторую иерархическую структуру - каталог требований. Некоторые системы управления требованиями (и Requality — одна из них) позволяют хранить связанные с требованиями версии спецификаций и отслеживать связь между фрагментами текста спецификации и требованиями. Таким образом, каждому атомарному требованию каталога соответствует один или несколько выделенных специальным образом участков текста спецификации требований. При обновлении спецификации возникает потребность в обновлении каталога требований под новую версию документа и поиске в новом тексте фрагментов, соответствующих требованиям. Этот функционал становится особенно полезным при работе со стандартами, новые версии которых разрабатываются независимо от команд, их использующих.

Поскольку часто бывает так, что новая версия спецификации требований отличается от предыдущей незначительно, и текст большей части требований в двух версиях документа совпадает, задача переноса выделения текста тех требований, которые остались без изменений, в новую версию документа, становится актуальной. Автоматизация решения этой задачи упрощает и ускоряет выделение требований в новой версии спецификации требований. Исследованию задачи переноса требований между версиями документа и посвящена эта работа.

1.1 Используемая терминология

Для проектирования разрабатываемой системы и постановки задачи введем следующие понятия:

- **Документ** – текстовый файл, являющийся синтаксически корректным XHTML документом.
- **Теги требования** (в рамках системы Requality)– пара тегов

  *<span class=”requality_text id_ ***”> *,

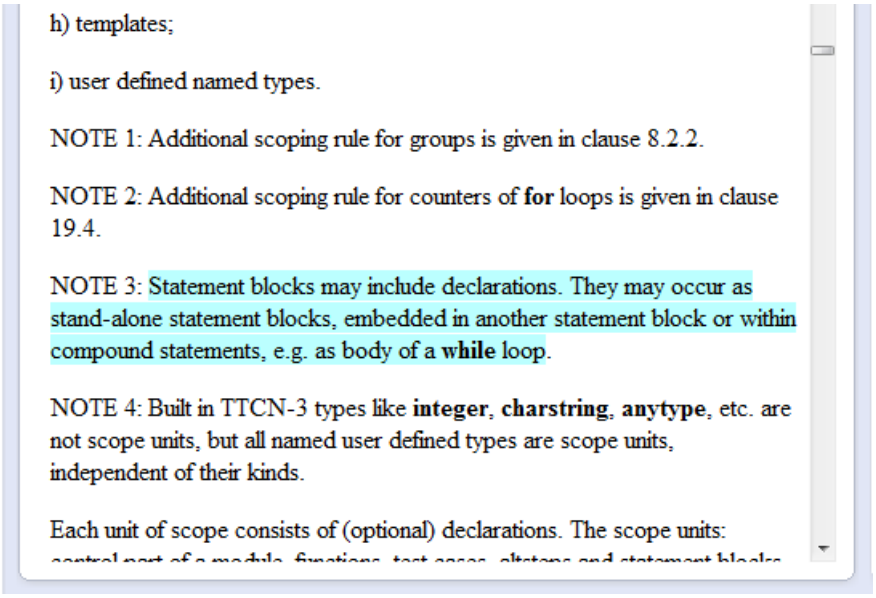
  где *** - идентификационный номер требования. Помимо этого, в Requality после открывающего тега требования и перед текстом внутри него, может находиться пара тегов

  **,

  позволяющая интерфейсу системы определять, на каком фрагменте текста центрировать редактор документов при выборе требования. Под **разметкой требований** будем понимать совокупность тегов требований в документе.
- **Фрагмент требования** – участок документа, ограниченный тегами требования. Фрагмент не может содержать XHTML теги внутри себя.

- **Объединение фрагментов требования** – несколько фрагментов одного и того же требования, идущих подряд.

  Пример выделенных в инструменте Requality фрагментов требований и xhtml тегов, ограничивающих их, продемонстрирован на рисунках 2 и 3 соответственно.


h) templates;

i) user defined named types.

NOTE 1: Additional scoping rule for groups is given in clause 8.2.2.

NOTE 2: Additional scoping rule for counters of **for** loops is given in clause 19.4.

NOTE 3: Statement blocks may include declarations. They may occur as stand-alone statement blocks, embedded in another statement block or within compound statements, e.g. as body of a **while** loop.

NOTE 4: Built in TTCN-3 types like **integer**, **charstring**, **anytype**, etc. are not scope units, but all named user defined types are scope units, independent of their kinds.

Each unit of scope consists of (optional) declarations. The scope units:

Рис. 2: *Пример объединения фрагментов требований в системе Requality*

Рис. 3: *Структура xhtml тегов для выделенных фрагментов*

- **Требование** (в рамках системы Requality)– логическое объединение фрагментов текста, ограниченных тегами требований с одинаковыми идентификационными номерами. **Идентификационный номер требования** – номер, использующийся в тегах требования, соответствующих ему. Идентификационный номер требования уникален, двух разных требований с одним номером не существует.

- **Исходный документ** – документ, представляющий рабочую версиию некоторой спецификации, может содержать разметку требований. Неформально - документ, разметка требований которого определена, и которую необходимо перенести.

- **Конечный документ** – документ, для которого разметка требований не определена. В общем случае конечный документ может быть никак не связан с исходным, однако тогда задача переноса разметки требований между версиями документа становится бессмысленной. Конечный документ представляет собой новую версию спецификации, в которую требуется перенести разметку требований.

- Часть конечного документа считается **полностью соответствующей** объединению фрагментов текста исходного документа, если содержимое объединения фрагментов текста исходного документа совпадает с частью конечного документа с точностью до незначащих символов и тегов XHTML. В противном случае часть конечного документа **не соответствует** содержимому объединения фрагментов исходного документа.

- Под **переносом объединения фрагментов требования** мы будем понимать перенос тегов требования, ограничивающих все фрагменты этого объединения, из исходного документа в конечный.

- Под **переносом требования** мы будем понимать перенос всех его объединений фрагментов, для которых в конечном документе было найдено полное соответствие.

- **Незначащие символы** - пробелы, переносы строк и знаки препинания.

- **Структурная разметка документа** – xhtml теги заголовков и абзацев (теги *<h1>* — *<h6>*, *<p>*, *<blockquote>*)

2 Постановка задачи

В рамках выпускной квалификационной работы необходимо разработать библиотеку, позволяющую осуществлять перенос требований из исходного документа в конечный.

Стоит отметить, что в поставленной задаче считается, что оба документа имеют схожую структурную разметку и названия подразделов, в которых находятся фрагменты требований. Перенос объединений фрагментов требований, для которых не найдено точного соответствия, выходит за рамки данной работы, хотя и может быть осуществлен путем построения функции схожести фрагментов текстов.

3 Проектирование системы

3.1 Функции, предоставляемые библиотекой

Для выполнения различных сценариев работы с документами и требованиями библиотека должна предоставлять следующие функции:

- **Получение объектной модели документа**

  Согласно [5], [6], **Объектная модель документа** (Document Object Model – DOM) – стандарт, предложенный веб-консорциумом, и регламентирующий способ представления содержимого документа (в частности – xhtml документа) в виде набора объектов. DOM позволяет представить любой документ в виде иерархической структуры (дерева) узлов, каждый из которых соответствует элементу, атрибуту, текстовому, графическому или любому другому объекту.

  Одним из инструментов, позволяющих извлекать DOM модель документа из текстового файла, является библиотека JDOM (в данной работе используется версия 2.0.6).

  По документу осуществляется получение структуры данных (дерева), соответствующего объектной модели документа. Схематичный пример полученного из документа DOM дерева изображен на рисунке 4.

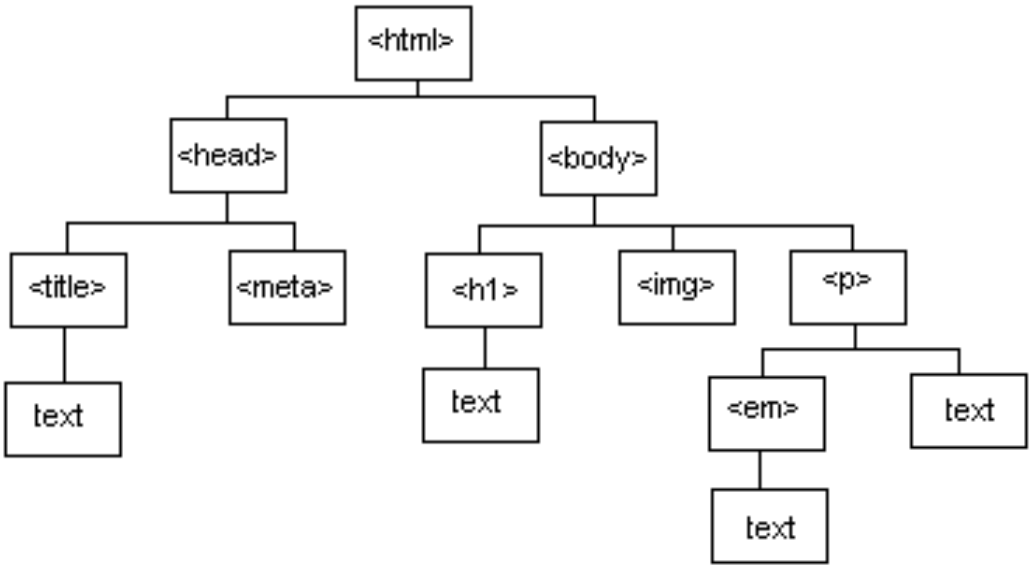


Рис. 4: *Пример DOM модели документа*

- **Выделение структурной разметки документа из его DOM дерева**

  По DOM модели документа осуществляется построение дерева, отражающего его структурную разметку.

- **Получение списка требований по дереву документа**

  Выделение фрагментов требований из дерева с дальнейшей группировкой по id.

  Из дерева структурной разметки документа извлекается список требований(объектов), каждое требование содержит идентификационный номер и список фрагментов текста, содержащихся в нем.

- **Синхронизация по фрагменту требования**

  По фрагменту требования находится вершина дерева разметки исходного документа, в тексте раздела которой содержится данный фрагмент.

- **Получение пути в дереве разметки от корня до элемента**

  По дереву разметки документа осуществляется поиск последовательности вершин – пути в дереве от корня до данной вершины.

- **Поиск элемента, соответствующего данному, в дереве другого документа**

  По элементу дерева разметки одного документа осуществляется поиск элемента разметки другого документа такого, что пути из корней деревьев до элементов имеют одинаковые типы вершин и идентичный (с точностью до незначащих символов) текст соответствующих заголовков.

- **Перенос объединения фрагментов требования**

  В конечном документе осуществляется поиск участка текста, соответствующего объединению фрагментов требования, и осуществляется перенос тегов фрагментов в найденные позиции (начало и конец найденного участка текста).

- **Перенос требования**

  Попытка переноса всех фрагментов требования.

3.2 Использование библиотеки JDOM

Важным этапом работы любого алгоритма, решающего поставленную задачу, является получение и хранение содержимого исходного и конечного документов. Одним из инструментов, позволяющих это сделать, является библиотека JDOM, используя которую можно получить DOM модель по XML документу. Также JDOM позволяет эффективно получить все фрагменты требований из исходного документа.

Помимо этого, JDOM предоставляет средства для редактирования XML документов, что является лучшим способом добавить разметку в конечную версию документа после осуществления переноса фрагментов требований.

В статье [7] описываются возможности и приводятся примеры использования JDOM.

3.3 Обзор решения, применяемого в Requality

3.3.1 Основная идея

Решение проблемы переноса разметки требований, использующееся на данный момент в Requality, основано на сравнении исходного и конечного документа без XHTML разметки, как строк текста (plain text). Для сравнения используется библиотека google-diff, в качестве результата сравнения двух текстов возвращающая список объектов Diff, каждый из которых содержит описание операции, которую нужно выполнить с текстом исходного документа.

Статья [8] описывает ключевой алгоритм, использующийся в google-diff.

Операции имеют формат *(<Тип операции>, <Текст>)*, где *Тип операции* - одна из трех команд "EQUAL" "DELETE" "INSERT" а *Текст* - часть текста исходного документа, над которой операцию необходимо совершить. Гарантируется, что при последовательном выполнении операций из списка из текста исходного документа получится текст конечного. Пример результата работы Diff алгоритма изображен на рисунке 5.

Исходный текст:	Конечный текст:
alpha 1 betta 2 gamma 3	alpha 1 gamma 3 betta 1

(EQUAL, "alpha 1"),
(DELETE, "betta 2"),
(EQUAL, "gamma 3"),
(INSERT, "betta 1")

Рис. 5: *Пример результата работы diff алгоритма*

3.3.2 Алгоритм

Исходный документ преобразуется в DOM дерево, и из него извлекаются все выделенные фрагменты требований, которые нужно перенести. Затем осуществляется преобразование исходного

и конечного документа в текстовые данные без XHTML разметки, при этом позиции в исходном тексте извлеченных ранее фрагментов требований запоминаются. Два полученных текста подаются на вход Diff алгоритму, который возвращает список операций, описывающих разницу между исходным и конечным документом.

После окончания работы Diff алгоритма для каждого фрагмента требования из исходного текста определяется, находился ли он в блоке EQUAL результата работа Diff, и если да, то осуществляется перенос соответствующих тегов требования в DOM дерево конечного документа. В противном случае считается, что фрагмент перенести не удалось. Обновленный конечный документ восстанавливается по измененному DOM дереву.

Таким образом, алгоритм можно представить в виде диаграммы, изображенной на рисунке 6.

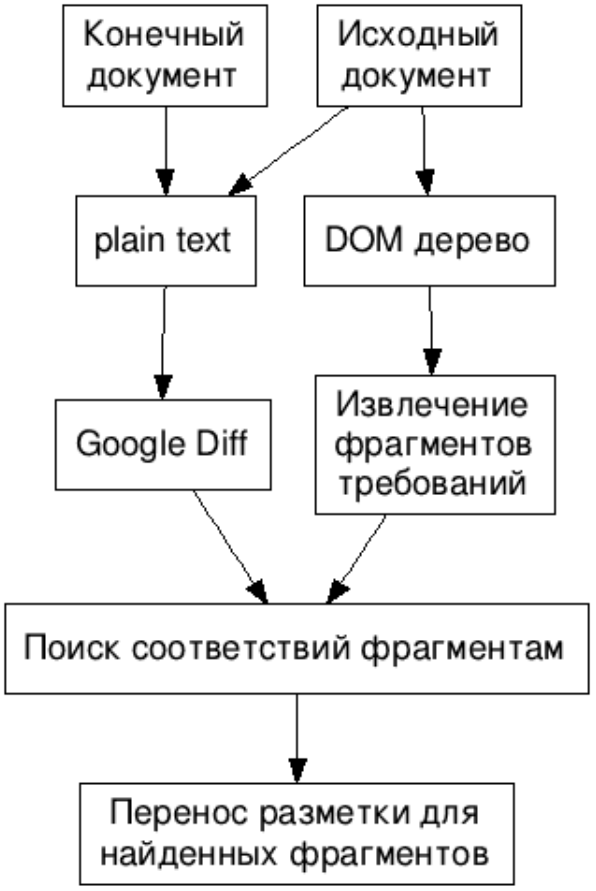


Рис. 6: *Диаграмма алгоритма переноса разметки требований системы Requality*

3.3.3 Характеристики алгоритма, реализованного в Requality

Важной особенностью такого подхода к решению задачи является независимость получения результата от разницы между разметкой исходного и конечного документов. Если в новой версии

документа была использована разметка, отличная от старого, то это не повлияет на эффективность работы алгоритма, поскольку разметка документов используется только при поиске фрагментов требований. С другой стороны, решение, основанное на использовании diff-алгоритма, перестает работать в случае перестановки разделов в конечном документе, и может работать существенно хуже в случае добавления большого количества новых разделов и текста.

Помимо этого, в текущей реализации этого алгоритма есть фрагменты, соответствие которым находится, однако перенос не осуществляется в связи с различными трудностями работы с DOM моделью документа.

3.4 Предлагаемое решение

Основной идеей решения поставленной задачи, которое описывается в данной работе, является локализация места конечного документа, в котором осуществляется поиск соответствия фрагменту, выделенному в исходном документе. В качестве места, в котором нужно осуществлять поиск, в конечном документе выбирается подраздел, соответствующий (в дальнейшем будет описано, как) подразделу исходного документа, содержащего искомый фрагмент требования.

После проведения этой операции размер текста, в котором нужно найти соответствие фрагменту требования, считается достаточно маленьким для применения алгоритма прямого поиска. В том случае, если совпадение с текстом фрагмента было найдено, осуществляется перенос разметки в соответствующее место конечного документа.

3.5 Внутреннее представление документа

Однако DOM дерево, полученное из исходного и конечного документов, не удовлетворяет полностью потребностям такого решения - текст раздела в таком дереве не привязан к заголовку, а заголовки, являющиеся вложенными в документе, в DOM дереве находятся на одном уровне иерархии. Таким образом, DOM модель не отражает структурную разметку документа.

Поэтому для удобства работы с иерархией разделов документов было решено осуществлять построение на основе DOM моделей документов деревья, позволяющие проще отслеживать вложенность разделов и связь текстов разделов с их заголовками.

Пример DOM дерева документа, и построенного по нему дерева структурной разметки можно наблюдать на рисунках 7 и 8 соответственно:

Основываясь на введенном новом представлении документа, определим следующие понятия:

Путь до раздела документа – список вершин в пути в дереве разметки документа до вершины, соответствующей разделу.

Путь до фрагмента требования – путь до вершины дерева разметки максимальной глубины, соответствующей разделу, в котором содержится этот фрагмент.

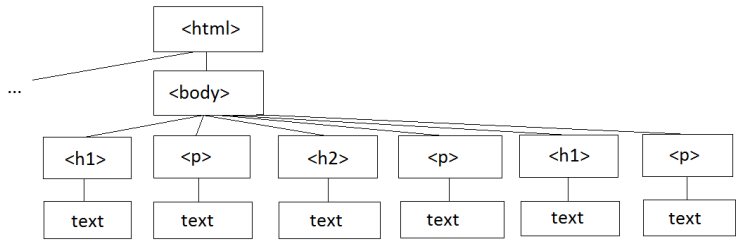


Рис. 7: *DOM дерево документа.*

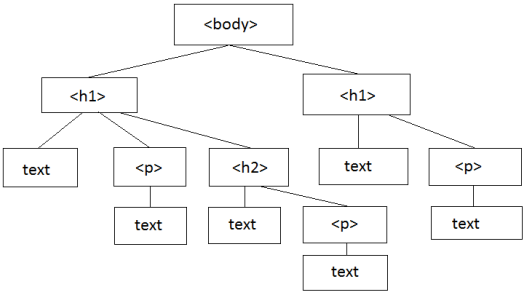


Рис. 8: *Построенное дерево структурной разметки.*

3.6 Алгоритм

Таким образом, алгоритм переноса разметки требований между версиями документа выглядит следующим образом:

1. Извлечь содержимое исходного и конечного документов при помощи JDOM
2. Построить на основе DOM деревьев документов деревья структурной разметки
3. Извлечь требования из дерева исходного документа
4. Для каждого объединения фрагментов требования:
   (a) Получить путь до объединения фрагментов требования
   (b) Получить по дереву разметки конечного документа путь до соответствующего раздела
   (c) Найти вхождение текста объединения фрагментов в полученном разделе.
   (d) Если путь в дереве разметки конечного документа или вхождение текста объединения фрагментов в разделе конечного документа не было найдено, считать, что перенос каждого фрагмента из объединения осуществить не удалось. В противном случае - изменить структуру части дерева разметки, соответствующей разделу конечного документа, добавив в неё фрагменты требования
   (e) Изменить структуру соответствующей разделу части DOM дерева конечного документа
5. Изменить конечный документ в соответствии с изменением структуры его DOM дерева

3.7 Характеристики предложенного алгоритма

В силу особенности построения дерева разметки для исходного и конечного документов, добавление, изменение или удаление разделов, не содержащих фрагментов требований, не влияет на эффективность работы алгоритма. Помимо этого, перестановка разделов в конечном документе так же не влияет на перенос фрагментов.

Однако изменения заголовков разделов, по которым локализуется место поиска соответствия объединению фрагментов требования, а так же изменения вложенности подразделов, в которых ищутся фрагменты, влияют на перенос разметки в конечный документ - если полное соответствие пути до объединения фрагментов в дереве разметки исходного документа не было найдено, перенос фрагментов не осуществляется.

Помимо этого стоит отметить, что предложенный алгоритм гораздо проще обобщить на случай, когда в конечном документе существует лишь частичное соответствие тексту объединения фрагментов, чем алгоритм, основанный на применении diff. Подробнее об этом будет рассказано в заключении.

4 Устройство системы

4.1 Дерево структурной разметки документа

Каждая вершина дерева структурной разметки имеет следующие поля: тип, текст содержимого и список вершин - сыновей и вершину-родителя. Изначально текст содержимого для вершины не определен, и заполняется в том случае, если в процессе переноса фрагментов требований вызывалась функция получения текста раздела, описанная ниже. Тип вершины дерева не фиксирован и зависит от соответствующей вершины DOM модели. Тремя зафиксированными типами вершин являются:

- **"text"** - в случае если вершина содержит текстовую информацию. Может находиться только в листьях дерева. Если тип вершины - *"text"*, в ней заполняется специальное поле, в котором запоминается содержимое соответствующего вершине участка текста документа.
- **"body"** - содержится в единственном экземпляре в корне дерева.
- **"requirement"** - в случае, если вершина содержит фрагмент требования. Может иметь только одного потомка типа *"text"*. Если тип вершины - *"requirement"*, в ней заполняются специальные поля, служащие для формирования списка требований и дальнейшего переноса фрагмента во в конечный документ.

  Поле id содержит идентификационный номер требования, к которому относится фрагмент, соответствующий вершине. Поле a содержит *true* или *false* в зависимости от того, нужно ли добавлять пару тегов ссылки на требование
  (**) после переноса фрагмента в конечный документ.

4.2 Построение дерева структурной разметки по DOM модели документа

Построение вершин дерева структурной разметки осуществляется в порядке обхода DOM дерева в глубину [9]. При этом вершины, не влияющие на структуру документа, игнорируются и не участвуют в построении дерева структурной разметки. В данной работе к вершинам, влияющим на структуру документа, относятся вершины DOM модели документа, имеющие имена *"span" "a"(в случае, когда вершина с именем "a"является потомком вершины, соответствующей фрагменту требования) "h1—h6" "p" и "blockquote"*. Соответствующая вершина дерева структурной разметки создается на основе текущей рассматриваемой вершины DOM модели.

Для организации иерархии параграфов и разделов используется стек (эта структура данных подробно описана в [9]): При прохождении в DOM дереве вершины заголовка раздела проверяется, есть ли такой тип раздела в стеке. Если его нет, то считается, что текущий раздел является подразделом последнего встреченного раздела и соответствующая ему вершина добавляется в стек. В противном случае из стека извлекаются вершины дерева до тех пор, пока извлеченная вершина не будет иметь тот же тип, что и добавляемая, а затем родителем добавляемой вершины становится родитель извлеченной из стека. Таким образом вершины подразделов одного и того же раздела в дереве структурной разметки имеют одинаковую глубину.

В случае, если текущая рассматриваемая вершина DOM модели документа имеет имя *span* и атрибут *class*, начинающийся с *"requality_ text"*, в дереве разметки создается вершина, имеющая тип requirement, и в ней заполняются поля *id* и *a*. По умолчанию считается, что такие вершины содержатся только в исходном документе.

4.3 Устройство требования

- **Фрагмент требования (Location)**

  Объект этого типа содержит ссылку на соответствующую фрагменту вершину дерева разметки. Предоставляет метод, позволяющий определять порядковый номер вершины в списке сыновей её родителя.

- **Объединение фрагментов (ActualLocation)**

  В силу особенностей выделения фрагментов требований в системе Requality, подряд идущие участки текста, соответствующие одному требованию, могут находиться в разных фрагментах из-за наличия разметки. Перед переносом такие участки нужно объединить в один, для которого и должен осуществляться поиск соответствия в конечном документе.

- **Требование (Requirement)**

  Объект каждого требования содержит *id* требования, список фрагментов, относящихся к нему, и список объединений фрагментов, полученный в результате обработки списка фрагментов. Объединение фрагментов происходит в том случае, если они имеют общего родителя, и их порядковые номера, как сыновей, отличаются на единицу. Помимо этого, объект типа Requirement содержит информацию о том, нужно ли добавлять пару тегов ссылки на требование, при переносе этого фрагмента.

4.4 Извлечение текста из дерева структурной разметки

Текстом вершины X дерева структурной разметки будем называть объединение полей text тех вершин дерева, которые имеют тип text и содержатся в поддереве, корнем которого является

вершина X.

1. **Текст заголовка**

   Текстом заголовка считается объединение текстов всех подряд идущих вершин, являющихся прямыми потомками вершины, соответствующей этому заголовку, до первой вершины, соответствующей абзацу. На рисунке 9 фрагмента дерева разметки выделены вершины, текст которых участвует в тексте заголовка h1.

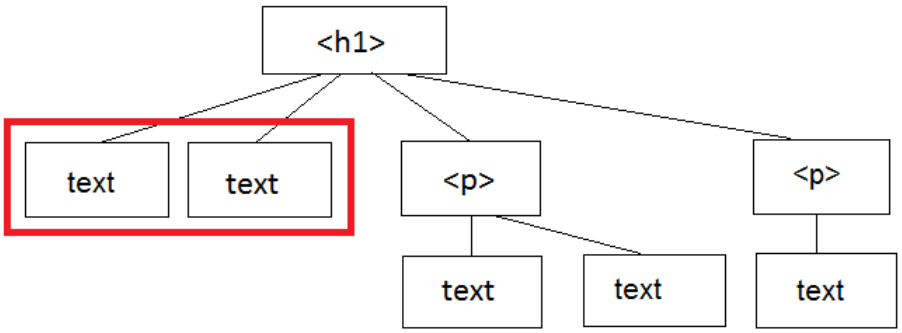


Рис. 9: *Текст заголовка дерева разметки*

2. **Текст раздела**

   Текст раздела содержит в себе текст соответствующего заголовка и текст всех вершин, не являющихся подзаголовками этого раздела. На рисунке 10 фрагмента дерева разметки выделены вершины, текст которых включается в текст раздела h1.

   После получения текста раздела он преобразуется при помощи регулярных выражений и запоминается в специальном поле соответствующей вершины дерева структурной разметки документа.

3. **Использование регулярных выражений**

   Поскольку используемые незначащие символы в исходном и конечном документах могут отличаться, для осуществления сравнения и поиска в частях текстов их необходимо привести к унифицированному виду. Для этого используется механизм регулярных выражений, описанный в [10], [11]. Перед дальнейшим использованием текста раздела или заголовка, над ним осуществляются следующие преобразования:

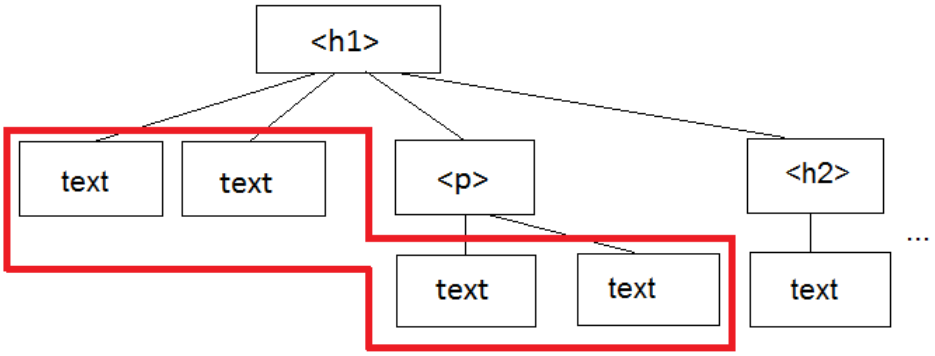


Рис. 10: *Текст раздела дерева разметки*

- Все пробельные символы заменяются на пробел (символ ' ' в Java)
- Удаляются все пробельные символы и переводы строк(whitespace characters) в начале и конце обрабатываемой строки
- Несколько пробельных символов или переводов строк, идущих подряд, заменяются на один.
- Пробельные символы, идущие перед знаками препинания, удаляются. После знаков препинания добавляется один пробел, если его нет.
- Перед открывающими скобками добавляется один пробел, если его нет.
- После открывающих и перед закрывающими скобками удаляется пробел, если он есть.

4.5 Устройство поиска соответствия объединению фрагментов требований

По объединению фрагментов требования осуществляется поиск пути в дереве разметки исходного документа до подраздела, содержащего это объединение фрагментов. При этом из множества найденных путей выбирается тот, который имеет максимально возможную длину. Путь хранится в виде последовательности указателей на вершины дерева, начиная от корня.

В дереве разметки конечного документа осуществляется поиск соответствующего пути. Путь B считается соответствующим пути A, если тексты всех заголовков в пути B совпадают с точностью до незначащих символов и цифр с текстами соответствующих заголовков в пути A. B

случае если такой путь был найден, считается, что соответствие объединению фрагментов должно находиться в конечном документе в тексте подраздела, соответствующего последней вершине (вершине наибольшей глубины) найденного пути. В противном случае каждый фрагмент из этого объединения считается не перенесенным, и процесс переноса для данного объединения фрагментов требования останавливается.

Если в конечном документе путь, соответствующий пути до объединения фрагментов требования в исходном документе, был найден, то текст соответствующего подраздела конечного документа извлекается, преобразуется с использованием регулярных выражений, и в нём осуществляется поиск текста объединения фрагментов методом прямого поиска [12]. Результатом поиска является позиция (номер символа), где в обработанном тексте подраздела начинается участок текста, полностью совпадающий с текстом объединения фрагментов требования, либо -1, если такая позиция не была найдена. По этой позиции далее осуществляется добавление тегов требований в конечный документ и элементов в соответствующее ему дерево разметки.

4.6 Добавление элементов требований в дерево разметки документа

В общем случае найденное соответствие объединению фрагментов лежит в нескольких вершинах дерева разметки конечного документа типа *"text"*. Это объясняется тем, что вершины типа *"text"*, идущие подряд, не объединяются при построении дерева разметки документа, при этом элементы DOM модели документа, не отвечающие за разметку, в построении дерева не используется. В ходе обхода поддерева, соответствующего найденному подразделу, возможны следующие варианты взаимного расположения найденного соответствия объединению фрагментов и вершин типа *"text"*, и соответствующие проводимые операции:

1. Текст вершины дерева разметки не пересекается с найденным соответствием (находится до номера символа, с которого соответствие объединению фрагментов было найдено). В этом случае текст текущей рассматриваемой вершины пропускается.

2. Текст вершины дерева разметки полностью содержит в себе текст, соответствующий объединению фрагментов.

   В этом случае осуществляется разбиение рассматриваемой текстовой вершины на три части (в общем случае, в частных случаях одна или обе пограничных текстовых вершины могут отсутствовать), согласно рисунку 11, добавляется вершина требования (тип “requirement”) с соответствующим *id*. Процесс преобразования дерева разметки для данного объединения фрагментов после проведения этой операции считается завершенным.

3. Текст вершины дерева разметки частично содержит в себе текст, соответствующий объединению фрагментов.

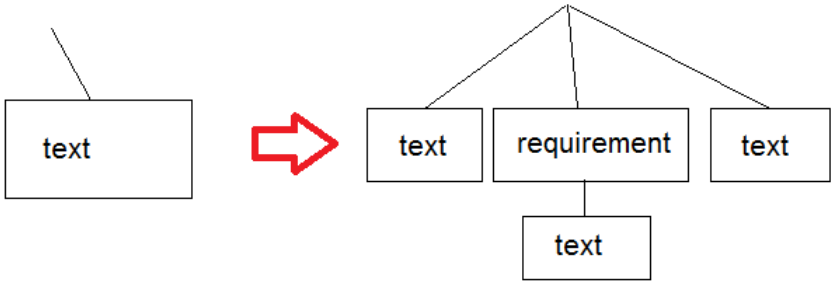


Рис. 11: *Случай полного вхождения текста объединения фрагметов в текстовую вершину дерева разметки*

В этом случае осуществляется разбиение рассматриваемой текстовой вершины на две части (в общем случае, в частном случае одна пограничная текстовая вершина может отсутствовать) согласно рисунку 12, добавляется вершина требования с соответствующим id. Затем часть текста, для которой была создана вершина типа requirement, удаляется из текста, соответствующего объединению фрагментов требования, и процесс продолжается для последующих текстовых вершин.

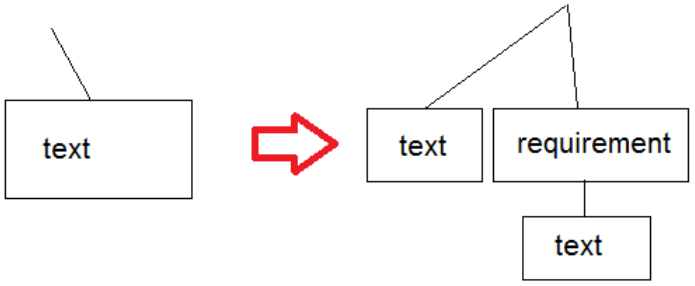


Рис. 12: *Случай неполного вхождения текста объединения фрагметов в текстовую вершину дерева разметки*

4.7 Добавление разметки требований в DOM модель

По измененному дереву разметки конечного документа и его DOM дереву синхронно осуществляется обход в глубину - на каждом шаге обхода текущая рассматриваемая вершина дерева разметки соответствует вершине DOM дерева. В случае если рассматриваемая вершина дерева

разметки и соответствующая ей вершина DOM дерева имеют тип *"text"*, осуществляется проверка на совпадение содержимого текущих вершин DOM модели и дерева разметки. Если они не совпадают, считается, что эта вершина была изменена в дереве разметки конечного документа из-за добавления элементов требований, и вершину DOM дерева нужно разбить [6] на несколько таким же образом, как было описано в предыдущем пункте. Вершина типа *"requirement"* дерева разметки при этом заменяется на вершину DOM дерева с именем *span* и атрибутами *class*, начинающимся *"requality_ text"*,

Поскольку на данном этапе вершины дерева разметки и DOM дерева, тип которых отличен от *"text"*, не влияют на работу этой части программы, их рассмотрение можно опустить, реализовав функции, получающие следующие в порядке обхода дерева в глубину вершины типа *"text"* по текущим в дереве разметки и DOM дереве соответственно.

После изменения DOM модели конечного документа по ней создается его измененная версия, содержащая теги перенесенных требований.

5 Полученные результаты

Описанный ранее алгоритм был реализован на языке программирования Java, и работа реализации была проверена на наборе документов ETSI [13], IEEE [14], [15] и Linux Foundation [16], [17]. Результаты экспериментов приведены в таблице 1.

Таблица 1: Результаты работы предложенного алгоритма на некоторых документах

Название документа	**Номер исходной версии**	**Номер конечной версии**	**Примерный объем (тыс. символов)**	**Перенесено/ найдено фрагментов**	**%**
ETSI TS 103 097	1.1.6	1.1.12	75	191/430	44.4
TTCN-3 core language part 3 head 5	4.5.1	4.6.1	32	323/335	96.4
TTCN-3 core language part 4 head 6	4.5.1	4.6.1	104	936/969	96.5
TTCN-3 core language part 5 head 7	4.5.1	4.6.1	20	138/154	89.6
POSIX*, fprintf	Issue 6, 2004	Issue 7, 2008	32	721/1014	71.1
POSIX, fwprintf	Issue 6, 2004	Issue 7, 2008	23	651/954	68.2
POSIX, environ(exec)	Issue 6, 2004	Issue 7, 2008	33	335/487	68.7
POSIX, fscanf	Issue 6, 2004	Issue 7, 2008	20	414/610	67.9
LSB**, zlib-deflateinit2	3.1	4.0	3.5	139/147	94.6
LSB, zlib-deflate-1	3.1	4.0	5.2	186/200	93.0
LSB, libutil-getopt-3	3.1	4.0	5.5	232/232	100
POSIX, все документы	Issue 6, 2004	Issue 7, 2008	∼8000	27683/39341	70.4
LSB, все документы	3.1	4.0	∼2000	5754/6767	85.0
Test Document 1	1	2	0.5	4/4	100
TTCN-3 core language part 3 head 5 (changed)	4.5.1	4.6.1	32	323/335	96.4

** The Open Group Base Specifications IEEE Std 1003.1*

*** Linux Standard Base Core Specification*

Помимо этого, на некоторых документах было проведено сравнение реализации разработан-

ного алгоритма и алгоритма, реализованного в инструменте управления требованиями Requality по трем параматрам - количеству найденных фрагментов требований, количеству перенесенных фрагментов требований и времени работы. Результаты сравнения приведены в таблице 2.

Таблица 2: Результаты сравнения эффективности двух алгоритмов

		Решение в Requality* **				**Предложенное решение**			
Название документа	**Примерный объем (т.с.)**	Найдено	Перенесено	Время (мс)	%	Найдено	Перенесено	Время (мс)	%
TTCN-3 core language part 3 head 5	32	335	231	3212	69	335	323	4439	96
TTCN-3 core language part 4 head 6	104	969	649	7993	67	969	936	10627	97
TTCN-3 core language part 5 head 7	20	154	96	3239	62	154	138	2641	90
ETSI TS 103 097	75	430	160	3194	37	430	191	2757	44
POSIX*, fprintf	32	1014	500	1520	49	1014	721	5059	71
POSIX, fwprintf	23	954	473	1499	50	954	651	4633	68
POSIX, environ(exec)	33	487	245	1811	50	487	335	4934	69
POSIX, fscanf	20	610	296	1640	49	610	414	2281	68
LSB**, zlib-deflateinit2	3.5	147	104	1079	71	147	139	1390	95
LSB, zlib-deflate-1	5.2	200	134	1250	67	200	186	1235	93
LSB, libutil-getopt-3	5.5	232	127	1064	55	232	232	1250	100
Test Document 1	0.5	4	0	1111	0	4	4	200	100
TTCN-3 core language part 3 head 5 (changed)	32	335	177	3569	69	335	323	4422	96

** The Open Group Base Specifications IEEE Std 1003.1*

*** Linux Standard Base Core Specification*

**** При тестировании была использована версия Requality, актуальная на момент 7.05.2015*

Документ, указанный в таблицах 1 и 2, как *"Test Document 1"* версий 1 и 2, является небольшим тестовым документом, демонстрирующим основное преимущество разработанного алгорит-

ма перед алгоритмом, испольующим Diff — в конечной версии этого документа некоторые разделы переставлены местами по сравнению с исходной.

Конечная версия документа, указанного в таблицах 1 и 2, как *TTCN-3 core language part 3 head 5 (changed)*, получена из соответствующей конечной версии документа *TTCN-3 core language part 3 head 5* перестановкой некоторых абзацев и разделов.

Из таблиц 1 и 2 видно, что в целом реализация приведенного в данной работе алгоритма переносит больше фрагментов требований, однако работает дольше. Но это связано преимущественно не с изменениями положения разделов в конечном документе и различиями рассмотренных алгоритмов, а с недостатками текущей реализации алгоритма, основанного на использовании Diff. В среднем алгоритм переносит на $\sim 41\%$ больше фрагментов и работает на $\sim 67\%$ медленнее.

Код программы, реализующей предложенный алгоритм, занимает ∼1300 строк, и приводить его в тексте данной работы было бы нецелесообразно. Посмотреть его можно в публичном репозитории GitHub [18].

6 Заключение

Таким образом, в данной работе было проведено исследование проблемы переноса разметки требований между многоверсионными текстовыми документами в рамках инструмента управления требованиями Requality. Было изучено решение, использующееся в Requality на данный момент, и разработан альтернативный алгоритм, основанный на предположении об отсутствии изменений названий и структуры разделов в новой версии спецификации требований.

Была написана реализация алгоритма на языке программировния Java и проведены эксперименты на наборе документов с размеченными требованиями, а также сравнение по эффективности с реализацией алгоритма, использующегося в Requality.

6.1 Возможные улучшения

Заметим, что приведенный в данной работе алгоритм обобщается на случай поиска в тексте раздела нечеткого соответствия тексту объединения фрагментов. Для этого достаточно заменить алгоритм прямого поиска объединения фрагментов в тексте раздела на алгоритм нечеткого поиска строки в тексте, использующий методы, описанные, например, в [19].

Список литературы

[1] Процессы разработки ПО.
http://rsdn.ru/article/Methodologies/SoftwareDevelopmentProcesses.xml (дата обращения: 07.05.15).

[2] Wiegers K. E., Beatty J. Software Requirements (3rd Edition). Microsoft Press, 2013. 672 p.

[3] Guide to the Software Engineering Body of Knowledge: 2004 version. CA: IEEE Computer Society Press, 2005.

[4] Система управления требованиями Requality.
http://requality.ru/en/index.html (дата обращения: 07.05.15).

[5] Акимов С. В. Технологии Internet / Intranet в почтовой связи: учебное пособие. СПб.: СПбГУТ, 2005.

[6] Harold E. R. Processing XML with Java: A Guide to SAX, DOM, JDOM, JAXP, and TrAX: Volume 1. Addison-Wesley Professional, 2002.

[7] Упрощение XML-программирования при помощи JDOM.
http://www.ibm.com/developerworks/ru/library/j-jdom/ (дата обращения: 07.05.15).

[8] Myers E. W. An O(ND) Difference Algorithm and Its Variations.
https://neil.fraser.name/software/diff_match_patch/myers.pdf (дата обращения: 07.05.15).

[9] Knuth D. E. The Art of Computer Programming: Volume 1: Fundamental Algorithms. Addison-Wesley Professional, 1997. 720 p.

[10] Smyth W. F. Computing Patterns in Strings. Addison-Wesley Professional, 2003. 440 p.

[11] Vogel L. Java and Regular Expressions.
http://www.vogella.com/tutorials/JavaRegularExpressions/article.html (дата обращения: 07.05.15).

[12] Алгоритмы поиска в строке.
http://habrahabr.ru/post/111449/ (дата обращения: 07.05.15).

[13] Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3; Part 1: TTCN-3 Core Language.
http://www.etsi.org/deliver/etsi_es/201800_201899/20187301/04.06.01_60/es_20187301v040601p.pdf (дата обращения: 07.05.15).

[14] The Open Group Base Specifications Issue 6 IEEE Std 1003.1, 2004 Edition.
http://pubs.opengroup.org/onlinepubs/009695399/ (дата обращения: 07.05.15).

[15] The Open Group Base Specifications Issue 7 IEEE Std 1003.1, 2008 Edition.
http://pubs.opengroup.org/onlinepubs/9699919799/ (дата обращения: 07.05.15).

[16] Linux Standard Base Core Specification 3.1.
http://refspecs.linux-foundation.org/LSB_3.1.0/LSB-Core-generic/
LSB-Core-generic/book1.html (дата обращения: 07.05.15).

[17] Linux Standard Base Core Specification 4.0.
http://refspecs.linux-foundation.org/LSB_4.0.0/LSB-Core-generic/
LSB-Core-generic/book1.html (дата обращения: 07.05.15).

[18] Diploma project GitHub repository.
https://github.com/Revoltt/ReqSystem.

[19] Нечеткий поиск в тексте и словаре.
http://habrahabr.ru/post/114997/ (дата обращения: 07.05.15).