Правительство Российской Федерации
Федеральное государственное автономное образовательное
учреждение высшего профессионального образования

"Национальный исследовательский университет
"Высшая школа экономики"

Отделение программной инженерии
Кафедра Управления разработкой программного обеспечения

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

На тему: Автоматизация проектирования тестов на основе требований

Студента группы No 271ма
Кильдишева Дениса Степановича
(Ф.И.О.)
Научный руководитель
д.ф.- м.н., профессор
(должность, звание)
Петренко Александр Константинович
(Ф.И.О.)
Консультант
к.т.н
(должность, звание)
Гринкруг Ефим Михайлович
(Ф.И.О.)

Москва, 2014 г.

Аннотация

Данная работа посвящена вопросу автоматизации проектирования тестов. В работе рассмотрен подход к разработке тестов для которого определен ряд связанных с ним проблем. Приведено сравнение программных средств поддержки требований и проектирования тестов, сделан вывод о необходимости в решении нескольких задач.

Для задач компактного описания схожих артефактов и поддержки модельного ряда выбраны пути решения в форме использования механизмов переиспользования и параметризации. Рассмотрено несколько возможных подходов к их реализации, произведена разработка механизмов в программном средстве Requality.

Количество страниц в работе без учета приложений – 78. Работа содержит таблиц – 1, иллюстраций – 28, использованных источников – 23.

Ключевые слова: ПРОЕКТИРОВАНИЕ ТЕСТОВ, ПЕРЕИСПОЛЬЗОВАНИЕ, ПОВТОРНОЕ ИСПОЛЬЗОВАНИЕ, ПАРАМЕТРЫ, ПАРАМЕТРИЗАЦИЯ, ТЕСТОВЫЕ СИТУАЦИИ, КАТАЛОГ ТРЕБОВАНИЙ.

Оглавление

Введение

Разработка систем, в том числе программных, должна сопровождаться процессом тестирования. Используемые при разработке систем стандарты, в том числе DO-178C[1] для разработки программного обеспечения в сфере авионики, рассматривают необходимость тестирования системы с использованием требований, в том числе представленных в форме документов.

Стоит отметить, что процесс тестирования жизненно важных систем, в соответствии со стандартами, должен производиться с использованием функциональных требований или моделей поведения, которые могут быть рассмотрены как форма требований[1].

На данный момент разработка систем во многих случаях производится в соответствии с набором документов, в том числе спецификаций и стандартов. При этом, использование документов в исходной форме затруднено по причине нечеткой формализации и возможного наличия дублирования и противоречий [2, 3]. Возникает задача анализа и выделения отдельных требований из текста, которые впоследствии могут быть использованы для построения систематизированного каталога требований[2].

После получения каталога требований становится возможным процедура проектирования тестов. Под проектированием тестов понимается описание тестовых ситуаций в рамках которых может производится проверка отдельного требования. При этом могут быть определены проверяемые условия, а также указаны входные и выходные данные, например, представленные в форме ожидаемых значений индикаторов для некоторого устройства. Подобного рода описание также называются тестовыми ситуациями.

После этапа проектирования тестов происходит этап разработки тестов, во время которого для выделенных тестовых ситуаций происходит описание процедуры проверки требований в указанных условиях. В ряде источников [2, 4] результат этапа разработки определяется как тестовая процедура, содержащая набор шагов для проверки соответствия системы требованиям в каталоге.

После разработки набора тестовых процедур становится возможным провести их выполнение, после обработки результатов которого можно утверждать о соответствии системы требованиям в рамках имеющегося набора спроектированных и разработанных тестов. Также можно рассматривать вопрос покрытия каталога требований тестовыми ситуациями и непосредственно тестами.

Стоит отметить, что подробное рассмотрение этапов разработки и выполнения тестов выходит за рамки данной работы.

Изложенный процесс тестирования может быть произведен над разными формами представления требований. В данной работе будет рассмотрено использование текстового представления. Подобное решение, в частности, связано с наличием большого количества документов, приведение которых в более формализованное представление является затруднительным.

Рассмотренный подход к тестированию позволяет выделить жизненный цикл требования в ситуации отсутствия изменений (Рис. 1). Вопрос о влиянии изменений на процесс будет рассмотрен подробней в пункте 1.1.2.

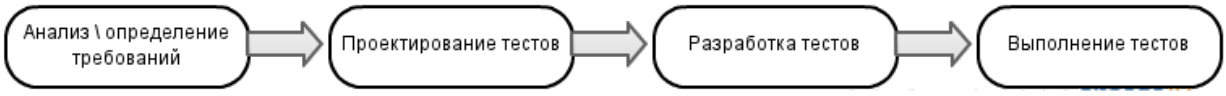

Рисунок 1. Жизненный цикл тестов

Проблемы, связанные с проектированием тестов

При рассмотрении жизненного цикла тестов (Рис. 1), становится возможным выделить ряд проблем, связанных как с процессом проектирования тестов, так и с поддержкой уже имеющегося набора требований и тестовых ситуаций.

- Первая из них - получение структурированного каталога требований из набора документов. Как было упомянуто выше, проблема может разделена на выделение требований и анализ их взаимоотношений.

- Следующей проблемой является отслеживание изменений, в том числе как исходных документов, так и каталога требований и тестовых ситуаций. Можно отметить, что требования, в том числе описанные в форме документов, время от времени изменяются. Также возможны изменения каталога требований и тестовых ситуаций. В связи с этим возникает необходимость в поддержке актуальности состояния всех участвующих в проектировании тестов элементов.
- Третья проблема может быть обозначена как поддержка набора схожих тестовых ситуаций. Возникают ситуации, при которых необходимо определять и поддерживать набор артефактов (в частности, тестовых ситуаций и требований) описание которых является схожим, отличаясь рядом параметров. В роли параметров могут выступать наборы значений, в том числе числовых.
- Последняя из рассмотренных проблем заключается в поддержке схожих фрагментов каталога артефактов для нескольких программных продуктов или разных версий одного продукта.

При поддержке каталогов требований для разных версий одного продукта, также обозначаемой как поддержка модельного ряда,

некоторые фрагменты каталога требований и спроектированных тестов могут быть использованы для других версий одного продукта.

Цель работы

Целью данной работы является применение механизмов переиспользования и параметризации для решения актуальных задач в области проектирования тестов с реализацией указанных механизмов в программном средстве Requality.

Для достижения поставленной цели будут рассмотрены связанные с процессом проектирования тестов проблемы. После этого будет проведено рассмотрение программных продуктов, позиционируемые как средства управления требованиями и проектирования тестов. Для решения проблем описания поддержки схожих элементов, рассмотрено использование переиспользования и параметризации. Будут обозначены подходы к реализации указанных механизмов с указанием особенностей текущего решения в программном средстве Requality.

Актуальность работы

На данный момент повторное использование фрагментов каталога требований и поддержка описаний схожих артефактов во многих случаях осуществляются вручную и не могут обеспечить отслеживание изменений логически связанных фрагментов.

Для программных средств в области поддержки требований и проектирования тестов функции повторного использования объектов реализованы по большей части для коммерческих программных продуктов. При этом имеющиеся решения в большинстве своем не позволяют описывать параметризованные объекты.

Стоит отметить, что в то же время параметризация широко используется в отдельных средствах, предназначенных для генерации тестовых наборов.

Задачей данной работы является разработка механизмов повторного использования и параметризации в рамках программного средства Requality, призванная обеспечить средство компактного описания схожих фрагментов каталога требований, требований и тестовых ситуаций с возможностью внесения определенных изменений в копии.

1. Обзорно-постановочная часть

1.1. Особенности проектирования тестов

Рассмотрим ряд особенностей процесса проектирования тестов принимая во внимание обозначенные во вступлении проблемы.

1.1.1. Выделение каталога требований

Как уже было упомянуто во вступлении, в рамках данной работы рассматривается представление требований в форме текстовых описаний. При этом фрагменты подобных описаний могут быть получены из документов, что также обязательно при соблюдении ряда стандартов[1]. Также в данной работе требования содержат уникальный в пределах каталога требований идентификатор.

В более общем случае требование может быть не связано с документом, что не исключает необходимость анализа и обработки подобных требований.

Рассматривая подходы к построению иерархии каталога требований, можно рассмотреть один из распространенных вариантов - использование отношения детализации требований. Так, более детализированное требование будет являться дочерним по отношению к требованию с меньшей детализацией.

Подобное решение не является единственно возможным, но в рамках данной работы рассматриваться примеры каталогов требований построенных с использованием этого принципа.

Для примера процедуры получения каталога требований, рассмотрим фрагмент документа, представляющего собой требования к программной системе (рис. 2). На нем представлен пример получения из фрагментов технического задания элементов каталога требований.

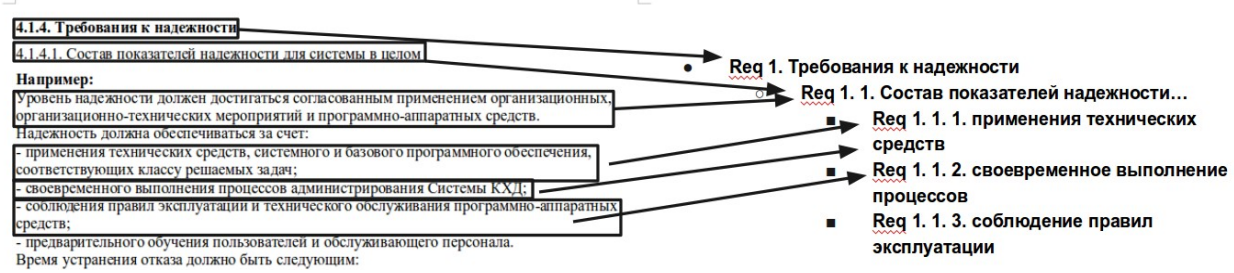

Рисунок 2. Получение каталога требований из документа

Как уже было сказано выше, требование с большей детализацией является дочерним по отношению к требованию с меньшей детальностью. Так, Req 1. соответствует обозначению семейства требований связанных с обеспечением надежности системы, а Req 1.1 определяет подкласс требований к надежности.

После построения каталога требований становится возможным описание тестовых ситуаций над полученными требованиями. Подробней этот процесс будет рассмотрен в 1.1.3.

1.1.2. Управление изменениями

Для поддержки актуальности каталога требований и разработанных тестовых ситуаций требуется организовать процесс управления изменениями.

Для примера возможных изменений можно рассмотреть изменения исходного документа, представленные на рис. 3. На данном примере обозначены два вида изменений - добавление нового фрагмента требований и удаление одного из фрагментов, связанного с требованием в каталоге.

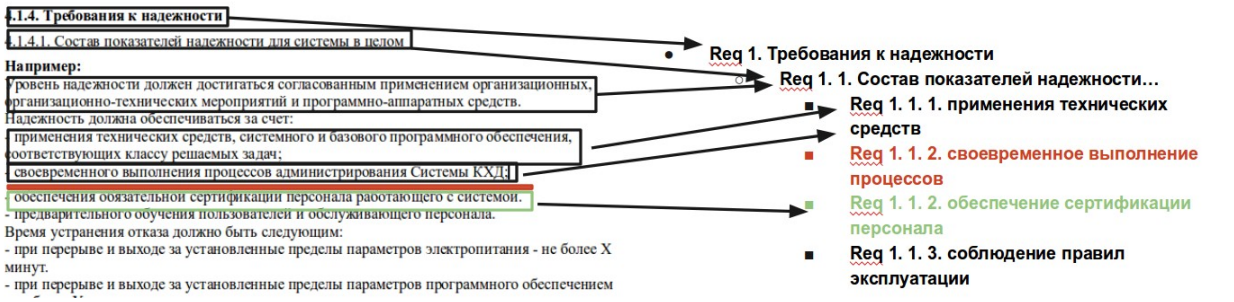

Рисунок 3. Изменение исходного документа

Можно отметить, что кроме указанных в примере изменений документа, можно также выделить еще один вид изменений - изменение фрагмента требований.

При возникновении изменений требуется провести соответствующие модификации в каталоге требований и связанных с ним тестовых ситуациях. Исчезновение и изменение фрагмента текста может послужить причиной удаления связанных с ним требований. Добавление нового фрагмента может послужить причиной добавления нового требования или привязку фрагмента к уже имеющемуся.

При этом, изменение или удаление требования может послужить причиной потери актуальности тестовых ситуаций. Добавление нового требования или привязка к требованию нового фрагмента документа может послужить причиной возникновения необходимости в описании новых тестовых ситуаций. Модификация текста документа связанного с требованиями также может послужить причиной возникновения ряда ситуаций, при которых набор связанных тестовых ситуаций будет изменен.

Стоит отметить, что поддержка механизмов управления изменениями является одной из ключевых функций для инструментов управления требованиями и проектирования тестов.

Можно утверждать об актуальности проблемы поддержки механизмов ответа на изменения и подходов, позволяющих снизить затраты на реагирование.

1.1.3. Схожие артефакты и фрагменты каталога требований

Рассматривая вопрос описания схожих артефактов в каталоге требований, приведем набор тестовых ситуаций для проверки требования к надежности прибора к высоким температурам:

- Req 1. Требования к надежности прибора

- Req 1. 1. Обеспечение устойчивости корпуса прибора к высоким температурам
  - Req 1. 1. _ TestPurpose 1. При помещении корпуса прибора в область с температурой 200 градусов Цельсия температура внутренней стороны корпуса не должна превышать 50 градусов
  - Req 1. 1. _ TestPurpose 2. При помещении корпуса прибора в область с температурой 300 градусов Цельсия температура внутренней стороны корпуса не должна превышать 75 градусов
  - Req 1. 1. _ TestPurpose 3. При помещении корпуса прибора в область с температурой 400 градусов Цельсия температура внутренней стороны корпуса не должна превышать 100 градусов

Для проверки требования становится возможным определить некоторый набор параметров, в частности, исходной и проверяемой температур. Становится возможным выделение обобщенного описания тестовой ситуации:

- Req 1. 1. _ TestPurpose N. При помещении корпуса прибора в область с температурой K1 градусов Цельсия температура внутренней стороны корпуса не должна превышать K2 градусов

В примере символами N, K2, K3 обозначены места описания параметров. Можно также определить некоторую зависимость их значений. Так, для K1 можно определить множество допустимых значений, включающее в себя набор [200, 300, 400], а значение K2 имеется возможность определить по формуле K1 / 4.

Особенности различных вариантов реализации описаний параметров будут рассмотрены в пункте 2.2.

Проблема поддержки описания схожих артефактов во многом похожа на проблему поддержки механизмов реагирования на изменения. Причина этому - необходимость в дополнительном отслеживании изменений для некоторых артефактов из набора схожих.

Примером может послужить изменение текста одной из рассмотренных тестовых ситуаций которое может потребовать изменение описаний для всех предложенных значений параметров.

Последняя из рассмотренных во вступлении проблем, а именно поддержка описания схожих фрагментов каталога требований для различных версий одного продукта, позволяет облегчить процесс управления изменениями и обеспечить удобный способ описания фрагментов каталога требований для дальнейшего переиспользования.

1.2. Сравнение программных средств

Рассматривая процесс проектирования тестов, стоит рассмотреть возможности имеющихся на данный момент программных средства в этой области. Для этого в первую очередь требуется определить характеристики для сравнения, в том числе связанные с поддержкой рассмотренных в пункте 1.1. механизмов.

1.2.1. Рассмотрение характеристик для сравнения

Для рассмотрения различных инструментов управления требованиями и проектированием тестов можно выделить набор характеристик с указанием возможными значениями, включая:

- Поддержку связи между фрагментом документа и требованием
  - Отсутствие поддержки привязки фрагмента документа к требованиям

- Привязка фрагмента хранимого в инструменте документа к требованиям с отслеживанием изменений
- Привязка фрагмента хранимого вне инструмента документа к требованиям с отслеживанием изменений

- Особенности интерфейса редактирования данных
  - Формат представления данных
    - Табличный
    - Иерархический
    - Графический
  - Возможность добавления иллюстраций
- Возможности расширения
  - Программный интерфейс описания расширений(API)
  - Расширение модели
    - Элементов иерархии
    - Логической части
- Доступные средства автоматизации
  - Генерация документов
  - Генерация отчетов
  - Поддержка импорта артефактов
  - Поддержка модельной серии
  - Поддержка механизма компактного описания схожих артефактов
- Лицензирование продукта
  - Свободное программное обеспечение
  - Закрытый продукт

1.2.1.1. Поддержка связи между фрагментом документа и требованием

Первой из рассмотренных характеристик является наличие возможности поддержания связи между фрагментом текста документа и определенным требованием в каталоге требований.

Подобная связь позволяет в ряде случаев решить проблему дублирования и противоречивости требований, облегчает процесс проектирования тестов.

Второе применение данной связи - использование ее при управлении изменениями - для изменений документа становится возможным проследить

потенциально необходимые изменения каталога требований и связанных тестовых ситуаций.

Третье применение - использование связи с фрагментами документа для получения покрытия документа требованиями и тестовыми ситуациями.

Рассмотрим возможные подходы к реализации подобной функции:

- Хранение связи в документе - данный подход предполагает хранение информации о связи требований и документа в файле документа . Для этого способа характерно удобство использования конечным пользователем при усложнении быстрого переноса каталога требований, что может потребоваться при появлении новой версии исходного документа. Также хранение информации в документе в ряде случаев может усложнить процесс совместной работы над ним.
- Использование для построения связи хранимого в программной системе документа. Подобный подход позволяет решить проблему переноса требований для различных версий документа, но при этом возможна частичная потеря представленных в документе данных, в частности, при наличии преобразований в хранимый в системе формат.

1.2.1.2. Особенности интерфейса редактирования данных

Вторая из рассмотренных характеристик - особенности интерфейсов редактирования данных. В рамках управления требованиями и тестовыми ситуациями возможности представления данных являются достаточно значимым.

Можно отметить, что использование исключительно текстового представления в ряде случаев может оказаться недостаточным. Например, может потребоваться использование таблицы, или иллюстраций, в текст

требований или описания текстовых ситуаций, что для текстового представления затруднительно.

Как одно из решений подобного рода проблем можно рассмотреть использование визуальных редакторов (английский термин: rich editor). При использовании данного средства становится возможным форматирование фрагментов текста и добавление дополнительных элементов, таких как ссылки, или упомянутые таблицы и иллюстрации.

Стоит отметить возможность существования различных видов связей между элементами. В общем случае возможно наличие множества сложных связей, что служит причиной необходимости поиска дополнительных способов их отображения. Подобное служит причиной использования в некоторых программных средствах более сложных представлений, таких как графическое.

Рассмотрены три формы визуализации: табличная, иерархическая и графическая.

Под табличном представлением подразумевается просмотр и редактирование информации в форме таблиц текстовых данных без выделения иерархии над элементами (рис. 4). Связи между элементами могут быть описаны в форме ссылок в полях таблицы. Табличное представление в ряде случаев может оказаться полезным, например, при редактировании атрибутов определенного артефакта.

Иерархическое представление предполагает выделение и отображение для артефактов иерархии на основе определенного параметра или отношения. Подобное отображение позволяет облегчить понимание зависимостей между артефактами в рамках выбранного отношения.

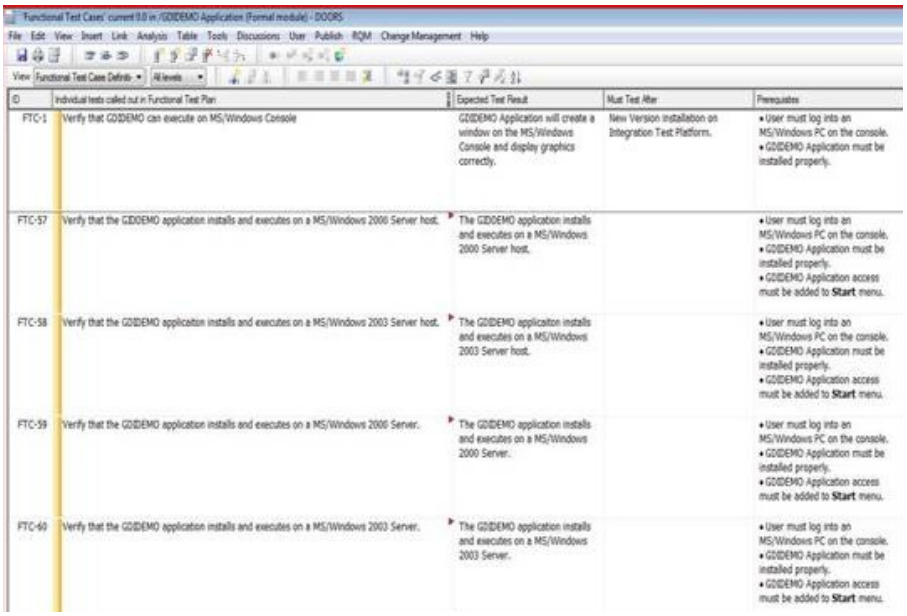

Рисунок 4. Пример табличного представления в интерфейсе IBM DOORS

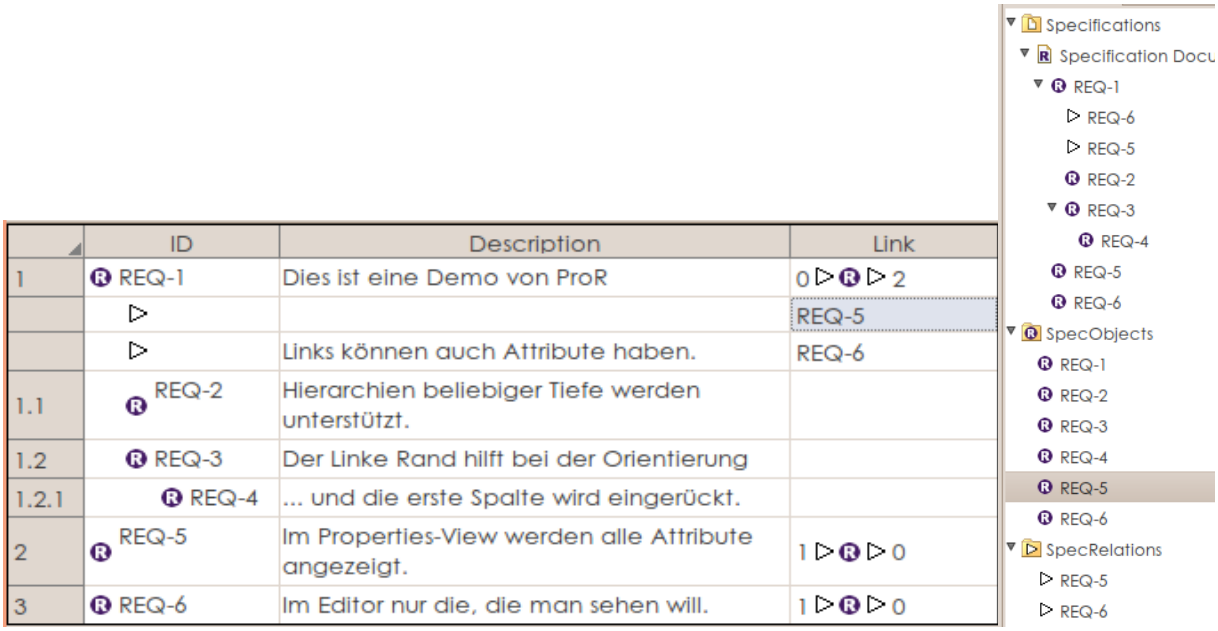

Рисунок 5. Примеры иерархического представления в интерфейсе Eclipse ProR

Последняя из рассмотренных актуализаций пользовательского интерфейса - графическая, предполагающая отображение данных артефактов и связей между ними в форме графа. Использование иерархического представления ограничено одним определенным видом связей, в то время как

графическое представление позволяет определять несколько видов связей на одном отображении.

В связи с этим, представление в форме графа является одним из наиболее удобных для получения представления о связях и взаимодействиях между артефактами, но сложно применимо в ситуации наличия большого количества артефактов или связей между ними по причине визуального замусоривания интерфейса. На рисунке 6 рассмотрен пример графического представления на котором показан интерфейс редактора описания сценариев инструмента Caliber RM (пункт 1.2.4.).

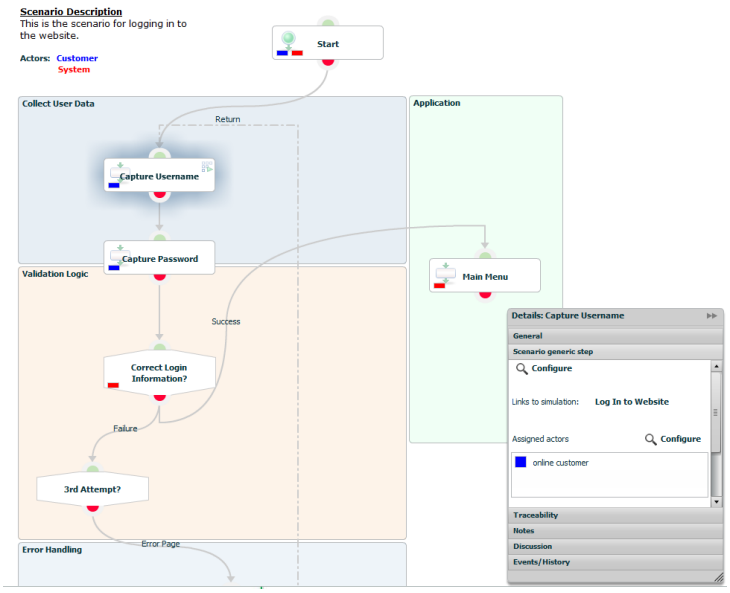

Рисунок 6. Пример графического представления интерфейса в приложении Caliber Requirements Management

1.2.1.3. Возможности расширения

Третья из рассматриваемых характеристик определяет доступные механизмы расширения программных систем. В рамках данной характеристики рассматриваются механизмы, позволяющие расширить функционал рассмотренных программных продуктов.

Были рассмотрены два способа описания дополнительных функций - путем написания дополнений, для чего было рассмотрено наличие программного интерфейса разработки приложений (английский термин: Application Programming Interface), и расширение программного средства с использованием доступных конфигурационных средств, включая изменение имеющегося набора элементов модели и расширение логической части модели.

Под расширением элементов модели подразумевается возможность описания новых видов артефактов, их свойств или отношений, под расширением логической части - возможность изменения поведения модели включая, но не ограничиваясь описанием дополнительных реакций на изменения элементов и определением ограничений над операциями.

1.2.1.4. Доступные средства автоматизации

Следующая рассматриваемая характеристика - это доступные возможности автоматизации процесса управления требованиями и проектирования тестов. Рассмотрим подробнее пять предложенных средств.

Под генерацией документов понимается наличие возможности перевести фрагменты модели в представление в форме некоторого документа. Подобная функция, в частности, может быть использована при выводе из системы текущего актуального набора требований.

Генерация отчетов по сути также является выводом некоторой информации из системы, но предоставляет более широкие возможности. Отчет предполагает возможность использования как имеющихся данных, так и обработки данных по имеющейся модели.

Примером могут послужить отчеты о покрытии, предоставляющие информацию для определенного вида артефактов о полноте наличия связи с другим видом артефактов. Частный случай - отчет о покрытии требований

тестами, в рамках которого для набора требований может быть указан процент требований, для которых разработаны тесты или обозначены требования, для которых нет тестов, в том числе в графической форме.

Третье средство - поддержка импорта артефактов, включает в себя механизмы импорта артефактов из внешних источников данных. Под подобными источниками могут пониматься как документы, так и другие программные средства. Применение подобного механизма позволяет, в частности, использовать набор требований из одного программного средства в другом.

Последние два механизма - поддержка модельного ряда и присутствие возможности компактного описания набора артефактов, означают возможности, призванные решить последние из рассмотренных актуальных задач.

Поддержка модельного ряда может быть определена как возможность описания нескольких каталогов требований и связанных с ними тестов. Подобные возможности присутствуют во многих программных средствах, заключаясь в доступности создания различных проектов.

При этом, как уже было замечено, проблема возможного дублирования фрагментов каталога требований усложняет поддержку нескольких каталогов, в рамках которых подобное может возникнуть. Ряд программных средств позволяет избежать дублирования при использовании определенных возможностей, но для многих других подобные возможности не заявлены.

Компактное описание артефактов надо рассматривать как возможность создания компактного описания набора артефактов, с которым впоследствии будет иметься возможность работать в рамках приложения. В том числе, предполагается возможность использования подобного описания для

создания артефактов. Среди инструментов, реализующих подобный механизм, можно отметить Qualifying machine.

Использование компактных описаний способствует облегчению процесса реагирования на изменения, позволяя в ряде случаев снизить затраты на создание и хранение набора артефактов со схожим набором атрибутов (свойств).

1.2.1.5. Лицензирование продукта

Последняя из рассматриваемых характеристик — лицензирование программного продукта. Данный параметр рассматривается в связи с особенностями разработки дополнений к коммерческому программному обеспечению и получения информации по его возможностям. Можно отметить, что большая часть платных программных продуктов поддерживает некоторый набор актуальных данных о доступных функциях, предоставляя также доступ к пробной версии с ограниченным функционалом, но проверка на фактическое наличие многих из функций может оказаться затруднительной.

Рассматривая необходимость модификации исходного кода программного средства, стоит отметить преимущества свободного программного обеспечения, часто предоставляющего подобную возможность.

1.2.2. Обзор инструментов поддержки требований

Был рассмотрен ряд имеющихся программных средств, используемых как в области анализа и проектирования требований, так и непосредственно для проектировании тестов.

Инструменты были выбраны после изучения различных источников, в том числе обзоров на средства управления требованиями, а также статей на

тематику управления требованиями и проектирования тестов. Примером может послужить обзор от International Council on Systems Engineering (INCOSE) [5]. Были выбраны как коммерческие продукты, так и свободное программное обеспечение.

- Коммерческие средства
  - IBM Reqsuite Pro
  - IBM DOORS
  - IBM Rational Requiremens Composer
  - Caliber Requirements Management
  - RTIME
  - Accompa
- Свободное ПО
  - Open Do Qualifying Machine
  - Eclipse RMF - ProR
  - Requality

Рассмотрим указанные программные средства подробнее.

1.2.2.1. IBM Reqsuite Pro

В рамках данной работы будет рассмотрено несколько средств управления требований принадлежащих корпорации IBM. Большая часть из них изначально разрабатывалась сторонними компаниями, но была в итоге приобретена IBM. На данный момент каждое средство имеет собственный набор функций, при этом поддерживая возможности переноса объектов в другие средства или предоставляя возможность интеграции с другими средствами.

IBM Rational ReqsuitePro представляет собой интегрируемый с Microsoft Word инструмент для управления требованиями и прецедентами для проектных групп[6]. На официальном сайте также заявлена поддержка интеграции с рядом средств семейства Rational с широкими возможностями в области трассируемости и анализа требований.

Сторонний обзор [7] выделяет поддержку методологии IBM RUP, гибкую систему авторизации пользователей и настраиваемую автоматическую генерацию и публикацию отчетов.

Из особенностей данного инструмента также можно отметить возможность поддержания соответствия между фрагментом текста и каталогом требований. В пункте 1.2.1.1. были рассмотрены достоинства и недостатки подобного подхода.

В плане поддержки модельной серии, можно отметить отсутствие этой возможности. Механизмы описания схожих артефактов не были упомянуты как поддерживаемые в рамках обзоров [6, 7] и данных сайта IBM.

1.2.2.2. IBM DOORS

IBM Rational Dynamic Object-Oriented Requirements System(DOORS) представляет собой масштабируемое средство управления требованиями в масштабах организации[8]. Отмечается наличие широких возможностей в области отслеживания и контроля изменений, инструменты построения связей между артефактами и использование подобных связей для различных анализов.

Отмечаются возможности в области интеграции с другими инструментами, в частности семейства IBM Rational, предоставляется возможность связи с инструментами поддерживающими спецификацию Open Services for Lifecycle Collaboration (OSLC) [9].

Среди особенностей отмечается возможность работы через Web интерфейс с базой требований, возможности масштабирования, различные средства расширения, наличие возможности вовлечения в процесс управления требованиями персонала не связанного с непосредственным управлением требованиями.

Стоит также отметить наличие продукта Rational DOORS Next Generation, представляющего собой инструмент на базе платформы IBM Rational Jazz. Данный инструмент представляет собой многопользовательский инструмент с клиентом, схожим с DOORS. При этом упрощается модель взаимодействия между пользователям и расширяются возможности по использованию инструмента за счет наличия Web интерфейса.

Рассматривая DOORS в рамках выбранных характеристик, можно отметить, что его интерфейс во многих случаях предполагает работу с текстовыми данными, что может ограничить возможности описания требований и наглядности проектирования тестов.

Инструмент является одним из трех поддерживающих связи между фрагментами документа и требованиями и одним из трех инструментов, для которых была заявлена поддержка переиспользования артефактов одновременно с возможностью поддержки модельного ряда.

Согласно описанию функциональности в официальном обзоре инструмента [9], в данном инструменте существует возможность описания копирования артефакта с поддержкой связи между исходным и копируемым артефактами. При этом имеется возможность определения изменения атрибутов связанного артефакта при изменении исходного.

Подобный подход является одной из возможностей описания схожих артефактов. Стоит отметить, что подобная реализация имеет ряд недостатков, которые будут рассмотрены в пункте 2.1.1.

1.2.2.3. IBM Rational Requiremens Composer

Последний из рассмотренных инструментов управления требованиями от компании IBM. Представляет собой средство коллективного управления

требованиями и отчетностью [10] в форме web сервиса на базе платформы Jazz.

Сторонний обзор [11] отмечает что данный инструмент в первую очередь нацелен на максимальное вовлечение клиента в процесс разработки заказанного программного продукта. Достигается это за счет поддержки различных форм обсуждений, расширенного отслеживания покрытия требований как другими требованиями, так и заметками клиентов, различных средств визуализации, таких как редактор вариантов использования.

Рассматривая инструмент с точки зрения интерфейса редактирования данных, стоит отметить наличие разнообразных средств визуализации, в числе которых уже отмеченный редактор сценариев использования. Можно отметить наличие возможности получения артефактов из средств Rational, наличие генерации документов и отчетов. Возможности расширения инструмента несколько ограничены в связи с представлением программного средства в форме web сервиса.

Возможности поддержки модельного ряда не заявлены[10], но имеющиеся механизмы позволяют повторно использовать артефакты и фрагменты каталога требований применяя так называемые коллекции артефактов.

Поддержка механизмов описания схожих артефактов не заявлена, но имеется возможность определения копий артефактов при помощи вышеупомянутых коллекций.

1.2.2.4. Caliber Requirements Management

Caliber Requirements Management[12] представляет собой семейство программных средств, предназначенное для управления требованиями. Содержит как непосредственно средства управления требованиями, так и

инструменты для работы с заинтересованными лицами по функциям подобные реализованным в IBM Rational Requirements.

Данный инструмент во многом похож на уже упомянутым IBM Rational Requirements - в виде преимуществ Caliber отмечаются механизмы, позволяющие получить непротиворечивое описание требований при содействии с клиентом и будущими пользователями.

Стоит отметить наличие широкого спектра возможностей среди рассмотренных программных средств - в Caliber доступны почти все рассматриваемые функции, исключая поддержание связи между фрагментом документа и требованием и наличие механизма описания схожих артефактов.

Для данного семейства инструментов заявлено наличие механизмов переиспользования фрагментов каталога требований, но при этом нет информации о возможностях Caliber в области описания схожих артефактов.

1.2.2.5. RTIME

Данное средство представляет собой инструмент управления требованиями и проектирования тестов[13]. Содержит несколько видов редакторов, в том числе имеется возможность подключения графических редакторов. Можно также отметить возможности для доступа заинтересованных лиц к текущему дереву требований и возможности описания различных версий каталога требований.

Среди особенностей заявлена широкая поддержка различных отчетов и наличие возможности поддержки модельного ряда без информации о поддержки механизма переиспользования. Также выделяется возможность отслеживания полной истории изменений артефактов. Поддержка описания схожих артефактов не была выявлена на основе доступной информации.

Можно отметить, что данное программное средство отличается сравнительно невысокой стоимостью для одного рабочего места, но при этом

осуществляется продажа RTIME как услуги, что может быть более затратно при длительном использовании.

1.2.2.6. Accompa

Accompa представляет собой интернет сервис для управления требованиями[14]. Среди заявленных особенностей можно отметить удобные средства взаимодействия между пользователями и управление изменениями с возможностью отслеживания их истории. Также можно отметить наличие средств группировки требований.

Рассматривая возможности данного инструмента, стоит отметить, что, как и IBM Rational Requirements Composer, Accompa предоставляется как web сервис, что усложняет процесс его расширения.

Для данного инструмента заявлена возможность переиспользования фрагментов каталогов требований, но отсутствует информация о наличии механизмов описания схожих артефактов.

1.2.2.7. Open Do Qualifying Machine

Open Do Qualifying Machine представляет собой средство управления процессом непрерывной верификации[15]. Данный инструмент является первым из рассматриваемых свободных программных средств. С точки зрения архитектуры представляет собой сервер с тонким клиентом в виде браузера.

Данное программное средство предназначено для управления требованиями и их изменениями в форме, облегчающей процесс сертификации. Среди особенностей отмечается возможность частичной автоматизации реакции на изменение в широких пределах. Также можно отметить возможности по модификации имеющейся модели при помощи конфигурации инструмента и наличие механизмов поддержки дополнений.

Обладает рядом недостатков интерфейса, проявляющихся в том числе в необходимости дополнительных навыков при настройке. Можно утверждать о наличии возможности компактного описания схожих артефактов. При этом поддержка модельного ряда для данного артефакта не была заявлена, но технически возможна.

1.2.2.8. Eclipse RMF - ProR

ProR представляет собой дополнение (английский термин - plugin) к среде разработки Eclipse [16] направленное на поддержку базовых функций работы с требованиям согласно стандарту ReqIf [17]. На данный момент является проектом Eclipse Foundation.

Среди особенностей данного программного средства можно отметить отсутствие возможностей графического описания свойств артефактов исходной версии дополнения, при этом подобные возможности могут быть добавлены при помощи доступных программных расширений (API).

Базовая версия предоставляет только некоторый набор артефактов, в том числе условного объекта и документа, для которых имеется возможность описания новых видов артефактов с определением доступных атрибутов. При этом также имеется возможность указания связи между артефактами с возможностью задания вида для объекта связи.

Поддержка модельного ряда для данного инструмента не заявлена, хотя имеется возможность описывать параллельно несколько каталогов требований без заявленной возможности описания переиспользуемого набора объектов. Поддержка описания схожих артефактов для этого инструмента заявлена не была.

1.2.2.9. Requality

Как и ProR, представляет собой расширение для среды разработки Eclipse [18]. Предназначен для управления требованиями и проектирования тестов.

Поддерживает механизм разработки расширений позволяющий дополнять имеющуюся модель, в том числе в форме дополнения новыми видами артефактов и описанием расширений логической части модели. Предоставляет несколько интерфейсов доступа к артефактам и их свойствам.

В Requality имеется возможность генерации отчетов, с помощью которой можно получить отчет по одному из хранимых шаблонов. Имеется возможность описать собственную форму отчета.

До выполнения данной работы инструмент не содержал средств компактного описания схожих артефактов и не обладал заявленной поддержкой переиспользования фрагментов каталога требований.

1.2.3. Таблица сравнения инструментов поддержки требований

После изучения информации по рассмотренным программным средствам, в том числе из представленных источников, была построена сравнительная таблица. Цветами обозначена оценка применимости значений характеристик для рассмотренного в 1.1. процесса проектирования тестов. При этом принят ряд обозначений:

Поддержка связи между фрагментом документа и требованием:

ОП - Отсутствие поддержки привязки фрагмента документа к требованиям

ПХ - привязка фрагмента хранимого в инструменте документа к требованиям с отслеживанием изменений

ПВ - привязка фрагмента хранимого вне инструмента документа к требованиям с отслеживанием изменений.

Особенности интерфейса редактирования данных:

Формат представления данных:

Т - табличный

И - иерархический

Г - графический

-И - возможность добавления иллюстраций

Возможности расширения:

API - наличие программного интерфейса описания расширений

РМЭ - расширение элементов иерархии модели

РМЛ - расширение логической части модели

Доступные средства автоматизации:

ГД - генерация документов

ГО - генерация отчетов

ИА - Поддержка импорта артефактов

Поддержка модельной серии:

“-+” - поддержка параллельного описания нескольких каталогов требований без заявленной поддержки модельного ряда

“+-” - заявленная поддержка модельного ряда без указания на наличие механизма переиспользования артефактов

“+” - наличие заявления о поддержке модельного ряда с упоминанием о наличии механизма переиспользования

Поддержка механизма компактного описания схожих артефактов

“+” - наличие механизма

“-” - отсутствие механизма

серый цвет - отсутствие как подтверждающей, так и опровергающей информации о наличии механизма.

Таблица

Сравнение инструментов управления требованиями и проектирования тестов

	Open Do Qualifying Machine	Eclipse RMF - ProR	IBM Rational Reqsuite Pro	IBM Rational DOORS	IBM Rational Requirements Composer	Caliber Requirements Management	RTIME	Accompa	Requality
Поддержание связи между фрагментом документа и требованием	ОП	ОП	ПВ	ПХ	ОП	ОП	ОП	ОП	ПХ
Интерфейс редактирования	Т-И, И-И	Т, И, -И*	И-И, Т-И	Т, И, Г	И-И, Т-И, Г	И-И, Т-И, Г	И, Т-И, Г*	И-И, Т-И, Г	И, Т-И
Возможности расширения	API, РМЛ, РМЭ	API, РМЛ	РМЛ, РМЭ	РМЛ, API	РМЛ	РМЛ, API	РМЛ	РМЛ	API, РМЛ
Средства автоматизации	ГО, ИА	ПА, ИА	ИА, ГО	ГД, ИА, ГО	ИА, ГО, ГД	ГО, ГД, ИА	ГО, ИА, ГД	ГО, ГД, ИА	ГО, ГД, ИА
Поддержка модельной серии	-+	-+	+-	+	+-	+	+-	+	-+
Механизм описания схожих артефактов	'+-	-	-	'+-	-	-	-	-	-
Лицензирование	Open Source	Open Source	Закрытый продукт	Закрытый продукт	Закрытый продукт	Закрытый продукт	Закрытый продукт	Закрытый продукт	Open Source
Цена	бесплатно	бесплатно	3000	4000	4000	2000	$100 месяц	$399 месяц	бесплатно

* - при использовани и сторонних дополнений

Рассматривая таблицу можно сделать ряд наблюдений. Первое что стоит отметить - отсутствие информации о поддержке механизмов описания компактных артефактов для большинства представленных продуктов. При этом можно утверждать о наличии подобного механизма в инструменте Open Do Qualifying Machine и в IBM DOORS.

Второе - вывод о наличии поддержки модельной серии во всех представленных продуктах. При этом поддержка данной функции одновременно с возможностью переиспользования артефактов встречается только у трех из рассмотренных программных продуктов.

Можно утверждать, что одновременная поддержка компактного описания схожих артефактов и наличие возможностей переиспользования практически не представлено, за исключением IBM DOORS.

Третье, что может быть отмечено - инструмент Requality содержит набор функций, сравнимый как с аналогами в области свободного программного обеспечения, так и с рядом коммерческих продуктов. При этом

стоит учесть доступные возможности для реализации механизмов компактного описания артефактов и переиспользования фрагментов каталога требований в рамках поддержки модельного ряда.

2. Подходы к реализации механизмов переиспользования и параметризации

2.1. Поддержка компактного описания схожих артефактов

Рассматривая проблему поддержки набора схожих артефактов можно выявить несколько связанных подзадач. Первая заключается в поддержке некоторого набора артефактов с возможностью его использования в различных местах каталога требований. Подобную проблему можно обозначить как проблему переиспользования.

Вторая проблема - обеспечить набор генерируемых параметров для артефакта с возможностью их использования в свойствах отдельных артефактов. Подобный механизм в ряде источников обозначается как параметризация.

2.1.1. Механизм переиспользования

2.1.1.1. Полная копия объекта

Рассматривая вопрос повторного использования артефакта стоит отметить ряд возможных реализаций подобного механизма.

Первый рассматриваемый вариант - использование полной функциональной копии. Первая из возникающих при этом проблем - проблема отслеживания изменений исходного элемента. Создание полной копии исходного объекта для некоторых систем потребует ручного отслеживания изменений исходного артефакта.

Данная проблема может быть разрешена путем автоматического отслеживания изменений, например, при помощи подписки копии на события изменений исходного артефакта как это показано на рисунке 7.

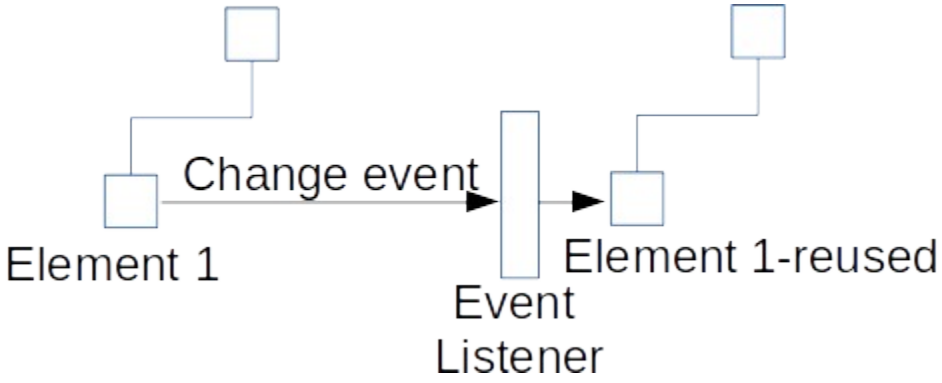

Рисунок 7. Пример создания полной копии с подпиской на изменения исходного артефакта

Рассматривая возможность подписки на изменения, стоит определить, что именно понимать под событием изменения.

При понимании события как пользовательского или внешнего запроса на изменение, дублирование артефакта потребует также дублирования вызова обработчиков пользовательского изменения.

Если под событием понимать изменение хранимых и отображаемых данных - реализация отслеживания подобных изменений возможна, но при этом часть функций, связанных с обработкой данных, не будет использоваться.

Можно сделать вывод о том, что использование полной копии исходного объекта в подобных условиях не будет являться целесообразным.

2.1.1.2. “Символьная ссылка”

Вторая реализация предполагает организацию связи между исходным артефактом и копией по аналогии с механизмом символьной ссылки в операционной системе Linux. При этом в месте переиспользования создается артефакт-ссылка, главной функцией которого является передача запросов на получение и обработку данных на исходный артефакт. Схематически подобное представлено на рисунке 8.

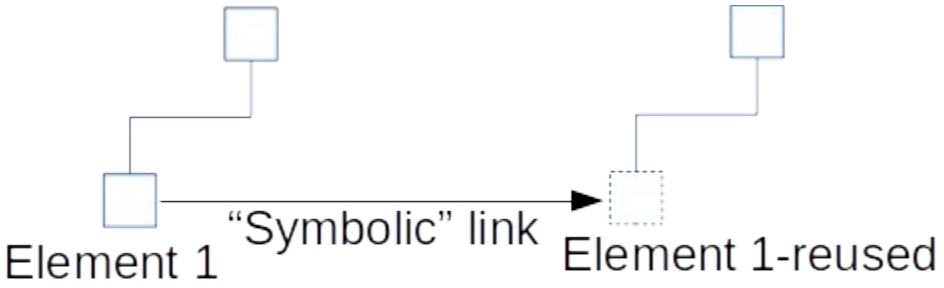

Рисунок 8. Пример создания “символьной” ссылки

В данной реализации также возможно возникновение определенных проблем.

Первое что стоит отметить - наличие функций, выполнение которых предполагается в определенном контексте. Как пример можно рассмотреть получение полного пути к артефакту, включающее идентификаторы всех его родителей. При использовании передачи запросов с места переиспользования, вызов данной функции вернет путь к исходному артефакту.

В данном случае решением может послужить добавление контекста в обработчики артефакта, но подобное решение является сложным в разработке и поддержке, в связи с чем его применение затруднительно.

Можно отметить более сложные варианты реализации артефакта - ссылки, при которых в копию добавляются механизмы обработки данных, но подобное решение выходит за рамки подхода.

Рассмотрим возможность возникновения необходимости в изменении свойств копии артефакта. Подобная ситуация может возникнуть, когда некоторые свойства отдельного артефакта отличаются от исходного, в то время как большинство остается схожими с исходным.

Самый простой вариант действий с подобными изменениями предполагает запрет на изменения копии артефакта. При этом в виде

дополнительной возможности можно рассмотреть создание в месте переиспользования независимого артефакта. Подобное решение, несмотря на свою ограниченность, позволяет описать несколько отличающихся артефактов для набора схожих за счет потери связи с повторно используемым объектом.

Более сложные варианты предполагают хранение изменений копии артефакта. Подобный подход позволяет избежать дублирования свойств исходного артефакта, при этом позволяя поддерживать изменения в копии.

Для символьной ссылки возможна реализация только механизмов запрета и создания полноценной копии, хранение дополнительной информации для артефакта - ссылки предполагает расширение возможностей копии и выходит за рамки подобного представления.

2.1.2.3. Промежуточный слой данных

Последняя из рассматриваемых реализаций предполагает создание промежуточного слоя хранилища данных, что становится возможным при разделении хранения и отображения данных, так, например, как это реализовано при использовании, Model-View-Controller (MVC) [19] или Model-View-Presenter.

При правильной реализации подобного подхода избегается дублирование данных, но сохраняются возможности вызова специфичных для класса артефактов функций и описания изменений каталога требований и свойств артефактов.

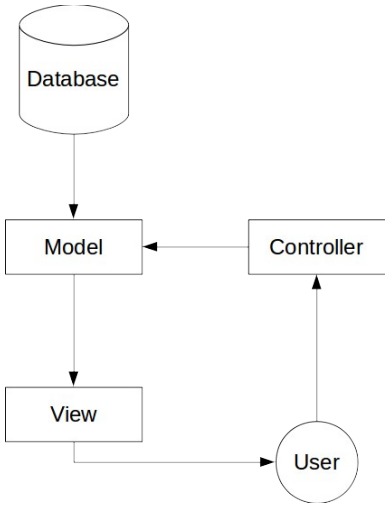

Рисунок 9. Model-View-Controller модель

По запросу на представление данных из хранилища для копии артефакта, возвращаются данные исходного объекта, дополненные хранимыми изменениями. При наличии изменений имеющихся свойств артефакта, механизм переопределяет соответствующее свойство для копии.

Для примера, рассмотрим наличие информации об изменении в копии свойства артефакта под названием text. При запросе на получение данных об этом свойстве промежуточный слой передаст хранимое измененное значение. Если же изменения для данного свойства не хранится, то передается значение свойства исходного артефакта.

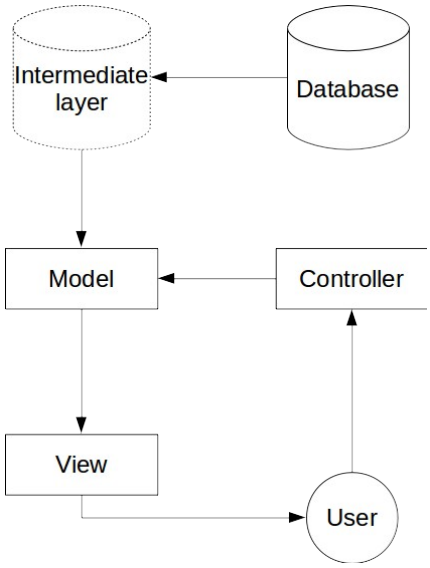

Рисунок 10. Model-View-Controller модель с промежуточным слоем данных

Использование промежуточного слоя позволяет обеспечить хранение информации об изменении, обеспечивая при этом отсутствие изменений в подходе к работе с данными. При этом предполагается использование копии артефакта с хранилищем данных в промежуточном слое.

Данное решение будет являться избыточным при запрете на изменения копий, но при этом дает возможность хранения изменений над повторно используемым объектом.

2.1.2. Механизм параметризации

Параметризация представлена во многих программных средствах, в том числе, является одной из ключевых функций для генераторов тестовых данных и тестовых наборов, которые в большинстве своем направлены на создание набора тестов и тестовых данных для проверки набора возможных значений определенных параметров.

Возможности описания параметров также встречаются в различных средах моделирования, например, в средствах управления бизнес процессами, таких как YAWL[20].

Рассматривая механизмы параметризации, стоит отметить две основные задачи - первая заключается в поддержке механизма описания параметров, вторая включает процесс определения значений параметров.

2.1.2.1. Поддержка описания параметров для механизма параметризации

Поддержка описания параметров заключается в определении возможных видов параметров и формата их представления.

В рамках данной работы было решено использовать часть видов свойств требований из стандарта ReqIF [17] для описания видов параметров.

Подобный выбор, в частности, позволяет адаптировать реализацию параметризации к выбранной программной области. Также выбор обусловлен широкой распространенностью стандарта и соответствием выбранного набора уже поддерживаемому программным средством, в рамках которого производится разработка.

Были выбраны представления параметров в виде строк (String), целочисленных значений (Integer), чисел с плавающей точкой, логического представления (Boolean) и списка строковых значений (List). Выбранный набор позволяет описать большую часть параметров, представленных в рассматриваемых на данный момент описаниях требований и тестовых ситуаций.

Рассмотрение возможных описаний механизмов поддержки параметров будет производится для древовидного описания каталога артефактов, при котором для артефакта определен отношение артефакта-предок и могут быть описаны несколько артефактов-потомков. При этом, в рамках данной работы будет применяться сквозная нумерация идентификаторов, как это показано на рисунке 11.

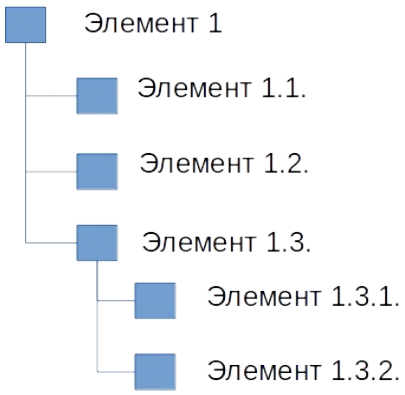

Рисунок 11. Пример иерархического представления каталога требований

Стоит уточнить, что понятие параметра может быть связано с понятием свойства артефакта. Под свойством артефакта (Attribute) будет пониматься

обладающая уникальным в пределах данного артефакта идентификатором переменная одного из возможных типов с определенным значением или набором значений (в случае списка строковых значений).

Свойство определяется только для одного артефакта, для параметров стоит отметить необходимость в определении доступности для разных артефактов. В частности, доступность артефактов применима для описания компактного описания набора схожих артефактов, что будет рассмотрено в пункте 2.2. данной работы.

Существует ряд способов, при помощи которых можно реализовать доступность значений переменных для разных артефактов. Часть из них, например, передача параметра к следующему заданию в YAWL[20], характерна только для определенного рода программных средств, но существуют более универсальные подходы.

В рамках данной работы реализовано использование нескольких вариантов доступности параметров, определение которой использует место артефакта в иерархии каталога требований. Для определения форм доступности параметров была проведена аналогия с YAWL [20] и средствами генерации тестовых наборов.

Среди реализованных форм доступности (scope) параметров были реализованы (рис. 12):

- доступность в пределах одного артефакта. По аналогии с представлением свойств, подобное решение предполагает доступность значений параметра только для одного артефакта
- доступность в пределах прямых потомков. Предполагает определение параметра для артефакта и его прямых потомков. Не предполагает доступность для потомков потомков. Подобное представление в частности удобно для описания схожих

артефактов (подробнее в пункте 2.2.), при этом препятствуя распространению значений параметра

- доступность для всего дерева потомков. Подобное описание может быть использовано для представления глобальных параметров на каталоге требований

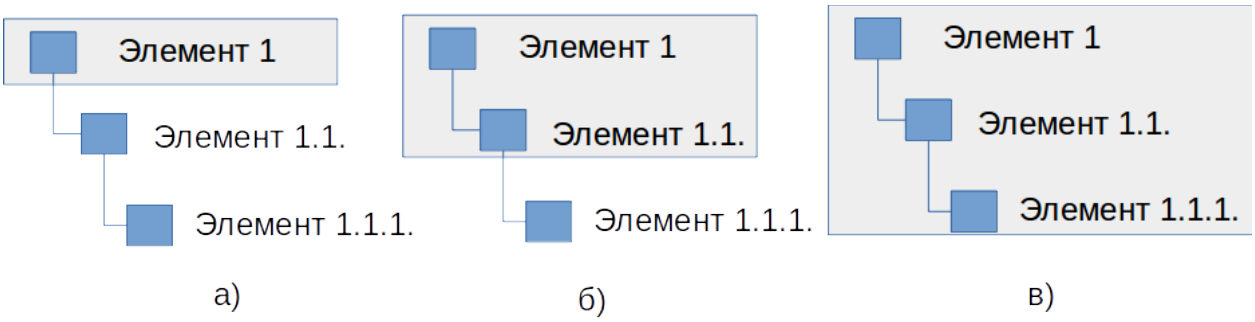

Рисунок 12. Виды доступности параметров (scope). Слева направо - а)доступность для артефакта, б)доступность для прямых потомков и в)доступность для всех потомков

Подобное представление позволяет определять параметры для набора артефактов, но в то же время избежать появления лишних параметров для отдельного артефакта.

Кроме рассмотрения особенностей описания представления параметров, стоит рассмотреть особенности реализации, в том числе порядка определения значения параметров при наличии конфликтов, которые рассмотрены подробней в пункте 3.

2.1.2.2. Определение допустимых значений параметров

Вопрос определения возможных значений параметров является одним из основных при описании механизмов параметризации. Можно рассмотреть ряд возможных подходов в этой области по аналогии с существующими решениями [21].

Первый и наиболее простой подход предполагает использование списка возможных значений параметров. Несмотря на то, что данное решение позволяет поддерживать необходимые значения параметра, при этом возникает проблема отсутствия формализации, оставляя возможность возникновения проблем с управлением изменениями значений, а также с потенциальным дублированием и противоречивостью.

В связи с этим, возникает необходимость в использовании более формальных описаний возможных значений, позволяя упростить процесс изменения списка значений и получить более компактное описание значений параметра. Решением проблемы может послужить использование генераторов значений параметра, подобно используемым в средствах генерации тестовых наборов механизмам.

Среди подходов к генерации значений параметров можно отметить возможность использование набора часто используемых генераторов значений простых типов.

Так, можно рассмотреть генерацию возможных значений целочисленного параметра в виде значений от указанного минимального до максимального с фиксированным шагом. Кроме этого, возможно описание генерации набора псевдослучайных значений в выбранном диапазоне и указанном количестве, для параметров с плавающей точностью - с указанным числом знаков после запятой.

Можно отметить возможность генерации значений согласно определенной математической функции. Примером может послужить необходимость описания параметра, для которого все значения могут быть представлены некоторым распределением. Подобный параметр, в частности, может быть использован для моделирования некоторых сигналов.

Подобный подход позволяет обеспечить генерацию наиболее часто встречающимся способам определения параметров, но не позволяет определить более сложные значения, например, сумму двух параметров. Для решения этой проблемы возможно описание генераторов для простых зависимостей между параметрами, но количество возможных зависимостей велико, так что подобное решение потребует больших затрат на разработку и поддержку.

Рассматривая проблему ограниченности количества генераторов, возможно использование механизмов расширяемости, с использованием которых становится возможным добавить новый генератор при появлении необходимости, но при этом возникает вопрос отображения и выбора доступных генераторов, количество которых при длительном расширении доступных функций будет расти.

Следовательно, возникает необходимость в возможностях для описания переменных, значения которых зависят от других переменных. Рассмотрим решения, призванные реализовать подобную функцию.

Одно из наиболее простых в реализации, но сложных в поддержке - это уже упомянутое использование простых генераторов для предопределенного набора отношений между параметрами. При этом возникает проблема поддержки большого и увеличивающегося набора генераторов.

Более перспективным выглядит использование языка описания отношений между параметрами. Рассматривая значения параметров как множества, можно выделить возможность описания значений параметров при помощи операций над множествами. При этом описание зависимости будет представлять собой описание операции с некоторым набором доступных множеств.

Подобный подход позволяет описать ряд операций над параметрами, но при этом более простым в реализации и удобным в использовании является описание операций над значениями параметров по определенной формуле, при которых параметры используются как одно из своих значений.

Подобное использование ограничивает возможности для работы с наборами значений, но при этом позволяет решить большую часть описания зависимости параметров, являясь более понятным и менее требовательным к навыкам для пользователя.

Независимо от выбранной реализации, описание параметров с зависимым от других параметров значением поднимает вопрос определения порядка вычислений значений этих параметров. Одна из наиболее явных причин подобного - возможность появления циклической зависимости между параметрами.

В виде частичного решения проблемы определения порядка вычислений возможно использование нескольких проходов генераторов значений параметров. При этом значение в случае зависимости между параметрами может оказаться некорректным.

Более корректным будет являться анализ описанных в генераторах формул с последующем построением графа зависимости параметров [21]. Данное решение позволяет определить порядок вычислений методом топологической сортировки [22]. При наличии в полученном графе циклов потребуется свести граф к дереву, например, путем удаления одной из связей или информирования пользователя о наличии проблемы с отказом от вычисления значения одного из параметров.

Использование моделей зависимости усложняется при наличии различных уровней видимости параметров - по сути, требуется отдельное определение зависимостей для каждого артефакта. При этом стоит отметить, что подобное решение будет являться наиболее корректным.

В виде промежуточного решения можно рассмотреть хранение графа зависимостей не в виде отдельной модели, а в виде описания зависимостей непосредственно в параметрах. Для получения значений в месте описания параметров подобный подход выглядит подходящим.

Был рассмотрен ряд подходов к реализации механизма определения значения параметра, но при этом остался не рассмотренным вопрос определения места вычисления значения параметра. Подобный вопрос, в

частности, возникает для параметров, значение которых зависит от других параметров, видимость которых определена не только для одного артефакта.

Рассмотрим ситуацию представленную на рисунке 12. в варианте б). Если определить в Элементе 1. параметр x, значение которого зависит от некоторого параметра y, видимость которого определена для прямых потомков, то может возникнуть ситуация, в которой y определен для Элемента 1. но не определен или имеет другое значение для Элемента 1.1.

В данной ситуации наглядно прослеживается необходимость определения определенного способа вычисления значений - в момент определения параметра или же в момент использования его значений.

Решение этого вопроса является достаточно важным с точки зрения количества требуемых вычислений - определение значений в момент появления параметра является более простым решением, но значение параметра для потомков артефакта в котором он был определен могут оказаться для пользователя неожиданным.

Вычисление значения в момент использования параметра потребует вычисления его значения для каждого артефакта, в свойствах которого он встречается, что может быть затруднительным, особенно при необходимости построения модели зависимостей параметров.

При этом вычисление значения в момент использования позволит более корректно определить значение переменной, позволяя определять параметры - формулы, например, параметр ожидаемая температура равный параметру входная температура деленному на 4.

На данный момент использование подобных формул в указанном виде не предполагается, в связи с чем значение параметров будет вычисляться в месте описания генератора значений параметра. При этом сохраняется

возможность использования вычисления в момент использования параметра при расширении механизмов отслеживания использования параметров.

2.2. Поддержка компактного описания схожих артефактов

Рассматривая вопрос поддержки компактного описания набора схожих артефактов можно выделить ряд проблем, часть из которых была рассмотрена в пункте 2.1.

В различных программных средствах для получения компактного описания используется несколько подходов, из которых можно отметить описание модели и поддержку обобщенного описания объекта с возможностью определения его копии с использованием набора значений параметров.

В рамках данной работы предполагается использование обобщенных описаний артефактов. Использование генераторов требует более серьезных изменений механизмов программного средства, при этом не предоставляя значимых преимуществ.

Здесь и далее может быть упомянута генерация артефактов - при этом будет имеются в виду создание некоторого набора требований с использованием обобщенного описания артефакта.

Использование обобщенного описания предполагает обеспечение для артефакта возможности указания в его свойствах (в том числе в описании (description) требования) имен параметров для последующей замены их значениями (подробней в разделе реализации).

На данный момент использование параметров предполагается только при повторном использовании, но в будущем механизм возможно будет расширен до использования в каталоге требований. Подобное решение позволит сократить затраты на поддержание каталога требований.

Использование данного подхода предполагает создание нового вида артефактов, задачей которого является хранение данных об описании набора схожих артефактов, для которого задан исходный артефакт и набор параметров.

Стоит рассмотреть вопрос определения количества копий при переиспользовании. В связи с использованием механизма компактного описания артефактов для целей тестирования, было решено использовать сочетания значений выбранных параметров.

На данный момент применяется метод получения декартова произведения всех возможных значений участвующих параметров, но в дальнейшем предполагается использование других методов покрытия, включая но не исключая покрытие всех пар значений.

Отдельно стоит отметить, что использование Requality не ограничено поддержкой каталога требований и проектированием тестов - полученный каталог может быть применен в дальнейшем для верификации системы. При этом для конечного пользователя во втором случае удобнее не производить отображение артефакта для повторного использования, показывая копии артефакта как детей артефакта-родителя артефакта для переиспользования как это показано на Рисунке 13.

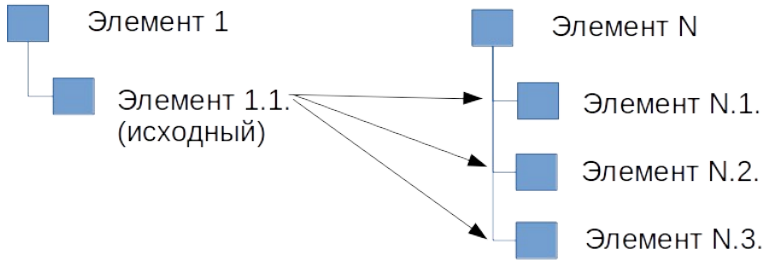

Рисунок 13. Результат использования механизма компактного описания схожих артефактов(справа), полученных с использованием артефакта 1.1.

Дополнительно стоит отметить, что для нового артефакта происходит замена параметров, которые применялись при получении копий исходного артефакта значениями, участвующими в построении декартова произведения.

Для примера рассмотрим построение набора схожих артефактов для параметров x со значениями [0,1] и параметра y со значениями [1,2]. При построении сочетаний значений по правилам декартова произведения получим 4 возможные пары - [x=0,y=1], [x=0,y=2], [x=1,y=1], [x=1,y=2].

Каждому сочетанию в соответствие будет приведен новый артефакт с порядковыми номерами от 1 до 4. При этом, при упоминании параметров x и y в свойствах используемого исходного артефакта, они будут заменены значением из соответствующего сочетания. Так, для артефакта номер 3 x будет заменен на 1, а y заменен на 1.

3. Особенности реализации механизмов переиспользования и параметризации

3.1. Особенности программного средства Requality.

3.1.1. Рассмотрение особенностей программной реализации

Рассматривая расширение возможностей программного средства, следует отметить необходимость внедрения новых функций в уже существующую программную систему.

До написания данной работы, Requality содержал набор механизмов, функциональные возможности которых были затронуты в процессе реализации механизмов параметризации и переиспользования. Для более подробного рассмотрения изменений, стоит рассмотреть особенности строения и функционирования инструмента.

Реализация Requality на данный момент может быть рассмотрена как совокупность трех наборов классов - представления данных, модели каталога

требования, и элементов пользовательского интерфейса, использующих операции над моделью.

Реализация рассмотренных в пункте 2. механизмов затронула все указанные модули, в большей степени влияя на модель и механизмы ее работы с данными. Рассмотрим подробнее описания измененных в ходе выполнения данной работы механизмов.

Для получения представления о функционировании программного средства Requality до внесения изменений, приведем подмножество классов исходного кода, представленное на рисунке 14.

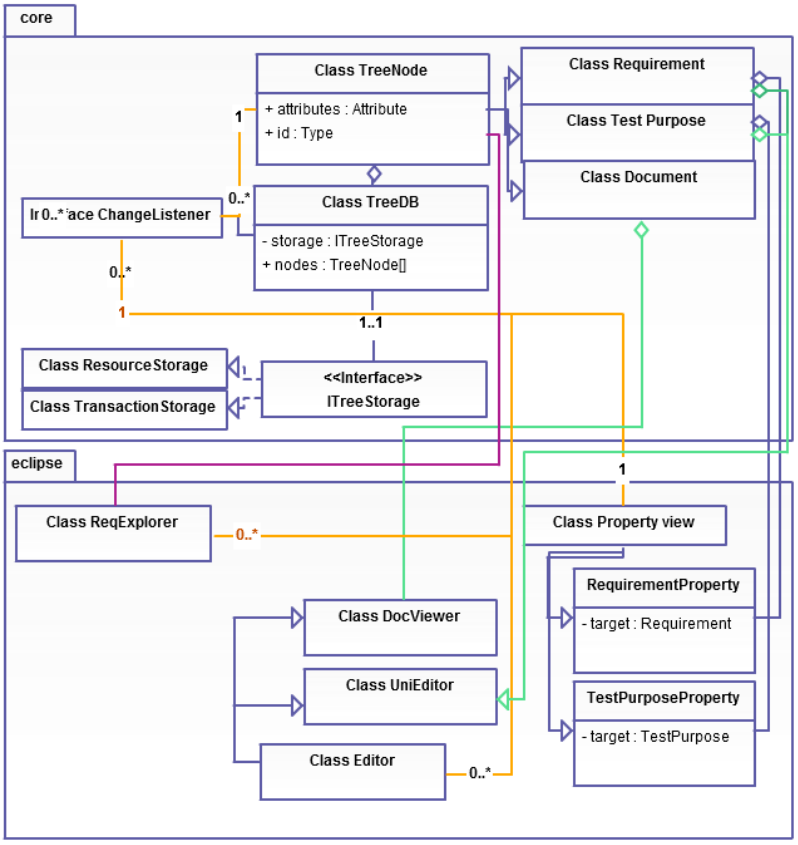

Рисунок 14. Модель архитектуры Requality

На данном рисунке классы условно разделены на две группы - работающих со средой eclipse и обеспечивающих основные функциональные возможности в группе core. В данной модели обозначен ряд основных

механизмов, изменение которых было произведено в ходе выполнения данной работы.

На модели обозначен интерфейс доступа к данным ITreeStorage с двумя реализациями, которые будут упомянуты позднее. Также указан механизм поддержки модели каталога артефактов TreeDB.

На схеме обозначен класс артефакта Tree Node, содержащего основные методы работы с артефактом, включая описания свойств одного из видов рассмотренных в пункте 2.1.2.1. Также модель содержит некоторые виды артефактов, включая требование (Requirement), документ (Document) и тестовую ситуацию (Test Purpose), применимые в рамках приведенного во введении жизненного цикла тестов[2].

Последний из рассмотренных в модели элементов из группы core - механизм подписки на изменения, обозначенный как Change Listener. Данный интерфейс предполагает возможность подписки одного объекта на изменения другого.

До выполнения данной работы предполагалась возможность подписки на все возможные изменения в модели, описываемой классом TreeDB, или подписки на изменения отдельного объекта. При этом возможно каскадное нарастание числа вызовов реакции на изменение при наличии сложных зависимостей между артефактами.

Реализованные в данной работы подходы к параметризации и повторному использованию предполагают возможность реагирования на набор связанных изменений. Так, при рассмотрении параметризации, в частности, параметров, видимость которых определена над набором артефактов, при изменении артефакта возникает необходимость в вызове сразу нескольких реакций на его изменение в местах его использования для обновления пользовательского интерфейса.

При рассмотрении переиспользования подобная ситуация возникает при изменении исходного артефакта, которое требует вызова реакции на изменение во всех местах переиспользования.

Программная реализация механизмов подписки на события предполагала реакцию системы на каждое событие независимо от остальных. В вышеизложенных ситуациях данное решение может послужить причиной многократного вызова методов обновления данных, которых в ряде случаев возможно избежать.

Стоит также упомянуть упомянутый в модели класс Attribute, который представляет собой реализацию хранимого свойства артефакта. До выполнения данной работы существовала определенная схема работы с свойствами, предполагающая единовременную выгрузку свойств в кэш свойств артефакта при его загрузке с отслеживанием изменений набора артефактов в кэше, призванном обновить информацию в хранилище данных.

Перейдем к рассмотрению классов группы Eclipse, включающих в себя средства пользовательского интерфейса, реализуемые с использованием механизмов работы со средой Eclipse. Для этого следует выделить ряд объектов, используемых для реализации дополнений для данной среды разработки.

В рамках представления архитектуры Eclipse применяется ряд сущностей, включая активное рабочее поле (workspace), представление объекта в виде описания ресурса (Resource), проекта, которому в соответствие можно сопоставить каталог требований и элементов пользовательского интерфейса, обозначаемых как виды (view).

Среда предоставляет механизмы расширения описания ресурсов, доступных к отображению видов (View), в том числе окон свойств выбранного в рабочем поле ресурса (Property), механизмов реакции на

команды, включая нажатия сочетаний клавиш и выбор пунктов меню, описания дополнительных меню, включая контекстные, механизмы истории изменений, включая отмену (Undo) и повтор (Redo) действий.

При рассмотрении указанных возможностей, стоит отметить наличие возможного обмена данными между активными элементами пользовательского интерфейса, включая, но не исключая установку упомянутого выше выбранного в рабочей среде ресурса.

Более наглядно элементы среды будут показаны в пункте 3.1.1. в рамках рассмотрения реализации с точки зрения пользовательского интерфейса.

Предложенная на рисунке 14 модель описывает реализацию трех видов расширений среди Eclipse, включая свойства выбранного ресурса, определенные для артефакта и обозначающие доступные параметры выбранного артефакта, вид иерархического представления каталога требований (ReqExplorer), отображающего элементы каталога требований в форме их уникальных идентификаторов и виды-редакторы выбранного артефакта, обозначенные как классы наследники класса Editor. Остался не рассмотренным в модели, но расширенным в ходе выполнения работы, механизм работы с обработчиками (handlers), в том числе используемый для контекстных меню и команд.

Все указанные элементы используют один из видов подписки на изменения, для представления каталога требований и для редактора выбранного артефакта - подписка на изменения Tree DB, для окна свойств выбранного артефакта - подписка на изменения конкретного артефакта.

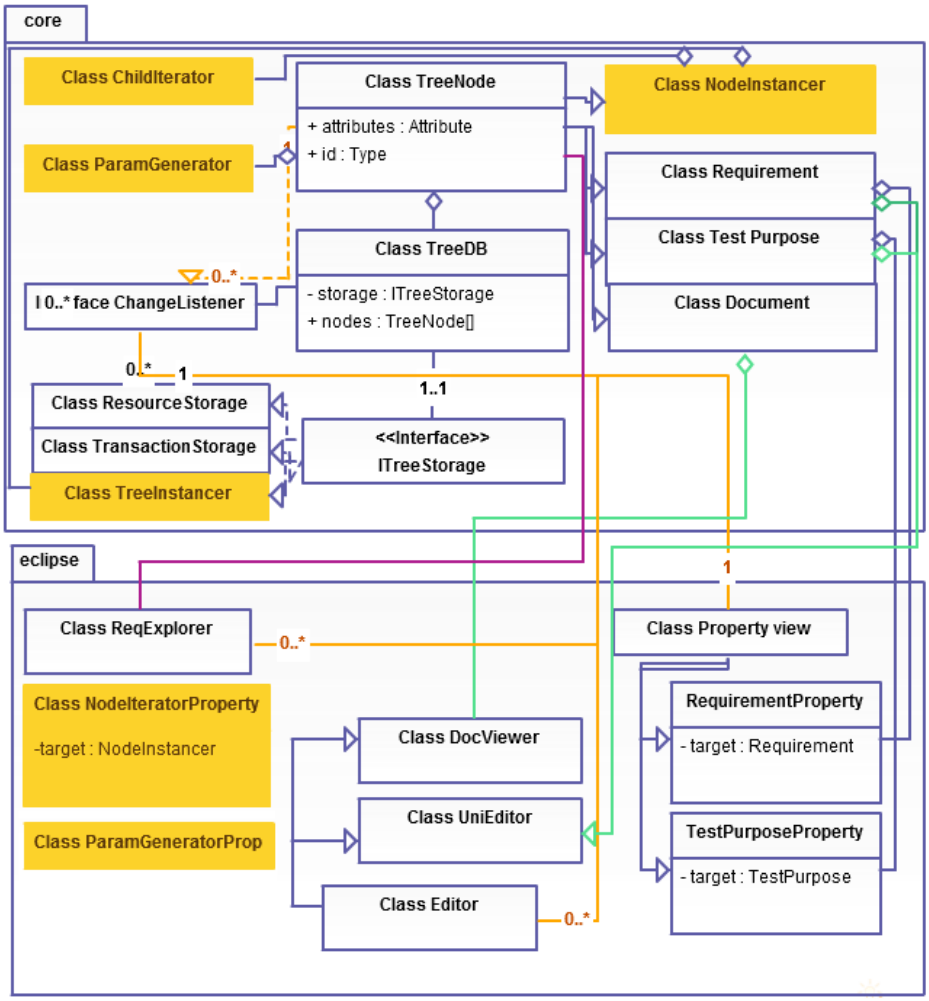

Рисунок 15. Измененная модель архитектуры Requality

Рассмотрим изменения модели в порядке рассмотрения ее элементов до изменения.

Как уже было сказано выше, механизмы хранения данных до выполнения данной работы были представлены классами - наследниками интерфейса ITreeStorage, среди которых активно используются в работе Requality два - Resource Storage и Transaction Storage, отвечающие за хранение данных механизмами среди Eclipce и хранение данных во время запущенной транзакции (которые будут рассмотрены позднее).

В рамках выполнения данной работы был разработан механизм представления данных, функционирующий как промежуточный слой между

действующим хранилищем данных, в роли которого в большинстве случаев выступает хранилище с использованием механизмов Eclipse и механизмами работы с данными.

Задачей подобного хранилища является поддержка свойств артефактов-копий в механизме переиспользования артефактов. Подобное решение, реализованное согласно рассмотренному в пункте 2.1.1. варианту реализации с использованием промежуточного хранилища данных, обладает рядом преимуществ.

Подход позволяет реализовать как решение с отсутствием возможности изменения данных, так и механизмы хранения некоторого набора изменений. В рамках выполнения данной работы было реализовано решение с блокировкой возможностей редактирования копий исходного артефакта с возможностью создания полноценной версии артефакта при выполнении определенных действий. При этом часть свойств артефактов-копий, в частности их идентификаторы являются измененными, хотя редактирование их пользователем на данный момент невозможно.

Несмотря на недостатки, данное решение может быть использовано для описания сложных форм параметризации. Например, становится возможным переиспользование фрагмента каталога требований в котором также было применено переиспользование.

Возможность редактирования артефактов - копий на данный является одним из направлений дальнейшего развития механизма переиспользования, но при этом поднимает ряд вопросов, среди которых:

- вопрос хранения изменений отдельного артефакта, на данный момент решенный за счет хранения набора измененных параметров

- вопрос хранения и обработки изменений каталога требований, на данный момент не решенный, но также предполагающий возможность решения за счет хранения в атрибутах артефакта для переиспользования набора пар номер - значение, где номер обозначат определенную операцию над деревом требований

Стоит отметить, что реализация возможностей переиспользования с применением промежуточного представления данных позволяет использовать артефакты - копии как обычный артефакт, созданный при помощи стандартных механизмов работы с артефактами.

Доступный набор артефактов был расширен введением Instancer Node, представляющего собой артефакт, реализующий функцию переиспользования.

Данный объект предоставляет интерфейс выбора исходного артефакта и определения метода выбора параметров по которым будет строится сочетания их значений.

Включает в себя промежуточное хранилище данных, наследника Tree Storage для которого возможно хранить описание копии фрагмента каталога требований. Для хранилища данных в дальнейшем предполагается реализация хранения изменений фрагментов каталога требований и их элементов в свойствах Instancer Node.

Также рассматриваемый вид артефактов содержит объект класса Child Iterator, содержащего методы построения для набора параметров множества наборов сочетаний значений параметров.

При этом, в классе Child Iterator производится переопределение значений параметров на локальное представление в виде значений переменных, используемых для построения набора и равных значению переменной в соответствующем сочетании.

Для определения идентификаторов копий повторно используемого артефакта имеется возможность описания формулы, по которой они будут определяться, подобно генератору значения параметра по формуле, реализация которого будет описана далее.

Элементы-копии в общем случае отображаются программным средством как потомки артефакта Instancer Node. Подобное решение обусловлено тем, что отображение копий для выбранного артефакта и отображение копий как потомков предка Instancer Node, не обозначают связей между копией и артефактом ее породившем.

Также была реализована возможность скрытия Instancer Node, при использовании которой, вместо данного артефакта механизмы Requality будут отображать его потомков. Для примера можно рассмотреть рисунок 16, на котором обозначено требование с потомком в виде Instancer Node, для которого реализовано переиспользование некоторого артефакта, на основе которого было получено две копии. После скрытия Instancer Node механизмы работы с каталогом требований будут воспринимать копии как потомков “Требования” из примера.

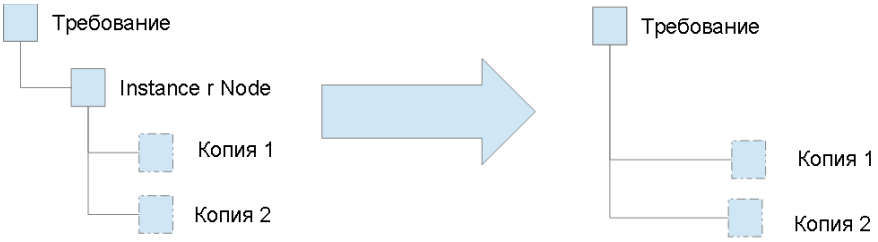

Рисунок 16. Скрытие Instancer Node

Также стоит отметить, что введение механизмов переиспользования послужило причиной повсеместного ввода атрибута _name до этого используемого только в требованиях для всех видов артефактов.

До выполнения данной работы большая часть требований использовало id, то есть изначально генерируемые, но редактируемые идентификаторы артефактов, связанные с фактическим способом хранения артефактов.

При этом для некоторых видов объектов имелась возможность описания дополнительного идентификатора _name, отображаемого вместо исходного идентификатора.

Механизм переиспользования ввел представление виртуальных артефактов, для которых идентификатор по ряду причин, включая, но не исключая то что фактически _id не является свойством артефакта и не участвует в механизмах работы с параметрами.

В данной ситуации, чтобы сохранить возможность описания пользовательских названий артефактов, было решено расширить возможности атрибута _name, включая изменения пользовательского интерфейса.

Механизм реагирования на изменения также был изменен по рассмотренным выше причинам. Было определено два вида возможных реакций на изменения - названный gui вид изменений предполагает отсутствие дальнейших изменений в рамках реакции на изменения и второй вид оставляет возможным новых изменений модели в ходе реакции на изменение.

Первый вид изменений используется по большей части для подписчиков на изменения со стороны пользовательского интерфейса, откуда и получил свое обозначение.

В рамках появления нового способа подписки на изменения, становится возможным подписка на набор gui изменений в виде одной реакции.

Подобное становится возможным, в частности, в связи с особенностями реализации изменений данных в инструменте Requality, которая определяет набор изменений модели в форме набора действий с единым названием, так называемую транзакцию.

Примером подобного могут послужить действия связанные с удалением артефакта. При этом будет проводиться транзакция “Удаление артефакта”, в рамках которой будет произведен набор элементарных изменений данных.

Система транзакций поддерживается для использования механизмов Undo\Redo среды Eclipse, при этом, как уже было сказано, позволяя получить связанный набор изменений в форме одного пакета.

Возможность объединения событий позволяет сократить их количество. На данный момент реализован ряд простых сокращений набора событий. Например, события об изменении требования может быть удалено при наличии события о его создании, в связи с стем, что появление событие о создании для пользовательского интерфейса загрузит текущий набор данных.

На данный момент выделение набора событий позволяет сократить их количество и обеспечить небольшой прирост скорости работы за счет последовательности реакции на изменения.

В дальнейшем предполагается реализация возможности подписки на наборы событий, а также поддержка наборов событий для слушателей из второй категории для которых возможно изменение в рамках реакции на изменение.

Для описания значений параметров было принято решение использовать представление значений параметров при помощи имеющегося механизма описания свойств артефактов. При этом указанные значения атрибутов отображаются в пользовательском интерфейсе, но не сохраняются

в хранилище данных. В дальнейшем возможно добавление отдельного способа представления параметров.

Сами параметры определяются генераторами значений, при этом для генераторов значений имеется возможность определения видимости генерируемых значений. Порядок замещения параметров с одинаковым именем предоставлен на рисунке 17.

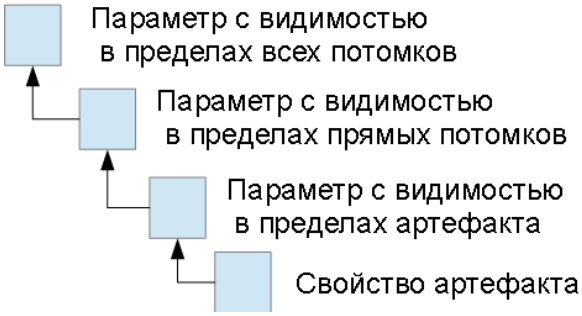

Рисунок 17. Приоритет видимости свойств и значений параметров

Согласно данному рисунку, наиболее приоритетными являются определенные пользователем свойства, при отсутствии свойств с указанным именем используется значение параметра с видимостью для конкретного артефакта. При отсутствии их будут использовано значение параметра с видимостью для прямых потомков.

Стоит отметить, что на данный момент реализовано несколько видов генераторов значений параметров, большинство которых представляет собой примеры генераторов значений независимого параметра по определенной закономерности.

На данный момент реализована генерация набора псевдослучайных целочисленных значений от указанного минимума до максимума, генерация последовательности целочисленных значений от указанного минимума до максимума с определенным шагом. Список доступных простых генераторов будет расширяться при получении запросов со стороны пользователей.

Реализован более сложный механизм генерации вычислимых значений параметров, для которого возможно определить формулу на языке JavaScript результат вычисления которой будет возвращен в виде одного вычисленного значения.

При этом в формуле имеется возможность использования имеющихся имен параметров и атрибутов, значение которых передается в механизм вычисления, представленный средствами библиотеки Rhino. Данное средство было добавлено в механизм работы с артефактами для возможности использования доступных атрибутов и значений параметров.

Стоит также отметить механизм описания зависимостей между параметрами, реализованный для генераторов значений параметра по формуле. При упоминании в формуле идентификаторов других параметров или атрибутов производится подписка генератора на изменения параметра. Инициализация зависимостей производится при загрузке генератора и при изменениях формулы.

Отдельно стоит отметить механизм описания нередактируемых артефактов, для которого реализован механизм генерации исключения при попытке изменения свойств артефакта или фрагмента каталога требований, являющихся переиспользованными копиями. Подобное решение призвано обеспечить корректность работы с виртуальными артефактами даже при работе дополнений к Requality.

Для некоторых не обозначенных в модели классов дополнительно была произведена адаптация используемых механизмов для возможностей работы с виртуальными требованиями. Подобное изменение, в частности, потребовалось по причине привязки ряда методов к определенному виду хранилища данных. Примером могут послужить методы, использующие исключительно ресурсы из Resource Storage.

Изменения классов группы eclipse включают частичное изменение механизмов отображения артефактов, в том числе возможность скрытия артефактов, на данный момент применяемая для Instancer Node.

Также изменен способ отображения идентификаторов артефактов, распространяющий использование атрибута _name, являющегося назначаемым пользователем свойством - идентификатором.

Добавлены отображения (view) для редактирования свойств артефакта Instancer Node и изменения доступных генераторов параметров для выбранного артефакта. Подробней изменения пользовательского интерфейса будут рассмотрены в пункте 3.1.1.

Стоит дополнительно отметить механизм замены копии повторного используемого элемента полнофункциональной копией. Подобное решение было реализовано для поддержки возможности модификации одной из копий, что может быть использовано для частичного редактирования подобного списка. Стоит отметить, что при этом подписка на изменения исходного артефакта отменяется.

При этом как уже было упомянуто, полнофункциональная копия представляется как потомок предка Instancer Node. Если Instancer Node скрыт и имеются два артефакта с одинаковым идентификатором, то приоритет имеет полнофункциональная копия.

3.1.1. Рассмотрение изменений пользовательского интерфейса

Для рассмотрения изменений пользовательского интерфейса Requality следует рассмотреть основные элементы пользовательского интерфейса предоставляемые средой. Первое что стоит отметить - Eclipse поддерживает так называемую перспективу, под которой понимается определенный набор открытых отображений (view) и их расположений. Плагин Requality содержит

описание соответствующей перспективы, отображение которой можно видеть на рисунке 18.

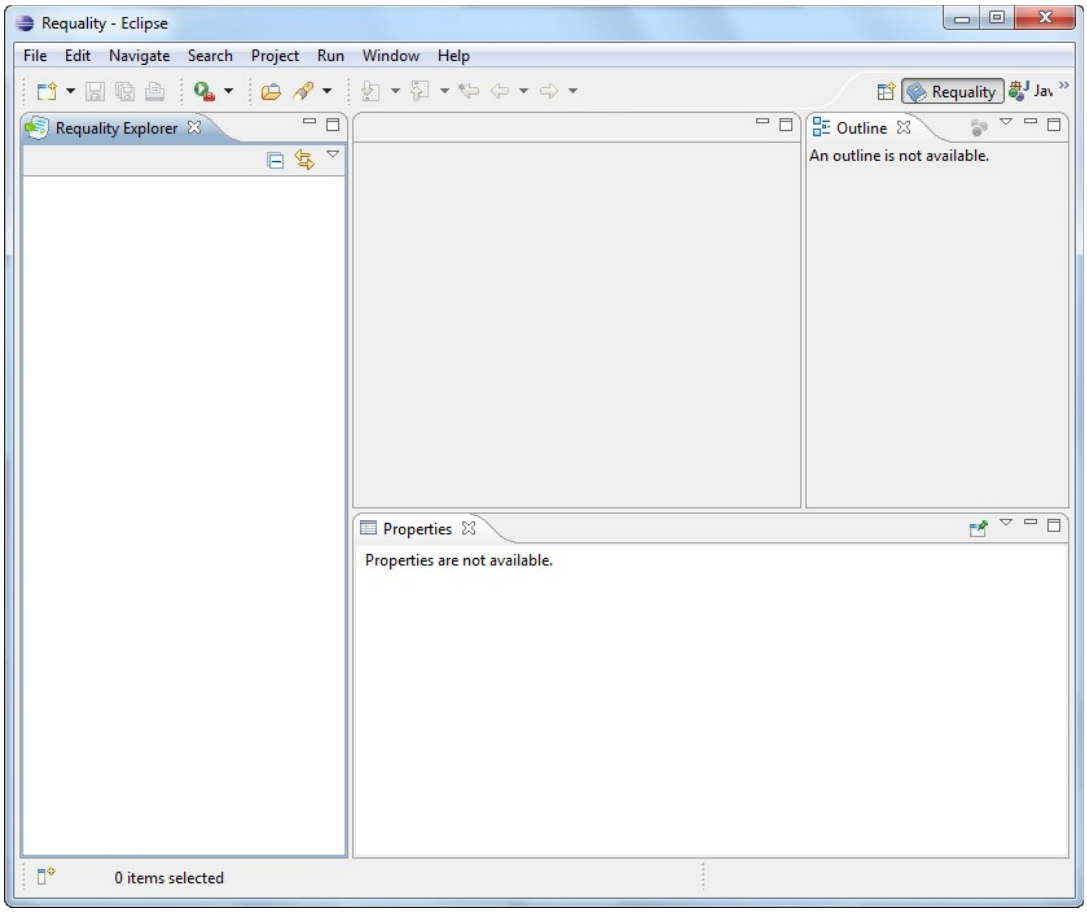

Рисунок 18. Перспектива Requality

Данное представление содержит несколько основных отображений (view), включая[18]:

1. **Requality Explorer** – отображает каталоги артефактов разделенные на отдельные проекты. Каждый каталог содержит поддерево документов, поддерево требований с тестовыми ситуациями и комментариями и поддерево отчетов проекта. *'Requality Explorer'* аналогичен стандартному для Eclipse виду *'Package Explorer'*, но предназначен для работы с проектами в формате *'Requality'*. Проект *'Requality'* содержит следующие компоненты:

a. **Папка Documents** – в ней размещаются все импортированные в проект документы с требованиями.

b. **Корневое требование Requirements** – контейнер для всех требований проекта, при этом сам являющийся требованием. Не может изменять свое корневое положение в иерархии требований и имеет неизменный идентификатор *'Requirements'* (но имя(атрибут _name) данного узла может изменяться). Кроме требований, в этом контейнере также располагаются тестовые ситуации и комментарии.

c. **Папка Reports** – в ней размещаются отчеты.

2. **Requality Markup Editor** – редактор документов. В нем открываются импортированные документы с требованиями, здесь же они размечаются на фрагменты для требований.

3. **Properties** – окно, в котором отображаются свойства выделенного артефакта. Содержимое окна Properties разбито на несколько вкладок. Для различных объектов набор и содержимое вкладок отличаются.

4. **Outline** – при открытом Requality Markup Editor отображает список отмеченных фрагментов требований. Причем отображаются фрагменты только того документа, который в данный момент открыт в *'Markup Editor'* и находится в фокусе. При открытом UniEditor отображает требование над которым редактор был открыт со всеми его потомками.

5. **UniEditor** – редактор, предоставляющий возможность удобного манипулирования и редактирования требований, тестовых ситуаций и комментариев. Открывается в том же окне, что и *'Requality Markup Editor'*.

Как уже было упомянуто выше, были внесены изменения в работу всех указанных представлений, исключая редактор разметки документа.

Первое что можно отметить относительно произведенных изменений пользовательского интерфейса — введение нередактируемых артефактов, в результате которого потребовалось поменять большинство окон свойств артефактов.

При повторном использовании для исходного элемента поля редактирования свойств артефакта активны (рис. 19), а для его копии — не активны, при этом для копии отображается специальный символ в левой-нижней части артефакта (отмечено красной стрелкой, рис. 20).

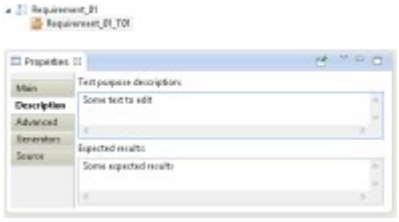

Рисунок 19. Тестовая ситуация до повторного использования

Рисунок 20. Копия тестовой ситуации после повторного использования

Появление нередактируемых копий в реализации механизма переиспользования послужило причиной возникновения возможности замены нередактируемой копии полной копией артефакта (рис. 21).

Подобное решение позволяет частично обеспечить возможность изменения повторно используемые артефактов. На данный момент

реализована возможность замены непосредственной копии повторно используемого элемента полной копией. Для элементов поддерева артефакта описание подобного механизма является затруднительным, в связи с необходимостью описания механизма схожего с хранением изменений каталога требований.

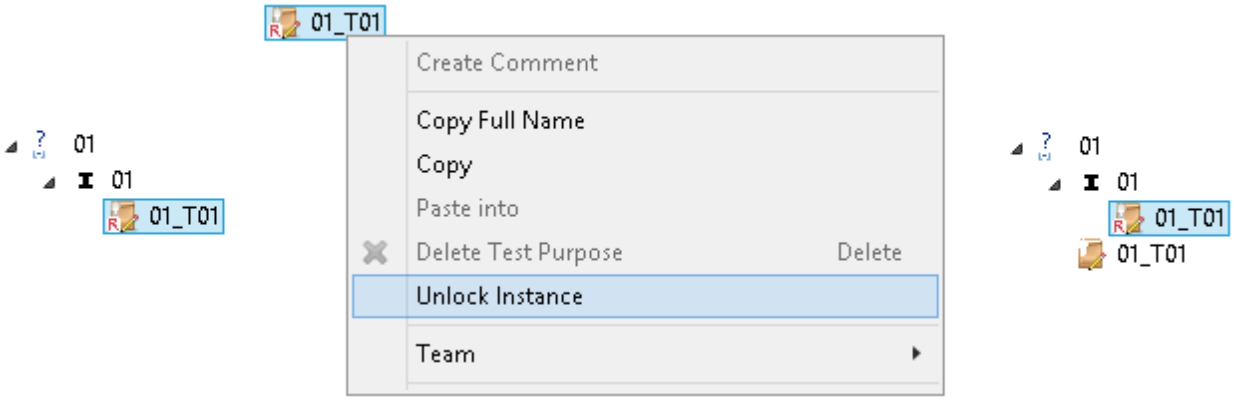

Рисунок 21. Получение полной копии из нередактируемой копии - потомка Instancer Node

Кроме появления нередактируемых артефактов стоит отметить добавление уже упомянутого выше артефакта Instancer Node, для которого реализовано отображение в дереве каталога требований с возможностью скрытия(рис. 22).

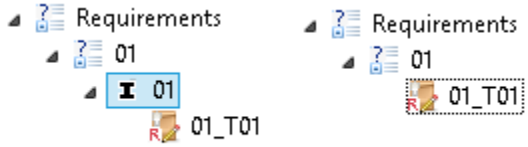

Рисунок 22. Instancer Node в каталоге требований в обычном (слева) и скрытом (справа) виде

При этом данный артефакт также может быть нередактируемым, если он находился в повторно используемом фрагменте каталога требований, как это показано на рисунке 23.

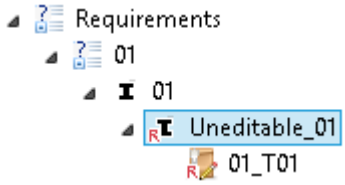

Рисунок 23. Нередактируемый артефакт Instancer Node

Рассмотрим изменения редактора фрагмента каталога требований UniEditor. На рисунке 24 можно видеть отображение редактора для фрагмента каталога требований из рисунка 23. При этом артефакт Instancer Node не отображается даже в случае если он не является скрытым. Вместо него отображаются потомки подобно функциональности отображения каталога требований в Req Explorer. Редактирование нередактируемых артефактов блокируется.

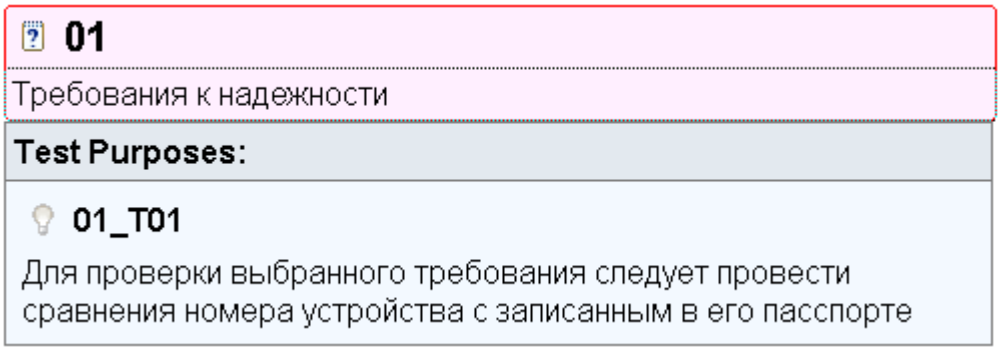

Рисунок 24. Отображение фрагмента дерева требований из Рисунка 23 в Uni Editor

Рассматривая изменения окон свойств стоит рассмотреть добавление новой вкладки под названием Generators отвечающей за отображение генераторов параметров. На рисунке 25 можно видеть пример отображения вкладки на котором показан список выбора доступных генераторов значений параметров с возможностью добавления нового элемента, удаления или редактирования выделенного.

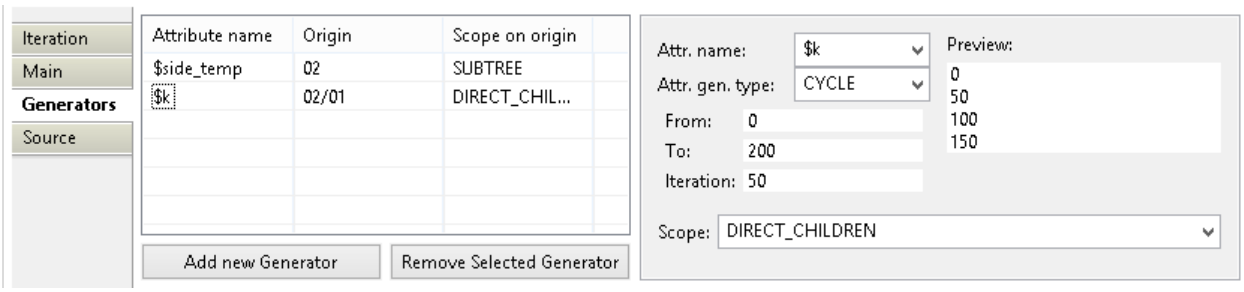

Рисунок 25. Вкладка Generators для артефакта из каталога требований

На данный момент вкладка Generators отображается для артефактов встречающихся в поддереве требований проекта по причине отсутствии необходимости на данный момент в применении параметров для документов и отчетов, хотя подобная возможность технически поддерживается.

Как уже было сказано выше, поддерживаются три вида генераторов - генератор целочисленных значений от указанного минимума до максимума с определенным шагом, генератор случайных значений в выбранном диапазоне и количестве и генератор значения параметра по формуле.

На рисунке 26 показана функция выбора формы доступности генератора - LOCAL соответствует видимости для одного артефакта, DIRECT_CHILDREN соответствует прямым потомкам, а SUBTREE обозначает доступность для всех потомков. Примеры разных видов видимости можно видеть на рисунке 26.

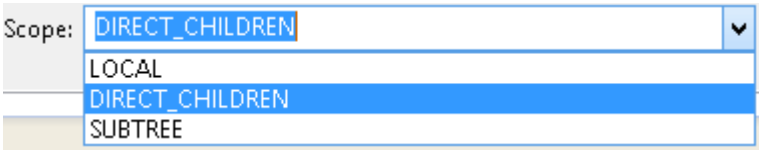

Рисунок 26. Выбор доступности генератора параметров

Стоит также рассмотреть окно свойств артефакта повторного использования. На рисунке 27 показана вкладка Main свойств на которой можно видеть идентификатор артефакта, список его свойств и сгенерированных значений параметров. Подобные вкладки реализованы для большинства имеющихся видов артефактов.

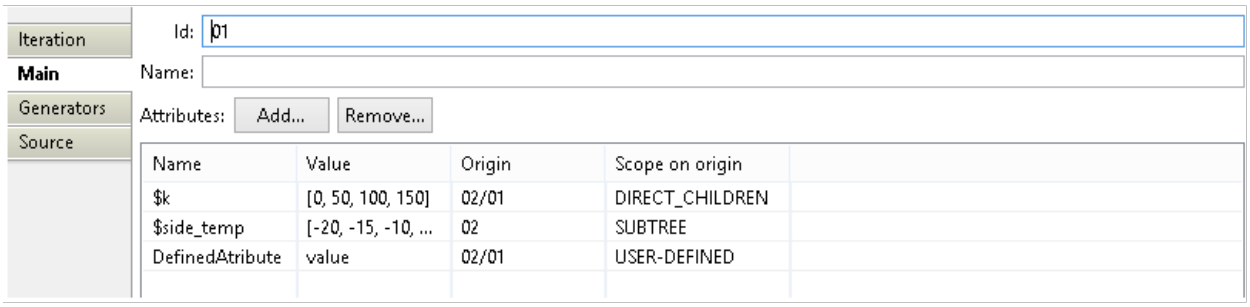

Рисунок 27. Вкладка Main свойств Instancer Node

Вторая значимая вкладка для данного артефакта называется Iteration и обозначает свойства механизма переиспользования. На этой вкладке можно видеть (рис. 28) механизм выбора одного из способов итерации по параметрам, набор выбранных на данный момент параметров представленный несколькими элементами типа List и кнопки для удаления одного из параметров или добавления нового параметра.

Рисунок 28. Вкладка Iteration свойств Instancer Node

Стоит отметить, что при отсутствии выбранных параметров, для Instancer Node создается ровно одна копия повторно используемого артефакта. При наличии выбранных параметров и подхода к итерированию количество копий будет вычисляться исходя из выбранных значений.

Кроме механизмов итерации вкладка содержит поле выбора целевого артефакта, обозначенное на рисунке 28 как “Target”. В нем отображается путь к выбранному артефакту.

Также вкладка Iteration содержит поле обозначенное как “Id formula”, определяющее формулу, по которой будет вычисляться идентификатор копий. При наличии ошибок в идентификаторе в Req Explorer и Uni Editor будет отображаться соответствующее сообщение о JavaScript ошибке. Стандартное значение этого поля - $i обозначающее номер копии.

Заключение

В данной работе были рассмотрены этапы процесса проектирования тестов, обозначен ряд связанных с данной предметной областью проблем (вступление). Было произведено сравнение программных средств направленных на поддержку требований (пункт 1.2.) и проектирование тестов (пункт 1.3.).

Были рассмотрены подходы к решению проблем связанных с автоматизацией процесса проектирования тестов, включая компактное описание схожих артефактов и фрагментов каталога требований для модельного ряда. Для поддержки указанных функций рассмотрено использование механизмов переиспользования (пункт 2.1.1.), параметризации (пункт 2.1.2.) и итерации по сочетанию значений параметров (пункт 2.2.).

Реализация механизмов параметризации и повторного использования произведена в рамках программного средства Requality. Для указанного продукта были рассмотрены особенности его строения и функционирования и обозначен набор произведенных над его структурой изменений (пункт 3.).

Использованные подходы позволили реализовать описание схожих артефактов и фрагментов каталога требований при этом оставляя возможности для последующего расширения использования реализованных механизмов.

Стоит отметить, что использование текущей реализации может потребовать дополнительного обучения пользователей за счет появления новых элементов. При этом стоит учесть, что использование формул для вычисления значений параметров может быть частично описано в документации на новые функции, но в общем случае потребует навыков написания формул на языке Java Script.

Реализованная в ходе выполнения данной работы версия механизмов переиспользования и параметризации позволяет описывать схожие артефакты, имеющие различия в наборе параметров. При этом реализован механизм итерации по наборам сочетаний всех выбранных параметров.

По сравнению с другими средствами управления требованиями и проектирования тестов реализованные механизмы обладают возможностями более гибкого описания свойств копий. При этом были рассмотрены пути расширения возможностей описания, реализация которых в дальнейшем позволит описывать единым способом различающиеся артефакты.

На данный момент имеется несколько потенциальных путей дальнейшего развития как механизмов переиспользования и параметризации, так и инструмента Requality в целом.

Один из них связна с дальнейшим усовершенствованием механизмов генерации параметров, при этом возможна смена механизмов вычисления значения по формуле или расширение возможностей самой генерации.

Второе потенциальное направление - разработка подхода к использованию параметров в виде фрагмента формальных описаний параметров. Так, возможно их использования для определения видимости фрагментов каталога требований или использование для автоматизации процесса верификации.

Третье направление - расширение возможностей редактирования повторно использованных фрагментов каталога требований, позволяющее расширить сферу применения реализованных механизмов.

Четвертое направление напрямую не связанное рассмотренными механизмами, но напрямую связанное с автоматизацией проектирования тестов - это рассмотрение возможностей получения требований путем использования семантических деревьев.

К сожалению, на данный момент рассматриваемое решение не позволяет полностью автоматизировать процесс выделения требований из текста спецификаций, но, возможно позволит обнаружить определенный класс текстовых фрагментов, которые можно отнести к требованиям. Требуется рассмотреть возможности применения данного подхода и изучить его эффективность в применении к анализу требований.

Список использованных источников

1. DO-178C "Software Considerations in Airborne Systems and Equipment Certification" / RTCA, 2011.
2. В.В. Кулямин, Н.В. Пакулин, О.Л. Петренко, А.А. Сортов, А.В. Хорошилов. Формализация требований на практике / В.В. Кулямин и др.- Препринты Института системного программирования РАН, Препринт 13, 2006.
3. В.В. Кулямин, А.К. Петренко. Развитие подхода к разработке тестов UniTESK / В.В. Кулямин и др. - Труды Института системного программирования РАН, том 26, выпуск 1, 2014. - Стр. 9-26.
4. Guide to the Software Engineering Body of Knowledge (SWEBOK), 2004.
5. INCOSE Requirements Management Tools Survey [Электронный ресурс]. Режим доступа: http://www.incose.org/productspubs/products/rmsurvey.aspx, свободный.
6. Описание программного продукта IBM Reqsuite Pro [Электронный ресурс]. Режим доступа: http://www-03.ibm.com/software/products/ru/reqpro/, свободный.
7. IBM Rational RequisitePro [Электронный ресурс]. Режим доступа: http://www.finecosoft.ru/reqpro, свободный.
8. Управление требованиями для систем и сложных ИТ-приложений [Электронный ресурс]. Режим доступа: http://www-03.ibm.com/software/products/ru/ratidoor/, свободный.
9. Обзор Rational DOORS [Электронный ресурс]. Режим доступа: http://pic.dhe.ibm.com/infocenter/doorshlp/v9r5/index.jsp?topic=%2Fcom.ibm.doors.requirements.doc%2Ftopics%2Fc_welcome.html, свободный.
10. Описание программного продукта IBM Rational Requirements Composer [Электронный ресурс]. Режим доступа: http://www-03.ibm.com/software/products/ru/rrc/, свободный.
11. IBM Rational Requirements Composer [Электронный ресурс]. Режим доступа: http://www.finecosoft.ru/rrc, свободный.
12. Caliber - learn more. Collaborative requirements and definition software [Электронный ресурс]. Режим доступа: http://www.borland.com/products/caliber/read/, свободный.

13. Simplify Requirements Life Cycle Management [Электронный ресурс]. Режим доступа: http://www.sdlctools.com/rtimeE2.asp, свободный
14. Category:RMF/User Guide - Eclipsepedia [Электронный ресурс]. Режим доступа: http://wiki.eclipse.org/RMF/ProR, свободный
15. Requirements Management Software - Product Tour [Электронный ресурс]. Режим доступа: http://www.accompa.com/requirements-management-software-tour.html, свободный.
16. QM - Design [Электронный ресурс]. Режим доступа: http://www.open-do.org/projects/qualifying-machine/design/, свободный.
17. Requirements Interchange Format. Version 1.1 [Электронный ресурс]. Режим доступа: http://www.omg.org/spec/ReqIF/1.1/PDF/, свободный.
18. Requality: общие сведения [Электронный ресурс]. Режим доступа: http://requality.ru/ru/general.ru.html#destination, свободный.
19. Э. Гамма, Р. Хелм, Р. Джонсон, Д. Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны. / Э. Гамма и др. - П.: Питер, 2007 г.
20. YAWL - User Manual. Version 2.3 [Электронный ресурс]. Режим доступа: http://www.yawlfoundation.org/manuals/YAWLUserManual2.3.pdf, свободный
21. А.В. Демаков, С.В. Зеленов, С.А. Зеленова. Генерация тестовых данных сложной структуры с учетом контекстных ограничений / А.В. Демаков, С.В. Зеленов, С.А. Зеленова - Труды Института системного программирования РАН, том 9, 2006. - Стр. 83-96.
22. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ
23. Introduction to Algorithms / Под ред. И. В. Красикова. — 2-е изд. / М.: Вильямс, 2005. — С. 632-635.

Приложения

Приложение 1.

Аннотация по-русски для размещения на портале

Тестирование является важной фазой жизненного цикла разработки программных систем. Обеспечению качества ответственных систем уделяется особое внимание. Важнейшим стандартом на процессы обеспечения качества, в частности, на процессы тестирования является группа стандартов DO-178 (A, B и C). В соответствии с этими стандартами основой для разработки тестов являются требования к программной системе.

Традиционным источником требований являются текстовые документы, например Технические задания. Такие документы являются неформальными как по форме, так и по содержанию. Для того, чтобы иметь четкую базу для проектирования набора тестов, а потом для оценки полноты тестового набора, неформальный документ требуется проанализировать с целью выделения отдельных требований, а затем построить каталог требований. Только после этого можно переходить к проектированию тестов - к определению тестовых ситуаций, разработке и выполнению тестов. Тем самым процессы анализа требований и тестирования тесно связаны между собой.

Данная работа посвящена вопросу автоматизации процессов управления требованиями и первой фазы проектирования тестов. Были рассмотрены вопросы получения каталога требований и набора тестовых ситуаций из документов, рассмотрен вопрос управления реакцией на изменения объектов, определена необходимость в описании схожих артефактов и фрагментов каталога требований.

Дается обзор и проводится сравнение программных средств поддержки требований и проектирования тестов. Сделан вывод о необходимости

разработки новых средств поддержки схожих описаний артефактов и фрагментов каталога требований. Особое внимание в работе уделяется механизмам переиспользования и параметризации требований. Рассмотрено несколько возможных подходов к реализации этих механизмов, выполнена реализация этих механизмов в программном средстве Requality.

В результате выполнения работы реализована поддержка механизмов переиспользования и параметризации, позволяющих решить рассмотренные проблемы описания и поддержки схожих тестовых ситуаций, требований и совпадающих фрагментов каталога требований.

Количество страниц в работе без учета приложений – 78. Работа содержит таблиц – 1, иллюстраций – 28, использованных источников – 23.

Ключевые слова: проектирование тестов, переиспользование, параметризация, тестовые ситуации, каталог требований.

Приложение 2.

Аннотация по-английски для размещения на портале

Testing is one of the important part of development of software systems. Process of quality assurance for safety critical systems is usually maitained under special overside. Processes from group of specification DO-178(A, B and C) are one of the most important for quality assurance processes, especially, for testing . In accordance with this standard, the basis of testing development is set of requirements on software system.

Traditional source of requirement is test documents, as an example, technical specifications. Those documents are usually informal on both structure and content. To build a clear basis for test design and further assessments on completeness of test set it is needed to extract single requirements from informal documents and to build an requirements catalog. Only after that, test design becomes possible – test purposes can be described and test procedures can be designed. So, processes of requirements analysis and testing are closely related to each other.

This paper considers a question of automation of processes related to requirements management and first phase of test design. It overviews problem of requirements elicitation, requirements catalog building, describes a necessity of change management, defines an importance of design and support of similar artifacts and fragments of requirements catalog.

Author introduces an overview and comparison of software products for requirements management and test design. He concluds that new mechanisms for support of similar artifacts and fragments of requirements catalog are required. Special attention is paid to re-use and parametrization mechanisms. This paper lists some possible approaches to their implementation, the mechanisms are implemented in software product Requality.

As a result, mechanisms for re-use and parametrization are implemented. This fact allows to solve listed problems of description and support of similar test purposes, requirements and equal fragments of requirements catalog.

Number of pages without attachments - 76. Number of tables - 1, pictures - 28, sources - 23.

Keywords: test design, re-use, parametrization, test purposes, requirements catalog.

Приложение 3

Исходные коды программного средства Requality

Ввиду большого объема документа текст программы приводится в электронном виде и находится на приложенном к бумажной копии диске, в папке sources.