АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ И ДОКУМЕНТИРОВАНИЯ ИНФОРМАЦИОННЫХ СИСТЕМ

Антон Цыбин, Людмила Лядова

***Abstract**: Статья посвящена описанию подходов к автоматизации тестирования и создания пользовательской документации в программных системах с графическим пользовательским интерфейсом. Для автоматизации тестирования применён комбинированный подход на основе машин состояний и data mining. Для автоматического составления документации применён подход, основанный на использовании описания системы с помощью метаданных, представленных связанными списками. Предложенные методы позволяют ускорить и повысить качество процессов тестирования и документирования приложений без требования задания сложных входных данных.*

***Keywords**: тестирование программ, автоматическое создание документации, data mining, фиксированные грамматики, изменяемые грамматики.*

***ACM Classification Keywords**: D.2 Software Engineering: D.2.2 Design Tools and Techniques; D.2.4 Software - Program Verification; I.2 Artificial Intelligence: I.2.2 Automatic Programming.*

Введение

На сегодняшний день одним из базовых понятий методологии проектирования программных систем является понятие жизненного цикла. Жизненный цикл – это непрерывный процесс, который начинается с момента принятия решения о необходимости создания информационной системы (ИС) и заканчивается в момент окончания её эксплуатации.

Структура жизненного цикла включает три группы процессов: основные, вспомогательные и организационные. Основные процессы включают следующие этапы:

- создание,
- внедрение и эксплуатация,
- сопровождение.

Этап создания программы также является сложным и включает в себя:

- оценку жизненного цикла системы,
- анализ требований,
- проектирование структуры компонентов,
- реализацию проекта.

На этапе внедрения и эксплуатации производятся контрольное тестирование и приёмо-сдаточные испытания. Данный этап направлен на обеспечение качества программного продукта.

Все рассмотренные этапы создания программной системы сопровождаются вспомогательным процессом – документированием. В рамках данного процесса производятся работы по созданию различных инструкций и руководств, документации пользователя и программиста.

Из всех описанных этапов жизненного цикла наиболее формализованными и обеспеченными автоматизированными средствами являются этапы анализа требований и проектирования модели предметной области. Меньше внимания в жизненном цикле уделяют этапам тестирования и создания документации. Такая закономерность наблюдается, прежде всего, в небольших компаниях по разработке программного обеспечения (ПО). Причина состоит в сложности автоматизации тестирования и составления документации. Несмотря на обилие средств автоматизации, среди них не существует

универсального продукта, подходящего для любой предметной области, для которой создается ПО. На выполнение данных этапов вручную требуется большой объём времени и средств, зачастую эти затраты не окупаются для небольших компаний. Однако тестирование – очень важный этап жизненного цикла, так как недостаточное внимание к нему не позволяет получить качественный, надежно функционирующий программный продукт. Для измерения степени автоматизации тестирования в жизненном цикле Институтом Иллинойса была предложена модель зрелости тестирования (TMM – Test Maturity Model) [1], содержащая набор приёмов повышения степени автоматизации.

Многие организации игнорируют этап создания документации, особенно на поздних стадиях создания продукта, что резко снижает эффективность внедрения и сопровождения, так как при отсутствии описания требований, описания функций системы затруднено изменение программного кода системы в случае обнаружения ошибок или переноса системы на новую программно-аппаратную платформу. При отсутствии или ненадлежащем состоянии пользовательской документации осложняются этапы внедрения и сопровождения системы, так как в этом случае снижается эффективность обучения пользователей, их неверные действия при работе с ПО приводят к необходимости исправления ошибок в данных и увеличению числа вопросов к службе технической поддержки.

Подходы к тестированию

На сегодняшний день рынок программного обеспечения предлагает огромный выбор продуктов, позволяющих автоматизировать тестирование программ на различных стадиях разработки. Существуют средства, проверяющие полноту и непротиворечивость требований к программе, тестирующие сетевое взаимодействие, основную функциональность приложений методами «чёрного», «белого» и «серого» ящиков, надёжность работы системы при больших и запредельных нагрузках. Существуют даже инструменты, позволяющие автоматизировать тестирование удобства использования графического интерфейса приложения.

Исторически первым методом тестирования является так называемый *метод «чёрного ящика»* [2]. Данный метод подходит для тестирования небольших программных модулей и сложен в использовании при тестировании больших программных систем, так как требует выполнения большого количества тестов для достижения приемлемого уровня уверенности в правильности программы.

Качественно новый подход к тестированию был предложен Энтони Хоаром – *метод доказательства корректности* программ [3]. Единственным серьёзным недостатком данного подхода является чрезвычайная сложность построения математических моделей для больших программ. После некоторых попыток применения подхода, от него решено было отказаться.

Позднее были предложены иные методы тестирования, основанные на *анализе исходного кода программ для выявления ошибок*. Наиболее известным является метод «белого ящика», предложенный Филлисом Франклом и Элайном Веюкером в 1988 году [2]. Недостаток данных методов состоит в том, что тестирование программы производится без наличия знаний о правильном результате работы программы для каждого теста. При использовании данных методов предполагается, что результаты работы программы на тестах проверит человек. Но инструментальные средства тестирования с помощью покрытия кода генерируют огромное количество тестовых наборов, так что проверить результат работы каждого теста вручную затруднительно. В связи с этим был предложен *критерий выборочной проверки* (так называемая *N-выборка*), основанный на статистических методах.

Среди программ автоматизированного тестирования существуют *средства записи-воспроизведения действий человека*, осуществляющего тестирование. В Rational Robot имеется возможность определения правильности результата по свойствам визуальных элементов управления.

Существуют также научные разработки, использующие средства записи-воспроизведения (такие, как Rational Robot) для комбинирования достоинств нескольких подходов к организации тестирования. Например, в работе [4] описан оригинальный способ тестирования правильности работы графического

интерфейса пользователя в программах. Недостаток такого подхода состоит в сложности построения и сопровождения диаграмм, описывающих состояния интерфейса.

Подходы к автоматическому созданию документации

На сегодняшний день на рынке ПО наибольшее распространение получили программы автоматического документирования исходного кода проектов. Рассмотрим несколько подходов.

Для создания программной документации существует множество программ, использующих одинаковый подход на основе грамматик. Грамматикой будем называть любой формализованный набор правил построения какой-либо системы.

Использование фиксированных грамматик

Большинство современных систем документирования исходного кода программных проектов используют фиксированные грамматики. Идея данного метода состоит в *использовании информации, содержащейся в правилах построения программ*. Правила построения программ иначе называют *грамматикой языка программирования*. Данные правила описаны в спецификациях языка программирования и, как правило, не могут быть изменены, т.е. являются фиксированными.

Например, существует программа NDoc для документирования исходного кода проектов на платформе dotNET. В данном примере в качестве грамматики можно рассматривать правила построения объектно-ориентированной программы для платформы dotNET, а также правила обработки XML-тегов комментариев.

Полностью аналогичный подход используется системой JavaDoc при построении программной документации проектов на языке Java.

Существуют также системы, создающие документацию для специфических языков программирования. Система LPdoc создает документацию для языков Lisp и Prolog. В качестве грамматики здесь можно рассматривать правила синтаксиса и семантики языков Lisp и Prolog.

Использование изменяемых грамматик

Реализация системы документирования программных проектов, полностью построенная на идее *конечных автоматов*, описана в работе [5].

Реализуемая в этом подходе архитектура позволяет создать общее описание проекта, а затем в любое время преобразовать данное описание в документацию. Аналогичные принципы построения компонента документирования используются и в представленном здесь исследовании.

Создание документации происходит на основе *шаблонов*, заранее заданных пользователем. Шаблоны содержат информацию о том, какую структуру должен иметь полученный документ.

В данном случае грамматика – это набор правил извлечения информации из определённым образом структурированного файла.

Отличие от подхода, основанного на фиксированных грамматиках, состоит в том, что наборы правил по выделению информации из различных файлов вынесены в XML-файл и хранятся вне системы документирования.

Предлагаемые подходы к автоматизации тестирования и создания документации

Данная работа посвящена исследованию возможности автоматизированной проверки правильности выходных данных программы с графическим пользовательским интерфейсом (GUI), а также разработке компонента автоматической генерации документации пользователя для информационных систем.

Приложения с графическим интерфейсом были выбраны в качестве объекта исследования, так как большинство современных прикладных систем реализуют такой интерфейс. Документация пользователя

содержит описание функций системы, выполнение типовых операций через графический интерфейс. Поэтому необходимо реализовать автоматизированную проверку приложения через GUI с последующим описанием основной функциональности приложения через формы, с которыми работает пользователь.

Основной проблемой тестирования была и остаётся сложность описания правил, в соответствии с которыми должна работать проверяемая программа. Если для программ, осуществляющих научные вычисления, количество правил составляет 10-20, то для сложных программных комплексов их число составляет сотни тысяч. Несмотря на достоинства автоматического создания тестов метода «белого ящика», данный метод не в состоянии автоматически проверить правильность выполнения теста. Для этого либо необходим набор правил спецификации, либо требуется помощь человека.

Идея предлагаемого подхода состоит в том, чтобы при тестировании программы через графический интерфейс отслеживать значения ряда параметров, представляющих состояние системы. В качестве параметров предлагается выбрать внутренние переменные или значения свойств визуальных элементов управления. Благодаря возможности проверки внутренних переменных данный подход шире, чем тот, что используется в Rational Robot [1]. Входные данные для интерфейса можно генерировать на основе исходного кода. Например, используя критерий покрытия структур или потоков данных, можно создать множество тестовых наборов для элементов графического пользовательского интерфейса таким образом, чтобы максимально проверить существующий код тестируемой программы. От пользователя требуется лишь указать набор внутренних переменных для определения результата работы.

Генерацию данных для пользовательского интерфейса можно производить случайным образом. Такой метод тестирования менее эффективен, чем метод «белого ящика», поскольку не предусматривает целенаправленного тестирования узких мест в коде.

В процессе тестирования программы происходит сбор данных о значениях указанных параметров (состояния программы) и событиях, вызывающих переходы из одного состояния в другое. Поскольку отсутствует информация о спецификации на программу, необходимо каким-либо образом проверить, правильно или нет выполняется каждый тест. Для решения данной проблемы предлагается использовать методы Data mining.

Причина выбора нечёткого метода для идентификации правильности выполнения теста связана со сложностью задания спецификации на тестируемую программу. Поскольку создание набора правил, в соответствии с которыми должна работать программа, – сложная задача, можно привлечь набор нечётких методов для извлечения знаний из результатов тестирования. Иными словами, без вмешательства человека постараться выявить в результатах тестирования определённые закономерности, ассоциации, зависимости, последовательности и т.п. На основе полученных знаний можно делать выводы об исключительных ситуациях, аномалиях в результатах тестирования, а также в соответствии с определёнными критериями выбирать определённые тесты для проверки пользователем. Таким образом, после выполнения сотен тестов программы через графический интерфейс, система тестирования выдаст программисту несколько тестов для ручной проверки правильности. Получив информацию о правильности тестов от пользователя, система учитывает её при построении закономерностей и впоследствии ищет ошибки с учётом полученной информации.

В качестве методов Data mining подходящими для применения в данном подходе являются:

- *Алгоритм ограниченного перебора*. Проверяет простые логические события в различных подгруппах данных. Выбор логических событий производится эвристически.
- *Алгоритм выявления ассоциаций*. Перевычисляет меры доверия к комбинациям фактов на основе байесовской условной вероятности и понятия шанса.

[1] Система Rational Robot не позволяет отслеживать значения внутренних переменных тестируемой программы

– *Статистические методы*. Использование усреднённых величин для проверки статистических параметров распределений, доказательство гипотез о распределении случайных величин, проверка линейной корреляции.
– *Нейронная сеть*. Аппроксимация зависимостей с помощью нелинейных функций.

Применив перечисленные методы к данным о состояниях системы и переходам между состояниями, можно получить знания о взаимосвязях между указанными параметрами программы, а также об их связи с воздействиями через графический интерфейс. На основе полученных знаний выделяются значения параметров, отклоняющиеся от найденных закономерностей.

В рамках данной исследовательской работы разрабатывается компонент, позволяющий автоматизировать процесс создания пользовательской документации для информационных систем [6].

На данном этапе разработки созданный компонент поддерживает создание документации для CASE-системы METAS.

METAS представляет собой программную систему, позволяющую проектировать и создавать её средствами различные информационные системы (ИС), динамически настраиваемые на условия эксплуатации и потребности пользователей [7]. Система METAS использует метаданные, описывающие предметную область, для которой создается система, интерфейс пользователя и создаваемые в системе документы, бизнес-процессы и т.д. в режиме интерпретации, что позволяет создавать информационные системы без изменения программного кода приложений, а также осуществлять динамическую настройку системы в ходе ее эксплуатации. Алгоритмы компонента документирования основаны на использовании метаданных METAS, представленных в виде многоуровневых связанных списков.

В процессе исследовательской работы была разработана и реализована двухслойная структура компонента документирования. Разработанный компонент имеет структуру, показанную на рис. 1.

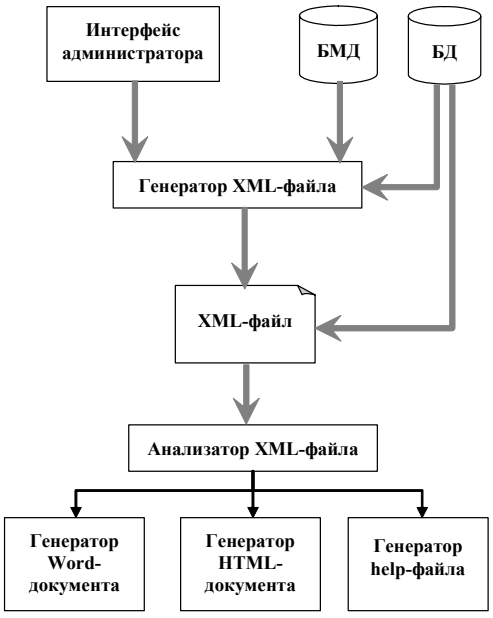

Рис. 1. Общая схема работы компонента документирования

Реализованная схема компонента документирования позволяет разработчику задать структуру документа. Параметры представления документа подаются на вход генератора XML-файла шаблона. Использование XML-файла документации позволяет хранить данные описания ИС независимо от представления документа, то есть в дальнейшем возможен анализ данного файла и преобразование информации, находящейся в нём в конкретный вид. Например, файл Microsoft Word, HTML или Help-файл.

Разработчик документации должен иметь возможность определения структуры и содержания документации, а именно: задавать различные сочетания вложенности элементов документации, возможность добавлять или удалять элементы описания.

Для решения поставленной задачи был разработан специальный интерфейс администратора.

Структура документа представляется в виде дерева. Дерево содержания можно задавать как визуально, так и с помощью текстового описания.

Текстовое описание синхронизируется с деревом. В интерфейсе реализована функция проверки синтаксических и семантических ошибок описания структуры документа (недопустимая вложенность элементов содержания документа). Задача проверки семантических ошибок возникла по причине ограниченного числа комбинаций вложенности элементов описания.

Интерфейс передаёт информацию о структуре документа в виде XML-файла структуры генератору XML-файла.

Основная задача генератора XML-файла – создать XML-файл, содержащий описание ИС в соответствии со структурой, заданной в интерфейсе разработчика документа. Генератор обходит списки метаданных в порядке, заданном структурой документа. Каждая вершина XML-файла описывает определённый объект.

В генераторе XML-файла также реализован алгоритм поиска путей к сущности в рекурсивном дереве. Задача возникла в связи с тем, что система METAS позволяет на главной форме настроить дерево объектов (сущностей) ИС для удобного доступа пользователей к сущностям. Поэтому может потребоваться описать путь к вершине, представляющей сущность в дереве, показать, как пользователь может до неё добраться.

Для описания форм в генератор была добавлена функция «фотографирования» формы. Изображение формы сохраняется в отдельном файле, а затем в XML-файл добавляется ссылка на данное изображение.

Анализатор XML-файла выполняет разбор созданных ранее XML-файлов описаний ИС и передаёт данные описания в специализированные генераторы конечных документов, поддерживающие определённый интерфейс. Анализатор использует технологию событийного разбора документов SAX.

Каждая вершина XML-файла описывает определённый объект. Кроме атрибутов, содержащих данные описания, каждая вершина имеет специальный атрибут, значением которого является индекс соответствующей записи в таблице (рис. 2).

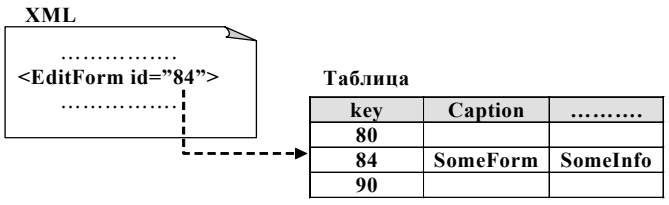

Рис. 2. Связь атрибута в XML и ключа в БД

Такой подход позволяет впоследствии при анализе XML-файла быстро находить нужную запись в БД, а также проверять на основе имеющихся описаний, тот ли это объект и существует ли он вообще. Несоответствие данных может быть следствием двух действий:

- произведена переиндексация таблиц,
- вручную изменены данные в XML-файле.

Во всех таких случаях анализатор спрашивает пользователя, откуда брать описание объекта – из XML или из ИС.

Для иллюстрации работы созданной схемы реализован генератор документа в представлении Microsoft

Word. Взаимодействие с Word происходит через механизм OLE. Данные об ИС поступают непосредственно от анализатора XML-файла.

Разработанный компонент позволяет:

- представить данные ИС в универсальном виде (XML);
- разделить процессы выборки информации и создания документации;
- преобразовать данные ИС к любому представлению;
- по-разному структурировать документацию пользователя;
- поддерживать связь документации с базой данных ИС.

Пример работы компонента создания документации в соответствии с описанной ниже схемой показан на рис. 3.

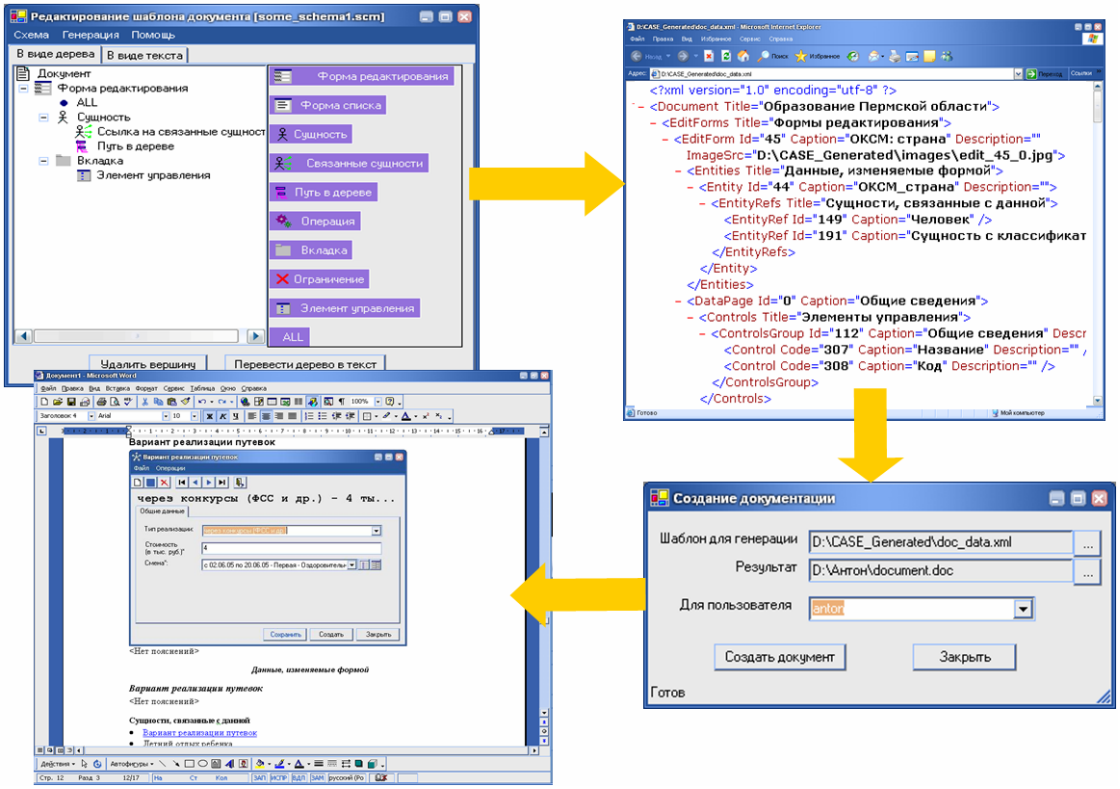

Рис. 3. Пример работы компонента создания документации

На рисунке изображён процесс создания документации, состоящий из шагов:

- создание структуры будущего документа,
- создание XML-шаблона, содержащего всю необходимую для документации информацию,
- указание параметров генерации конечного документа,
- создание конечного документа в формате Microsoft Word.

Реализованный компонент создания документации позволяет создавать XML-файлы шаблонов документов требуемой структуры, содержащие данные ИС. Также реализована возможность обработки XML-шаблонов и создания документации пользователя в формате Microsoft Word. Таким образом, имеется возможность в кратчайшие сроки создавать для информационной системы документацию различной структуры и представления.

В настоящее время исследуется возможность применения в генераторе шаблонов изменяемой грамматики. Алгоритм обработки элементов метаданных при построении документа меняется в

зависимости от смысла отношений элементов метаданных. В ходе выполнения работы была поставлена задача представления на декларативном уровне правил обработки сочетаний элементов метаданных. Требуется предоставить разработчику возможность изменять данные правила. При этом информация о связях элементов метаданных представляет собой метаметаданные, или метаданные второго уровня. Такой подход позволяет реализовать компонент в виде отдельного модуля с возможностью подключения к любой информационной системе, основанной на списках. Подход также предоставляет более широкие возможности для приведения структуры документации к требуемому виду.

Разработана математическая модель применения изменяемых грамматик в алгоритме документирования, формально доказана обоснованность принятых проектных решений.

Заключение

Предложенный метод тестирования позволяет в автоматизированном режиме осуществить тестирование программы с графическим интерфейсом на любой стадии разработки благодаря возможности выбора контролируемых параметров программы. Кроме того, существует возможность проведения регрессионного тестирования, если при тестировании система будет сохранять выполняемые тесты.

Структура компонента документирования обеспечивает дальнейшее расширение возможностей. В ходе исследований появились также новые идеи построения логики работы компонента документирования.

Реализованный компонент планируется использовать для создания документации пользователей для различных информационных систем, а также для создания «заготовок» документов, которые впоследствии могут быть доработаны специалистом по созданию документации.

Библиографический список

[1] Дастин Э., Рэшка Д., Пол Д. Автоматизированное тестирование программного обеспечения. М.: ЛОРИ, 2003. С. 15-20.

[2] Gregory M. Kapfhammer. Software testing: [Электронный документ] (http://cs.allegheny.edu/~gkapfham/research/publish/software_testing_chapter.pdf).

[3] Дейкстра Э. Программирование как дисциплина математической природы: [Электронный документ] (http://khpi-iip.mipk.kharkiv.edu/library/extent/dijkstra/pp/ewd361.html).

[4] Калинов А.Я., Косачёв А.С. Автоматическая генерация тестов для графического пользовательского интерфейса по UML диаграммам действий: [Электронный документ] (http://www.citforum.ru/SE/testing/generation_uml).

[5] Суясов Д.И., Шалыто А.А. Автоматическое документирование программных проектов на основе автоматного подхода: [Электронный документ] (http://is.ifmo.ru).

[6] Цыбин А.В. Автоматическая генерация документации пользователя в информационных системах, управляемых метаданными // Сб. тезисов конференции-конкурса «Технологии Microsoft в теории и практике программирования» / Новосибирск: НГУ, 2007. С. 78-80.

[7] Лядова Л.Н., Рыжков С.А. CASE-технология METAS // Математика программных систем: Сб. науч. тр. / Пермь: Перм. ун-т, 2003. С. 4-18.

Сведения об авторах

Антон Цыбин – Пермский государственный университет, студент магистратуры кафедры математического обеспечения вычислительных систем; Россия, г. Пермь, 614990, ул. Букирева, 15; e-mail: magicdr@mail.ru

Людмила Лядова – Пермский государственный университет, заведующий кафедрой математического обеспечения вычислительных систем; Россия, г. Пермь, 614990, ул. Букирева, 15; e-mail: LNLyadova@mail.ru