УДК 681.3.06

А. М. Андреев, Ив. С. Свирин

РЕГРЕССИОННОЕ ТЕСТИРОВАНИЕ АВТОМАТИЗИРОВАННЫХ СИСТЕМ УЧЕТА, ПОСТРОЕННЫХ НА БАЗЕ КОНФИГУРАТОРА

Рассмотрено регрессионное тестирование, которое является специализированным видом тестирования, для проверки изменений, сделанных в коде программы в связи с устранением ошибок и консолидацией вносимых изменений.

E-mail: ivansvirin@gmail.com

Ключевые слова: *регрессионное тестирование, автоматизированные системы учета, конфигуратор, псевдоестественный язык.*

Современный уровень развития информационных технологий, позволяет позиционировать программное обеспечение (ПО) как сложный продукт, качество которого необходимо контролировать на всех этапах разработки. Для верификации ПО используется большое число разнообразных методологий. Не все ошибки или недоработки можно выявить на этапе тестирования до эксплуатации ПО, что влечет дополнительную разработку релизов и обновление существующей версии ПО, которое устранит конкретные ошибки, замеченные пользователем, но влияние изменения кода на другие участки и другую функциональность в полном объеме не будет проверено. Для исключения возможности непреднамеренного внесения одних ошибок путем корректировки других необходимо использовать регрессионное тестирование в целях подтверждения работоспособности всей функциональности ПО [1].

Регрессионное тестирование осуществляется с использованием тест-кейсов, написанных на ранних этапах разработки и стадии тестирования, с помощью которых можно подтвердить либо опровергнуть корректность работы ПО.

Можно выделить три основных вида:

- регрессия багов (bug regression) — попытка доказать, что исправленная ошибка на самом деле не исправлена;
- регрессия старых багов (old bug regression) — попытка доказать, что недавнее изменение кода или данных нарушило исправление старых ошибок, т.е. старые баги стали снова воспроизводиться;
- регрессия побочного эффекта (side effect regression) — попытка доказать, что недавнее изменение кода или данных вывело из строя другие части разрабатываемого ПО.

В целях минимизации затрат и устранения человеческого фактора целесообразно использовать автоматизацию процесса регрессионного тестирования, т.е. обеспечить верификацию ПО с использованием специальных средств (Automation Test Tool).

Для организации автоматического тестирования используются следующие понятия:

- тест-скрипт (Test Script) — это набор инструкций для автоматической проверки определенной части ПО;
- тестовый набор (Test Suite) — это комбинация тест-скриптов для проверки определенной части ПО, объединенной общей функциональностью или целями, преследуемыми запуском данного набора;
- тесты для запуска (Test Run) — это комбинация тест-скриптов или тестовых наборов для последующего совместного запуска.

Структура построения регрессионного теста может быть представлена в следующем виде (рис. 1). В данной структуре скрипты и формируемые на базе них тестовые наборы разработаны на специальном скриптовом языке или на языках высокого уровня — это зависит от компилятора средства автоматизированного регрессионного тестирования.

Написанием тест-скриптов и компоновкой их в тестовые наборы занимается разработчик тестовых сценариев, который должен обладать хорошим знанием как самого ПО, которое подлежит верификации, так и языка построения скриптов, что заставляет выдвигать повышенные требования к сотруднику, занимающему эту должность. Под каждый конкретный проект или продукт необходимо выделять отдельного человека, данная должность не является унифицированной: при переходе к другому проекту необходимо обучать сотрудника предметной области “с нуля”, что приводит к появлению дополнительных материальных затрат.

При разработке ПО на каждое функциональное требование формируется набор тестовых скриптов, прогон которых проводится при поступлении дополнений и обновлений. На каждую обнаруженную ошибку в эксплуатируемом ПО также формируется тестовый набор,

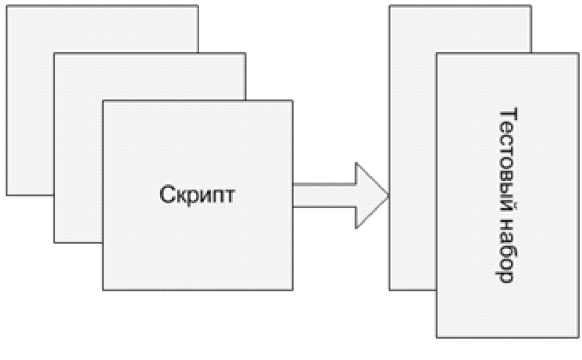

Рис. 1. Структура формирования теста в регрессионном тестировании

который добавляется в общий массив выполняемых программой автоматизированного тестирования сценариев.

Как бы ни было высоко стремление к автоматизации всех рутинных процессов, связанных с тестированием, полностью исключить ручную работу тестировщика-оператора не представляется возможным. Основной причиной возникновения таких ситуаций является необходимость проверки визуальных составляющих ПО либо сложных алгоритмов работы, на которые невозможно или слишком трудоемко писать тестовые скрипты. Тест-кейс для ручного тестирования представляет собой в упрощенном виде таблицу с двумя колонками: в первой описано действие, которое необходимо выполнить тестировщику, а во второй — результат, который должен показать, что тестовый шаг выполнен успешно, в случае несоответствия полученного результата формируется отчет для группы разработчиков.

При нынешней структуре построения регрессионных тестовых наборов, практически на всех этапах формирования теста необходимо наличие высококвалифицированного персонала, что сказывается на стоимости проекта либо на качестве производимого продукта.

Основной задачей является минимизация средств на формирование наиболее адекватных, полных тестов. При текущей схеме формирования регрессионных тестов минимизировать затраты не получается.

Большинство автоматизированных учетных систем в настоящее время разрабатываются с использованием специальных конфигураторов, это позволяет значительно сократить время разработки прикладных решений, а также обеспечивает функционирование в едином информационном пространстве. Изменение законов, нормативных актов, организационно-распорядительных документов, формы собственности — все это приводит к необходимости внесения большого числа изменений в существующую конфигурацию. В рамках каждой конфигурации выделяется ряд подсистем, каждая из которых автоматизирует работу определенного структурного подразделения. Как правило, заявки на адаптацию или доработку функционала не согласовываются с другими подразделениями, поэтому велик риск частичной или полной блокировки работоспособности отдельных подсистем учета. Остановка работы подразделения в крупных компаниях может повлечь значительные материальные потери, поэтому остро стоит вопрос регрессионного тестирования для данной области ПО.

Верификацию таких автоматизированных систем можно проводить, используя обычный метод регрессионного тестирования, и использовать встроенный язык конфигуратора в качестве скриптового языка, но это нецелесообразно и слишком трудоемко. Использование специфических особенностей среды разработки [2] позволит минимизировать затраты на создание и сопровождение регрессионного тестирования.

Отличительными особенностями всех конфигураторов являются:

- ограниченное число предопределенных классов;
- невозможность расширения набора классов;
- ограниченное число методов и свойств предопределенных классов.

Эти особенности позволяют разработать иной способ построения регрессионных тестов. Для формализации подхода необходимо ввести новые понятия:

действие — программная инструкция для автоматического или ручного выполнения;

результат — множество состояний объектов метаданных;

тестовый шаг — совокупность действия и соответствующего ему результата;

тестовый сценарий — набор тестовых шагов, объединенных общей функциональной направленностью.

Работа любого прикладного решения сводится к созданию, удалению или изменению состояний существующих объектов, в рамках классов имеется, как уже отмечалось, ограниченное число методов и свойств, т.е. на каждый класс можно разработать ряд шаблонов действий и шаблонов результатов, которые позволят упростить процесс регрессионного тестирования.

Рассмотрим новый метод построения регрессионного тестирования (рис. 2):

Действия и результаты разрабатываются программистом высокой квалификации, который отлично знает встроенный язык, описание классов, а также предметную область. Разработка ведется на встроенном языке с добавлением специально разработанных операторов, облегчающих работу программиста, а также повышающих уровень

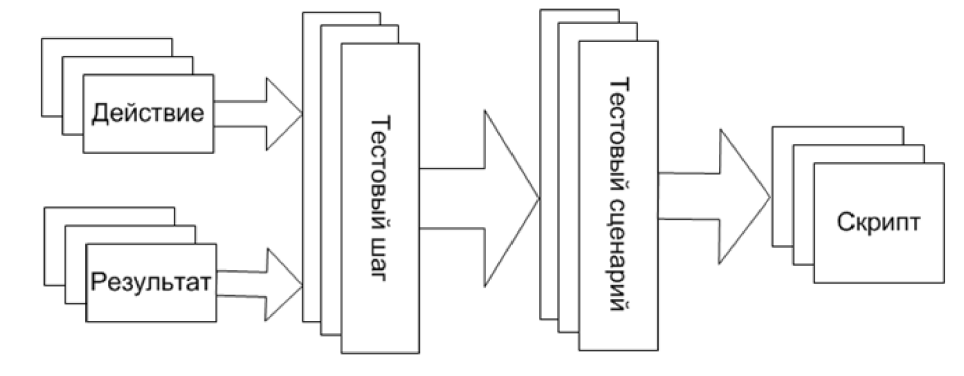

Рис. 2. Схема формирования регрессионного теста для автоматизированной системы учета, построенного с использованием конфигуратора

абстракции шаблонов, например: оператор [ЗаполнитьПараметры] заполняет все параметры конкретного объекта по списку реквизитов, извлеченных из метаданных.

Массив действий и результатов формирует псевдоестественный язык более высокого уровня, чем встроенный язык конфигуратора.

Тестовый шаг обязательно состоит из действия и его результата. Разработкой тестовых шагов может заниматься сотрудник, который не знаком непосредственно с встроенным языком и устройством конфигурации, а имеет поверхностное понятие о работе с конфигуратором даже без знания предметной области, квалификация сотрудника по сравнению с предыдущим этапом может быть значительно снижена. Элементы псевдоестественного языка помогут составителю тестовых шагов не вникать в суть решения проблемы: операторы действий и результат будут выбираться в соответствии с функциональными требованиями, написанными на естественном языке, и их сопоставлением. Если в функциональных требованиях написано, что необходимо создать объект с определенными параметрами, то разработчик выбирает из всего массива тот оператор, который соответствует созданию описанного объекта, а в качестве проверки результата — оператор проверки создания описанного объекта с заданными параметрами.

Любое действие порождает изменение состояния объектов базы данных; если это изменение будет соответствовать результату, то будет верно утверждение, что тестовый шаг пройден успешно. При выполнении скрипта действия возможно появление ошибок, которые могут быть вызваны одним из следующих факторов:

- ошибка разработки шаблона действия/результата;
- некорректное заполнение свойств объектов.

В случае, если компилятор не сумел выполнить скрипт действия или скрипт результата, тестовый шаг считается не пройденным или проваленным.

Тестовые шаги компонуются в тестовые сценарии для выделения теста проверки определенной функциональности. В один тестовый сценарий может входить неограниченное число тестовых шагов. Квалификация сотрудника, занимающегося разработкой тестовых сценариев, также может быть невысокой. Общая структура тестового сценария может быть представлена в следующем виде (рис. 3).

При запуске проверки тестового сценария запускается многоитерационный процесс исполнения действия и проверки результата, при успешном прохождении всех тестовых шагов тестовый сценарий считается успешно выполненным.

Поскольку для исполнения скриптов используется конфигуратор, то возникающие при их выполнении ошибки, автоматически фиксируются компилятором конфигуратора и попадают в отчет об ошибке

Тестовый сценарий					
Тестовый шаг 1		Тестовый шаг *i*		Тестовый шаг *n*	
Действие	Результат	Действие	Результат	Действие	Результат

Рис. 3. Структура тестового сценария

(bug report) в разрезе тестового сценария и тестового шага, который предоставляется разработчику прикладного ПО для выявления причин возникновения неполадок и устранения замечаний.

При данной структуре построения тестового набора высокую квалификацию должен иметь лишь программист, осуществляющий описание операторов псевдоестественного языка. Список созданных операторов может расширяться и корректироваться по мере отработки регрессионного механизма тестирования в рамках схожих по функциональности систем. Опыт, накопленный при тестировании конкретной конфигурации, может быть использован для проверки других конфигураций, так как независимо от того, какие задачи организации учета решает конкретное прикладное решение, объекты предопределенных классов остаются неизменными, а соответственно, и подходы к проведению тестирования могут быть применены для всех возможных конфигураций. Массив операторов псевдоестественного языка может быть использован для всех конфигураций практически без доработок или с минимальными изменениями.

Тестировщик с минимальными знаниями работы конфигуратора, имея описание требования или ошибки на естественном языке от заказчика, может, используя операторы разработанного псевдоестественного языка, описать порядок действия и порядок проверки результата, после чего транслятор автоматически преобразует скрипт на псевдоестественном языке в скрипт, воспринимаемый компилятором конфигуратора, что позволяет значительно сократить время формирования скриптов проверки.

Для проверки возможности реализации и эффективности регрессионного тестирования автоматизированных систем, построенных на базе конфигураторов, по новой структуре формирования тестов было выбрано средство “1С: Предприятие 8.1”. Была разработана унифицированная подсистема, обеспечивающая регрессионное тестирование любой конфигурации “1С: Предприятие 8.1”: элементы подсистемы интегрируются в существующее прикладное решение и позволяют реализовать новую схему построения регрессионного теста.

Рассмотрим простой пример. От разработчика прикладного ПО требовалось разработать функционал, обеспечивающий создание документа “Приходная накладная” с реквизитами:

- регистрационный номер;
- дата документа;
- склад, на который осуществили отгрузку;
- сотрудник, принявший груз.

В табличной части документа “Груз” имеются два поля:

- номенклатура;
- количество.

При создании документа нужно осуществлять проверку заполнения всех реквизитов и наличие хотя бы одной строки в табличной части.

После разработки необходимо создать тесты.

Первоначально следует создать операторы псевдоестестественного языка:

Действие “Создание документа” в реализованной подсистеме представлено на рис. 4. Рассмотрим выполняемый код, соответствующий данному действию.

Объект — это понятие специального хранилища, которым необходимо оперировать разработчику операторов псевдоестественного языка. Если в рамках действия или результата необходимо осуществлять некоторые операции над некоторым объектом конфигурации, следует использовать это понятие.

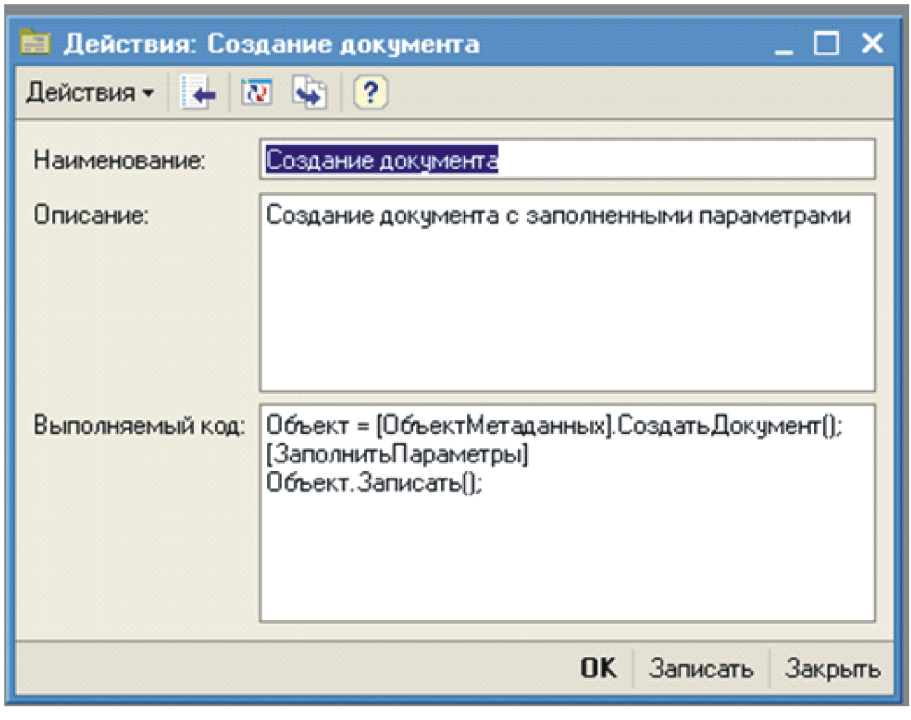

Рис. 4. Объект “Действие” подсистемы регрессионного тестирования, реализованной на “1С: Предприятие 8.1”

[ОбъектМетаданных] — дополнительный оператор подсистемы, осуществляющий подстановку конфигурационного объекта предопределенного класса. В рассматриваемом случае это будет “Документы.ПриходнаяНакладная”. Выбор конкретного значения подстановки будет осуществляться на уровне формирования тестового шага.

[ЗаполнитьПараметры] — дополнительный оператор подсистемы, добавляющий в скрипт на этапе формирования тестового шага код на встроенном языке программирования, осуществляющий заполнение реквизитов *Объекта*, указанных на этом этапе.

Записать(), СоздатьДокумент() — это методы предопределенного класса Документы конфигурации “1С:Предприятие 8.1”. Они могут быть вызваны для любого объекта.

Результат “Проверка создания документа” в реализованной подсистеме представлен на рис. 5. Рассмотрим выполняемый код, соответствующий данному результату.

[Номер], [Дата] — это шаблон *[Реквизит]*, где в качестве реквизита указывается наименование любого реквизита, который присутствует в конкретном предопределенном классе, на этапе формирования тестового шага.

УспешностьПрохождения — это специальный признак подсистемы, который обеспечивает контроль корректности выполнения скрип-

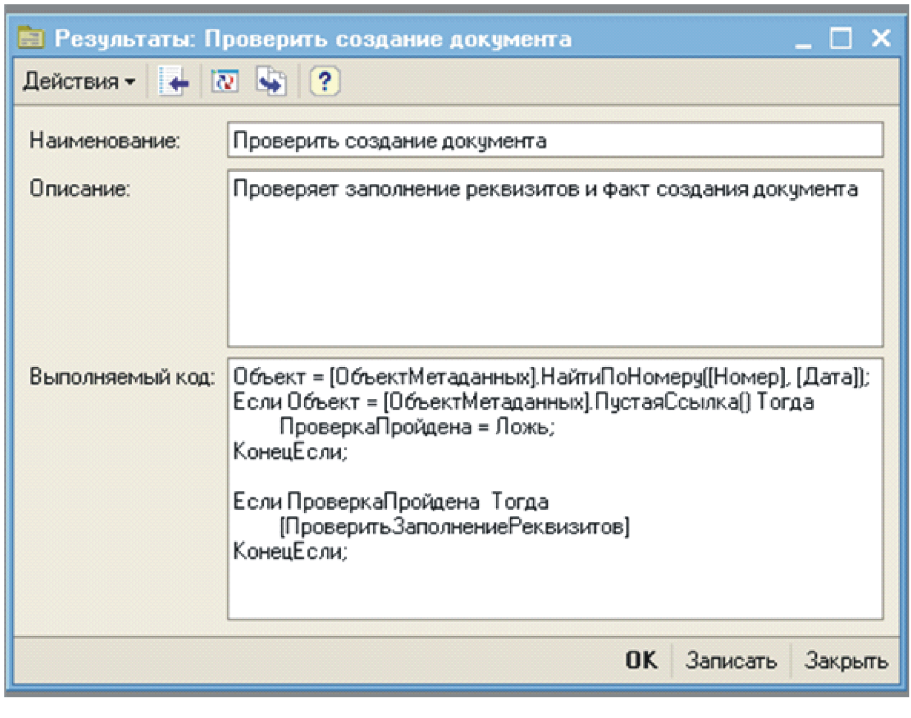

Рис. 5. Объект “Результат” подсистемы регрессионного тестирования, реализованной на “1С: Предприятие 8.1”

та. Если выполнение скрипта результата вернет этот параметр со значением “Ложь”, то тестовый шаг считается не пройденным.

[ПроверитьЗаполненностьПараметров] — оператор подсистемы для подстановки кода на встроенном языке, осуществляющего последовательную проверку всех заполненных реквизитов на этапе формирования тестового шага.

Все остальные операторы, указанные в действии и в результате, являются операторами встроенного языка “1С:Предприятие 8.1”, и код написан в соответствии с требованиями, предъявляемыми к разработке прикладных решений.

Разработанные операторы псевдоестественного языка направлены на подтверждение корректности работы механизмов, т.е. мы не пытаемся доказать, что механизм работает неправильно при вводе определенных параметров или действий.

После разработки элементов псевдоестественного языка к работе приступает составитель тестовых шагов, в данном случае будет один тестовый шаг “Проверка создания документа Приходная накладная”, реализация представлена на рис. 6.

На этапе формирования тестового шага, разработчик указывает конкретные объекты метаданных конфигурации в действии и результате, при его выборе автоматически формируется список реквизитов и табличных частей, имеется возможность заполнить значения для создания или проверки.

Тестовый шаг помещается в тестовый сценарий и отправляется на выполнение.

Посмотрим, как осуществляется работа подсистемы. Если указать не все параметры, необходимые для заполнения, то компилятор при попытке создать документ “Приходная Накладная” должен выдать ошибку. Не выполнение действия приведет к автоматическому проставлению признака провала тестового шага и соответственно тестового сценария, а поскольку, как было отмечено выше, в нашу задачу входит подтверждение корректности работы, то необходимо корректно заполнить все реквизиты и элемент табличной части. Если подсистеме не удалось создать объект со всеми корректными данными, то разработчик прикладной программы допустил ошибку механизмов проверки корректности заполнения данных. Соответствующую ошибку сформирует компилятор конфигуратора, и она попадет в отчет о тестировании. Если объект метаданных был создан, но в результате

Действие	Результат
Создание документа	Проверка создания документа

Рис. 6. Структура тестового шага

выполнения скрипта проверки признак успешности прохождения теста вернулся как “Ложь”, то это также указывает на ошибку, т.е. в результате обработки операции сохранения объекта произошло непреднамеренное изменение реквизитов документа, что и было выявлено. Анализ существующего документа, а также ошибка, сформированная уже подсистемой регрессионного тестирования, помогут разработчику локализовать проблему и устранить ее.

С помощью тех же операторов псевдоестественного языка могут быть протестированы любые другие объекты предопределенного класса “Документ” на любой конфигурации без какой-либо доработки, необходимо осуществлять лишь заполнение значения объекта метаданных и его реквизитов на этапе формирования тестовых шагов.

Таким образом, при использовании вновь разработанной схемы формирования регрессионных тестов удастся сэкономить значительные материальные средства, поскольку в рамках одной организации можно временно выделить одного высококвалифицированного сотрудника, разбирающегося в работе с конфигуратором, который разработает операторы псевдоестественного языка или адаптирует существующие массивы операторов под специфику тестируемой учетной системы, а затем вернется к своим прежним обязанностям, а все работы, связанные с дальнейшей работой по формированию тестовых шагов и тестовых сценариев, могут быть переданы менее квалифицированному сотруднику или даже стажеру, в чьи функции будет входить лишь корректировка параметров либо формирование новых шагов, которые можно скомпоновать без глубокого знания конфигуратора.

Если бы задача была реализована по традиционной структуре построения регрессионных тестовых наборов, то разработчику с высокой квалификацией пришлось бы заниматься длительной разработкой тестовых скриптов под каждый конкретный объект предопределенного класса конфигуратора на встроенном языке конфигуратора “1С: Предприятие”, после чего отправлять его на исполнение. При добавлении, удалении, изменении реквизита, его типа данных, необходимо осуществлять корректировку скриптов. Если к одному объекту метаданных относится большое число скриптов, то данная работа представляется достаточно трудоемкой и длительной. Высококвалифицированный сотрудник вынужден вникать в предметную область конкретного решения и на основе анализа формировать новые тесты, проверять валидность существующих тестов в рамках внесенных изменений.

Преимущество предлагаемой схемы формирования регрессионных тестов состоит в простоте использования, минимизации материальных затрат. Снижение стоимости формирования и проведения тестов, охватывающих весь функционал учетной системы, позволяет обеспечить

построение добротной и качественной системы, оперативное определение возможных неполадок и ошибок, возникающих при адаптации и изменении объектов конфигурации, с которыми ведут работу несколько различных функциональных подразделений. Подход позволяет минимизировать риски, связанные с возможным простоем в работе сотрудников из-за внесенных блокирующих ошибок.

СПИСОК ЛИТЕРАТУРЫ

1. ISTQB. Certified Tester. Foundation Level Syllabus 2011.
2. Габец А. П., Гончаров Д. И., Козырев Д. В., Кухлевский Д. С., Радченко М. Г. Профессиональная разработка в системе 1С: Предприятие 8 / Под ред. М.Г. Радченко. – М.: "1С-Паблишинг"; СПб.: Питер, 2006. – 808 с.

Статья поступила в редакцию 15.12.2011