На правах рукописи

ГРИППА Генри Леонидович

Автоматизация тестирования программных приложений методом ключевых состояний

Специальность 05.13.11 –
Математическое и программное обеспечение
вычислительных машин, комплексов и компьютерных сетей

Автореферат
диссертации на соискание ученой степени кандидата технических наук

Санкт – Петербург – 2006

Работа выполнена в Государственном образовательном учреждении высшего профессионального образования «Санкт-Петербургский государственный политехнический университет».

Научный руководитель – доктор технических наук, профессор
Черноруцкий Игорь Георгиевич

Официальные оппоненты – доктор технических наук, профессор
Александров Анатолий Михайлович
– кандидат технических наук
Зотов Алексей Алексеевич

Ведущая организация – Санкт-Петербургский институт информатики и автоматизации Российской академии наук

Защита состоится 18 мая 2006 г. в 16 часов на заседании диссертационного совета Д 212.229.18 в ГОУ ВПО “Санкт-Петербургский государственный политехнический университет” по адресу: 195251, Санкт-Петербург, Политехническая ул., д.29, 9 уч. корп., ауд. 325.

С диссертацией можно ознакомиться в Фундаментальной библиотеке ГОУ ВПО “Санкт-Петербургский государственный политехнический университет”.

Автореферат разослан 17 апреля 2006 г.

Ученый секретарь
диссертационного совета Шашихин В.Н.

Общая характеристика работы

Актуальность

В последнее время в области информационных технологий в связи с постоянно растущей конкуренцией особую актуальность приобретают вопросы тестирования, как деятельности, повышающей качество программных продуктов. На сегодняшний день одним из необходимых видов тестирования, применяемых в ходе процесса разработки и модификации программных продуктов, является, так называемое, регрессионное тестирование – повторное тестирование части программы, зависящей от внесенных в нее изменений.

Однако, в условиях крупных проектов промышленного масштаба тестирование в целом, и регрессионное тестирование в частности, является очень дорогостоящей и трудоемкой деятельностью. Это обусловлено необходимостью проводить регрессионное тестирование в случае внесения даже малейших изменений в код программы, в то время как процесс регрессионного тестирования может включать в себя исполнение достаточно большого количества тестов на скорректированной версии программы. И, несмотря на то, что усилия, требуемые для внесения небольших изменений, как правило, минимальны, они могут требовать достаточно больших усилий для проверки качества измененной программы. Тем не менее, надежная и эффективная разработка и сопровождение программного обеспечения невозможны без регрессионного тестирования. Одним из очевидных выходов в сложившейся ситуации является автоматизация процесса тестирования.

На сегодняшний день на рынке программного обеспечения существует ряд продуктов нацеленных на автоматизацию процесса тестирования. Однако, проведенное исследование выявило недостаточную эффективность существующих в этой области информационных технологий средств и методов автоматизации, особенно в применении их к крупным проектам промышленного масштаба.

В данной работе на основе анализа существующих концепций, методов и средств автоматизации тестирования и тенденций развития технологий предложен новый метод разработки автоматизированных тестов и созданы необходимые средства программной поддержки. Разработанный метод отличает высокая эффективность разработанных автоматизированных тестов, универсальность в применении и относительно низкая трудоемкость процесса внедрения автоматизированного тестирования.

Все вышеизложенное свидетельствует об актуальности данной работы.

Цель и задачи работы

Целью работы является создание метода разработки автоматизированных тестов, удовлетворяющего следующим требованиям:

- инвариантность относительно программного языка реализации тестируемого продукта;
- высокая эффективность разрабатываемых автоматизированных тестов, определяемая как отношение количества обнаруженных ошибок к количеству известных ошибок, содержащихся в покрываемом тестами коде программного продукта;
- относительно низкая трудоемкость, определяемая как отношение объема разработанных тестов к объему покрываемого ими кода тестируемого продукта.

Достоверность полученных результатов подтверждается достаточной аргументацией приведенных в работе выводов и рекомендаций, их практическим применением, а также конкретными результатами внедрения.

Научная новизна работы

Научную новизну работы представляют следующие результаты:

1. Создан метод разработки автоматизированных тестов программных приложений.
2. Разработан оригинальный язык описания автоматизированных тестов, разработанных в соответствии с новой методикой.
3. Создана система автоматизированного тестирования, осуществляющая поддержку автоматизированных тестов, разработанных при помощи новой методики и с использованием нового языка описания тестов.
4. Создана методика распознавания ключевых состояний тестируемых систем, в связи с чем решена задача системной классификации множества произвольных состояний.
5. Разработан новый подход к классификации кластеров выделенного множества состояний, построение на основе этого динамической библиотеки состояний

Практическая значимость работы

В результате работы были разработаны новый метод, создана методика распознавания ключевых состояний тестируемых систем, в связи с чем решена задача системной классификации множества произвольных состояний, создан язык описания тестов и готовый программный продукт являющийся средством поддержки данного метода. Применение результатов работы на практике позволило достичь значительного сокращения времени регрессионного тестирования, увеличения эффективности, значительного снижения

трудоемкости тестирования, и, как следствие, снижения конечной стоимости программных продуктов.

Апробация работы

Основные результаты диссертационной работы докладывались и обсуждались на семинарах корпорации Borland и межвузовских научных конференциях.

Внедрение

Разработанная система автоматизации тестирования была применена для автоматизации тестирования в процессе разработки и сопровождения шести программных продуктов корпорации Borland.

Публикации

По материалам диссертации сделаны доклады на трех межвузовских конференциях и опубликовано 7 печатных работ.

Структура и объем работы

Диссертация содержит 153 страницы основного текста, 19 рисунков, 6 таблиц и состоит из введения, трех глав, заключения, списка литературы и двух приложений.

Содержание работы

Во введении обосновывается актуальность работы, формулируются цель, задачи, объект и предмет диссертационного исследования, показана научная новизна и практическая ценность полученных результатов. Приводится структура диссертации.

Первая глава посвящена изучению различных методов, средств и аспектов тестирования программного обеспечения на протяжении их жизненного цикла, проведению различных оценок с целью выявления их недостатков.

На сегодняшний день существуют два основных метода автоматизации тестирования: метод unit-тестов, реализующий, так называемое, модульное тестирование, и применение различных программных продуктов, автоматизирующих тестирование при помощи механизма записи и воспроизведения действий пользователя.

Идея unit-тестов состоит в том, что для каждого класса или для небольшой их совокупности, в случае, когда один класс не является функционально автономным, пишется тестовая программа, которая проверяет функциональность модуля (класса или группы классов) на работоспособность. Т.е. тестируемая программа рассматривается не как некий целостный программный продукт, а как набор самостоятельных модулей со своей

функциональностью, которые тестируются не в контексте всей программы, а как самостоятельное приложение.

Unit-тестирование — это, как правило, проверка работоспособности на наборе контрольных примеров, то есть совокупности входных и соответствующих им выходных данных. По замыслу авторов технологии unit-тестирования, такой подход позволяет выявлять большое количество ошибок и быстро их локализовать, так как найти ошибку в пределах модуля легче, чем в пределах проекта.

Однако, unit-тестирование имеет свои недостатки. Создание полноценного unit-теста – это, фактически, отдельный проект со всеми присущими ему фазами: проектированием, документацией, написанием кода, отладкой, итерациями тестирования и исправления найденных ошибок. Время, необходимое для написания сколько-нибудь полезного теста, как минимум, не меньше, а, как правило, в несколько раз превосходит время, необходимое на разработку основного функционала, а их объем может достигать 80% от объема тестируемого приложения.

Кроме того, если разработчик не предполагал какого-либо контекста использования своего модуля при разработке, то он не перестроится и при написании теста к нему. Тест будет создаваться на основе тех же предположений, что и сам модуль, а это во многом противоречит самой идее тестирования.

И, наконец, самым значительным минусом подхода unit-тестирования является именно то, что отдельные модули системы тестируются обособленно от контекста всей системы в целом. И при таком подходе работоспособность каждого модуля в отдельности никоим образом не гарантирует нормальную работоспособность всей системы в целом.

Пожалуй, unit-тесты хороши только для проектов, модули которых представляют собой совокупность относительно автономных очень сложных и комплексных алгоритмов, а бюджет позволяет привлекать независимых от написания кода аналитиков и специалистов для подготовки контрольных примеров, и увеличивать время разработки в два-три раза. Но такие проекты нельзя назвать распространенным случаем разработки бизнес-приложений.

Программные средства автоматизации тестирования, такие как Rational Robot, Mercury Interactive WinRunner, Mercury Interactive QuickTest, Segue SilkTest и др., позволяют производить запись последовательности действий пользователя (специалиста по тестированию) и в дальнейшем воспроизводить эту последовательность автоматически неограниченное число раз.

Принцип работы таких продуктов заключается в следующем: после начала записи теста программа переходит в фоновый режим работы и начинает записывать события операционной системы при том или ином действии пользователя и объекты графического

пользовательского интерфейса (GUI) тестируемой программы, на которых данные события были сгенерированы. В ходе записи теста необходимые идентификационные данные задействованных объектов, а также параметры событий операционной системы записываются при помощи различных языков, в результате чего получается тестовый скрипт (файл с инструкциями), который может подвергаться дальнейшему редактированию с целью добавления дополнительных функций и при воспроизведении скрипта передается на вход программе автоматизации. В качестве момента, когда система полностью среагировала на возмущения, вызванные воспроизведенным действием, берется момент появления объекта, определенного в описании следующего действия.

Диаграмма действий для воспроизведения тестового скрипта современными программными средствами автоматизации тестирования, построенная в соответствии с требованиями спецификации языка объектно-ориентированного моделирования UML (Unified Modeling Language) версии 2.0 представлена на рис. 1.

Таким образом, данные инструменты позволяют организовать регрессионное тестирование программных продуктов путем однократной записи действий, совершаемых для воспроизведения тестового сценария, и возможностью многократного их повторения.

В то же время все подобные современные продукты объединены общим набором существенных недостатков.

Во-первых, современные продукты подобного рода, как уже говорилось, при записи действия пользователя запоминают, по сути дела, события операционной системы и объекты GUI, на которых данное событие было сгенерированно. Однако, объекты, события на которых данные инструменты в состоянии записывать, изначально ограничены достаточно стандартным набором типов. И, несмотря на то, что, как правило, подобные продукты позволяют расширять этот список внесением описания своих собственных графических компонентов, на практике это работает только лишь в исключительных случаях. Связано это с тем, что подобные продукты не имеют доступа к API тестируемой системы, без использования которого в большинстве случаев просто невозможно описать нестандартный графический компонент. Так, к примеру, в большинстве графических редакторов все компоненты на поле графического редактора являются отдельными объектами с точки зрения самого редактора, но для операционной системы представляют один большой графический образ.

Таким образом, если тестируемое приложение содержит какие-то нестандартные объекты, являющиеся частью GUI, то подобного рода инструменты будут давать сбои при записи последовательности ваших действий, т.е. будут просто несовместимы с тестируемой системой.

Во-вторых, подобные продукты не в состоянии обойти, так называемую, проблему синхронизации действий, заключающуюся в определении момента окончания отклика тестируемой системы на произведенное действие.

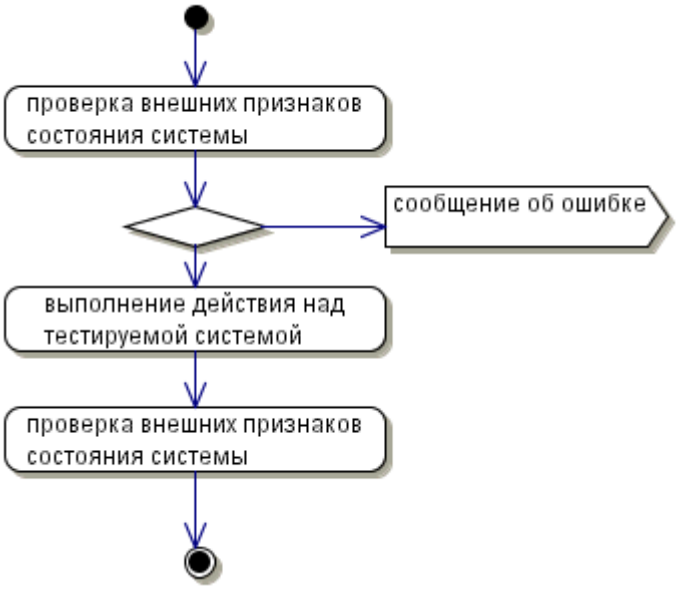

Рис. 1 Диаграмма действий для воспроизведения одного шага тестового скрипта современными программными средствами автоматизации.

И, наконец, одним из существеннейших недостатков подобных инструментов является то, что они не в состоянии проверить правильность реакции тестируемой системы на выполнение каждого действия в отдельности или же всей последовательности в целом. Т.е. правильность выполнения действия определяется только появлением объекта GUI в заранее заданный промежуток времени. И при этом никак не проверяется внутреннее состояние тестируемой системы, что приводит к существенному снижению количества отлавливаемых ошибок и затруднению их последующей локализации.

Все перечисленные проблемы связаны с отсутствием доступа к API тестируемого продукта. Очевидно, что выход из сложившегося положения заключается в тесной интеграции продукта автоматизации тестирования с тестируемой системой, что позволило бы не только решить имеющиеся проблемы, но и повысить уровень эффективности процесса тестирования.

Краткие характеристики описанных методов указаны в сводной таблице 1.

Во второй главе автор предлагает к рассмотрению собственный метод разработки автоматизированных тестов, язык разработки этих тестов и краткое описание реализации системы программной поддержки. Для наглядности его перспективности приводится расширение разработанного метода, позволяющее автоматизировать саму процедуру составления тестовых сценариев и практически полностью исключить ручной труд из процесса тестирования. Работа над расширением метода является темой дальнейшей работы автора, однако, на сегодняшний день уже создан прототип системы поддержки расширенного метода.

Новый подход к разработке автоматизированных тестов сродни идее unit-тестирования, но в отличие от unit-тестов оперирует не физическими, а, в каком-то смысле, логическими модулями тестируемой системы.

Для реализации данного подхода необходимо весь тестовый сценарий разбить на последовательность состояний тестируемой системы и логически завершенных действий – случаев использования в терминологии языка объектно-ориентированного моделирования UML, осуществляющих перевод системы из одного состояния в другое.

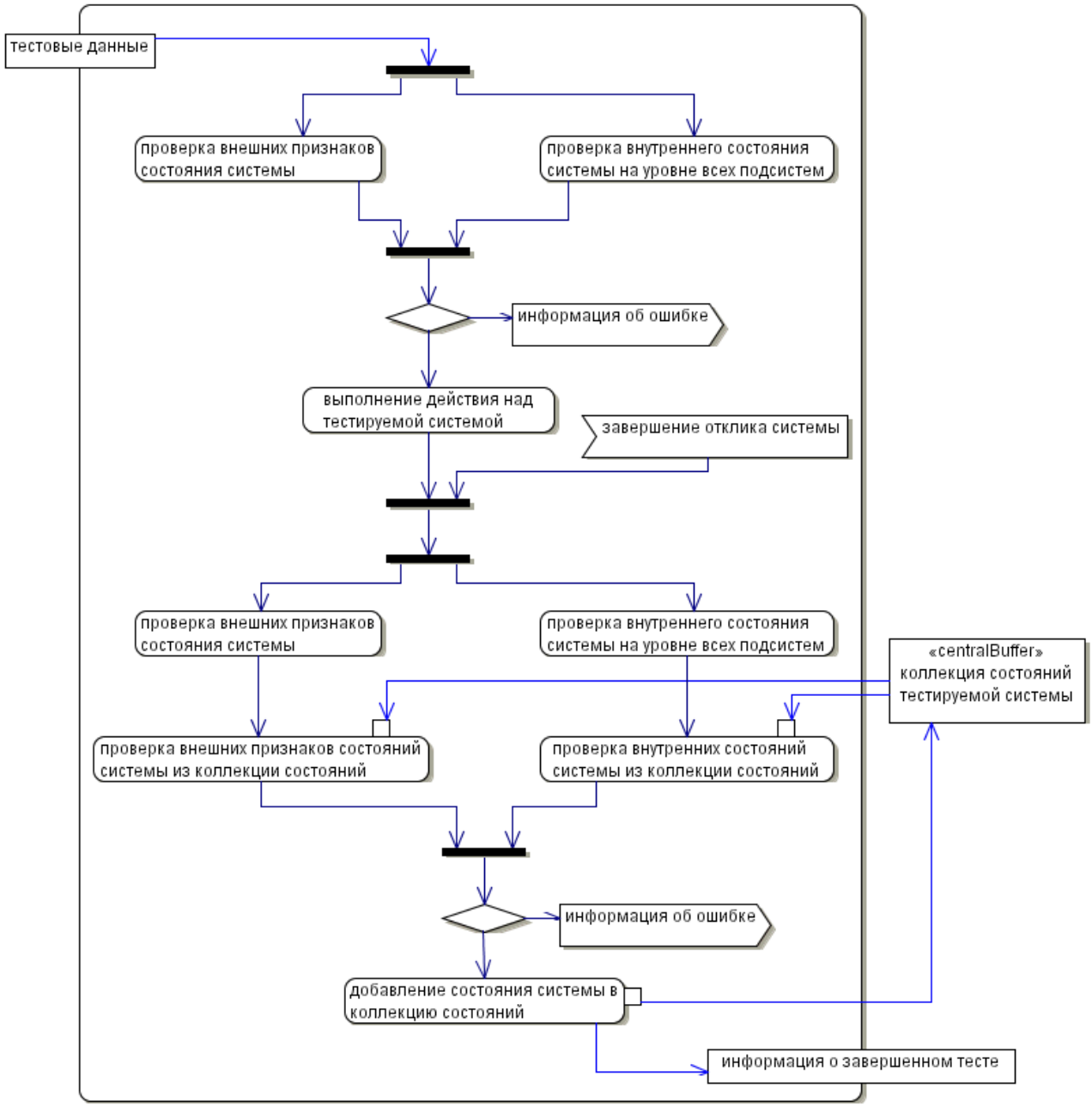

Рис. 2 Диаграмма действий для выполнения одного шага тестового скрипта.

Отдельное логически завершенное действие рассматривается как, своего рода, логический модуль тестируемой системы со своей функциональностью. В таком случае необходимо написать набор тестов на каждое логически завершенное действие из последовательности,

составляющей тестовый сценарий, проверяющих правильность состояния системы, в которое она переходит в результате совершенного действия. Т.к. тестируемая система может состоять из множества связанных подсистем, то и проверка состояния всей системы будет заключаться в проверке всех ее подсистем. Например, подсистемы работы с файловой системой, подсистемы интерпретации кода, и т.д.

Таким образом, формируется набор тестов логически завершенных действий (производимых через API тестируемого продукта или через GUI операционной системы), совершаемых над тестируемой системой. При этом в декларации самих действий должны быть инкапсулированы проверки состояний, в которые тестируемая система переходит в результате этих действий, осуществляемых на различных уровнях тестируемой системы, а также проверки, предшествующие действию и проверяющие возможность его выполнения (т.е. идентифицирующие состояние системы, в котором она находится перед совершением очередного действия). В дальнейшем из этого набора строятся различные последовательности, являющиеся тестовыми сценариями, т.е. тестовым скриптом.

Диаграмма действий, составленная в терминах языка объектно-ориентированного моделирования UML версии 2.0 и описывающая исполнение одного шага тестового скрипта – тестирования одного логически завершенного действия, изображена на рис. 2.

Как видно из диаграммы, при выполнении одного тестового шага, прежде всего, выполняется проверка, как внешних признаков состояния тестируемой системы, так и внутренних состояний тестируемой системы на уровне различных подсистем. После чего, если состояние системы не удовлетворяет указанным требованиям, происходит прерывание выполнения тестового сценария и выдается сообщение об ошибке. В противном случае – выполняется необходимое воздействие на тестируемую систему.

После произведенного действия система управления воспроизведением тестового скрипта должна дождаться момента окончания реагирования тестируемой системы на совершенное действие, что должен обеспечить модуль синхронизации действий управляющей системы. Информация об окончании отклика тестируемой системы на воспроизведенное действие должна получаться модулем синхронизации действий непосредственно от самой тестируемой системы посредством ее API.

После завершения синхронизации необходимо проверить корректность результатов произведенного действия. Здесь, опять таки, на основании входных данных проводится проверка, как внешних признаков состояния тестируемой системы, так и внутреннего состояния системы на уровне различных подсистем.

Далее необходимо проверить, что последнее совершенное действие не оказало негативного влияния на не связанную с ним часть тестируемой системы. Так, к примеру, в

случае тестирования многооконного текстового редактора необходимо проверить, что при открытии второго файла не было внесено никаких изменений в отношении первого открытого файла, что он не был изменен, удален или закрыт. Для этого вводится некая буферная конструкция, именуемая динамической коллекцией состояний. Ее задача заключается в хранении информации о результатах воздействия предшествующих действий на состояние тестируемой системы. По окончании проведения проверок результата воздействия последнего совершенного над тестируемой системой действия проводится проверка результатов воздействия предшествующих действий, совершенных над тестируемой системой ранее, информация о которых добавляется в динамическую коллекцию состояний после каждого совершенного действия и хранится там до окончания выполнения скрипта. Кроме того, должна быть реализована возможность запоминания внутренних состояний описанной динамической коллекции и последующее обращение к ним. Это необходимо для тестирования воспроизведения действий, обратных предшествующим (таких как undo/redo).

В случае, если в ходе осуществляемых проверок была обнаружена ошибка, как и в случае проверок, предшествующих выполнению действия, выдается сообщение об ошибке, а выполнение тестового скрипта прекращается.

Однако, стоит заметить, что в данной ситуации возможно просто сбросить информацию об ошибке в какой-нибудь буфер информации и продолжить дальнейшее выполнение тестового скрипта. Это возможно потому, что, в отличие от случая с проверками перед выполнением действия, проверки выполняемые после совершенного действия никоим образом не говорят о возможности или невозможности выполнения следующего действия. Однако, подобная схема работы нежелательна по причине того, что при обнаружении первой же ошибки система входит в непрогнозируемое состояние и, при обнаружении возможных последующих ошибок, нельзя с уверенностью сказать являются ли они самостоятельными ошибками или же всего лишь последствиями первой обнаруженной ошибки.

В дальнейшем из таких тестовых шагов строится тестовый сценарий, описанный в файле тестового скрипта.

Научной новизной системного подхода является распознавание ключевых состояний тестируемых систем, в связи с чем возникает задача системной классификации множества произвольных состояний.

Данная модель описывает кластерное представление произвольных состояний в системах, которая позволит на их основании построить систему распознавания узловых состояний.

Предлагаемая модель основана на использовании информации о совместном появлении исследуемых состояний вблизи друг от друга. При этом группы часто встречающихся состояний образуют роды, являющиеся, согласно рассматриваемой модели, атомарными смысловыми единицами, которые отображаются в виде динамической библиотеки состояний. Таким образом, необходим метод определения наиболее часто соседствующих состояний и построения множества узловых состояний тестируемой системы.

Пусть D - множество всевозможных тестируемых объектов, то есть, систем. $W = \{w_1, ..., w_N\}$ - множество состояний системы. Задача построения контекстного представления состоит в том, чтобы для некоторого априорно заданного множества атомарных состояний гипотетической области S найти отображение $\gamma : D \times S \to R$, задающее для каждого объекта $d \in D$ и каждого состояния $s \in S$ некоторую меру принадлежности состояния данному объекту d. Множество таких отображений обозначим через $\Gamma(D, S)$. Контекстное представление объекта основано на объединении его состояний в контекстные группы и определения родов состояний, характерных системе, на основе этих групп. Контекстные группы формируются на основе близости этих явлений в объекте. При этом принимается гипотеза о том, что явления, соседствующие в объекте, зависят друг от друга и вместе образуют узлы предметной области.

Обозначим через $F(W)$ множество всех конечных последовательностей состояний из W. Тогда можно считать, что на множестве объектов D задано отображение $\eta : D \to F(W)$, ставящее в соответствие каждому объекту $d \in D$ некоторую последовательность состояний.

Контекстные группы либо формируются по принципу одинакового количества элементов в них, либо соответствуют определенным типам исследуемых в системе состояний. Для простоты будем рассматривать стратегию разбиения, основанную на равном количестве элементов. Каждая стратегия A задает отображение $h_{\text{A}} : F(W) \to F(F(W))$, где $F(W)$ - множество всех последовательностей состояний, соответствующих системам, $F(F(W))$ - множество всех последовательностей контекстных групп, которые, в свою очередь, также соответствуют последовательностям состояний. На этом этапе, поскольку контекстные зависимости уже учтены при построении данного разбиения, можно далее не учитывать информацию о последовательности состояний и контекстных групп, и ввести отображение ϑ, вида $\vartheta : F(F(W)) \times S \to R$, задающее для каждой последовательности контекстных групп и каждого состояния области их S меру принадлежности данного состояния этой контекстной группе. Однако для построения этого отображения необходимо сначала рассмотреть каждую контекстную группу в отдельности и определить состояния,

содержащиеся в ней. Для этого необходимо ввести отображение $\xi : F(W) \times S \to R$, задающее для каждой последовательности состояний каждой контекстной группы меру принадлежности состояний этой контекстной группе. Объединяя результаты действия этого отображения на контекстные группы системы, можно получить образ системы при отображении ϑ, задаваемом при помощи отображения ξ. Объединяющее отображение φ производит построение окончательного вектора признаков системы. При этом каждая контекстная группа путем действия отображения ξ преобразуется в вектор признаков $\sigma_{i_1},...,\sigma_{i_S}$, где $i = 1,...,n$. Отметим, что действие отображения ξ для каждой контекстной группы можно рассматривать как получение вектора признаков системы, перенумеровав признаки, которыми в данном случае являются роды исследуемой области и рассмотрев векторное пространство, размерности которого соответствуют этим признакам. Затем отображение φ производит построение окончательного вектора признаков системы, состоящего из значений $v_1,...,v_n$, соответствующих элементам множества узлов исследуемой области S. Представленная модель имеет абстрактный характер в силу того, что зависит от отображений ξ и φ, вид которых не был конкретизирован. Выбор этих отображений основан на использовании вероятностного подхода к классификации контекстных групп и документа. Каждая контекстная группа f_t, в результате действия отображения ξ, отображается на вероятностное распределение $P(s_k \mid f_i),\ k = 1,...,K$ родов в области на данной контекстной группе.

Отображение φ производит обобщение полученных при помощи ξ классификационных данных. Результатом его действия является вероятностное распределение $P(s_k \mid d)$ состояний гипотетической области родовых состояний динамической библиотеки состояний на рассматриваемой системе d. Данное отображение действует, опираясь на априорно равную вероятность всех контекстных групп в системе, выражающуюся в результатах проведения опыта по случайному выбору контекстной группы системы. Эта априорная равновероятность при рассмотрении вероятностных представлений контекстных групп означает одинаковую информативность контекстных групп и равный их вклад в общую характеристику системы.

Для функционирования системы поддержки автоматизированного тестирования описанным методом необходим язык описания скриптов. В качестве такового использован язык, основанный на расширяемом языке разметки текста – XML. Достоинства использования данной формы языка заключаются в “прозрачности” скриптов, описанных при его помощи, легкости обучения персонала данному языку, в отсутствии требований

специальных сред разработок при работе с ним и в удобстве работы с данными, описанными при помощи такого языка.

Основными конструкциями данного разработанного языка являются

- декларации акций, описывающие действия, производимые над системой;
- декларации состояний тестируемой системы, по сути своей, описывающие состояние каких-либо отдельных подсистем тестируемой системы и состоящие из набора проверок;
- множества состояний – множества атомарных состояний системы;
- динамические состояния – состояния создаваемые, изменяемые и удаляемые во время исполнения тестового скрипта, обладающие возможностью сохранения состояния самого динамического состояния, с последующей адресацией к данным "срезам" динамических состояний;
- динамические коллекции состояний – множества динамических состояний;
- процедуры – блоки скриптового кода выделенные в отдельные скрипты с возможностью их вызова из основного кода;
- фабрики состояний, создающие динамические коллекции состояний;
- фильтры состояний – выполняющие выборку из всех существующих состояний лишь тех, которые удовлетворяют условиям, заданным в фильтре и используемые в паре с модификаторами параметров состояний;
- модификаторы параметров состояний.

В третьей главе приводятся основные результаты применения на практике разработанного метода и проводится их анализ.

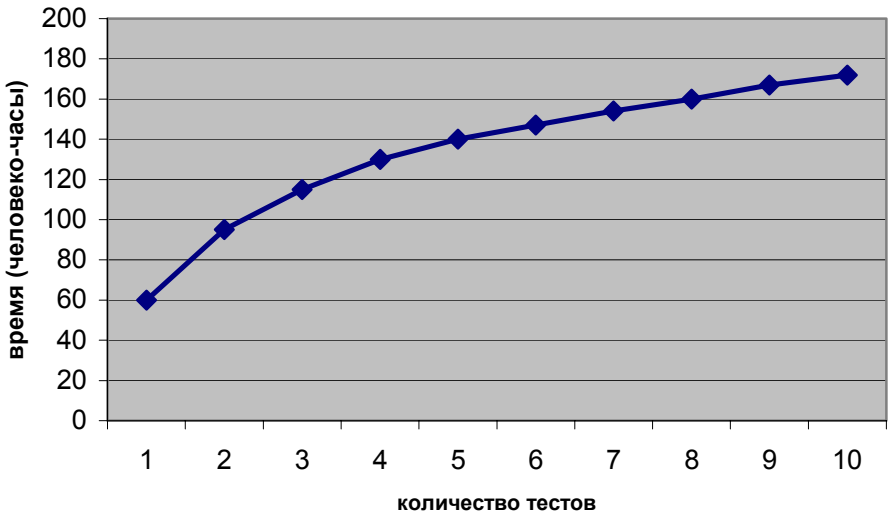

Рис. 3 Время разработки тестов в зависимости от их количества.

Метод автоматизации тестирования	инвариантность относительно языка реализации продукта	тестирование целостного продукта	тестирование на уровне внутренних подсистем продукта	тестирование через пользовательский интерфейс	трудоемкость	трудоемкость при переносе на другие платформы	эффективность
unit-тесты	-	-	+	-	**72%**	**80%**	**50%**
программные средства автоматизации тестирования	-	+	-	+	**47%**	**0%**	**5%**
Разработанный метод	+	+	+	+	**28%**	**16%**	**115%**

Таблица 1 Функциональные и технические характеристики методов автоматизации тестирования.

Критерии		Весовые коэффициенты	используемые методы		
			P_1	P_2	P_3
			unit-тесты	программные средства автоматизации тестирования	разработанный метод
Q_1	инвариантность относительно языка реализации продукта	0,01	0	0	1
Q_2	тестирование целостного продукта	0,02	0	1	1
Q_3	тестирование на уровне внутренних подсистем продукта	0,01	1	0	1
Q_4	тестирование через пользовательский интерфейс	0,15	0	1	1
Q_5	трудоемкость	0,08	0	0,57	1
Q_6	трудоемкость при переносе на другие платформы	0,03	0	1	0,80
Q_7	эффективность	0,70	0,41	0	1

Таблица 2 Нормализованные критерии оценки методов автоматизации тестирования.

Относительно трудоемкости метода следует отметить, что время, затраченное на разработку автоматизированных тестов, уменьшается с увеличением количества

разработанных тестов. Это связано с тем, что с увеличением количества разработанных тестов увеличивается набор реализованных действий и проверок, которые используются в разработке последующих тестов. Более того, начиная с некоторого момента времени, когда будет реализован весь набор действий и проверок, необходимых для тестирования продукта, время разработки одного теста будет зависеть лишь от времени написания теста в формате языка тестов. Таким образом, график суммарного времени, затраченного на разработку всех тестов, в зависимости от их количества будет иметь вид графика, изображенного на рис. 3.

Краткие сравнительные характеристики отражены в таблице 1. В таблице 2 указаны критерии, нормализованные по следующим формулам

$$q(N,L)=\frac{Q(N,L)-Q_{\min}}{Q_{\max}-Q_{\min}},\ \text{если Q}\to\max;\ q(N,L)=\frac{Q_{\max}-Q(N,L)}{Q_{\max}-Q_{\min}},\ \text{если Q}\to\min \qquad (1),$$

где {Q(N,L)} – множество критериев, N – номер варианта, L – номер критерия, q(N,L) – нормализованный критерий.

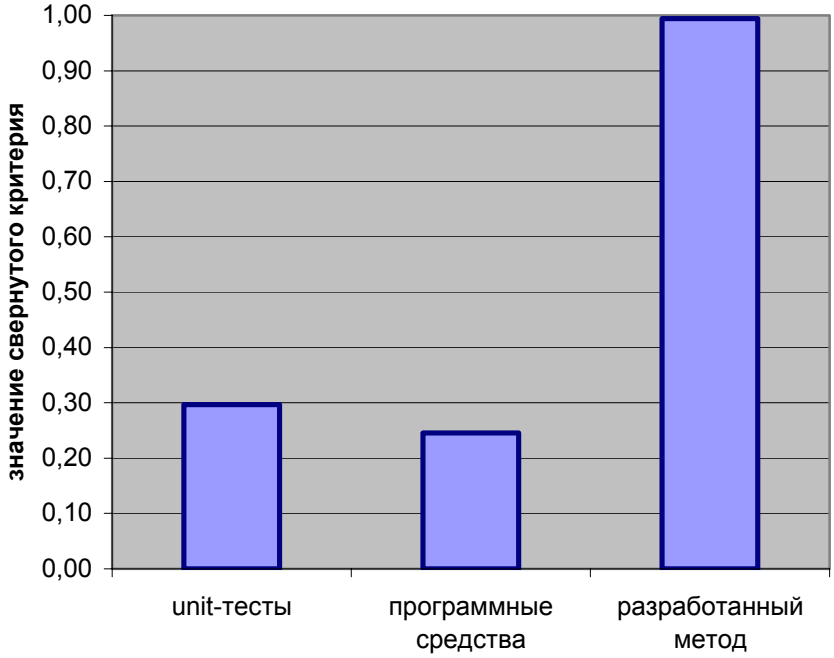

Рис. 4 Диаграмма аддитивных сверток критериев.

аддитивная свертка нормализованных критериев $F(\alpha,N)=\sum\alpha_i\cdot q_i(N),\ где\ \sum\alpha_i=1,$ (2)

при этом несложно заметить, что $F(\alpha,3)>F(\alpha,1)\ \ \forall\alpha\in\left\{\alpha_i|\ \sum\alpha_i=1\right\}$, а также то, что $F(\alpha,3)<F(\alpha,2)\Leftrightarrow\alpha_6>5\cdot(\alpha_1+\alpha_3+0.43\cdot\alpha_5+\alpha_7)>5\cdot\alpha_7$. Последние два утверждения означают, что разработанный метод эффективнее метода unit-тестов при любом наборе весовых коэффициентов. А также всегда эффективнее методов с применением программных средств автоматизации за исключением случаев, когда весовой коэффициент критерия

“трудоемкости при переносе на другие платформы” превосходит весовой коэффициент критерия “эффективности” не менее чем в пять раз, что абсолютно невозможно в условиях реальных проектов.

В заключении дается итоговая оценка проделанной работы, и формулируются основные результаты работы.

Основные результаты работы

Для достижения поставленной цели в работе были решены следующие задачи:

1. Проведен анализ существующих методов и средств автоматизации регрессионного тестирования программных продуктов, указаны их основные достоинства и недостатки.
2. Предложен новый метод разработки автоматизированных тестов. Этот метод является универсальным в плане применения при автоматизации любого программного продукта, эффективным с точки зрения обнаружения ошибок, требующий относительно низких трудозатрат, позволяющий тестирование пользовательского интерфейса и ориентированный на глубокое тестирование программных продуктов.
3. Разработан язык описания тестов, построенных по предложенной методике. Данный язык легок для обучения и применения, и ориентирован на создание тестов по предложенной методике.
4. Разработан новый подход к вопросам организации тестирования приложений с графическим интерфейсом, позволяющего автоматизировать процесс тестирования, задав информационную потребность иерархическим множеством категорий и обучив соответствующую модель на имеющихся состояниях. При этом распределенная архитектура разрабатываемой системы является масштабируемой и позволяет обеспечить высокую скорость обработки.
5. Предложена методика распознавания ключевых состояний тестируемых систем, в связи с чем решена задача системной классификации множества произвольных состояний.
6. Разработан новый подход к классификации кластеров выделенного множества состояний, на основе чего строится динамическая библиотека состояний.
7. Создан инструментарий, поддерживающий процесс автоматизации регрессионного тестирования на основе разработанного метода.
8. В качестве перспективы предложено расширение разработанного метода, ориентированного на применение при автоматизации не только регрессионного, но и системного тестирования и создан прототип системы программной поддержки данного расширения.

9. Разработанный метод, язык описания тестов и инструментарий были применены при автоматизации регрессионного тестировании промышленных программных проектов в корпорации Borland.

Практические результаты, полученные в ходе работы, позволяют сделать вывод, что поставленные задачи решены, и констатировать достижение цели всей работы. По своей значимости работа относится к новым технологическим средствам поддержки автоматизации регрессионного тестирования. Практические приложения работы важны для практики и инвестируются конкретными заказчиками.

Публикации:

1. Гриппа Г.Л., Чернoруцкий И.Г. Современные методики тестирования компьютерного программного обеспечения. Автоматизация тестирования // Материалы межвузовской научной конференции “XXXI неделя науки СПбГПУ”. Ч. 6. СПб.: Издательство СПбГПУ, 2003. – С. 24-25.
2. Гриппа Г.Л. Проблемы автоматизации тестирования приложений с графическим интерфейсом // Вычислительные, измерительные и управляющие системы.: Сборник научных трудов аспирантов и молодых ученых факультета технической кибернетики. СПб.: Изд-во Политехн. ун-та, 2004. – С. 141-142.
3. Гриппа Г.Л. Метод action-тестов автоматизации тестирования приложений с графическим интерфейсом // Вычислительные, измерительные и управляющие системы.: Сборник научных трудов аспирантов и молодых ученых факультета технической кибернетики. СПб.: Изд-во Политехн. ун-та, 2004. – С. 142-146.
4. Гриппа Г.Л. Система поддержки action-тестов. Язык action-тестов // Вычислительные, измерительные и управляющие системы.: Сборник научных трудов аспирантов и молодых ученых факультета технической кибернетики. СПб.: Изд-во Политехн. ун-та, 2004. – С. 146-156.
5. Гриппа Г.Л. Расширенный метод action-тестов автоматизации тестирования приложений с графическим интерфейсом // Вычислительные, измерительные и управляющие системы.: Сборник научных трудов аспирантов и молодых ученых факультета технической кибернетики. СПб.: Изд-во Политехн. ун-та, 2004. – С. 156-157.
6. Гриппа Г.Л., Чернoруцкий И.Г. Автоматизация тестирования программных приложений. Метод логических модулей // Технологии Microsoft в теории и практике программирования. Материалы межвузовского конкурса-конференции студентов, аспирантов и молодых ученых Северо-запада. СПб.: Изд-во Политехн. ун-та, 2005. – С. 20-22.
7. Гриппа Г.Л., Чернoруцкий И.Г. Автоматизация тестирования программных приложений. Метод логических модулей // XXXIII Неделя науки СПбГПУ: Материалы Всеросийской межвузовской научно-технической конференции студентов и аспирантов. Часть V. СПб.: Изд-во Политехн. ун-та, 2005. – С. 20-22.