МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра математической кибернетики и компьютерных наук

ТЕСТИРОВАНИЕ ДЕРЕВА ДИАЛОГОВ В СИСТЕМЕ СО СКРЫТЫМ ВНУТРЕННИМ СОСТОЯНИЕМ

АВТОРЕФЕРАТ
дипломной работы

студента 5 курса 551 группы
направления 09.03.04 – Программная инженерия
факультета КНиИТ
Малова Артёма Алексеевича

Научный руководитель
Профессор, д.ф.-м.н. _____ В.А. Романов

Заведующий кафедрой
Доцент, к.ф.-м.н. _____ С.В. Миронов

Саратов 2024

ВВЕДЕНИЕ

Разработка программного обеспечения является ключевым элементом современного технологического прогресса и играет значительную роль в различных сферах нашей жизни. В эпоху цифровизации и стремительного развития информационных технологий, программное обеспечение выступает как фундаментальная составляющая, обеспечивающая автоматизацию, инновации и оптимизацию процессов. Разработка и внедрение ПО охватывает множество этапов – от анализа и сбора требований до проектирования, тестирования, внедрения и поддержки, что требует высокого уровня профессионализма на каждом из этапов.

Разработка программного обеспечения постоянно усложняется вследствие того, как растут масштабы бизнес-процессов, увеличивается быстродействие аппаратной части, растёт количество платформ, на которых надо представлять своё решение (Android, iOS, web, Win, Linux, macOS). В качестве примера можно взять браузер Chrome или ядро Линукса. В начале оба продукта содержали несколько тысяч строк кода, в настоящее же время репозиторий Chromium содержит более 40 миллионов строк кода. За несколько десятилетий разработка программного обеспечения перешла от энтузиастов, пишущих свой код в одиночестве, к огромным командам. Современная команда разработки вполне может включать в себя сотни и даже тысячи человек. В таких условиях программное обеспечение и его разработка являются независимым процессом обладающим большим количеством артефактов и стадий, вследствие чего тестирование стало необходимой частью разработки комплексного ПО. В последующем оно стало представлять из себя отдельную профессию на рынке труда.

На текущем этапе развития программного обеспечения, тестирование стало неотъемлемой частью общего цикла разработки. Без него трудно представить существование любой современной IT-компании. Тестирование позволяет выявлять и исправлять ошибки на ранних стадиях, что существенно снижает риски и затраты на исправление проблем в будущем. Качество программного продукта напрямую зависит от эффективности тестирования, и квалификации команды. Современные методики тестирования, помогают ускорить процесс разработки и обеспечить более высокую стабильность выпускаемых версий ПО. Таким образом, тестирование играет ключевую роль в достижении высокого уровня качества и конкурентоспособности программных продуктов. К сожалению, одним только тестированием качества

не добиться. Рецепт получения высокого качества: смешивайте разработку и тестирование в блендере, пока они не станут единой субстанцией [1].

Основной задачей тестирования является выявление недочетов, для исключения вероятности попадания их к конечному пользователю. Одна из целей тестирования – проверка максимально возможного количества сценариев работы ПО при затратах времени, отвечающих установленным временным рамкам.

Для достижения качества необходимо постараться покрыть все возможные сценарии использования ПО. Для этого моделируются действия пользователя. В случае диалоговых систем с большим количеством вложенных диалогов и сложными бизнес-процессами (CRM, игры, экспертные системы) количество возможных состояний и сценариев, по которым к ним приходят, растёт нелинейно. При большом количестве исходных данных и их взаимодействиях между собой, количество сценариев для проверки растет экспоненциально, поэтому возникает проблема нехватки времени на покрытие проверками достаточного количества функционала исходного ПО. Как следствие, в данном случае желательно будет автоматизировать часть проверок и сценариев работы с программным обеспечением для сокращения времени, затрачиваемого на тестирование.

При тестировании многоуровневых диалогов и систем без возврата может возникнуть интересная проблема технического характера: когда у системы есть скрытое внутреннее состояние, то дефекты, связанные с этим состоянием, тяжело найти и локализовать, поскольку отмена и повтор действия могут быть невозможны, а повторный запуск – может не гарантировать воспроизведения исходного состояния.

Цель работы: разработать способ тестирования системы с неизвестным внутренним состоянием.

Из цели вытекают следующие задачи, которые должны быть решены в ходе написания настоящей работы:

- изучить существующие виды и подходы к тестированию диалоговых систем;
- изучить методы сохранения и воспроизведения состояния процесса ОС;
- реализовать метод эмуляции действий пользователя при работе с десктопным приложением;

- спроектировать архитектуру автоматизированного тестирующего комплекса;
- разработать программу тестирования системы со скрытым внутренним состоянием;
- проверить комплекс на тестовой задаче (диалог с взаимоисключающими вариантами развития действий).

Дипломная работа состоит из введения, 4 разделов, заключения, списка использованных источников и 1 приложения. Общий объем работы – 59 страниц, из них 40 страницы – основное содержание, включая 15 рисунков и 1 таблицу, список использованных источников из 17 наименований.

КРАТКОЕ СОДЕРЖАНИЕ

Раздел 1 данной работы посвящен основным сведениям теории тестирования, необходимым для лучшего понимания специфики работы. В частности, в подразделе 1.1 приведена классификация подходов к тестированию по доступу к коду и архитектуре приложения, даны их определения, необходимые для рассмотрения темы работы и разобраны плюсы и минусы подходов.

Подразделы 1.2 и 1.3 проводят классификацию по степени автоматизации и запуску кода соответственно, что необходимо так как итоговый комплекс будет автоматизироваться.

В подразделе 1.4 рассматриваются типы тестирования ПО, а также даются определения типам которые будут использованы в дальнейшем в настоящей работе.

Подраздел 1.5 дает определение качеству ПО и рассматривает показатели качества.

Качество программного обеспечения – сумма функциональности и технических характеристик программного продукта, отвечающих за возможность выполнения сформулированных или подразумевающихся задач.

Раздел 2 содержит сведения о системе с неизвестным внутренним состоянием, тестирование которой является одной из задач работы. В качестве примера подобной системы приводится игра в наперстки: наглядной моделью подобной системы со случайным фактором и одноуровневым деревом диалогов является игра в “напёрстки”.

Данная игра является системой, содержащей в себе стохастический фактор. В игре есть внутренний диалог – случайное перемещение шарика и неизвестное внутреннее состояние для пользователя (где именно находится шарик).

Правила игры:

- При входе в игру размещается горошина.
- Необходимо проверить, что горошина есть всегда.
- Кнопки назад нет.

Проблема:

- При повторном запуске горошина окажется в другом месте.

Задача: необходимо убедиться, что в системе нет каких-либо системных искажений:

- горошина есть;
- горошина только одна.

В подразделе 2.1 рассматривается тестирование вариантов диалога, дается определения диалога, а также объясняется основная проблема, возникающая в случае тестирования подобных систем.

Состояние, когда система в ответ на несколько действий может менять свое состояние – будет называться диалогом. Когда вариантов диалога много, количество операций для тестирования начинает расти экспоненциально, а значит задачу необходимо автоматизировать. В случае автоматизации процесса тестирования диалога у нас возникает следующая техническая проблема: если у системы есть внутреннее состояние и у программного продукта есть дефект, связанный с описанием того, как это состояние меняется в зависимости от действий пользователя, то дефекты связанные с ошибками описания или перехода в таком случае тяжело найти и локализовать.

Подразделы 2.2 и 2.3 дают классификацию планируемого решения, а также рассматривают его архитектуру.

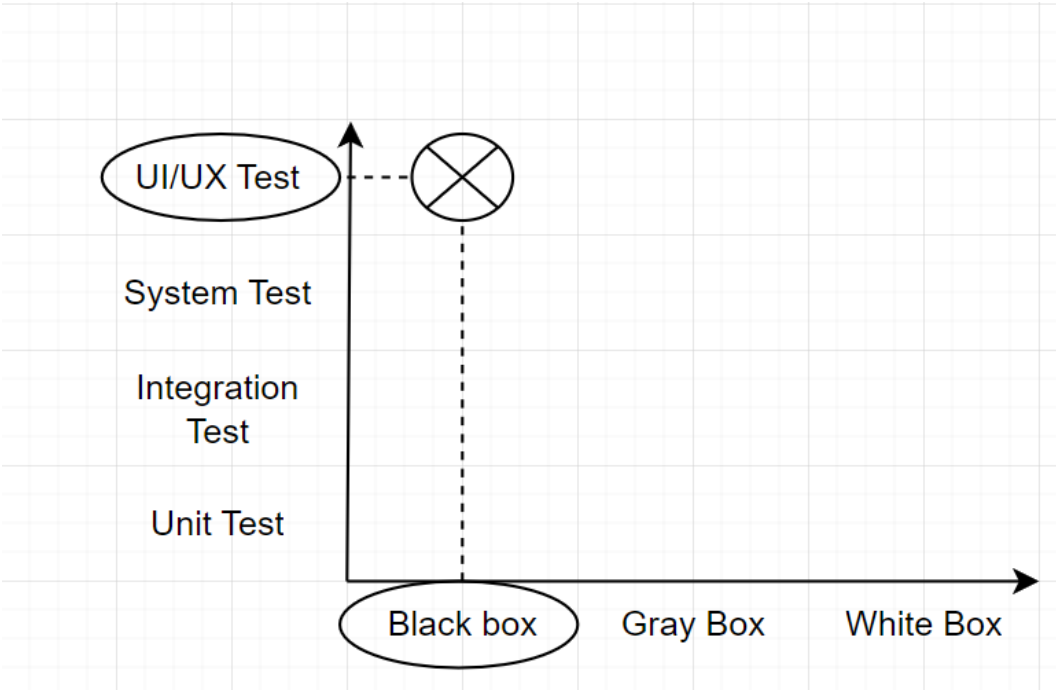

Рисунок 1 — Классификация решения

Для наглядного представления классификации решения нужно обратиться к рисунку 1. По горизонтали указана классификация методов тестирования по доступу к коду и архитектуре приложения, по вертикали – уровень тестирования.

В данной работе используется виртуальная машина для фиксации состояния тестируемого приложения: восстановленное из снапшота приложение имеет то же самое состояние которое оно имело на момент заморозки / создания снапшота. Внутренний драйвер для управления тестируемым приложением, и драйвер, и приложение при этом находятся внутри виртуальной машины. Само тестируемое приложение, и оркестратор виртуальной машины, выполняющий действия со снапшотами.

В таком случае итоговая архитектура тестирующего комплекса получается следующей и отображена на рисунке 2:

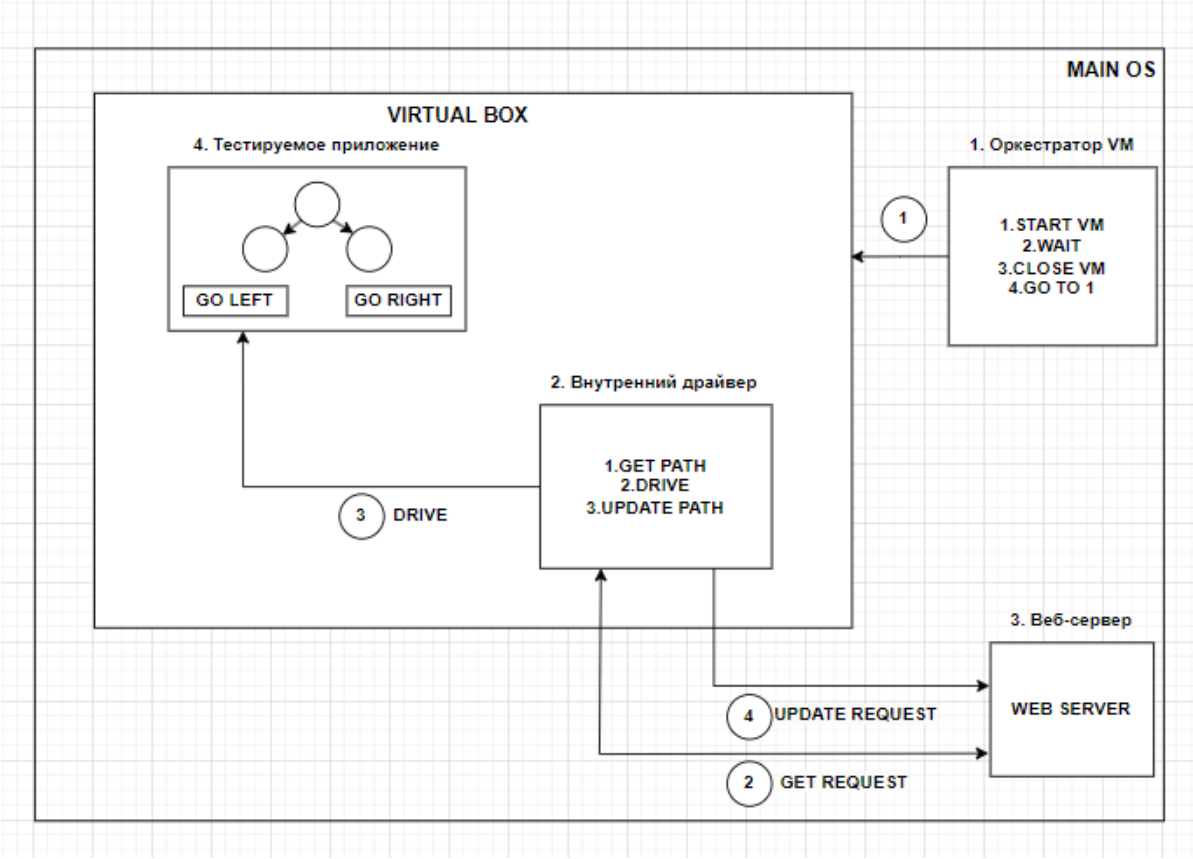

Рисунок 2 — Архитектура решения

Также, в разделе 2.3 приводится таблица 1 с функциональным разбиением итогового комплекса, для лучшего понимания архитектуры реализуемого приложения.

Таблица 1 - Функциональное разбиение

Приложение №	**Название приложения**	**Сценарий работы**
1	Оркестратор виртуальной машины	1.Запуск сохраненного снапшота виртуальной машины. 2.Ожидание выполнения работы внутреннего драйвера. 3.Закрытие виртуальной машины. 4.Возврат к пункту.
2	Внутренний драйвер	1.Обращение и отправка GET запроса к веб-серверу. 2.Получение и обработка ответа от веб-сервера 3.Управление тестируемым приложением 4.Отправка UPDATE запроса к веб-серверу для его сохранения.
3	Веб-сервер	1.Обработка входящих запросов 2.Отправка и сохранение переменной, отвечающей за путь, который необходимо пройти внутреннему драйверу
4	Тестируемое приложение	1.Представляет собой бинарное дерево диалогов позволяющее перейти к одному из потомков

Раздел 3 рассматривает состав итогового комплекса и посвящен разбору технической реализации каждого из компонентов финального решения.

Раздел 3.1 рассказывает о тестируемом приложении, реализованном для проверки комплекса. Для удобной отладки и реализации комплекса в качестве приложения был выбран и реализован наглядный пример исходной программы (программа 4) — тестируемое приложение представляющее из себя модель дерева диалогов. Структура диалога не является фиксированной и имеет возможность расширения для облегчения тестирования возможностей комплекса путём создания сложных нелинейных диалогов без изменения программы.

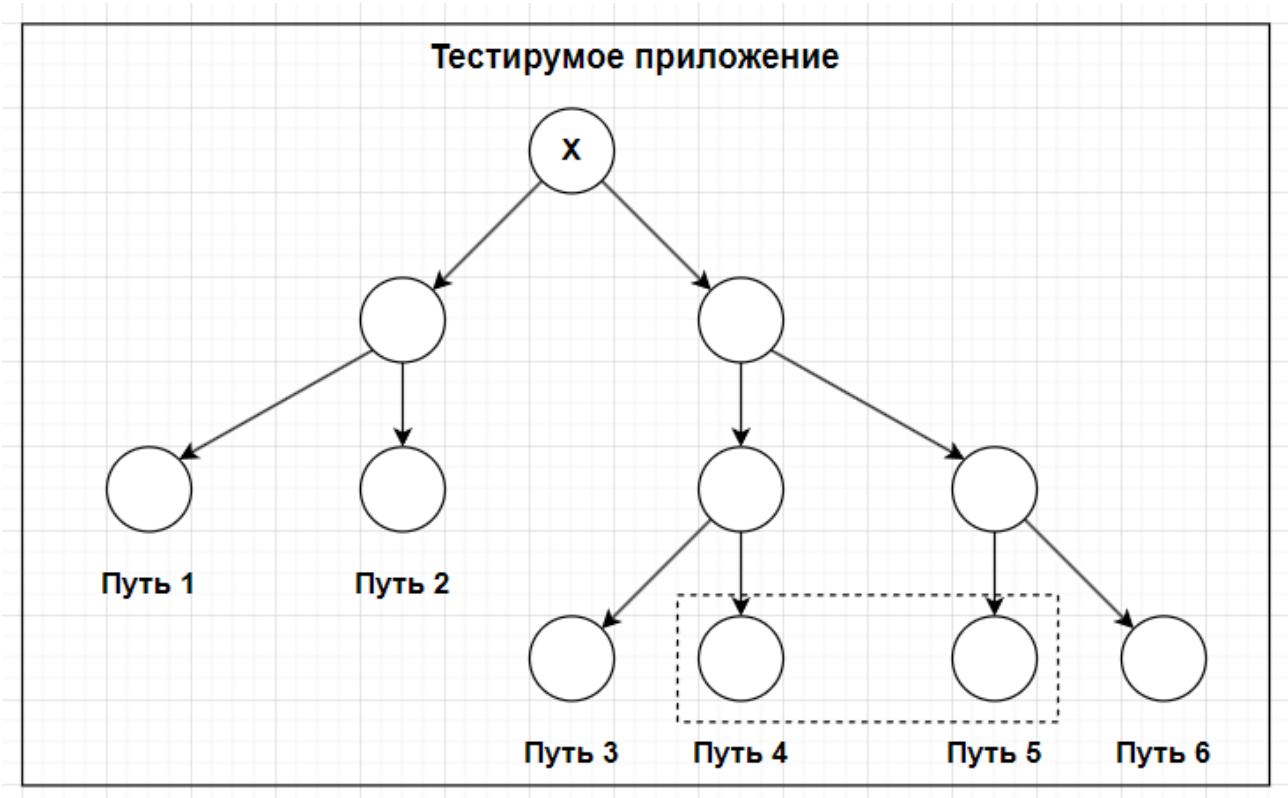

Рисунок 3 —Тестируемое приложение

На рисунке 3 представлена реализация исходной программы, которую предстоит тестировать. Правила работы с приложением в данном случае будут схожи с подходом к игре в наперстки:

- приложение состоит из 6 путей
- возможности вернуться назад на уровень выше — нет
- случайное состояние генерируется в точке X

- итоговые точки путей 4 и 5 являются взаимоисключающими и не могут существовать одновременно
- генерация финальной точки путей 4 и 5 зависит от случайного состояния в точке X

Необходимо:

- проверить общее количество путей
- убедиться в том, что концы путей являются взаимоисключающими

Проблема:

- если попробовать провести несколько запусков тестируемого приложения это поможет пользователю понять о существовании или отсутствии интересующих его путей, но не даст понимании о связанных с ними

путями так как при переходе к одному из вариантов мы не можем получить информацию о втором

В подразделе 3.2 настоящей работы рассматривается реализованный Web-server, а также дается информация о его технической реализации и способах взаимодействия с другими приложениями посредством API, который был развернут с использованием swagger, ngrok, open-api.

Подразделы 3.3 и 3.4 рассматривают внутренний драйвер управления тестируемого комплекса и оркестратор виртуальной машины, отвечающий за управление общим решением. Помимо этого, в разделе рассматриваются технические аспекты реализации написанного решения. Для реализации функционала использовался WinAppDriver, позволяющий эмулировать действия пользователя в среде Windows. Фактически является Selenium`ом для Windows. Финальная программой в данном списке является оркестратором, необходима для запуска и выключения виртуальной машины, созданию снапшота и перезапуска общего цикла.

Итоговым разделом работы выступает раздел 4, содержащий в себе Sequence диаграмму, представленную на рисунке 4 и позволяющую визуализировать общий ход итогового решения.

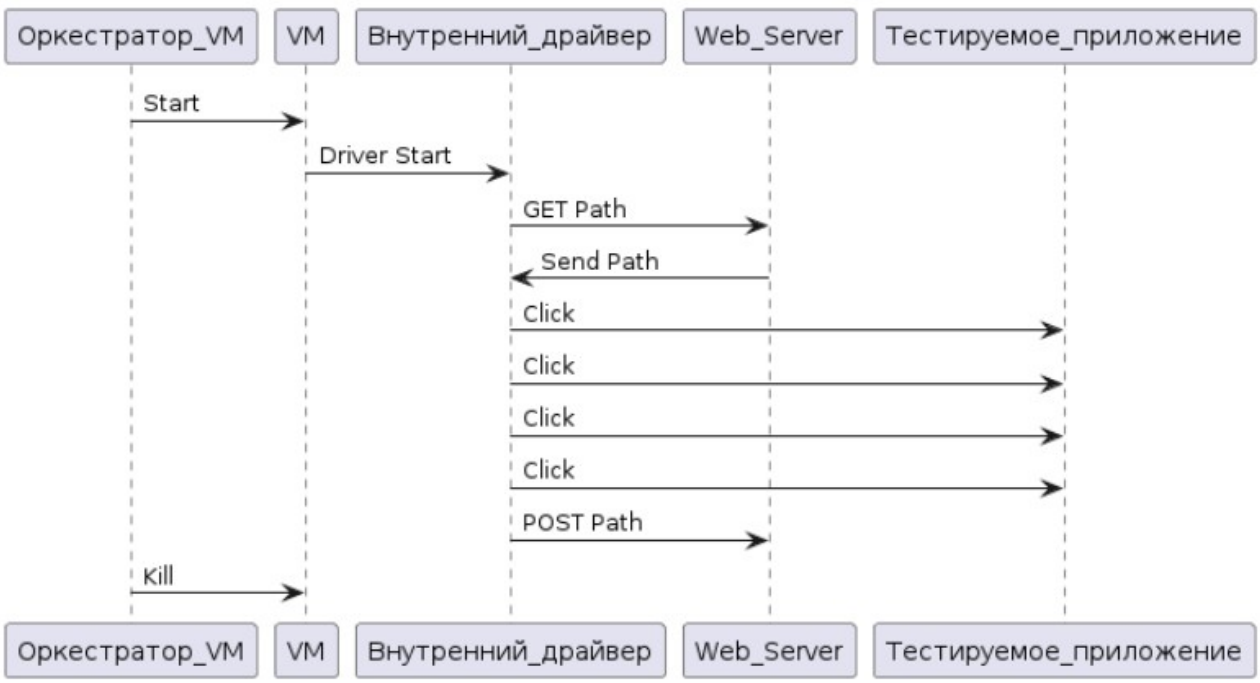

Рисунок 4 – Sequence диаграмма

1. Оркестратор виртуальных машин производит запуск слепка виртуальной машины.
2. Виртуальная машина запускает внутренний драйвер предназначенный для тестируемого приложения
3. При запуске внутреннего драйвера он отправляет GET запрос на веб-сервер для получения информации о текущем последнем сохраненном пути.
4. Веб-сервер в ответ на запрос внутреннего драйвера отправляет информацию о сохраненном пути. Если запуск виртуальной машины первый, то на веб-сервере хранится фиксированное значение равное 1, что позволит драйверу тестируемого приложения понять, что необходимо приступить к первому пути и исключит необходимость обработки null значения хранящегося на веб-сервере.
5. Внутренний драйвер производит манипуляции с тестируемым приложением, доводя его до необходимой точки на дереве диалоговой системы.
6. После выполнения всех необходимых действий с приложением, внутренний драйвер отправляет POST запрос на Web-server для обновления хранящейся там информации о последнем пройденном пути.
7. В самом конце действие переходит к оркестратору виртуальных машин для завершения работы с текущим слепком виртуальной машины.

После полного прохождения этапов описанных на sequence диаграмме происходит зацикливание системы и действия повторяются для нового снапшота виртуальной машины.

ЗАКЛЮЧЕНИЕ

Разработанный комплекс позволяет тестировать оконные приложения на Windows и фиксировать внутреннее состояние системы с помощью заморозки виртуальной машины, это дает возможность тестировать системы с неизвестным внутренним состоянием, что в случае без использования снапшотов системы было возможно только в том случае если тестовый контур был заложен на стадии проектирования приложения или при непосредственном вмешательстве в работу тестируемого приложения — изменение параметров в системе или базе данных и т.д.

Использование оркестратора в совокупности с веб-сервером позволяет создавать гибкие сценарии тестирования с автоматизацией так как имеется возможность создавать задачи на тестирование в большом количестве— создание слепков виртуальных машин их запуск и отключение.

В будущем данный комплекс допускает доработку в виде создания промежуточным снапшотов для тестирования больших систем содержащих в себе варианты диалога и неизвестное состояние, что может привести к увеличению производительности и скорости решения поставленной задачи так как необходимость в избыточных действиях отпадет.

Так как драйвер тестируемого приложения получает информацию с помощью Web-сервера, то имеется возможность горизонтального масштабирования решения. Как пример — запуск виртуальных машин на нескольких устройствах параллельно, что потребует переработки веб-сервера для добавления к нему возможности параллельной обработки поступающих запросов, но кратно увеличит скорость обработки всех ветвей диалога.

В ходе написания дипломной работы поставленная цель была достигнута: был разработан способ тестирования системы с неизвестным внутренним состоянием. Таким образом, все поставленные задачи решены, цель работы достигнута.