РОЧЕВ К. В., ЖИФАРСКИЙ В. Д., ЖЕРЕБЦОВ В. П. АВТОМАТИЗАЦИЯ ТЕСТИРОВАНИЯ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ БЕЗ НАПИСАНИЯ КОДА

УДК 004.4'24, ГРНТИ 50.05.13

Автоматизация тестирования программного обеспечения без написания кода

К. В. Рочев[1], В. Д. Жифарский[1], В. П. Жеребцов[2]

[1]Ухтинский государственный технический университет, г. Ухта;
[2]Сервисный центр, г. Ухта;

В данной статье рассматриваются основные принципы и методы тестирования приложений, и предлагается способ автоматизации ручного тестирования. Представлена разработка системы тестирования, которая упрощает и ускоряет процесс ручного тестирования за счет записи и воспроизведения тестовых сценариев без написания кода, а также обеспечивает визуальное сравнение результатов записанного тесового сценария и его воспроизведения.

Ключевые слова: *система, тестирование, дефекты, ошибки, тестовые сценарии, сравнение, Windows API*

Software Test Automation without writing code

K. V. Rochev[1], V. D. Zhifarsky[1], V. P. Zherebtsov[2]

[1]Ukhta State Techincal University, Ukhta;
[2]Service center, Ukhta;

This article discusses the fundamental principles and methods of application testing and proposes a way to automate manual testing. It presents the development of a testing system that simplifies and accelerates the manual testing process by recording and replaying test scenarios without the need for coding. Additionally, the system provides visual comparison of the results between the recorded test scenario and its replay.

Keywords: *system, testing, defects, errors, test scenarios, comparison, Windows API*

Введение

Информационные системы играют важную роль в современном мире, обеспечивая автоматизацию процессов [1] и повышение эффективности работы организаций. Однако, разработка и внедрение информационных систем требует значительных затрат времени и ресурсов. Для минимизации рисков и повышения качества разрабатываемых систем, необходимо проводить тестирование приложений.

Тестирование программного обеспечения (ПО) [2] является критически важным этапом в процессе разработки, обеспечивающим проверку и валидацию

программного продукта на соответствие заявленным требованиям и ожиданиям пользователей. Основной целью тестирования является выявление дефектов и ошибок в программном обеспечении, чтобы обеспечить его надежность, безопасность и корректность работы.

Процесс тестирования включает в себя несколько этапов: планирование тестирования, разработка тестовых сценариев, выполнение тестов, анализ результатов [3] и документирование обнаруженных дефектов. Существуют различные виды тестирования, такие как функциональное, нефункциональное, регрессионное, нагрузочное и стресс-тестирование, каждое из которых решает свои специфические задачи (Таблица 1) [4-8].

Таблица 1 – Основные типы тестирования, их цели, методы и особенности

Вид тестирования	**Цель**	**Методы/Подходы**	**Особенности**
1	2	3	4
Функциональное	Проверка соответствия функциональности ПО требованиям	Тестирование по требованиям, тест-кейсы, сценарии использования	Проверяет, что система работает так, как ожидается
Регрессионное	Убедиться, что изменения не повлияли на существующую функциональность	Повторное выполнение ранее написанных тестов	Проводится после внесения изменений в код
Интеграционное	Проверка взаимодействия между модулями или системами	Снизу вверх, сверху вниз, "сэндвич" (комбинированный)	Оценивает корректность взаимодействия компонентов
Системное	Проверка системы в целом на соответствие требованиям	Сквозное тестирование, тестирование в среде, близкой к реальной	Охватывает все аспекты системы
Модульное (Unit)	Проверка отдельных компонентов или модулей	Написание юнит-тестов (например, с использованием JUnit, NUnit)	Выполняется разработчиками на уровне кода
Нагрузочное	Проверка производительности системы под нагрузкой	Нагрузочные тесты, стресс-тесты, тестирование стабильности	Оценивает, как система ведет себя при высокой нагрузке
Юзабилити	Оценка удобства использования интерфейса	Тестирование с участием пользователей, A/B-тестирование	Фокусируется на пользовательском опыте
Безопасность	Проверка уязвимостей и защищенности системы	Пентест, анализ кода, тестирование на проникновение	Оценивает устойчивость системы к атакам

Вид тестирования	Цель	Методы/Подходы	Особенности
1	2	3	4
Продолжение Таблицы 1			
Дымовое (Smoke)	Быстрая проверка основных функций системы	Выполнение минимального набора тестов	Используется для проверки работоспособности после сборки
Приемочное	Проверка готовности системы к передаче заказчику	Тестирование по критериям приемки, UAT (User Acceptance Testing)	Проводится заказчиком или конечными пользователями
Кросс-браузерное	Проверка корректности работы в разных браузерах	Тестирование на различных браузерах и устройствах	Важно для веб-приложений
Локализационное	Проверка адаптации продукта для разных языков и регионов	Тестирование перевода, форматов дат, валют и т.д.	Оценивает корректность локализации
API-тестирование	Проверка работы API и взаимодействия между системами	Использование инструментов (Postman, SoapUI), написание скриптов	Оценивает корректность запросов и ответов
Автоматизированное	Ускорение выполнения тестов и повышение их точности	Использование фреймворков (Selenium, Cypress, JUnit и др.)	Требует навыков программирования и настройки
Ручное	Проверка функциональности вручную	Выполнение тест-кейсов без использования скриптов	Подходит для exploratory-тестирования и проверки сложных сценариев
Exploratory	Исследование системы без заранее подготовленных тест-кейсов	Свободное тестирование, основанное на интуиции и опыте тестировщика	Полезно для поиска неочевидных багов
Конфигурационное	Проверка работы системы в разных конфигурациях	Тестирование на различных ОС, устройствах, версиях ПО	Оценивает совместимость системы
Тестирование доступности	Проверка доступности системы для людей с ограниченными возможностями	Использование стандартов (WCAG), инструментов для анализа доступности	Оценивает соответствие стандартам доступности

Автоматизированное тестирование приложений является одним из наиболее эффективных методов проверки работоспособности и соответствия требованиям. Оно позволяет сократить время на проведение тестирования, повысить его качество и снизить вероятность ошибок.

Целью данной работы является разработка системы автоматизированного тестирования приложений. Цель системы – упростить и ускорить процесс ручного тестирования приложений за счет записи и автоматизированного воспроизведения тестовых сценариев.

В рамках работы приведены основные принципы и методы автоматизированного тестирования, а также проведен анализ существующих систем и инструментов для тестирования.

Процесс ручного тестирования приложений до автоматизации включает в себя следующие шаги:

1. Определение целей и задач тестирования, выбор методов и инструментов тестирования;
2. Сбор требований к приложению, определение функциональных и нефункциональных требований;
3. Разработка тест-кейсов на основе требований к приложению;
4. Проведение тестирования приложения с использованием ручных методов тестирования;
5. Анализ результатов тестирования, выявление ошибок и проблем;
6. Внесение изменений в приложение для исправления ошибок и проблем;
7. Проведение повторного тестирования после внесения изменений.

Проблемы, которые могут возникнуть в процессе тестирования, включают в себя:

− Недостаточная проверка функциональности: если тестировщик не проверил все функции приложения, то могут возникнуть ошибки [9], которые приведут к некорректной работе приложения;

− Недостаточная проверка производительности: если тестировщик не проверил производительность приложения, то могут возникнуть проблемы с задержками и зависаниями.

Внедрение информационной системы автоматизированного тестирования позволит улучшить качество тестирования, исключая ошибки тестировщика и предоставляя полные данные о производительности приложения и ошибках, произошедших во время тестирования. Также автоматизация тестирования позволит повторно запускать тесты необходимое количество раз для проверки стабильности работы тестируемого приложения.

Существующие системы автоматизации ручного тестирования

При поиске программ для возможного решения задач проекта были рассмотрены такие системы как:

1. Selenium – это популярный набор инструментов для автоматизации веб-браузеров. Он предоставляет возможность автоматизировать взаимодействие с веб-страницами, поддерживает различные языки программирования и интеграцию с различными фреймворками для тестирования. VSProfiler – коммерческий профайлер от корпорации Microsoft, входящий в состав пакета Visual Studio Team System (VSTS) и версии Development Edition среды разработки Visual Studio.

2. Ranorex Studio – это инструмент для автоматизации тестирования, поддерживающий тестирование веб-, мобильных и настольных приложений. Он предоставляет функции для записи и воспроизведения тестов.

3. Appium – это инструмент для автоматизации мобильных приложений на платформах Android и iOS. Он поддерживает множество языков программирования и фреймворков.

Проведём сравнение разрабатываемой системы с возможными аналогами (Таблица 2):

Таблица 2 – Сравнение преимуществ разрабатываемой системы и ее аналогов

Преимущества	SfAT (разрабатываемая система)	Selenium	Ranorex Studio	Appium
Создание кейсов	+	-	+	-
Запись действий	+	-	+	+
Воспроизведение действий	+	+	+	+
Сравнение и результатов	+	-	-	-
Создание кейсов без написания кода	+	-	+	+

На рынке представлено множество инструментов для автоматизированного тестирования [10] приложений, таких как Selenium, Appium и Ranorex Studio. Однако ни один из них не предоставляет полный набор возможностей, которые предполагает разрабатываемая система:

1. **Создание кейсов без написания кода:** только Ranorex Studio поддерживает эту функцию.

2. **Запись и воспроизведение действий:** Selenium и Appium требуют дополнительных инструментов или знаний программирования для записи действий.

3. **Сравнение скриншотов:** ни один из рассмотренных инструментов не поддерживает встроенное сравнение скриншотов.

Функциональные требования к системе:

1. Система должна позволять создавать и редактировать тест-кейсы;
2. Система должна предоставлять возможность запуска тест-кейсов для записи действий.
3. Система должна предоставлять возможность запуска тест-кейсов для воспроизведения действий в автоматическом режиме.
4. Система должна предоставлять возможность анализа результатов тестирования;

Границы системы

1. Тестировщик – создает сценарии тестирования, инициирует запуск тестирования, получает и анализирует результаты тестирования;
2. Тестируемое ПО – получает последовательность действий в соответствии со сценарием тестирования.

Границы системы можно показать при помощи контекстной диаграммы (Data Context Diagram) [11] (**Ошибка! Источник ссылки не найден.**1).

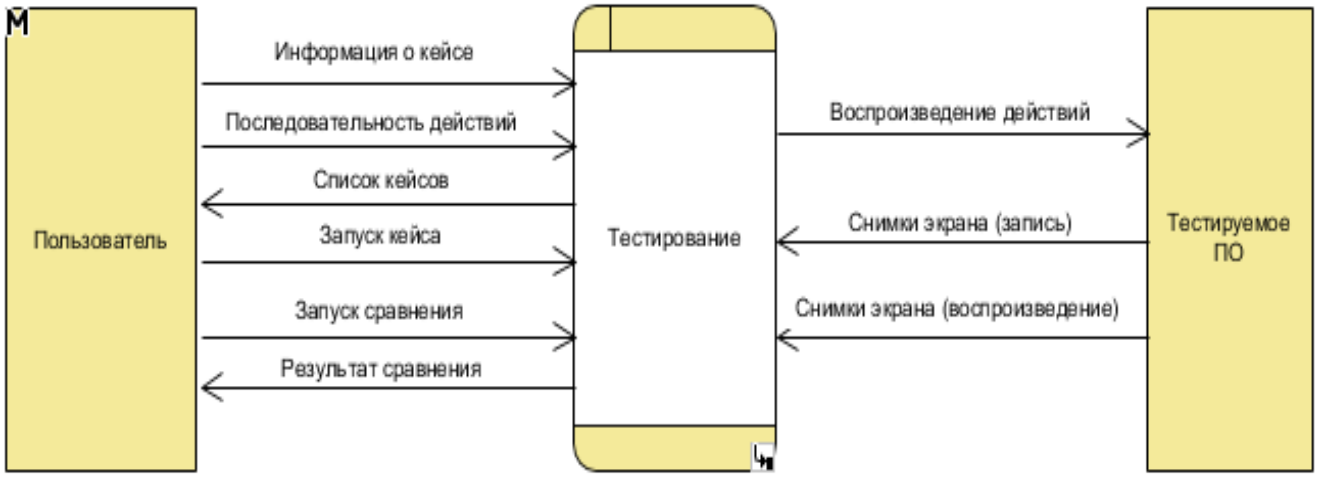

Рисунок 1 – Контекстная диаграмма «как будет»

Далее можно определить процессы, на которые система может быть декомпозирована.

1. Создание кейсов
2. Запись действий
3. Воспроизведение действий
4. Сравнение и результаты

Процессы отображены на диаграмме потоков данных первого уровня [11] (Рисунок 2).

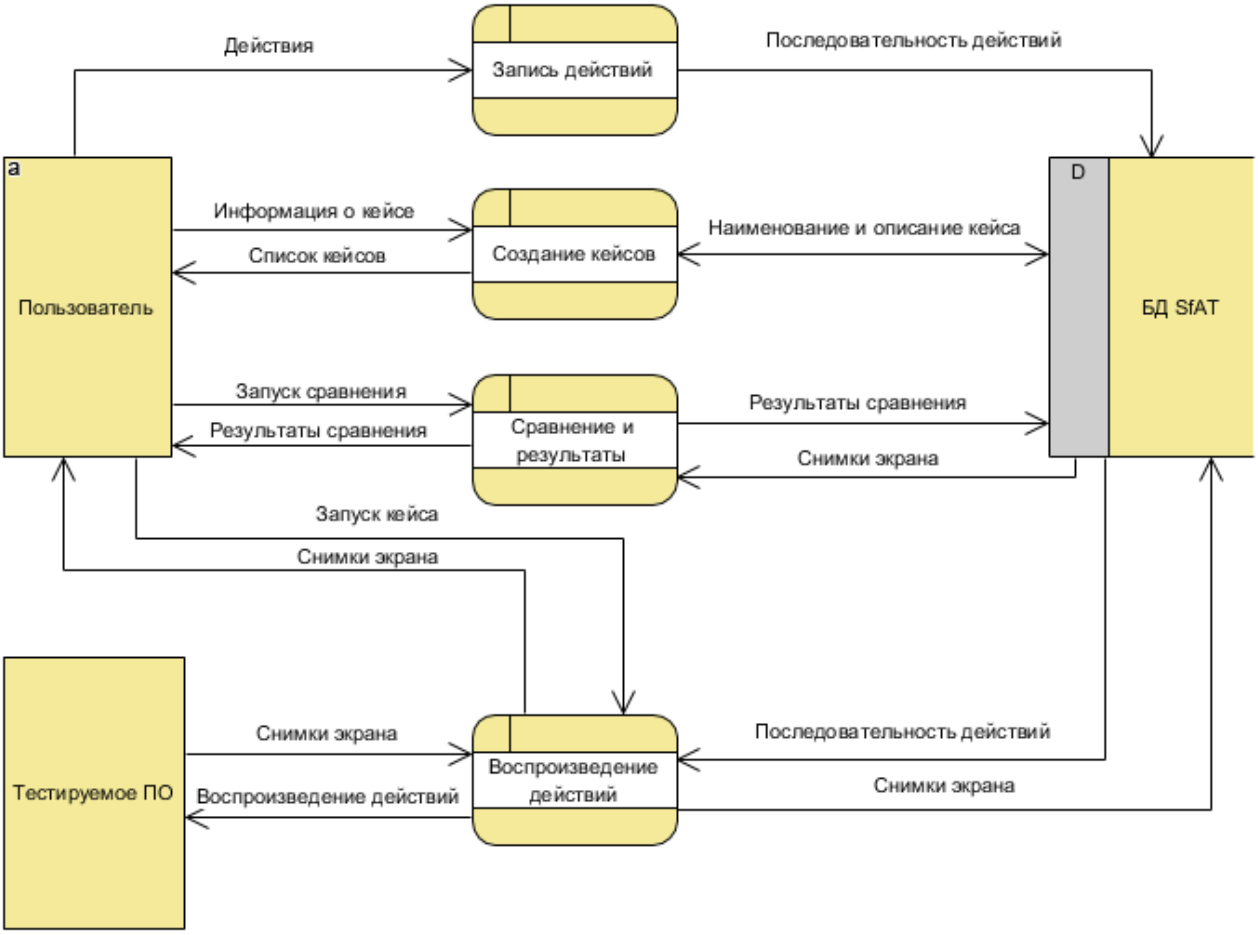

Рисунок 2 – Диаграмма потоков данных «как будет» 1-го уровня

Для создания тест-кейсов необходимо устанавливать перехватчики ввода мыши. Перехват ввода осуществляется при помощи статического класса MouseHook [12]. Функция `SetHook` устанавливает глобальный низкоуровневый перехватчик мыши, вызывая Windows API функции SetWindowsHookEx() с параметрами текущего процесса и его основного модуля, чтобы операционная система знала, где находится функция обратного вызова. Это необходимо для корректной загрузки и выполнения функции обратного вызова в правильном контексте. Указание модуля обеспечивает стабильность и корректность работы перехватчика, позволяя перехватывать события мыши по всей системе (Листинг 1).

Листинг 1 – Установка перехватчика мыши

```
private static IntPtr SetHook(LowLevelMouseProc proc)
{
   using (Process curProcess = Process.GetCurrentProcess())
   using (ProcessModule curModule = curProcess.MainModule)
   {
      return SetWindowsHookEx(WH_MOUSE_LL,
         proc, GetModuleHandle(curModule.ModuleName), 0);
   }
}
```

Далее события мыши обрабатываются в функции обратного вызова HookCallback. Когда происходит событие мыши, функция проверяет код события nCode и тип сообщения wParam. В случае если событие соответствует интересующему типу WM_LMBUTTON, то функция увеличивает счётчик событий который помогает отслеживать количество зарегистрированных событий мыши и извлекает информацию о событии. Затем создаётся объект MouseButtonEventArgs для передачи информации о событии мыши в обработчик событий. После завершения обработки события происходит сброс флага isClickPending для предотвращения обработки одного и того же события повторно.

Листинг 2 – Обработчик хука для отслеживания событий мыши

```
private static IntPtr HookCallback(int nCode, IntPtr wParam, IntPtr lParam)
{
   if (nCode >= 0 && (MouseMessages)wParam == MouseMessages.WM_LBUTTONUP)
   {
      if (!isClickPending)
      {
         isClickPending = true;
         EventCounter++;
         Console.WriteLine("Registered events count: " + EventCounter);

         MSLLHOOKSTRUCT hookStruct = (MSLLHOOKSTRUCT)
              Marshal.PtrToStructure(lParam, typeof(MSLLHOOKSTRUCT));
         MouseButtonEventArgs args = new MouseButtonEventArgs(
              Mouse.PrimaryDevice, 0, MouseButton.Left)
         {
            RoutedEvent = UIElement.MouseLeftButtonDownEvent,
            Source = null
         };
         MouseActionEvent(null, args);

         isClickPending = false;
      }
   }
```

Для проверки корректности полученных результатов выполнения тестов производится сравнение скриншотов по формуле (1), в ней вычисляется степень совпадения цветовых компонентов:

$$K = \frac{\sum_{i=1}^{W}\sum_{j=1}^{H}(|r_{Aij}-r_{Bij}|+|g_{Aij}-g_{Bij}|+|b_{Aij}-b_{Bij}|)}{W\times H\times max(r)\times 3} \quad (1)$$

где: K – коэффициент совпадения скриншотов (0-1),
W – ширина изображения, пикселей,
H – высота изображения, пикселей,
A – исходный скриншот кейса,
B – скриншот тестового случая,
r, g, b – цветовые компоненты пикселей,
max(r) – максимальное значение цветовой компоненты.

После вычисления коэффициента совпадения, проводится его сравнение с порогом соответствия, который может быть задан для каждого конкретного тестируемого приложения. Хорошую эффективность сравнения показывают значения порогового коэффициента в диапазоне 0.03-0.08%.

На начальном экране приложения (Рисунок 3) производится создание и редактирование тест-кейсов. Начать, завершить, воспроизвести запись можно при помощи соответствующих кнопок.

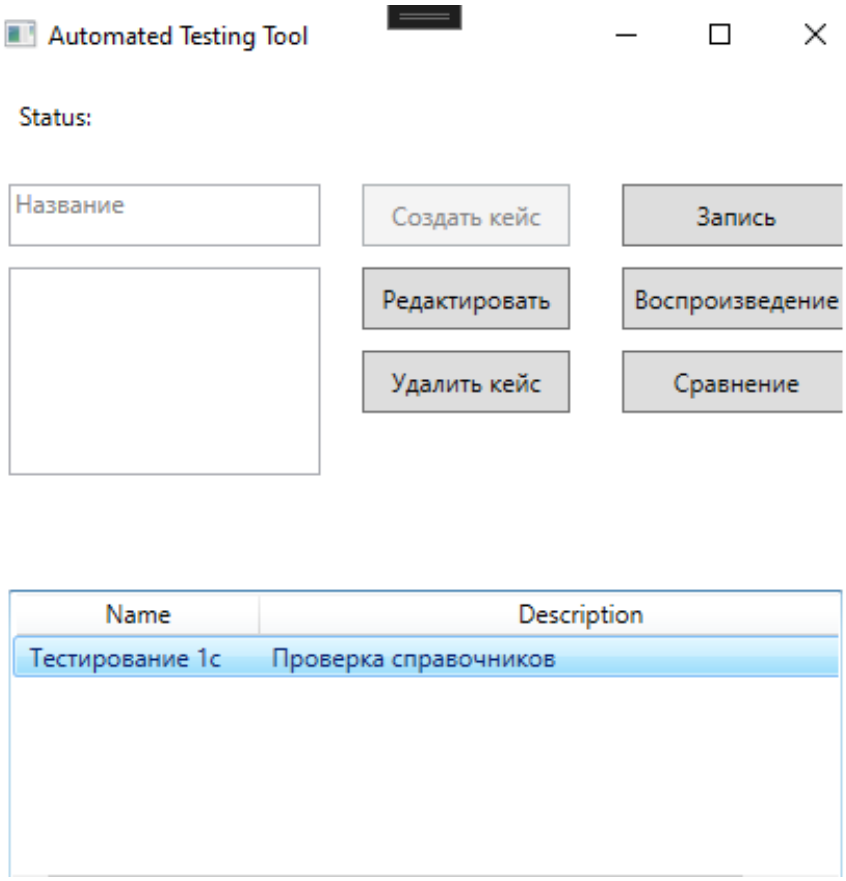

Рисунок 3 – Начальный экран приложения

Основной сценарий применения приложения предполагает, что тестировщик, который до этого тестировал приложения вручную, точно так же их тестирует, только со включенной программой записи, и отмечает, где начинаются, а где заканчиваются отдельные кейсы. Успешные кейсы сохраняет, остальные архивирует или удаляет.

После очередных изменений тестируемого программного обеспечения тестировщик может запускать успешные ранее тесты уже на автопилоте (например, для больших систем можно запускать набор сценариев на ночь, либо в ускоренном режиме), и смотреть, какие из кейсов работают, как раньше, а в каких что-то изменилось. Для изменившихся требуется перепройти сценарий вручную и либо заменить тестовые кейсы на новые (если поведение системы изменилось планово), либо, если произошли отклонения от заданного поведения и, например, выявлены ошибки, передать их разработчику, что можно сделать месте с записанными результатами воспроизведения кейсов.

Обсуждение

Система обладает широкой применимостью для тестирования десктопных приложений и мобильных приложений через эмуляторы, что выгодно отличает её от инструментов вроде Selenium, ориентированных на веб-приложения. Она подходит для тестирования графических интерфейсов, где важно имитировать действия пользователя, такие как клики, движения мыши и ввод с клавиатуры. Благодаря использованию имитации действий, система менее чувствительна к изменениям внутренней структуры приложения, если визуальные элементы остаются на своих местах. Это особенно полезно для тестирования приложений с часто меняющейся внутренней логикой, но стабильным интерфейсом.

Система поддерживает сложные сценарии взаимодействия, включая последовательности кликов, перемещений мыши и ввода текста, что позволяет тестировать пользовательские сценарии, требующие многократных действий. Возможность сравнения скриншотов добавляет ценность, так как позволяет выявлять визуальные отклонения, что важно для проверки корректности отображения интерфейса. Кроме того, отсутствие необходимости написания кода для создания тест-кейсов делает систему доступной для тестировщиков без глубоких знаний программирования, что снижает порог входа и ускоряет процесс создания тестов.

Однако система имеет ряд ограничений. Она сильно зависит от визуальных элементов интерфейса, что делает её уязвимой к изменениям в расположении или внешнем виде элементов. Если интерфейс приложения меняется, тест-кейсы могут потребовать перезаписи. Это ограничивает её применимость для приложений с динамически изменяющимся интерфейсом, где элементы могут перемещаться случайным образом. Кроме того, система может быть менее гибкой для тестирования сложных сценариев, требующих условной логики или обработки данных, таких как динамическое изменение данных или взаимодействие с внешними системами.

Производительность и масштабируемость также могут стать проблемой. Имитация действий пользователя, например, движение мыши, может быть медленнее, чем прямое взаимодействие через API. Это может затруднить тестирование больших систем или выполнение множества тестов одновременно. Сравнение скриншотов, особенно при высоком разрешении экрана, может быть ресурсоёмким процессом. Кроме того, система ограничена в поддержке платформ: хотя она работает с десктопными приложениями и мобильными эмуляторами, её применимость к другим платформам, таким как встроенные системы или IoT-устройства, может быть ограничена.

Ещё одним ограничением является отсутствие интеграции с API приложения. Это означает, что система не может тестировать функциональность, которая недоступна через графический интерфейс, например, backend-логику или интеграцию с внешними системами. Это снижает её универсальность и требует комбинирования с другими инструментами для полного покрытия тестирования.

Заключение

Внедрение системы автоматизированного тестирования приложений повышает качество тестирования, исключая ошибки, возникающие при ручных методах, и обеспечивая полный анализ стабильности тестируемого ПО. Представленная система демонстрирует значительные преимущества по сравнению с существующими решениями, предлагая создание тест-кейсов без необходимости написания кода и встроенные функции записи и воспроизведения действий. Это позволяет ускорить процесс тестирования и повысить его эффективность. Дальнейшее развитие системы направлено на расширение функционала, управление наборами сценариев, а в перспективе на интеграцию с современными инструментами DevOps, что позволит ещё больше упростить и автоматизировать процесс тестирования.

Список использованных источников и литературы

1. Новиков М. Д. Автоматическое тестирование сложных программ на языке Паскаль // Национальная Ассоциация Ученых. 2023. – № 88-1. С. 56-59.

2. Мейер Б. Семь принципов тестирования программ // Открытые системы. СУБД. – 2008. – № 7. – С. 52-54.

3. Андриянов Ю. В., Матухин П. Г., Матяш Г. А. Программно-методическая разработка "Программа для проведения компьютерного тестирования и анализа его результата" // Хроники объединенного фонда электронных ресурсов Наука и образование. – 2013. – № 5 (48). – С. 13.

4. Латин Ю. Э. Автоматизация тестирования программного обеспечения средствами фреймворков // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и технические науки. – 2023. – № 4. – С. 78-81.

5. Ремезова А. П., Яковлев Е. И. Автоматизация тестирования программного обеспечения // Математическое и программное обеспечение систем в промышленной и социальной сферах. – 2022. – Т. 10. № 2. – С. 20-24.

6. Хасанов А. С., Шишкин В. В. Автоматизация интеграционного тестирования программного обеспечения с повышенными требованиями к критичности // Вестник Ульяновского государственного технического университета. – 2021. – № 2 (94). – С. 61-67.

7. Тюшнякова И. А., Пискунова Н. Автоматизация тестирования программного обеспечения: ключевые методологии и подходы // Вестник Таганрогского института имени А. П. Чехова. – 2024. – № 2. – С. 107-116.

8. Латыпов Б. Ф., Зиязетдинов Т. Р., Мухетдинов А. Р. Исследование ресурсоемкости применения автоматизированного тестирования при разработке программного обеспечения // Современная наука: актуальные проблемы теории и практики. Серия: Естественные и технические науки. 2022. № 10-2. С. 63-69.

9. Ярмонов А.С., Верещагина Е.А., Фролов А.В. Тестирование программ и математическая модель поиска ошибок в программном комплексе // Промышленные АСУ и контроллеры. – 2024. – № 2. – С. 42-44.

10. Клёнова Р.А., Валиуллин Д.Р. Разработка программы для проведения автоматизированного электронного тестирования // Современные наукоемкие технологии. – 2010. – № 3. – С. 41-42.

11. Рочев К. В. Классификация средств графического моделирования для разработки информационных систем // Информационные технологии в управлении и экономике. 2024. №1. Режим доступа: http://itue.ru/Issue/Article/275

12. Справочник по Windows API [Электронный ресурс]. — URL: https://vsokovikov.narod.ru/New_MSDN_API/globsdk.htm (дата обращения: 11.05.2024).

List of references

1. Novikov M. D. Automatic Testing of Complex Programs in Pascal. National Association of Scientists. 2023. No. 88-1. Pp. 56-59.

2. Meyer B. Seven Principles of Software Testing. Open Systems. DBMS. 2008. No. 7. Pp. 52-54.

3. Andriyanov Yu. V., Matukhin P. G., Matyash G. A. Software and Methodological Development "Program for Conducting Computer Testing and Analyzing Its Results." Chronicles of the United Fund of Electronic Resources Science and Education. 2013. No. 5 (48). P. 13.

4. Latin Yu. E. Automation of Software Testing Using Frameworks. Modern Science: Actual Problems of Theory and Practice. Series: Natural and Technical Sciences. 2023. No. 4. Pp. 78-81.

5. Remezova A. P., Yakovlev E. I. Automation of Software Testing. Mathematical and Software Support of Systems in Industrial and Social Spheres. 2022. Vol. 10. No. 2. Pp. 20-24.

6. Khasanov A. S., Shishkin V. V. Automation of Integration Testing for Software with Increased Criticality Requirements. Bulletin of Ulyanovsk State Technical University. 2021. No. 2 (94). Pp. 61-67.

7. Tyushnyakova I. A., Piskunova N. Automation of Software Testing: Key Methodologies and Approaches. Bulletin of the Taganrog Institute named after A. P. Chekhov. 2024. No. 2. Pp. 107-116.

8. Latypov B. F., Ziyazetdinov T. R., Mukhetdinov A. R. Research on Resource Intensity of Automated Testing in Software Development. Modern Science: Actual Problems of Theory and Practice. Series: Natural and Technical Sciences. 2022. No. 10-2. Pp. 63-69.

9. Yarmonov A. S., Vereshchagina E. A., Frolov A. V. Software Testing and a Mathematical Model for Finding Errors in a Software Complex. Industrial ACS and Controllers. 2024. No. 2. Pp. 42-44.

10. Klyonova R. A., Valiullin D. R. Development of a Program for Conducting Automated Electronic Testing. Modern High Technologies. 2010. No. 3. Pp. 41-42.

11. Rochev K. V. Classification of Graphical Modeling Tools for Developing Information Systems. Information Technologies in Management and Economics. 2024. No. 1. Available at: http://itue.ru/Issue/Article/275

12. Windows API Reference [Electronic resource]. URL: https://vsokovikov.narod.ru/New_MSDN_API/globsdk.htm (accessed: 11.05.2024).