**BC/NW 2018 № 2 (33):6.2**

**РАЗРАБОТКА ФРЕЙМВОРКА ДЛЯ АВТОМАТИЗАЦИИ ТЕСТИРОВАНИЯ ERP-СИСТЕМЫ ACUMATICA**

Гончаров Г.К.

Тестирование программного обеспечения представляет собой исследование, проводимое для определения заинтересованным сторонам информации о качестве продукта или сервиса при их испытании. Тестирование программного обеспечения также может дать объективную, независимую от программного обеспечения оценку, что позволит бизнес-сектору оценить и понять риски, связанные с реализацией тестируемого программного обеспечения. Методы тестирования включают в себя процесс выполнения программы с целью обнаружения ошибок в программном обеспечении.

Тестирование программного обеспечения включает в себя выполнение программного компонента или компонента системы для оценки одного или нескольких необходимых свойств, представляющих интерес [1]. В целом, эти свойства показывают уровень, где компонент или тестируемая система:

* отвечает требованиям, которыми руководствовались при их проектировании и разработке;
* правильно отвечают на все виды входных данных;
* выполняют свои функции в течение приемлемого времени;
* могут быть установлены и корректно работают в предназначенных для них средах;
* достигают общего результата, желаемого заинтересованными сторонами.

По мере того как число возможных тестов, даже для самых простых программных компонентов, практически бесконечно, специалисты по тестированию программного обеспечения используют некоторую стратегию, чтобы выбрать тесты, которые являются выполнимыми для имеющегося времени и ресурсов [2]. В результате составляются такие сценарии, которые, как правило, пытаются выполнить приложение с целью обнаружения ошибок в программном обеспечении. Задача тестирования представляет собой повторяющийся процесс, т.к. если одна ошибка исправлена, она может осветить другие, более сложные ошибки на более глубоком уровне, или создать новые.

Тестирование программного обеспечения может предоставить объективную, независимую информацию о качестве программного обеспечения и рисках воспроизведения ошибок пользователями [1]. Тестирование программного обеспечения может быть проведено только когда выполняется тестируемое программное обеспечение. Общий подход к разработке программного обеспечения часто определяет, когда и как проводится тестирование. Например, в поэтапном процессе большинство тестов проводится после того, как требования к системе определены и затем реализованы в тестируемых программ. Однако, иногда все процессы по разработке (выставление требований, программирование и тестирование) выполняются одновременно.

**Цели и задачи**

Облачные вычисления (также называемые «программное обеспечение как услуга» или SaaS) позволяют пользователям получать доступ к программным приложениям, которые работают на общих вычислительных ресурсах (например, процессорные элементы, память и дисковое хранилище) через интернет. Эти вычислительные ресурсы хранятся в удаленных центрах обработки данных, предназначенных для размещения различных приложений на нескольких платформах.

Облачная ERP – это программное обеспечение как услуга, которое позволяет пользователям получать доступ к программному обеспечению Enterprise Resource Planning (ERP) через интернет. Облачная ERP, как правило, имеет гораздо более низкие первоначальные затраты, поскольку вычислительные ресурсы сдаются в аренду на месяц, а не приобретаются напрямую и обслуживаются на месте. Облачная ERP также предоставляет компаниям доступ к своим критически важным для бизнеса приложениям в любое время из любого места.

Acumatica – международная IT компания, разработчик облачных CRM и ERP решений для среднего и малого бизнеса по всему миру. Acumatica представляет собой набор модулей, тесно связанных между собой: управление ресурсами и складские хранения, счет-фактуры и заказы, учет времени и отчеты, финансовые документы и покупки и другое.

Программное обеспечение для управления финансами предоставляет предприятиям полный набор функций бухгалтерского учета для отслеживания ежедневных финансовых операций и формирования квартальных и годовых финансовых отчетов. Кроме того, оно предоставляет инструменты для отчетности, анализа, составления бюджета и планирования.

Предназначенное для компаний со сложными требованиями, но простое в использовании в небольших организациях, программное обеспечение Acumatica Financial Management представляет собой многофункциональный бухгалтерский пакет, полностью интегрированный с наборами продуктов Acumatica для управления информацией о клиентах, распределения, управления производством, управления услугами на местах и учета проектов.

Acumatica Distribution Management - это программное обеспечение ERP, которое помогает компаниям управлять своей цепочкой поставок и логистической деятельностью, включая управление складом, управление запасами и заказами (продажи и заказы), а также интегрировать эти действия с финансовыми показателями и продажами компании. Программное обеспечение для оптовых продаж может помочь компаниям повысить степень удовлетворенности клиентов, сократить сроки заказов и контролировать расходы по всей цепочке поставок и распределения.

Модуль управления распределением включает в себя управление заказами на продажу, расширенный учет, управление заявками, управление заказами на покупку, расширенное выполнение и расширенные финансовые показатели. Он полностью интегрирован с CRM Acumatica, производством, обслуживанием на местах и проектным учетом для обеспечения наглядности во всей организации.

ERP-система Acumatica предлагает набор ресурсов для построения различного вида отчетов и аналитики, чтобы сделать работу проще и продуктивнее. Отчетность выходит за рамки только баланса и отчетов о движении денежных средств.

Поддержание качества такого продукта на высоком уровне является основной задачей инженеров, сопровождающих систему. В каждой программе имеется ряд дефектов, которые не проявляются до тех пор, пока пользователь не выполнит определенный набор действий, вызывающих сбой. Столкнувшись с такой проблемой разработчики исправляют неверное поведение системы, однако правки могут приводить к возникновению новых ошибок в совершенно неожиданных местах. Это усложняет работу инженеров отдела обеспечения качества. С развитием системы, увеличением числа модулей и усложнением интеграции между ними процесс поиска и исправления дефектов становится все тяжелее. В определенный момент человеческих ресурсов становится недостаточно для ручной проверки возможных сценариев. Тогда приходится вводить систему автоматизированного тестирования.

Целью данного проекта является разработка фреймворка для автоматизации процесса тестирования ERP-системы Acumatica. Требования, выдвигаемые к разрабатываемому фреймворку:

* возможность запуска тестовых сценариев в различных браузерах (Mozilla Firefox, Google Chrome);
* возможность запуска тестовых сценариев на мобильных устройствах под управлением ОС Android и IOS;
* бесплатное распространение;
* возможность адаптации под различные версии web-интерфейса;
* возможность запуска тестовых сценариев для оконных приложений;
* желательна поддержка языка C#.

**Анализ предложенных средств автоматизации**

Методы разработки программного обеспечения не стоят на месте, то же можно сказать и об используемых для разработки инструментах и технологиях. В первую очередь, такое совершенствование необходимо для того, чтобы продуктивность и качество создаваемого продукта увеличились, необходимое для разработки время сократилось, а заказчик остался доволен полученным результатом [3]. Таким образом, можно смело заявить, что тестирование играет важную роль в разработке качественного продукта.

Рассмотрим предложенные инструменты для автоматизированного тестирования с целью выбора наиболее подходящего и использования его в качестве основы при разработке фреймворка.

Sikuli – открытая кросс-платформенная визуальная среда создания сценариев-скриптов, которая ориентирована на программирование графического интерфейса при помощи изображений (скриншотов).

В качестве скриптового языка в Sikuli используется Jython, то есть в скрипте при желании можно использовать конструкции из языка Python [4]. В SikuliX появилась возможность использовать для написания скриптов язык Ruby в реализации jRuby. Sikuli доступна для работы в Windows, Mac OS X и Linux.

Sikuli появилась в 2008 г. как результат совместной работы доктора Rob Miller (профессора в департаменте EECS в MIT), студента Массачусетского технологического института (MIT) из Китая Sean Tsung-Hsiang Chang, и Tom Yeh – соискателя ученой степени в Университете Мэриленда (University of Maryland).

Sikuli переводится с древнего индейского языка Wixarica из Мексики как «глаз бога», что намекает на возможность видеть все на экране. Авторы языка считают, что некоторые задачи, например, по автоматизации тестирования пользовательского интерфейса, проще выполнять с помощью визуальных средств [4]. Sikuli использует алгоритмы распознавания текста и индексации изображений с помощью «визуальных слов». Встроенные функции языка принимают в качестве параметров графические данные (скриншоты).

Sikuli позволяет автоматизировать все, что видно на экране, без углубленного знания внутреннего API (что не мешает также писать сложные скрипты на Jython или Ruby).

Jython – это реализация языка Python на языке Java. Первоначальное имя проекта – JPython, пришлось поменять из-за конфликта с одноименным проектом.

Программы, выполняющиеся в среде Jython могут одновременно использовать классы языков Java и Python, используя, например, классы стандартной библиотеки Swing.

На языке Java можно разрабатывать апплеты для браузеров. Но в настоящий момент апплеты в Jython не включены. Компилятор был удален из пакета Jython 2.5.0 с целью внедрения новой технологии, которая лучше реализует создание апплетов.

Selenium (***Selenium WebDriver***)– это профессиональный инструмент для автоматизированного управления браузерами. Наиболее популярной областью применения Selenium является автоматизация тестирования web-приложений. Однако при помощи Selenium можно автоматизировать любые другие рутинные действия, выполняемые через браузер.

Разработка Selenium поддерживается производителями популярных браузеров, они адаптируют браузеры для более тесной интеграции с данным инструментом. Selenium является центральным компонентом целого ряда других инструментов и средств автоматизации. Присутствует поддержка компьютерных и мобильных браузеров.

Selenium позволяет разрабатывать сценарии автоматизации на многих языках программирования: Java, Javascript, Groovy, Python, C#, PHP, Ruby, Perl и GO. Можно организовывать распределенные стенды, состоящие из сотен машин с разными операционными системами и браузерами, и даже выполнять сценарии в облаках [5].

По назначению Selenium WebDriver представляет собой драйвер браузера, то есть программную библиотеку, которая позволяет разрабатывать программы, управляющие поведением браузера.

По своей сущности Selenium WebDriver представляет собой:

* спецификацию программного интерфейса (API) для управления браузером;
* реализации этого интерфейса для наиболее популярных браузеров (Internet Explorer, Google Chrome, Mozilla Firefox);
* набор клиентских библиотек для этого интерфейса на нескольких языках программирования.

Самое главное отличие WebDriver от всех остальных драйверов заключается в том, что это "стандартный" драйвер, а все остальные – "нестандартные". И это не простая фигура речи. Организация W3C действительно приняла WebDriver за основу при разработке стандарта интерфейса для управления браузером.

AutoIt – бесплатно распространяемый язык программирования, используемый при автоматизации задач для Microsoft Windows [6]. В изначальной версии AutoIt преимущественно использовался для разработки скриптов автоматизации (макросов) для программного обеспечения Microsoft Windows. Скрипты данного типа наиболее подходят для использования на часто запускаемых программах, например, установка одинакового набора программ на большое количество персональных компьютеров.

В свежих версиях AutoIt эволюционировал дизайн языка и неплохо выросла функциональность – данный шаг приблизил AutoIt к языкам общего назначения. После обновления до AutoIt 3 синтаксис языка стал реструктурирован и стал похож на язык семейства BASIC. Появилась возможность скомпилировать код в отдельный от интерпретатора файл.

AutoIt использует симуляцию нажатия клавиш, движений мыши и манипуляции с окнами, элементами управления. Это позволяет автоматизировать задачи пользователя Windows таким образом, какой невозможен или затруднен в других языках программирования [6]. Кроме того, этот язык компактен, самодостаточен и работает на всех версиях Windows «прямо из коробки», без внешних динамических библиотек и записей в реестре, что позволяет безопасно использовать его на серверах.

Интерпретатор AutoIt имеет маленький размер и не обязателен к установке. Для работы необходим основной файл, а для создания скрипта — любой текстовый редактор. В полной версии AutoIt находится интегрированная среда разработки, основанная на бесплатном редакторе SciTE. Компилятор, дополнительные программы и информационные материалы полностью совмещены, что позволяет сделать данный редактор стандартным окружением для программистов, пользующихся AutoIt. Компилятор AutoIt и среда разработки SciTE просто устанавливаются, вспомогательные настройки не требуются. Имеется полная документация по AutoIt и SciTE на русском языке с дополнительной установкой полной версии редактора.

Как и другие скриптовые языки, AutoIt – язык использующий классическую модель и переменные вариантного типа, разрешающий хранить типы данных различного рода, например, массивы. В AutoIt нет (или имеют ограниченную поддержку) пространства имен, объектно-ориентированное программирование, абстрактные типы данных, классы, расширенные графические функции. Также, помимо возможностей, встроенных в ядро, есть вариант использовать большую библиотеку готовых функций.

SilkTest является инструментом для автоматизированного и регрессионного тестирования корпоративных приложений. Он был первоначально разработан Segue Software, которая была приобретена компанией Borland в 2006 году. Borland была приобретена компанией Micro Focus International в 2009 году.

SilkTest предлагает различные клиенты:

* SilkTest Classic использует доменный 4Test специфический язык для написания сценариев по автоматизации. Этот объектно-ориентированный язык похож на C++. Он использует понятия классов, объектов и наследование;
* Silk4J позволяет писать тесты автоматизации в Eclipse, используя Java в качестве языка сценариев;
* Silk4Net позволяет то же самое в Visual Studio с помощью VB или C#;
* SilkTest Workbench позволяет писать тесты по автоматизации на визуальном уровне (по аналогии с бывшим TestPartner), а также с использованием VB.Net в качестве языка сценариев.

Особенности:

* SilkTest Хост: содержит все файлы сценариев источника, помимо дополнительных;
* SilkTest Агент: переводит команды сценария в GUI команд (действия пользователя). Эти команды могут быть выполнены на той же машине, как хозяин или на удаленной машине.

SilkTest определяет все окна и элементы управления тестируемого приложения как объекты, и определяет все свойства и атрибуты каждого окна. Исходя из этого, он поддерживает ООП реализацию (частично) [7].

SilkTest может быть запущен, чтобы определить движение мыши вместе с нажатием клавиш (полезно для пользовательского объекта). Он может использовать как запись и воспроизведение или описательные методы программирования для захвата диалоговых окон, что является достаточно важным моментом для разработчиков.

Расширения, поддерживаемые SilkTest: .NET, Java (Swing, SWT), DOM, IE, Firefox, SAP графического интерфейса пользователя Windows.

**Выбор программного средства**

Выбор технологии – вопрос достаточно сложный. Исходя из поставленных требований, рассмотрим и обоснуем решение касательно каждого средства.

**Sikuli**:

Плюсы данного варианта:

* простота: очень легко создать скрипт даже человеку, не понимающему в программировании;
* бесплатно распространяема;
* подробная документация.

Минусы:

* для мониторинга системы необходима разработка способов отправки данных и их хранения;
* нет привязок по id у объектов – также является минусом, потому что в этом случае при любом незначительном изменении изображения, либо шрифта, есть шанс того, что фреймворк не сможет определить объект.

Данное решение подходит скорее для частного использования и автоматизации рутинных каждодневных задач.

**AutoIt**:

AutoIt – бесплатное средство для тестирования, имеется развернутая документация, но, к сожалению, для разработки необходимо более глубокое понимание концепции данной программы. Данное ПО заслуживает внимания, но скорее для небольших проектов, использующих синтаксис Basic.

Плюсы:

1. бесплатное распространение;
2. может тестировать как оконные приложения, так и web-приложения.

Минусы:

1. синтаксис заявлен простым, но при разработке возникает множество сложностей;
2. сообщество хоть и существует, но его активность и поддержка небольшая;
3. придется изучать Basic, а помимо него привыкать к специфичному языку AutoIt.

**SilkTest**:

Хорошее программное обеспечение для тестирования, имеется и официальная поддержка от известной компании HP, также продаются готовые решения для различных проектов, исходя из условий заказчика, но такие решения достаточно дорогостоящие.

Плюсы:

1. поддержка;
2. имеется встроенное ПО для захвата видео тестов;
3. различные клиенты позволяют писать под различные языки программирования.

Минусы:

1. платный;
2. тип нахождения объектов – захват изображения, следовательно, при любом незначительном изменении изображения либо шрифта есть шанс того, что фреймворк не сможет определить объект.

**Selenium WebDriver**:

Несомненно, одним из лидеров по тестированию ПО в компаниях мирового уровня на различных платформах является Selenium. Имеется огромная база знаний и поддержка пользователей, также есть возможность разработки под различных платформы, что является очень удобным.

Вариант с Selenium имеет приоритет над остальными, так как встроенная система определения шаблонов позволяет определять ID объектов в браузере и создавать структуру объектов, привязанную к уникальным идентификаторам, тем самым точно определять различные свойства элементов, расположенных на странице (например, координаты объекта, его видимость, заблокирован ли элемент).

Плюсы данного варианта:

1. прост в освоении, также имеются сборки под различные языки программирования;
2. имеется IDE-плагин в Firefox в котором уже есть отслеживание логов;
3. имеется инструмент нахождения объектов по Id;
4. существует версия не только для Web, но и под операционные системы Android и IOS;
5. большое сообщество поддержки и пользователей;
6. бесплатное распространение.

Минусы:

1. для организации работающего стэка технологий необходимо изучить дополнительные инструменты;
2. нет возможности делать тесты для определенных типов приложений, только web-приложения.

Несмотря на то, что ни одно из средств полностью не удовлетворяет потребности, Selenium WebDriver является предпочтительным вариантом, т.к. позволяет использовать уникальные идентификаторы для поиска элементов на странице, поддерживает возможность разработки на языке C#, бесплатно распространяем и поддерживает мобильные операционные системы, для которых так же требуется проводить тестирование. Выберем его в качестве основы для разработки фреймворка.

**Разработка фреймворка**

 Разработка начинается с подключения динамической библиотеки WebDriver.dll в проект фреймворка. Данная библиотека содержит базовые методы, которые будут использоваться в качестве основы.

Разработка происходить в среде Visual Studio, подключение библиотеки осуществляется с помощью менеджера пакетов NuGet Package командой Install-Package Selenium.WebDriver [5]. После этого с помощью команды using можно подключать необходимые пространства имен и оборачивать базовые методы в желаемый код.

Одной из частей фреймворка является сбор элементов, содержащихся на каждом экране, и упаковка их в обертку, далее называемую враппер (от англ. wrap – упаковка). Все врапперы собираются в динамическую библиотеку, подключаемую в тестовых сценариях. Враппер представляет собой класс, описывающий один из экранов. Члены класса (поля, свойства, методы) могут быть вызваны из кода теста. В соответствии с соглашением о неразглашении коммерческой тайны, код, реализующий этот процесс, не может быть приведен в данной работе.

Для удобства использования в ядре фреймворка создаются классы, соответствующие элементам на экране: кнопки (Button.cs), выпадающие меню (Select.cs), поля для ввода текста (Input.cs), элементы с возможностью выбора (Checkbox.cs), таблица (Grid.cs), дерево элементов (TreeView.cs) и другие. Используя объекты этих классов можно получать доступ к элементам, расположенным на экранной форме, что существенно упрощает работу по сравнению с базовыми возможностями, предоставляемыми Selenium WebDriver. Например, рассмотрим процедуру авторизации на сайте.

В базовом варианте, предоставляемом Selenium WebDriver разработчику тестов каждый раз приходилось бы писать существенное количество кода:

FirefoxDriver firefox = new FirefoxDriver();

firefox.Navigate().GoToUrl(siteUrl);

WebDriverWait wait = new WebDriverWait(firefox, new TimeSpan.FromSeconds(15));

wait.Until(By.Id("txtUser"));

firefox.FindElement(By.Id("txtUser")).SendKeys(username);

firefox.FindElement(By.Id("txtPass")).SendKeys(password);

firefox.FindElement(By.Id("btnLogin")).Click();

Спрятав логику обработки в ядро, каждый раз описанный выше код будет вызываться, когда разработчик будет вызывать одну функцию:

PxLogin.LoginToDestinationSite();

Проект фреймворка содержит файл конфигурации в формате xml. В данном файле задается используемый браузер, адрес сайта назначения, логин и пароль для авторизации, название запускаемого тестового сценария и другие параметры. После запуска фреймворка проверяется наличие данного файла в сборке проекта и в случае успеха значения переписываются во внутренние переменные как это представлено на .



Рис. 1. Схема алгоритма работы фреймворка

Для каждого браузера в Selenium WebDriver реализован свой класс драйвера. Чтобы унифицировать код, используемый разработчиками, создадим метод, который будет создавать экземпляр класса для того браузера, который указан в конфигурационном файле.

public enum Browsers

{

Firefox, Chrome

}

public static Browsers SelectedBrowser

{

get

 {

 switch (Config.Config.BROWSER\_START\_COMMAND)

 {

 case "\*chrome": return Browsers.Chrome;

 default: return Browsers.Firefox;

 }

}

}

private Browser()

{

string expectedBrowserVersion;

 switch (SelectedBrowser)

 {

 case Browsers.Firefox:

 \_webDriver = StartFirefox();

 expectedBrowserVersion = Config.Config.FIREFOX\_VERSION;

 break;

 case Browsers.Chrome:

 \_webDriver = StartChrome();

 expectedBrowserVersion = Config.Config.CHROME\_VERSION;

 break;

 default:

 throw new CoreException("Unknown browser selected: " + SelectedBrowser);

 }

\_webDriver.Manage().Window.Maximize();

 CurrentWindowHandle = \_webDriver.CurrentWindowHandle;

}

После запуска фреймворка, если в конфигурационном файле верно указан тест, выбранный для исполнения, запускается механизм получения атрибутов и создания объекта класса.

testType = Support.GetTypes().FirstOrDefault(type => type.Name == test && type.IsSubclassOf(typeof(Check)));

if (testType == null) throw new ExecutionException("Can't find test: " + test);

var testDescription = testType.GetCustomAttribute<TestDescription>(false);

if (testDescription != null) Log.Information("[Description: {0}]", testDescription.Description);

var testBackups = testType.GetCustomAttributes<SiteAttribute>(false).Select(attribute => attribute.Backup?.Name ?? attribute.Snapshot).Where(backup => !string.IsNullOrEmpty(backup));

testInstance = (Check)Activator.CreateInstance(testType);

testInstance.BeforeExecute();

testInstance.Execute();

После вызова метода Activator.CreateInstance(testType) создается экземпляр тестового класса. В нем обязательно должен быть определен метод Execute(), в теле которого описываются автоматизируемые действия. В противном случае будет вызван метод Execute() родительского класса (т.к. все тесты являются наследниками класса Check), который вызовет исключение класса NotImplementedException сообщающее о том, что в исполняемом тесте нет реализации метода Execute(). Необязательным является метод BeforeExecute(), который может содержать в себе какие-либо предварительные действия, и AfterExecute(), который может содержать завершающие действия.

Исполнение метода testInstance.Execute() происходит в блоке try, поэтому непредвиденные ошибки, возникающие в ходе выполнения теста отлавливаются и передаются в блок catch, откуда происходит вывод этих ошибок в лог-файл.

catch (Exception e)

{

Launcher.ExceptionsInTestRun.Add(e);

 if (Support.WithQACenter && Browser.IsRunning) Log.Error(e, "#applicationstacktrace\n" + Core.Trace.Trace.ShowAcumaticaTrace());

 else Log.Error(e);

}

Если до момента выполнения метода testInstance.Execute() не происходит ошибок, то управление передается в тестовый класс, и запускается соответствующий тест. Пример алгоритма выполнения теста приведен на . В зависимости от типа, указанного в конфигурационном файле, запускается соответствующий браузер и создается экземпляр драйвера, с которым будет осуществляться работа.

Любое действие, в том числе открытие основного экрана, занимает некоторое время, поэтому в коде должны выставляться временные промежутки (называемые таймаутами), в течение которых происходит ожидание какого-либо результата. Самым простым решением в таком случае является остановка исполняемого потока на константное значение, например, 10 секунд. Однако у такого подхода есть существенные недостатки:

1. Действие может завершиться быстрее, тогда оставшееся время будет потрачено впустую. Для одной строчки кода это может быть приемлемым решением, но когда в проекте имеется несколько тысяч таких строк – время простоя может составлять несколько часов.
2. Действие может не успеть завершиться в отведенный промежуток времени по различным причинам. В результате получим ложное срабатывание и необходимость перезапускать тест, завершившийся с такой ошибкой.

Альтернативный вариант строится на подходе с использованием динамического ожидания, при котором устанавливается максимальное время, которое мы готовы ожидать результат, но если он наступит раньше таймаута – выполнение кода теста продолжится.

Wait.WaitForCondition(Username.IsVisible, Wait.LongTimeOut);

Данная строка задает ожидание появления на экране поля для ввода имени пользователя в течение промежутка времени, устанавливаемого в классе Wait в свойстве LongTimeOut.

public static T WaitForCondition<T>(Func<T> function, int timeOutInMilliseconds)

{

var endTime = DateTime.Now.Add(TimeSpan.FromMilliseconds(timeOutInMilliseconds));

while (DateTime.Now < endTime)

 {

 try

 {

 var result = function();

 if (typeof(T) == typeof(bool))

 {

 if (result as bool? == true) return result;

 }

 else if (typeof(T) == typeof(BooleanVerifiableValue))

 {

 if (result as BooleanVerifiableValue) return result;

 }

 else if (result != null) return result;

}

 catch (Exception e) when (e.GetType().Name == typeof(NoSuchElementException).Name)

 {

 lastException = e;

}

 catch (Exception e) when (e.GetType() != typeof(UnhandledAlertException))

{

 throw new WaitException($"An exception occurred during waiting for the {typeof(T).Name)} condition: {e.Message}", e);

 }

 Thread.Sleep(100);

}

 throw new WaitTimeoutException($"Timed out waiting for the {typeof(T).Name)} condition within the specified timeout: {timeOutInMilliseconds.ToString(Config.DefaultCulture)}ms", lastException);

В этом методе вычисляется конечный момент времени, до которого должен вернуться результат. С периодичностью в 100 мс происходит попытка выполнить переданный в качестве аргумента метод. В случае неудачного выполнения поток снова засыпает на 100 мс, после которых вновь будет произведена проверка. Если же результат выполнения метода вернется быстрее, то более нет необходимости тратить время на проверку, можно вернуть управление коду теста.

После открытия главного экрана сайта происходит ожидание появления элемента для ввода имени пользователя. По этому событию начинается ввод логина и пароля с последующей попыткой авторизации.

При успешной авторизации происходит переход на страницу «Заказы» и начинается процесс создания новых документов. После необходимо проверить корректность созданных документов. Для этого выбирается заказчик, для которого создавался документ и получаются сохраненные значения с помощью метода GetValue(), который возвращает текстовое значение, хранящееся в элементе. Метод VerifyEquals(string) осуществляет проверку того значения, которое вернул метод GetValue() с ожидаемым значением, передаваемым в качестве аргумента. В случае несовпадения ошибка выводится в лог-файл.

 

Рис. 2. Схема алгоритма одного из тестовых сценариев

Тестовый сценарий, соответствующий :

[Site(Name = "site", Backup = typeof(SalesDemo\_GInq700))] [Acumatica.Distribution.SalesOrders]

public class SalesOrders000 : Check

{

private readonly OrderSo \_orderSo = new OrderSo ();

public override void Execute()

{

PxLogin.LoginToDestinationSite();

\_orderSo.OpenScreen();

for (int i=0; i<10; i++)

{

 \_orderSo.Insert();

 \_orderSo.Summary.CustomerID.Select("C0000000"+i.ToString());

 \_orderSo.DocumentDetails.New();

 \_orderSo.DocumentDetails.Row.InventoryID.Select("AACOMPUT01");

 \_orderSo.DocumentDetails.Row.SubItem.Type(0);

 \_orderSo.DocumentDetails.Row.OrderQty.Type(3);

 \_orderSo.DocumentDetails.Row.CuryUnitPrice.Type(500);

 \_orderSo.DocumentDetails.New(); \_orderSo.DocumentDetails.Row.InventoryID.Select("CARRENT");

 \_orderSo.DocumentDetails.Row.OrderQty.Type(15);

 \_orderSo.DocumentDetails.Row.CuryUnitPrice.Type(300);

 \_orderSo.Summary.Hold.SetFalse();

 \_orderSo.Summary.OrderDate.Type(DateTime.Now.AddDays(-10));

 \_orderSo.Save();

 }

\_orderSo.RefreshScreen();

for (int i=0; i<10; i++)

{

 \_orderSo.Summary.CustomerID.Select("C0000000"+i.ToString());

 \_orderSo.DocumentDetails.SelectRow(1);

\_orderSo.DocumentDetails.Row.InventoryID.GetValue().VerifyEquals("AACOMPUT01");

\_orderSo.DocumentDetails.Row.OrderQty.GetValue().VerifyEquals(3); \_orderSo.DocumentDetails.Row.CuryUnitPrice.GetValue().VerifyEquals(500);

 \_orderSo.DocumentDetails.SelectRow(2); \_orderSo.DocumentDetails.Row.InventoryID.GetValue().VerifyEquals("CARRENT"); \_orderSo.DocumentDetails.Row.OrderQty.GetValue().VerifyEquals(15);

 \_orderSo.DocumentDetails.Row.CuryUnitPrice.GetValue().VerifyEquals(300);

 \_orderSo.Summary.Hold.GetValue().VerifyEquals(flase); \_orderSo.Summary.OrderDate.GetValue().VerifyEquals(DateTime.Now.AddDays(-10));

 }

}

}

Графический результат выполнения теста приведен на и .



Рис. 3. Результат создания документов для первого заказчика



Рис. 4. Результат создания документов для последнего заказчика

**Выводы**

Было проведено исследование четырех версий программного обеспечения для разработки автоматизированных тестовых сценариев с целью дальнейшего использования одной из них в качестве основы при построении фреймворка, который планируется использовать для мониторинга ERP-системы Acumatica. Был рассмотрен простейший пример задачи на создание заказов для нескольких покупателей и описан алгоритм работы фреймворка в случае ожидания возникновения события.

В результате разработан фреймворк на основе бесплатно распространяемого Selenium WebDriver, активно используемый для проведения автоматизированного тестирования ERP-системы Acumatica внутри компании и передаваемый партнерам вместе с выпускаемыми обновлениями программного обеспечения.

Ведется регулярная доработка и обновление компонентов в соответствии с новым функционалом, выпускаемым в основной программе

**Литература**

1. Рекс Блэк. «Ключевые процессы тестирования – планирование, подготовка, проведение, совершенствование» - Издательство Лори. 544 с. 2011
2. Элфрид Дастин, Джефф Рэшка, Джон Пол. Автоматизированное тестирование программного обеспечения. Внедрение, управление и эксплуатация. Издательство Лори. 592 с. 2003
3. Автоматизированное тестирование - [Электронный ресурс]. URL: http://automated-testing.info/
4. SikuliX by RaiMan - [Электронный ресурс]. URL: http://www.sikulix.com/
5. Документация Selenium – [Электронный ресурс]. URL: http://docs.seleniumhq.org/
6. Русское сообщество AutoIt – [Электронный ресурс]. URL: http://autoit-script.ru/
7. Tutorial SilkTest – [Электронный ресурс]. URL: https://documentation.microfocus.com/help/index.jsp?topic=%2Fcom.borland.silktest.workbench.doc%2FSILKTEST-AFA12AE2-SCRIPTWELCOME-CON.html