НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»

ННК "Інститут прикладного системного аналізу"
(повна назва інституту/факультету)

Кафедра Системного проектування
(повна назва кафедри)

«До захисту допущено»

Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)

"___"_____2015 р.

Дипломна робота

першого (бакалаврського) **рівня вищої освіти**
(першого (бакалаврського), другого (магістерського))

зі спеціальності 7.050102, 8.050102 Інформаційні технології проектування
7.050103, 8.050103 Системне проектування
(код та назва спеціальності)

на тему: «Розробка модуля розпізнавання елементів графічного інтерфейсу користувача»

Виконав: студент IV курсу, групи ДА-12
(шифр групи)

Галатенко Дмитро Володимирович _____
(прізвище, ім'я, по батькові) (підпис)

Керівник доцент, к.т.н. Корначевський Я.І. _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант з охорони праці доцент, к.б.н. Гусєв А. М. _____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище, ініціали) (підпис)

Рецензент д.т.н., проф. Бідюк П.І. _____
(посада, науковий ступінь, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Нормоконтроль ст. викладач Британов О.А.

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2015 року

Національний технічний університет України
«Київський політехнічний інститут»

Факультет (інститут) ННК "Інститут прикладного системного аналізу"
(повна назва)

Кафедра Системного проектування
(повна назва)

Рівень вищої освіти Перший(Бакалаврський)
(перший (бакалаврський), другий (магістерський) або спеціаліста)

Спеціальність 7.050102, 8.050102 Інформаційні технології проектування 7.050103, 8.050103 Системне проектування
(код і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ А.І.Петренко
(підпис) (ініціали, прізвище)
«___»_____2015 р.

ЗАВДАННЯ
на дипломний проект (роботу) студенту
Галатенку Дмитру Володимировичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи)) Розробка модуля розпізнавання елементів графічного інтерфейсу користувача
керівник проекту (роботи)) Корначевський Ярослав Ілліч, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом по університету від «02» квітня 2015 р. №30/1-ст

2. Строк подання студентом проекту (роботи) 08.06.2015

3. Вихідні дані до проекту (роботи)

Програмний продукт Mathematica, який буде тестуватись
Тест-план програмного продукту
База зображень елементів графічного інтерфейсу користувача
API розпізнавання графічних елементів Sikuli
Форма реалізації – у вигляді модуля на мові Java 7

4. Зміст розрахунково-пояснювальної записки (перелік завдань, які потрібно розробити)

1. Розглянути можливі варіанти засобів для розпізнавання графічних елементів та обрати варіант для розробки.

2. Розробити алгоритм роботи модуля розпізнавання елементів графічного інтерфейсу.
3. Розробити програмний модуль розпізнавання на мові Java.
4. Дати рекомендації по практичній розробці модуля розпізнавання.
5. Розглянути результат роботи модуля розпізнавання.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів тощо)

1. Тест-план системи Mathematica – плакат.
2. Діаграма класів модуля розпізнавання – плакат.
3. Результат роботи модуля розпізнавання ч.1 – плакат.
4. Результат роботи модуля розпізнавання ч.2 – плакат.

6. Консультанти розділів проекту (роботи)*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	Гусєв А.М., доцент, к.б.н.		
Основна частина			

7. Дата видачі завдання 01.02.2015

Календарний план

№ з/п	Назва етапів виконання дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Отримання завдання	01.02.2015	
2	Збір інформації	15.02.2015	
3	Вивчення засобів розпізнавання та вибір варіанту для розробки	28.02.2015	
4	Розробка алгоритму та структури модуля	10.03.2015	
5	Розробка плану тестування	15.03.2015	
6	Побудова бази зображень	25.03.2015	
7	Розробка модуля розпізнавання	25.04.2015	
8	Тестування програмного модуля	30.04.2015	
9	Оформлення дипломної роботи	31.05.2015	
10	Отримання допуску до захисту та подача роботи в ДЕК	19.06.2015	

Студент _____ Д.В.Галатенко
(підпис) (ініціали, прізвище)

Керівник проекту (роботи) _____ Я.І.Корначевський
(підпис) (ініціали, прізвище)

* Консультантом не може бути зазначено керівника дипломного проекту (роботи).

АНОТАЦІЯ

бакалаврської дипломної роботи Галатенка Дмитра Володимировича
на тему «Розробка модуля розпізнавання елементів графічного
інтерфейсу користувача»

Дана дипломна робота присвячена розробці програмного забезпечення для проведення автоматизованого тестування графічного інтерфейсу користувача. Метою роботи є розробка програмного модуля, базованого на технології розпізнавання графічних елементів.

В роботі розглянуто засоби розпізнавання елементів графічного інтерфейсу, проведений їх порівняльний аналіз, та визначено найкращий засіб для реалізації програмного модуля. Розроблений загальний алгоритм роботи модуля розпізнавання, створений детальний тестовий план для демонстрації роботи модуля, виділені основні цілі проведення тестування, наведені особливості процесу реалізації модуля розпізнавання, а також надані практичні рекомендації по розробці програмного забезпеченні орієнтованого на розпізнавання елементів графічного інтерфейсу.

Загальний обсяг роботи: 73 сторінки, 19 рисунків, 4 таблиці, 15 бібліографічних найменувань.

Ключові слова: комп’ютерний зір, графічний інтерфейс користувача, автоматизація тестування.

АННОТАЦИЯ

бакалаврской дипломной работы Галатенко Дмитрия Владимировича
на тему «Разработка модуля распознавания элементов пользовательского графического интерфейса»

Данная дипломная работа посвящена разработке программного обеспечения для проведения автоматизированного тестирования графического интерфейса пользователя. Целью работы является разработка программного модуля, основанного на технологии распознавания графических элементов.

В работе рассмотрены средства распознавания элементов графического интерфейса, произведен их сравнительный анализ, и определено лучшее средство для реализации программного модуля. Разработан общий алгоритм работы модуля распознавания, создан подробный тестовый план для демонстрации работы модуля, выделены основные цели проведения тестирования, приведены особенности процесса реализации модуля распознавания, а также даны практические рекомендации по разработке программного обеспечении ориентированного на распознавание элементов графического интерфейса.

Общий объем работы: 73 страницы, 19 рисунков, 4 таблицы, 15 библиографических наименований.

Ключевые слова: компьютерное зрение, пользовательский графический интерфейс, автоматизация тестирования.

ANNOTATION

a bachelor’s degree work of Dmytro Halatenko
entitled "Monitoring the quality of education in the cloud learning environment"

This course is devoted to the research of recognition GUI elements. The aim is development software module, based on image elements recognition technology.

The course consider the means of identification GUI spent their comparative analysis, and determined the best way to implement software module. The general algorithm of recognition module and detailed test plan to demonstrate the module was created in this work.

Also developed recognition module, highlighted it`s the main aspects of the design. The basic objectives of testing are features recognition module implementation process and provided practical advice on the development of software-based detection GUI.

Total volume of work: 73 pages, 19 figures, 4 tables, 15 bibliographic titles.

Keywords: computer vision, graphical user interface, test automation.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

ПЗ	програмне забезпечення
GUI (Graph. User Interface)	графічний інтерфейс користувача
IT (Information Technologies)	інформаційні технології
НС	надзвичайна ситуація
Скріншот	(англ. Screen – екран, shot - знімок) зображення, отримане комп'ютером, що відображає те, що бачить користувач на екрані монітора.
WPF	Windows Presentation Foundation
DDT (Data Driven Testing)	Тестування орієнтоване на дані
VCS (Version Control System)	Система контролю версій
JRE	Java Runtime Environment
VNC	Virtual Network Computing
GPL	General Public License
FFT (Fast Fourier Transform)	Швидке перетворення Фур’є
SAD (Sum of the Abs. Differences)	Сума абсолютних різниць

ВСТУП

Оцінка сучасного стану проблем

Відколи ПЗ консольного вигляду застаріло, почалася ера програм орієнтованих на користувача. Такий перехід відбувся завдяки появі GUI, який і сьогодні використовується в усіх десктопних програмних розробках. Саме GUI є найзручнішим засобом взаємодії програми з користувачем, серед усіх відомих на сьогоднішній день, тому при розробці ПЗ програмісти широко використовують елементи GUI.

Як відомо, невід'ємною частиною процесу розборки є тестування програмного забезпечення. Як замовник так і розробник, повинні бути впевнені, що програмний продукт відповідає вимогам висуненим в технічному завданні, а найбільш доцільним способом переконатись у цьому є проведення ефективного тестування. Така впевненість в працездатності програмного продукту коштує доволі дорого. Команда витрачає багато часу і ресурсів на якісне тестування, якого вимагає, без винятку, кожен програмний продукт. Тому дуже гостро стоїть питання економії зусиль витрачених на реалізацію продукту. Це питання вирішується ефективно за допомогою використання засобів автоматизації тестування. Такі засоби полегшують роботу тестувальників, і значно прискорюють сам процес тестування за рахунок того, що одна й та ж послідовність дій виконується не вручну людиною, а автоматично за допомогою машини.

Оскільки GUI став невід'ємною частиною сучасного ПЗ, то відповідно він теж потребує розробки методів ефективного автоматичного тестування. Одним із найбільш ефективних методів автоматизації тестування є метод оснований на візуальному розпізнаванні елементів управління. Цей метод використовує сучасні технології комп'ютерного зору для того що б розпізнати елементи графічного інтерфейсу на екрані та про взаємодіяти з ними, таким же способом як це робить людина під час мануального тестування ПЗ.

Варто відзначити найголовніші переваги тестування GUI базованого на розпізнаванні елементів графічного інтерфейсу в порівняні з стандартними методами:

- спрощений процес тестування;
- незалежність від платформи тестування;
- значно менші часові затрати на тестування.

Ці переваги приводять до популяризації технології тестування базованої на візуальному розпізнавання та сприяють подальшому її розвитку.

Актуальність

Розпізнавання елементів графічного інтерфейсу широко використовується в автоматизації тестування програм які мають GUI. При такому підході до тестування GUI актуальною проблемою є локалізація елементів управління програмою на екрані. Це є найбільш ресурсоємкою процедурою при тестуванні ПЗ. Тому увага в роботі приділяється саме питанню розпізнавання елементів графічного інтерфейсу користувача.

Такий підхід до тестування є новим і перспективним напрямком в розробці програмного забезпечення по ряду причин висвітлених нижче.

При автоматизації програмної логіки програмісти-тестувальники пишуть відповідні автоматичні тест-кейси тільки після детального аналізу коду. Приблизно так само виконується тестування GUI, але це доволі складний процес, адже програмісту-тестувальнику необхідно розібратися із внутрішньою структурою програми, перш ніж взятися за написання тестів для GUI. Такий метод тестування називається тестування білої скриньки (white-box testing) і не оправдано затрачує багато ресурсів. Тому доцільно було оптимізувати процес тестування GUI таким чином, щоб тестування GUI відбувалось по методології «чорної скриньки» (black-box testing). Black-box тестування (функціональне тестування) - це процес за якого тестувальник не має жодної інформації про внутрішню реалізацію функцій системи. Лише на основі вхідного набору параметрів та очікуваного набору вихідних значень можна зробити висновок

про працездатність системи. Тож на основі тестування базованого на розпізнавання графічних елементів можна зробити висновки про такі параметри системи:

- функціональність;
- прийом вхідних даних;
- отримання результатів;
- збереження цілісності програми.

Мета

Кінцевою метою даної роботи є розробка модуля розпізнавання елементів GUI, який буде базуватися на технології розпізнавання графічних елементів. Відповідно до цього на шляху досягнення мети будуть вирішені такі задачі:

- аналіз існуючих систем розпізнавання графічних елементів і вибір відповідних до поставленої мети;
- дослідження методів пошуку графічних елементів;
- створення програмного модулю реалізації автоматичного тестування графічного інтерфейсу користувача;
- оцінка ефективності такого підходу до автоматизації тестування у вирішеній задачі автоматизації тестування.

Методи досліджень

Для вирішення вище поставлених задач використовуються методи розпізнавання елементів графічного інтерфейсу основані на алгоритмі шаблонного співпадіння (template matching).

Результат виконаної роботи буде перевірений за допомогою проведення відповідних тестів на ЕОМ.

1. ЗАСОБИ РОЗПІЗНАВАННЯ ЗОБРАЖЕНЬ

1.1 Пошук графічних елементів

Тема розпізнавання зображень вивчалась доволі детально протягом останніх 5 років і світ отримав чимало напрацювань пов’язаних з цією галуззю. Розроблялись додатки, бібліотеки і фреймворки автоматизації тестування базовані на технології пошуку графічних елементів. Спочатку вони були недосконалі і надавали мінімальний набір функцій, тому для повноцінної роботи, а тим паче для процесу корпоративної розробки в компанія не були придатними. Але ці засоби постійно удосконалювались і покращувались в своєму функціоналі, адже сама ідея такого підходу до тестування є дуже привабливою.

Пошук графічних елементів базується на обробці інформації зчитаної із поточного зображення на екрані. Як наслідок він можливий лише на тих пристроях які допускають екранний вивід інформації. Під поняття обробки підпадає саме пошук відповідностей до заданого шаблону, тобто зображення. Подібно до того як людина знаходить необхідну їй інформацію на екрані комп’ютера, машина знаходить задане їй зображення на екрані.

Як технологія, пошук елементів представляє собою вирішення задачі локалізації графічного елемента на екрані пристрою. Працює це в дуже простий спосіб. Заздалегідь слід підготувати необхідне зображення для пошуку на екрані, на приклад зображення елемента управління. В певний момент роботи програми, коли запускається пошуковий процес, автоматично робиться скріншот екрану. Припускається, що в даний момент працює програма що проходить тест і на екрані знаходиться елемент управління, який необхідно знайти. Якщо даний елемент управління виходить за межі екрану чи вікна, або перекритий іншим вікном, пошук не дасть бажаного результату. Далі пошуковий процес зіставляє зразок зображення для пошуку із отриманим

скріншотом екрану і шукає такі координати на скріншоті, які максимально відповідають пошуковому зразку.

Пошуковий процес являє собою це дуже складні математичні розрахунки, для виконання яких працює ціла пошукова система. Конкретна реалізація такої пошукової системи буде детально розглянута в розділі 2.

Таким чином система отримує дані про положення відповідного елемента управління на екрані в даний момент часу і може відреагувати на них. На приклад, шляхом позиціонування мишки в цю точку. Так відбувається взаємодія системи з ПЗ, яке проходить тестування.

Отже пошук графічних елементів імітує процес взаємодії людини з комп'ютером, за допомогою вище описаного алгоритму. Така технологія є досить привабливою за рахунок простоти сприйняття цього підходу людиною. Не треба витрачати багато часу на те щоб розібратись з технологією пошуку. Прочитавши коротку довідкову інформацію, можна починати працювати безпосередньо з пошуком графічних елементів, не маючи відповідної спеціальної підготовки.

1.2 Ціль пошуку графічних елементів

Засоби автоматизації тестування базовані на технології пошуку графічних елементів - це новий, абсолютно інакший підхід до тестування в області GUI. І тут виникає питання: який зв'язок між тестуванням і пошуком графічних елементів? Тож роз'яснимо ситуацію. Під час проведення manual testing [1] (процес тестування ПЗ за допомогою людських ресурсів, тобто тестувальник виконує роль кінцевого споживача) тестувальнику необхідно:

- здійснити певну взаємодію з програмою (відповідно до тест-плану);
- проаналізувати отриману реакцію на цю взаємодію;
- здійснити оцінку отриманих даних.

Розкриємо кожен з цих пунктів детальніше.

Під першим пунктом мається на увазі, що необхідно задати початкові умови для роботи програми, тобто провести взаємодію з графічним

інтерфейсом. Це може бути як введення текстової інформації так і натискання кнопок, чекбоксів, випадаючих списків і т.д. Очевидно що для такої взаємодії спершу треба знати який вигляд має певний елемент і потім відшукати його на екрані.

Другий пункт порушує питання відповідності отриманої реакції на взаємодію до очікуваної реакції даного ПЗ. На даному етапі потрібно відслідковувати чітку послідовність подій, що вказані у тест-плані. Кожне вікно, або ж його елемент управління повинні відповідати очікуваним вимогам: існувати та мати відповідний вигляд. Також на даному етапі варто відслідковувати можливі помилки в роботі програми. Можуть з'являтися попереджувальні вікна, що містять звіт про помилку, чи аварійне завершення програми яке відслідковується на рівні ОС. Такі вікна також обов'язково мають відстежуватись, звісно, якщо можлива їх поява. Приведені дії реалізуються методом отримання очікуваного вікна чи елемента управління на екрані, тобто зводяться до пошуку графічних елементів, як і в попередньому твердженні.

В останньому пункті під оцінкою отриманих даних мається на увазі висновок про успішне або провальне проходження тесту. Таке рішення відбувається на основі даних отриманих з пункту 2.

Таким чином були наведені аргументи для того, щоб дати відповідь на питання: в чому ж полягає зв'язок між тестуванням і пошуком графічних елементів. За рахунок можливості пошуку «бачити» екран комп'ютера, всі наведені пункти, що виконуються тестувальником, можуть бути виконані за допомогою комп'ютерних засобів шляхом візуалізації пошуку. Таким чином з'являється новий підхід до автоматизації тестування GUI під назвою «автоматизація тестування GUI базована на пошуку графічних елементів».

Ідея цього процесу ще раз підкреслює можливість заміни людської праці машинною, навіть у тому випадку, де, здавалось би, без людини обійтись неможливо. Машина повністю бере на себе рутинну роботу: виконує вказані дії, слідкує за правильність проходження тесту, видає детальний звіт про всі виконані дії а також пройдені і не пройдені тести.

Отже основними цілями пошуку графічних елементів є створення нового методу тестування і спрощення процесу автоматизації тестування GUI. Така технологія є досить привабливою за рахунок простоти сприйняття цього підходу людиною. Не треба витрачати багато часу на те, щоб розібратись з технологією пошуку. Це приводить менеджерів проектів до значної економії часу а отже і економії коштів. А також варто сказати про найсильнішу можливість даної технології - максимізацію тестового покриття для даного екземпляру ПЗ, при тих же затратах, які були б використані в інших методах тестування.

1.3 Огляд існуючих засобів базованих на пошуку графічних елементів

Серед найпотужніших засобів автоматизації тестування на основі пошуку графічних елементів слід виділити Ranorex, SikuliX, T-Plan та EggPlant. Безперечно, всі ці системи є достатньо функціональними та складними, але кожна з них має певні особливості в роботі, тому далі в цьому розділі буде наведений детальний опис їх можливостей, і на основі порівняння буде обрано один з них для подальшого виконання роботи.

1.3.1 Ranorex

Перший потужний засіб автоматизації, який розглядається це Ranorex. Цей додаток все став досить розповсюдженим завдяки надійності роботи на Windows системах.

За допомогою Ranorex можна тестувати такі додатки: Windows (WinForms, WPF, Win32), Java, Qt, Delphi, Flex, а також звичайно HTML (браузери — IE, Firefox, Chrome, Safari). У останній версії з'явилася також підтримка тестування мобільних клієнтів на Android і iOS.

Ranorex підтримує запис дій за допомогою вбудованого рекордера, або ідентифікацію елементів графічного інтерфейсу за допомогою компонента RanorexSpy. Усі ідентифіковані елементи зберігаються в XML репозиторії, де

кожен елемент записаний за допомогою XPath нотації [2]. На основі запису, здійсненого рекордером, відбувається авто-генерація коду у C# або VB.NET. Будь-який крок також може бути написаний, або виправлений вручну.

Також даний продукт має власну IDE: Ranorex Studio. В результаті організація коду і тест-кейсів максимально нагадує MS Visual Studio: усі тест-кейси розкладені по проектах; проекти, які використовуються разом, можна об'єднувати в рішення. На виході після компіляції ми отримуємо по одному .exe файлу на рішення і по одному .dll на проект. Досить скопіювати їх на тестову машину і запустити як звичайний Windows додаток подвійним кліком миші.

Організація тест-кейсів відбувається таким чином: створюється запис або автоматично, або написаний на одній з підтримуваних мов програмування. Тут же в ручних модулях ми можемо використати будь-який необхідний нам код на C#, VB.NET та Python, оскільки ці мови використовуються для написання автоматичних тестів в Ranorex. Допускається процес DDT завдяки можливості пов’язати будь-який тест-кейс з даними з CSV, або Excel файлу чи з бази даних.

Ranorex зберігає увесь код, в тому числі створений авто-генерацією, і усі репозиторії в текстовому виді (наприклад для репозиторіїв використовується формат XML) це дає можливість для версіонності і спільної розробки використовуючи VCS, як на приклад Git, з усіма його можливостями. Ціна однієї ліцензії з прив'язкою до робочого місця коштує 1 480 € за купівлю і додатково ще 290 € за кожен рік користування (починаючи з другого) за подальші оновлення і підтримку.

1.3.2 SikuliX

Як і всі програми базовані на засобах пошуку графічних елементів, Sikuli аналізує екран комп’ютера для розпізнавання елементів управління GUI. Проект Sikuli є розробкою працівника та студентів MIT (Massachusetts Institute of Technology), має відкритий програмний код, а отже є безкоштовним програмним продуктом.

Написаний Sikuli на мові програмування Java. Для того щоб почати роботу варто лише завантажити установочний файл і почекати 10 секунд поки програма встановиться на комп'ютер. Даний програмний продукт не потребує окремої машини для можливості проведення тестування ,тобто може тестувати програму на тій же машині, де він запущений. Єдині вимоги до машини - це наявність екрану і встановлена JRE 5 версії і новіше.

Sikuli має досить добре розроблену API з широким набором функціональних можливостей. Всі ці функції значно покращують якість і швидкість тестування, і їх наявність робить Sikuli інструментом тестування досить високого рівня. Інструмент надає можливість роботи з API, що підтримують Java, Python, Jython та JRuby. Окрім набору для роботи з мишею і клавіатурою, який надає максимальні можливості для автомати заторів, Sikuli має певні засоби які дозволяють прискорити пошук. Найголовнішим із них є використання об'єктів Region, які задають регіон пошуку зображення замість того щоб шукати його на всьому екрані. Детально всі методи, які використовуються в роботі будуть розглянуті в розділі 2.

Обов'язковим для засобу автоматизації даного рівня є наявність модуля розпізнавання тексту, і Sikuli не є виключенням. У першій версії продукту цей модель був слабким, він не забезпечував необхідної надійності розпізнавання тексту. Але з виходом наступної версії 1.1 ця помилка була виправлена в модулі OCR. Звітність Sikuli не є такою, яку можна було назвати чудовою, але переважно є задовільною. Але можливості API дозволяють переписати цей логер і налаштувати його під себе для більш детального звіту по виконанню роботи. Також за рахунок API з'являється можливість використовувати такі популярні засоби як TestNG або JUnit а також технологію DDT в поєднанні з можливостями Sikuli, це створює достатньо потужні можливості для тестування програмного інтерфейсу користувача.

Крім усього переліченого, Sikuli має власну IDE з мінімальним набором функціоналу для тестування. На приклад, при пошуку зображення аргументом функції find() є безпосередньо скріншот шуканого елемента. Причому скріншот

не треба робити заздалегідь, в IDE існує вбудований засіб для створення скріншотів «на льоту», тобто під час написання скрипту. Це дійсно дуже зручна штука, особливо для людей взагалі не знайомих з технологіями програмування. Створений скрипт можна зберегти у вигляді вихідного коду (скріншоти, сценарій, і візуальний варіант в HTML), або ж експортувати в виконуваний .skl скрипт, який можна запускати із консолі робочої машини. Приклад написання Sikuli-скрипту в її власній IDE ілюструє рисунок 1.

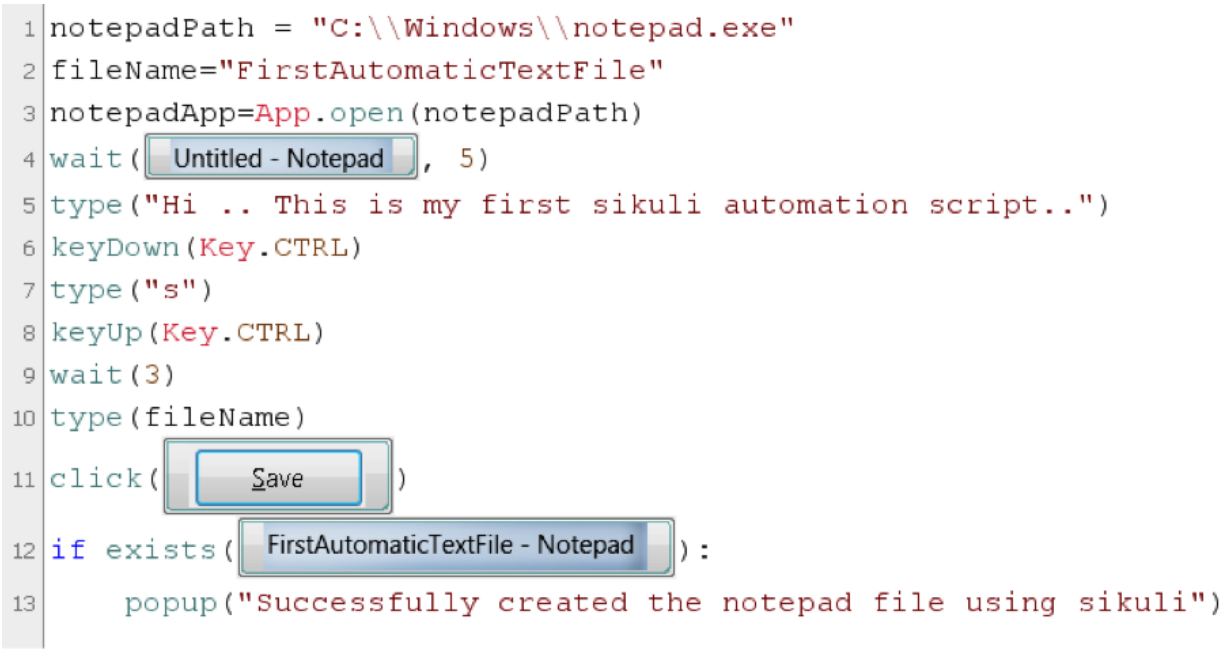


Рис. 1 Вигляд скрипту в Sikuli IDE.

Тож Sikuli є універсальним, потужним і безкоштовним засобом автоматизації тестування з відкритим вихідним кодом. Головною перевагою цього засобі є наявність досить гнучкого і функціонального API. А певні незручності що присутні в системі не завдають відчутного дискомфорту при автоматизації тестування.

1.3.3 T-Plan Robot

T-Plan Robot, також відомий як VNCRobot, базований на VNC [3] (використовує RFB протокол). Це надає незалежність від платформи на якій він запущений, єдина вимога до платформи це підтримка VNC server, який

підтримується всіма найпопулярнішими операційними системами. Також є можливість використовувати його для тестування мобільних пристроїв завдяки підтримці PocketVNC. T-Plan поєднує в собі два основні підходи: базований на зображенні і базований на абсолютних координатах екрану. Це дає повну впевненість в надійності роботи програми незалежно від зміни графічного інтерфейсу [4].

Функціонал T-Plan не надає специфічних можливостей по автоматизації тестування; він включає в себе лише стандартний набір процедур: пересування миші, клік, натискання на клавіші, текстове введенні інформації, знаходження зображення та очікування заданого зображення.

Встановлення T-Plan не є складною задачею. Оскільки система розроблена на мові Java, то все, що не обхідно це завантажити і розпакувати архів. Але розгортання системи займе певний час і потребує спеціальних вмінь. Спершу необхідно встановити VNC server і провести для нього всі необхідні налаштування. Далі підключити його до сервера за допомогою VNCconnection.

Система звітності в T-Plan дозволяє створювати прості HTML сторінки які містять опис виконуваних операцій, підкріплених скріншотом, який зберігається під час виконання тесту. Щодо DDT , його використання є неможливим через відсутність засобу для зчитування зовнішніх даних, файлів і т.д. T-Plan Robot надається у вільному доступі для завантаження, надається згідно з GPL [5]. Але існує також платна Enterprise версія яка надає широкий спектр підтримки цього програмного забезпечення.

Отже T-Plan Robot працює виключно на віддаленому сервері, тож потребує додаткову машину для встановлення. Його функціональні властивості надають необхідний мінімум для тестування і в той же час не є досить різноманітними та специфічними, що не є його сильною стороною. Проте, неодмінною перевагою не наявність системи у вільному доступі безкоштовно. А для професіоналів, які потребують постійної підтримки та оновлень існує можливість придбати Enterprise версію.

1.3.4 EggPlant

EggPlant це потужний корпоративний інструмент тестування, відповідно він має безліч можливостей для зручної маніпуляції даними. Як і у T-Plan, його робота основана на VNC з'єднанні з машиною на якій запускається тест.

EggPlant має достатньо сильний модуль розпізнавання тексту, що є невід'ємною частиною роботи програми автоматичного тестування. На приклад, коли елемент управління змінює свою форму або візуальний вигляд, то він (з високою ймовірністю) має той же надпис що і раніше (при наявності такого надпису), а отже цей елемент буде вдало розпізнаний системою. Також коли ми працюємо з технологією пошуку графічних елементів розпізнавання текстів дає можливість перевірити результати виконання програми.

EggPlant має високу стабільність роботи. Оскільки всі сценарії запускаються на віртуальній машині, вплив зовнішніх факторів мінімальний, а отже маємо високу надійність такої системи. Також вона має чудову систему звітування. Кожен запуск заноситься в лог, а кожен тест, незалежно від успішності його виконання, супроводжується скріншотом. Відповідно це дуже спрощує подальшу обробку результатів тестування. Окремо слід також виділити можливість групування сценаріїв у так звані колекції. Так однин і той же елемент управління виглядає по різному в залежності від того, виділений він чи неактивний. Саме для такого випадку слугує можливість групування зображень.

Даний програмний продукт наявний для всіх сучасних операційних систем. На даний момент з'явилась API, що підтримує такі мови програмування як Java, Ruby, Python. Також наявна можливість для розробки за методологією DDT, що дає величезну перевагу у порівнянні з T-Plan.

Оскільки цей програмний продукт є досить високоякісним і націлений переважно на використання в корпоративній розробці, він є платним, причому його ціна сягає 9000 € на рік.

Отже маємо прекрасний професійний засіб тестування, з широким спектром можливостей, але в той же час наявні досить негативні моменти. По-перше це складність у процесі розвертання, оскільки він працює тільки з віддаленою віртуальною машиною (використовує технологію VNC) і не дозволяє тестувати додатки на тій же машині на якій він працює. А по-друге висока ціна, що є більш недоцільним ніж негативним для виконання даної роботи.

1.4 Вибір оптимального засобу розпізнавання зображень

Кожен з наведених засобів тестування володіє потужною системою пошуку зображень, також кожен має свої позитивні і негативні сторони. Тож для того щоб визначити який додаток є найбільш підходящим для виконання даної роботи, необхідно ввести певні метрики оцінки якості тестового програмного продукту.

Перш за все варто звертати увагу на наявність API, адже робота передбачає виконання практичної частини: написання власної програми яка автоматизує тестування за допомогою саме засобів пошуку графічних елементів. Серед певної кількості API перевага надається мові програмування Java. Оскільки технологія тестування базована на пошуку графічних елементів є платформо незалежною, дуже доцільно не звужувати цей широкий спектр можливостей, тому логічно вибрати мову програмування яка також є платформо незалежною. Саме такою мовою і є Java. Серед розглянутих систем тестування сюди підпадають Sikuli та EggPlant.

Далі мова піде про VNC - програмне забезпечення яке надає можливість доступу у мережі TCP/IP до віддаленого комп'ютера з будь-якого іншого комп’ютера або мобільного пристрою з ціллю відслідковування (моніторингу) та дистанційного керування (*remote control*) [4]. Для здійснення такої взаємодії потрібно встановити (інсталювати) програмне забезпечення VNC, яке відображає у вікні вашого комп’ютера весь екран віддаленого комп’ютера та передає йому коди натиснутих клавіш та команди мишки, таким чином

надаючи користувачу повний «ефект присутності». Наявність VNC в системі звісно говорить про її серйозність та професійну орієнтованість, але такі системи потребують додаткового обладнання, а також значних зусиль для розгортання самої системи. Таким чином в даній роботі наявність VNC не вважається перевагою системи, хоча в процесі корпоративної розробки ситуація протилежна. З цим параметром лідирують Ranorex та Sikuli.

Важливою складовою оцінки засобів тестування є ціна програмного забезпечення, тобто наявність безкоштовної версії продукту у вільному доступі. Серед наведеної інформації спостерігаємо наступні цінові показники: Ranorex - 1 480 € за купівлю і додатково ще 290 € за кожен рік користування (починаючи з другого) за подальші оновлення і підтримку; EggPlant - 9000 € за рік користування. Sikuli та T-Plan в даному випадку можуть бути завантажені абсолютно безкоштовно і надаються згідно з GPL.

Модуль розпізнавання тексту є невід'ємною частиною професійно засобу тестування GUI. Якщо виникає ситуація, за якої, з різних причин, не вдалося знайти зображення на екрані, програма зможе автоматично здійснювати пошук по тексту, і не завершиться помилкою, а скоріш за все знайде шуканий елемент управління, адже їхні назви рідко коли змінюються. Тож сильним модулем для розпізнавання тексті володіють всі з наведених систем крім T-Plan.

DDT - це такий підхід до тестування, при якому тестові дані зберігаються окремо від скриптів, зазвичай в документі Excel, файлі CSV або у базі даних. Даний підхід надає можливість без додаткових зусиль швидко переглядати чи редагувати дані, а не шукати їх по всьому проекту, коли виникне така необхідність. Питання можливості використання DDT в системі базованій на пошуку графічних елементів стоїть дуже гостро, тому воно виноситься в критерії відбору оптимальної системи для вирішення задачі розпізнавання елементів GUI. Спостерігаємо програшну ситуацію по цій позиції у T-Plan. Всі інші системи цілком задовольняють поставленим вимогам.

Якщо розглядати питання звітності то абсолютно всі засоби мають відповідну систему звітності про виконані дії. Всі засоби крім Sikuli надають

дуже детальний звіт з усіма можливими даними, та підкріпляють кожну дію відповідним скріншотом. У Sikuli ж ситуація інакша: всі виконані дії логуються із збереженням абсолютних координат зображення або дії на екрані. В загальному випадку для роботи цього цілком достатньо, хоча є можливість вдосконалити існуючу систему звітності у Sikuli для кращого перегляду тестів.

Для зручності зведемо проаналізовані дані в таблицю 1.

Табл. 1 Порівняння засобів розпізнавання зображень

Назва	API	Відсутня VNC	Безкоштовний	Розпізнавання тексту	DDT	Звітність
Ranorex	C#, VB	+	-	+	+	+
Sikuli	Java, Ruby, Python	+	+	+	+	+/-
T-Plan	-	-	+	-	-	+
EggPlant	Java, Ruby, Python	-	-	+	+	+

Тепер можна чітко зробити висновок, що в даній роботі найоптимальнішим засобом автоматизації тестування є Sikuli. Він поступається місцем іншим засобам автоматизації лише за одним параметром, який, по суті, є найменш важливим серед усіх наведених метрик. Саме Sikuli буде використовуватись у подальшій роботі для написання демонстраційної програми тестування графічного інтерфейсу користувача.

1.5 Висновок

Тема розпізнавання зображень є досить цікавою і представляє собою широке поле для подальших досліджень в галузі автоматизації тестування ПЗ. Актуальність теми підкреслюється використанням пошуку графічних елементів

для автоматизації тестування GUI. Дана технологія ще не використовується масштабно в процесі розробки ПЗ, але заслуговує на увагу. Не виключено, що цей підхід до тестування в найближчі роки буде активно впроваджуватися в компанія, які займаються розробкою ПЗ. В розділі були визначені ключові вимоги до засобу автоматизації тестування. На основі проведеного аналізу встановлено, що Sikuli є найбільш оптимальним засобом автоматизації для виконання даної роботи.

2. АЛГОРИТМ РОБОТИ МОДУЛЯ РОЗПІЗНАВАННЯ

2.1 Алгоритм роботи OpenCV

Обраний для виконання роботи засіб Sikuli надає повноцінну API, яка надає зручний інтерфейс для пошуку зображень по шаблону. Але Sikuli не має безпосередньо власного алгоритму пошуку зображень, тому використовує алгоритм із відкритої бібліотеки OpenCV під назвою Template Matching [6].

Якщо подивитись всередину цього алгоритму, він приховує в собі алгоритм базований на швидкому перетворенні Фур'є , оскільки він демонструє вищу швидкість виконання, в порівнянні зі звичайним перетворенням Фур'є. Якщо розглянути застосування цього методу на практиці ми маємо 2 зображення: основне, і зображення пошуку (шаблон пошуку). Ціль алгоритму визначити точку основного зображення в якій можна спостерігати максимальну відповідність до шаблону пошуку, як показано на рисунку 2.

Рис. 2 Ціль алгоритму Template Matching

FFT забезпечує утворення так званого «ковзного вікна», тобто пошукове зображення «ковзає» по основному зображенню шукаючи максимально чітку відповідність з ним. Цей процес ілюструється рисунком 3. Дана відповідність виражається у числових значення і називається крос-кореляцією.

Рис. 3 Переміщення пошукового шаблону над оригінальним зображенням

По мірі попіксельного переміщення шаблонного зображення ми отримуємо відповідну кількість значень крос-кореляції, які утворюють крос-кореляційну матрицю . Таким чином крос-кореляційна матриця забезпечить отримання координат верхнього лівого краю шуканого зображення. Ця процедура відбувається шляхом пошуку максимуму по строчкам і стовбцям крос-кореляційної матриці. Точка максимуму і дає нам максимальне співпадіння з шуканим зображенням, як показано на рисунку 4.

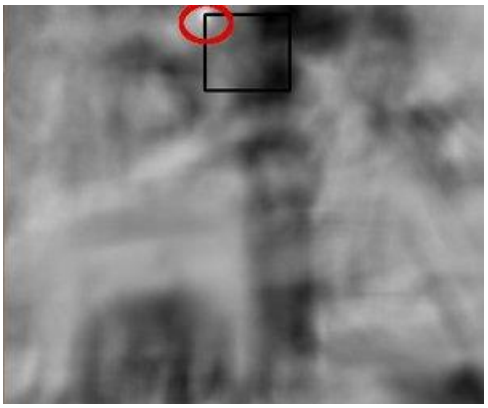


Рис. 4 Координати максимальної відповідності основного зображення до шаблону пошуку.

Загальна технологія пошуку зображення полягає у заповненні матриці шаблонного зображення нулями або середніми значеннями одного із зображень так щоб воно розширилось до розмірів основного зображення [7]. Отже ми

маємо можливість позбутися від областей, що оточують шаблон пошуку, встановлюючи значення за його межами рівними нулю. Але практика показує, що заповнення нулями не дає очікуваного результату. Справа в тому, що основне зображення може мати досить активні області, які даватимуть максимум крос-кореляційної матриці у тому місці яке не відповідає дійсності, а отже отримаємо хибні розрахунки.

Шляхом вирішення цієї проблеми є використання нормалізації [8]. Нормалізація дозволить зберегти попередні рішення (заповнення матриці нулями), а також працюватиме на локальному рівні у більшому зображенні, що дасть можливість вирівняти досить активні області. Загальний алгоритм нормалізації зображення має такий вигляд:

1. знайти середню величину відстані між пікселями використовуючи низькочастотний фільтр (ФНЧ);
2. відняти середню величину від значень відстані між пікселями початкового зображення;
3. обчислення середньоквадратичного відхилення відстані між пікселями;
4. нормалізація середньоквадратичного відхилення;

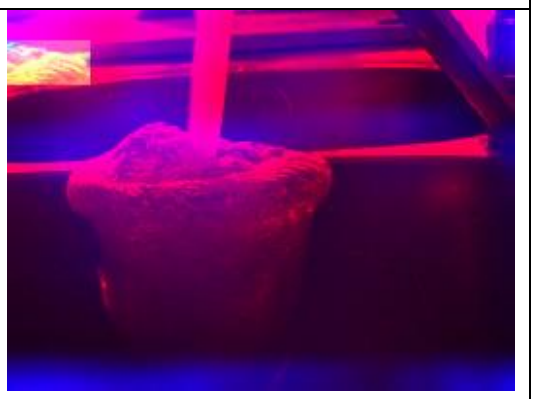
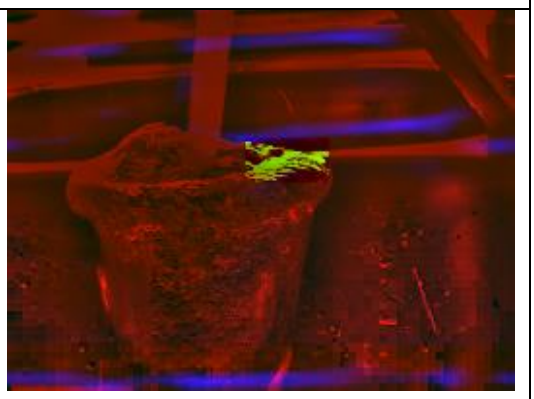
В результаті отримуємо нормалізоване зображення, яке придатне для перетворення Фур'є.

Однак існує ще одна проблема на границях основного зображення, коли шаблонне зображення виходить за його межі. Вона полягає в тому, що на границях ми маємо найменшу концентрацію корисної інформації. Очевидно, що залежно від розміру вікна, ми будемо отримувати різні результати крос-кореляції. Оскільки границі - найчастіші області помилок в обчисленні крос-кореляції і розмір вікна пов'язаний з точністю результату, ми повинні прийняти рішення щодо послаблення такого негативного впливу на загальний результат обчислень. Тому було вирішено затемнити ті пікселі, що входять в граничну область (граничною областю вважаємо та частину основного зображення яку по

краям перекриває шаблонне зображення). Цей процес називається *адаптивним контрастним збільшенням* і саме він дає можливість вирівняти дисперсію та середнє значення обох зображень і як наслідок дає коректні значення крос-кореляційної матриці.

В результаті проведеної нормалізації отримуємо цілком коректну процедуру локалізації зображення. Щоб порівняти вагомість проведення нормалізації розглянемо результат наведений в таблиці 2.1.

Табл. 2 Порівняння результатів без використання нормалізації та з нею.

Нормалізація	Крос-кореляційна матриця	Найкраща відповідність
Відсутня		
Наявна		

Завершальним кроком алгоритму Template Matching є обчислення сум абсолютних різниць. Обчислення SAD відбувається на основі отриманої кореляційної матриці за наступним алгоритмом [9]. Спершу матриця розбивається на комірки рівного розміру, які відповідають розміру шаблонного

зображення. Потім знаходиться локальний максимум для кожної комірки кореляційної матриці і по цьому максимум встановлюється потенційне положення шаблонного зображення. Отримавши певну кількість таких потенційних положень шуканого зображення вираховується значення SAD для кожної комірки. Значення SAD показує на скільки сильно відрізняється зображення у даній позиції від шуканого зображення, та обчислюється за наступною формулою:

$$\sum_{(i,j)\in W} |I_1(i,j) - I_2(x+i, y+j)| \qquad (1)$$

де W – кореляційна матриця;

i, j – координати пікселів в межах зображень;

I_1 – комірка кореляційної матриці;

I_2 – шаблонне зображення;

x, y – зміщення шаблону вздовж відповідних напрямків (в даному випадку $x=0, y=0$).

Таким чином прагнемо до мінімізації значення суми максимальних відмінностей серед всіх комірок кореляційної матриці. Отже серед усіх комірок вибираємо ту, яка містить область з найменшим значенням SAD. Рис. 5 детально описує фінальний крок алгоритму Template Matching.

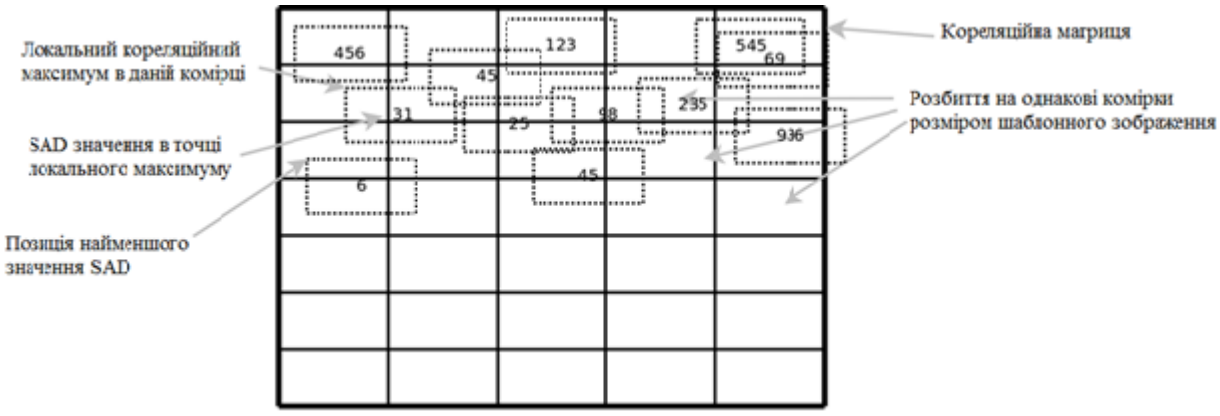


Рис. 5 Обчислення сум абсолютних різниць кореляційної матриці

Ця процедура дає значне прискорення пошуку максимальної відповідності по кореляційній матриці, тому на практиці вона широко застосовується саме з цією метою.

В результаті було отримано ефективний алгоритм пошуку зображення по шаблону, здатний правильно локалізувати шаблонне зображення за прийнятний час. Була застосована ефективна нормалізація, що надає можливість коректно застосувати прискорення пошуку за допомогою FFT, а також використаний алгоритм SAD задля прискорення фінального пошуку координат шаблонного зображення.

2.2 Загальний алгоритм роботи модуля розпізнавання

При розробці модуля розпізнавання елементів графічного інтерфейсу варто виділити загальний алгоритм, за яким працюватиме даний модуль. Блок-схема алгоритму наведена на рис. 6 Дотримання цих кроків є обов’язковим при розробці ПЗ для розпізнавання.

Рис. 6 Блок-схема загального алгоритму роботи модуля розпізнавання

Тож спершу необхідно запустити програмний продукт який є об’єктом тестування. Далі виконати ініціалізацію всіх ресурсів необхідних для проведення процесу тестування. Потім виконується розпізнавання елементів згідно до тест-плану і нарешті завершення роботи програми.

При розробці модуля розпізнавання в даній роботі будуть витримані ці кроки, тому розглянемо детальніше кожний з них.

Початковий крок ініціює запуск програмного забезпечення яке підлягає процесу тестування. На даному етапі відбувається створення об’єкту ініціалізації додатка App та виклик функції run() для нього. Також виділяються додаткові ресурси необхідні для проведення тестування. На приклад створюються буфери для читання інформації з файлу чи бази даних.

Далі відбувається ініціалізація елементів управляння. Тут мається на увазі що програма завантажує всі необхідні дані для тестування. При тестування графічного інтерфейсу за технологією пошуку графічних елементів це зазвичай будуть зображення. Ці зображення містять вигляд всіх необхідних елементів управляння, переходів, вигляд вікна в цілому, або якоїсь його частини в залежності від розробленого тест-плану.

Далі робота програми будується на виконанні тест-кейсів розробленого тест-плану, виконується безпосереднє розпізнавання зображення, які є елементами управління, очікування зображень що представляють вінка даної програми. Вікна, в свою чергу, мають свої елементи управління, утворюючи деревовидну структуру під назвою дерево елементів графічного інтерфейсу. При стандартному підході до тестування GUI необхідно було побудувати дерево елементів графічного інтерфейсу перш ніж виконувати тестування, адже процес відбувався всередині програми (по методу white-box testing). На противагу цьому підходу, розпізнавання елементів застосовується при методології black-box тестування, тому не потребує знань про внутрішню структуру ПЗ і в тому числі не потребує знання дерева елементів GUI, а навпаки, може забезпечувати побудову цього дерева на даному етапі виконання програми.

І останнім кроком загального алгоритму розпізнавання є завершення роботи програми з попереднім вивільненням усіх зайнятих ресурсів, закриттям з’єднань і т.д.

2.3 Можливості Sikuli для вирішення задачі розпізнавання

2.3.1 Загальні відомості та рекомендації

Можливості SikuliX базовані на роботі з зображеннями є чутливими для пікселів, тобто або зображення на екрані попіксельно співпадає або SikuliX провалить тест (з деякою незначною похибкою). Це зазвичай призводить до потреби мати безліч різних зображень для різних середовищ. Версія 2 матиме дещо більше особливостей, що спростить обробку таких ситуацій.

Інші аспекти, важливі для тестувальників:

- SikuliX необхідно мати реальний екран, що відображає виконання додатку або щонайменше деяке еквівалентне віртуальне рішення;
- SikuliX доступний тільки на машинах під Windows, Mac або Linux і маючи версію Java 6+.
- SikuliX користується пакетом OpenCV для виявлення зображення на екрані.

Особливість пошуку SikuliX заснована на OpenCV методі matchTemplate(). Основна схема пошуку заклечається в тому щоб, дочекатися зображення, що з'являється в певному регіоні екрану:

```
# деяка верхня ліва частина екрану
aRegion = Region (0, 0, 500, 500)
# file.png  - зображення збережене у файловій системі
# це зображення, яке ми хочемо шукати в заданому регіоні
aImage = "someImage.png"
# шукайте і отримуйте результат
aMatch = aRegion. find (aImage)
```

MatchTemplate() чекає навіть масштабованого або більшого зображення, де вхідне зображення має бути знайдене. Щоб підготувати це, ми внутрішньо робимо скріншот (використовуючи клас Java Robot) екранної області, визначеної цим регіоном. В цей час базове зображення зберігається в пам'яті. Цільове зображення зчитується з відповідного файлу. Обидва зображення потім перетворюються в потрібні об'єкти (CVMat) OpenCV.

Зараз ми виконуємо функцію matchTemplate() і отримуємо матрицю за розміром базового зображення, що містить для кожного пікселя відмітку схожості з цільовим зображенням. Значення відмітки пікселя в кожній позиції змінюються між 0.0 і 1.0: чим нижче значення, тим нижча вірогідність, що область з верхньої лівої сторони в цьому розташуванні пікселя містить цільове зображення. Значення відмітки вище за 0.7 - 0.8 говорить про високу вірогідність, що зображення знайдене в цій позиції.

У наступному кроці, ми користуємося іншим методом OpenCV, щоб отримати релевантне значення (результуючу відмітку) максимуму від згаданої матриці результату, тобто, що ми шукаємо піксель у базовому зображенні, для якого верхнє ліве зображення буде максимально співпадати з цільовим зображення.

Якщо не вказано додаткової інформації, то будуть взяті відмітки результату тільки більше 0.7. Інші значення повідомлять про виключення: FindFailed. В залежності від різних аспектів цільового зображення (головним чином, скільки фонових підзображень у напрямку до країв міститься в цільовому зображенні), зазвичай отримаємо результати > 0.8 або навіть 0.9. Якщо слідували рекомендації SikuliX, як створити цільові зображення, у більшості випадків отримаємо результуючу відмітку > 0.95 або навіть > 0.99 (за значення повного співпадіння взято величину 1.0).

Якщо відмітка результату прийнята, в наступному кроці ми створюємо об'єкт відповідності, це вказує на регіон екрану, що ймовірно містить зображення.

Якщо зображення не знайдене (відмітка не задовольняє мінімальному прохідному значенню), то або пошук закінчується з повідомленням про невдалий пошук, або, починається новий пошук з нового скріншоту цього регіону. Це повторюється доки зображення не буде знайдене, або закінчиться заданий користувачем чи стандартний(3 секунди) час очікування відповіді пошуку, який також призводить до виключення FindFailed. Норма цього повторення може бути вказана, щоб скоротити використання процесора засобом SikuliX.

Чим більше базове зображення, тим довше відбувається пошук. Чим менша різниця у розміру базового та цільового зображень, тим швидше відбувається пошук. На сучасних системах з великими моніторами, що шукають, маленькі або середні за розміром зображення (до 10 000 пікселів), тривалість обробки, можливо, була б між 0.5 і 1 секундою або навіть більше. Звичайний підхід, щоб скоротити пошуковий час - скоротити пошуковий регіон якомога більше до області, де очікується, що цільове зображення буде знайдене. Маленькі зображення приблизно 10 пікселів у пошуковому регіоні приблизно 1000 пікселів знаходяться за час приблизно в 10 мілісекунд або навіть швидше.

Остання версія SikuliX 1.1.0 включила в себе особливість: перед пошуком в пошуковому регіоні, спочатку перевіряється, чи зображення знаходиться все ще в тому ж місці, як під час останнього пошуку (якщо пошуковий регіон містить цю відповідність). У випадку успіху, ця облегшуюча операція забирає декілька мілісекунд, що істотно збільшує швидкість процесу пошуку, але тільки у випадку використання завдань, що повторюються.

Якщо ви хочете знайти всі співпадіння на екрані, існує findAll() метод, який повертає список відповідностей в порядку зменшення значення відмітки результату. Ви, можливо, пропрацювали б цей список згідно з їх позицією на екрані, користуючись (x, y) координатами верхнього лівого кута. findAll внутрішньо оцінює матрицю пошукового результату, методом пошуку

наступного значення максимуму після "виключення" деякої області навколо останнього максимуму.

2.3.2 Обробка зображень в SikuliX

Щоб користуватися зображеннями з особливостями SikuliX такими як клік, подвійний клік і т. д., вам треба зберігати ці зображення в файли бажано PNG форматі у файловій системі або в Інтернеті. Зображення в цьому контексті являють собою прямокутну піксельну область, узяту з екрану (створенням скріншоту).

Процес створення скріншотів підтримує IDE, також це можна зробити програмно через відповідні властивості SikuliX API. Щоб завантажити зображення SikuliX має 2 принципи:

- *шлях до зв'язки зображень* : зображення зберігаються разом зі скріптовим файлом сценарію (.py для Python, .rb для Ruby та .js для JavaScript) в теці з назвою someScript.sikuli, де файл сценарію має бути названим так само, як тека (наприклад someScript.py). Це все автоматично виконується, працюючи з SikuliX IDE.
- *шлях до зображення* : додатково SikuliX підтримує список місць як шлях до зображення. Можливі місця - теки у файловій системі, теки у jar-файлі і в Інтернеті. Є функції, доступні, щоб управляти вашим власним шляхом зображення. Коли зображення необхідно завантажити (виключення: абсолютний шлях даний), папки послідовно перевіряються на існування зображення. Перша папка що відповідає вимогам отримує пріоритет.

Рекомендовано мати схему найменувань для файлів зображень і не покладатися на основний файл, що називає SikuliX IDE за поточним часом системи, який створений в основному для нових користувачів з маленьким досвідом програмування.

Версія 2 матиме окремим додатком інструмент створення скріншотів, який підтримуватиме такі основні операції оброблювального зображення:

- створення і видалення скріншотів протягом технологічного процесу (деякий вид реєстратора);
- організація шляху до зображення;
- групування подібних зображень (може бути відключено в залежності від аспектів середовища);
- групування зображень, які так чи інакше мають відношення один до одного і мають бути знайдені разом;
- організація різних станів зображення (на приклад активна і неактивна кнопка);
- оптимізація скріншотів задля отримання найвищої можливої швидкості пошуку;

2.4 Висновок

У розділі було розглянуто деталі алгоритму пошуку зображень Template Matching, який використовується в OpenCV. Також розкриваються основні можливості Sikuli по роботі із зображеннями. Висвітлені ключові функції цієї системи та наведені практичні рекомендації по покращенню ефективності пошуку зображення на екрані.

Крім того важливим моментом є представлення загального алгоритму роботи модуля розпізнавання. Під час розробки тестового програмного забезпечення варто чітко дотримуватись наведених кроків алгоритму. Тому подальше виконання роботи буде відбуватись навколо представленого загального алгоритму.

3. РОЗРОБКА МОДУЛЯ РОЗПІЗНАВАННЯ ЕЛЕМЕНТІВ ГРАФІЧНОГО ІНТЕРФЕЙСУ КОРИСТУВАЧА

3.1 Побудова цілей тестування на основі тест-плану

Як було сказано раніше, основною ціллю використання розпізнавання зображень є автоматизація тестування графічного інтерфейсу. Замість мануального тестування отримуємо, таким чином, автоматизоване black-box тестування. Основною перевагою даного підходу є те, що він значно скорочує часові затрати на тестування.

Підхід до тестування базований на розпізнаванні графічних елементів доцільно використовувати у наступних випадках:

- відсутній доступ до властивостей елемента графічного інтерфейсу користувача;
- властивості елемента графічного інтерфейсу користувача постійно змінюються;
- функціонал більше не оновлюється;
- брак часу у команди;
- брак досвіду команди.

Для демонстрації використання Sikuli оберемо широковідомий інженерний пакет Wolfram Mathematica 10 версії. В цьому випадку ми маємо повністю закриту систему, що говорить про відсутність доступу до властивостей елемента графічного інтерфейсу користувача. Отже доцільно використати Sikuli для тестування певного функціоналу Wolfram Mathematica.

Wolfram Mathematica у новій версії 10 надає можливість динамічних обчислень використовуючи повзунки (sliders) за допомогою функції Manipulate [10].

Рис. 7 Зображення повзунка в Wolfram Mathematica.

Динамічні обчислення надають змогу створювати залежності між значенням змінної і результатом обчислень. Дані залежності можуть бути змінені динамічно у будь який час. Також існує можливість використовувати додаткове меню повзунка, яке розкриває весь потенціал цього засобу. Додаткове меню може бути викликане натисканням кнопки «+» справа від повзунка і містить такі можливості:

- встановлення кроку зміни параметра;
- збільшення/зменшення параметра;
- автоматичне «відтворення», тобто плавна зміна параметра в реальному часі.
- прискорення/уповільнення автоматичного відтворення
- 3 варіанти для напрямку відтворення: зациклений вперед, зациклений назад, зациклений в екстремальних межах.

Виконаємо тестування функції Manipulate за допомогою засобів що надає Sikuli API.

Процес тестування буде відбуватись по заздалегідь запланованому сценарію, що відповідає тест-плану. Розглянемо цей сценарій. Першим кроком, очевидно, є запуск програми яка тестується. Запуск може відбуватися двома шляхами: створення об’єкту App і виклик його методу run(), або ж шляхом стандартним для користувача подвійним кліком на іконку запуску програми. Далі програма повинна дочекатись стартове вікно Wolfram Mathematica і розпізнати його на екрані. Як тільки Sikuli розпізнає це вікно, вона переходить до наступного кроку тест-плану – створення нового документа. Це відбувається шляхом натискання кнопки New document, дана дія повинна перевести нас в нове вікно з порожнім документом (так званий notebook в термінах Wolfram Mathematica).

Вікно нового документу необхідно заповнити вхідною інформацією. Для прикладу візьмемо побудову графіка гармонічного сигналу, де за допомогою функції Manipulate можна змінювати частоту фазу та амплітуду сигналу. Тож код для реалізації даної ідеї має наступний вигляд:

```
Manipulate[Plot[A * Sin[f x + p], {x, 0, 6}], {{f, 2, "Frequency"}, 1, 6},
 {{p, 0, "Phase"}, 0, 2 * Pi}, {{A, 0.3, "Amplitude"}, 0.2, 0.5}]
```

Рис. 8 Код демонстрації роботи функції Manipulate

В результаті виконання цього коду Wolfram Mathematica дасть наступний результат:

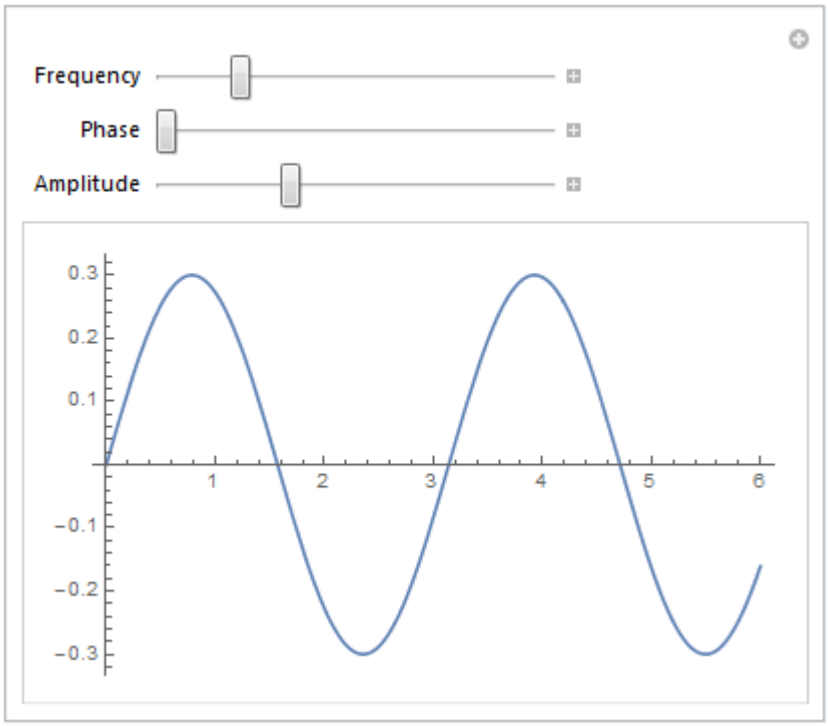


Рис. 9 Результат роботи функції Manipulate

Крім того що Wolfram Mathematica може знаходити зображення будь-яких елементів графічного інтерфейсу та виконувати взаємодію типу click() або doubleClick(), варто продемонструвати виконання інших варіантів взаємодії, таких як dragAndDrop(). З допомогою цієї функції, крім звичайної дії, можна

виконати переміщення повзунка. Тож, дочекавшись зображення на рисунку 3, виконається дія по переміщення повзунка вправо. Таким чином отримаємо зміну графіка демонстраційного сигналу в реальному часі.

Далі перевіримо роботу додаткового меню, яке відкривається натисканням на «+» справа від повзунка. Серед наведених вище можливостей меню змінимо напрямок автоматичного переміщення повзунка на протилежний і виконаємо відтворення в зворотному напрямку.

Коли наведені дії завершиться для всіх повзунків, виконується завершення роботи. Для цього необхідно закрити вікно створеного документу відхиливши запит на збереження цього файлу та закрити основне вікно Wolfram Mathematica.

3.2 Платформа реалізації

Основним критерієм до вибору платформи послугувала можливість кросплатформності у використанні технології пошуку графічних елементів. Оскільки дана методологія дозволяє абстрагуватися від властивостей і технологій використаних при створенні графічного інтерфейсу. Тож серед усіх відомих мов програмування вибір падає саме на Java. Віртуальна машина Java забезпечує кросплатформність, що поширюється на усі існуючі на сьогодні найбільш популярні платформи – Windows, Linux, Mac. Це дозволить повноцінно користуватися перевагами гафічного підходу до автоматизації GUI. Також вагомим аргументом для використання цієї мови програмування є наявність досить розвинутої і надійної бібліотеки SikuliX API, яка детально була описана в попередньому розділі.

Окрім кросплатформності сильними сторонами мови Java є також висока надійність роботи, розвиненість мови. Java була зароджена як об’єктно-орієнтована мова програмування, і надійність цього підходу з роками тільки виправдала себе. Парадигма ООП наділяє мову такою прекрасною властивістю як масштабованість, що дає змогу без особливих труднощів багаторазово розширювати систему. Мова Java створена так, щоб бути максимально простою

надійною. На приклад в ній забороняється використовувати множинне наслідування, для запобігання уникнення неоднозначності при зверненні до батьківського класу. Натомість було впроваджено поняття інтерфейсу, який вже не є класом, проте містить загальні рекомендації до створення класів і надає можливість множинного наслідування.

Окремою темою є безпека мови, на якій розробники сконцентрували найбільшу увагу. Тому щоб створити просту, безпечну і безвідмовну мову програмування, в Java існує система винятків або ситуацій, коли програма зустрічається з неочікуваними труднощами, наприклад:

- операції над елементом масиву поза його межами або над порожнім елементом;
- читання з недоступного каталогу або неправильної адреси URL;
- ввід недопустимих даних користувачем.

Одна з особливостей концепції віртуальної машини полягає в тому, що помилки (виключення) не призводять до повного краху системи. Крім того, існують інструменти, які «приєднуються» до середовища періоду виконання і кожен раз, коли сталося певне виключення, записують інформацію з пам'яті для відлагодження програми. Ці інструменти автоматизованої обробки виключень надають основну інформацію щодо виключень в програмах на Java.

Для ефективного керування пам'яттю під час життєвого циклу об'єкта Java використовує автоматичний збирач сміття. Програміст вирішує, коли створювати об'єкти, а віртуальна машина відповідальна за звільнення пам'яті після того, як об'єкт стає непотрібним. Коли до певного об'єкта вже не залишається посилань, збирач сміття може автоматично прибирати його із пам'яті. Проте, витік пам'яті все ж може статися, якщо код, написаний програмістом, має посилання на вже непотрібні об'єкти, наприклад на об'єкти, що зберігаються у діючих контейнерах. Збирання сміття дозволене у будь-який час. В ідеалі воно відбувається під час бездіяльності програми. Збірка сміття автоматично форсується при нестачі вільної пам'яті в купі для розміщення

нового об'єкта, що може призводити до зависання. Тому існують реалізації віртуальної машини Java з прибиральником сміття спеціально створеним для програмування систем реального часу.

Java не має підтримки вказівників у стилі C/C++. Це зроблено задля безпеки й надійності, аби дозволити збирачу сміття переміщувати вказівникові об'єкти.

3.3 Особливості процесу реалізації

3.3.1 Архітектура

Детальний опис архітектури модуля тестування представлений на рисунку 10.

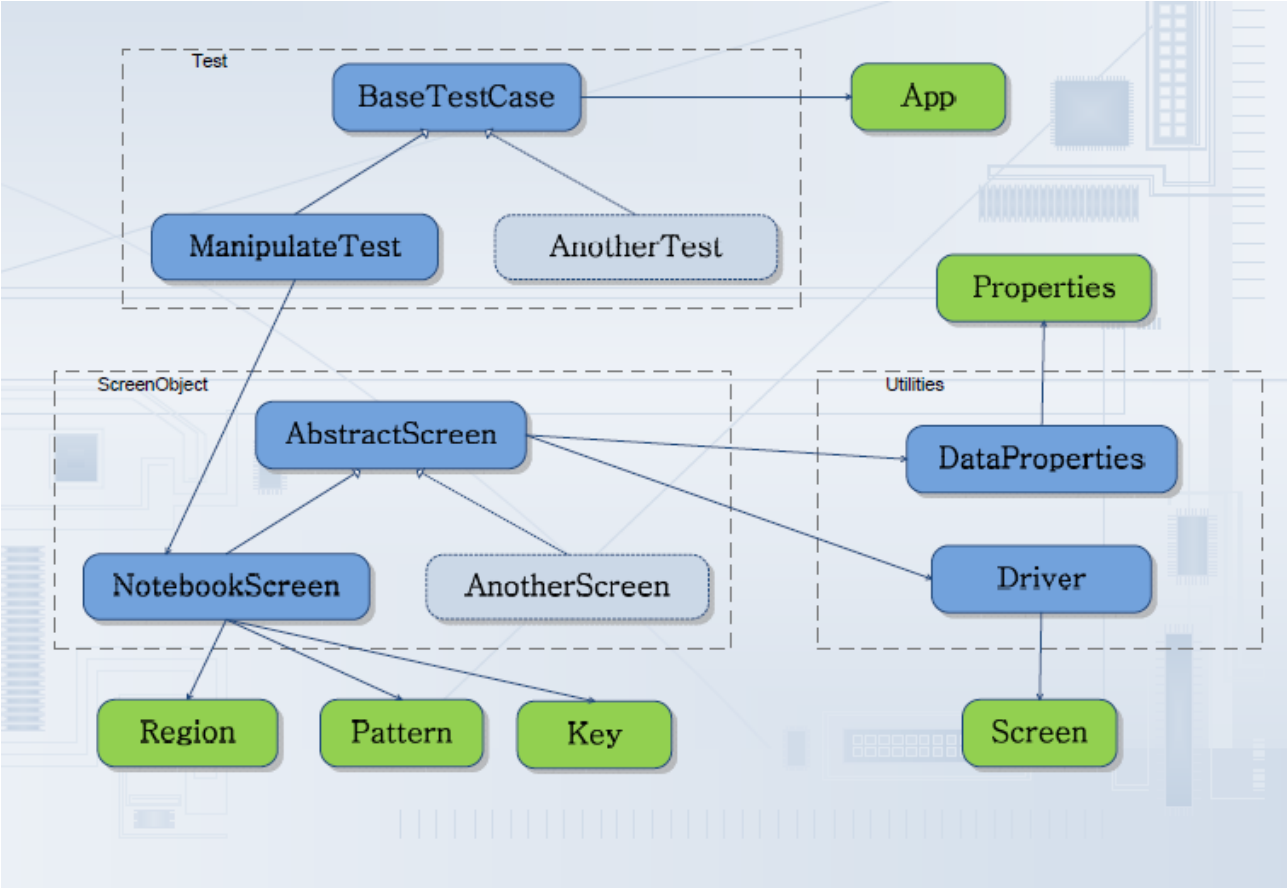


Рис. 10 Діаграма класів модуля розпізнавання елементів GUI

Створення модуля розпізнавання відбувається згідно з розробленою діаграмою класів. Вона на конкретному прикладі демонструє загальну архітектуру програм автоматизації тестування GUI, базованої на розпізнаванні елементів графічного інтерфейсу. Розглянемо складові діаграми на рисунку 10, щоб детальніше зрозуміти роботу програми.

Модуль в цілому поділений на 3 пакети: screen, test та utility. Пакет screen демонструє логіку роботи програми. Його класи містять методи для роботи з усіма елементами графічного інтерфейсу над якими буде відбуватись процес тестування. Кожен клас відповідає за функціонал певного вікна і містить унікальний набір методів для роботи з елементами які включає в себе це вікно. Пакет utility містить допоміжні засоби які спрощують та узагальнюють доступ до ресурсів необхідних до тестування. Ресурсами в даному випадку виступають скріншоти вікон і елементів управління графічного інтерфейсу. Доступ до зображень надається класом DataProperties. Ще один клас присутній в пакеті – Driver, він надає монопольний доступ до екрану з будь якої частини програми. Driver організований згідно шаблону проектування singleton, таким чином створюється тільки один об’єкт для доступу до екрану пристрою.

І третій пакет створений для зберігання тестів. При використанні спеціалізованих засобів для тестування, як на приклад TestNG, це є особливо актуальним. Класи цього пакету містять методи для безпосереднього запуску програми та проведення тестування. Вони переважно використовують об’єкти із пакету screen для доступу до функціоналу програмного забезпечення.

Далі мова піде про описання деталей системи, власні класи та класи із API, які використовуються в модулі. Основною ідеєю створення даного модуля є виділення в роботі програми, яка підлягає тестуванню, екранних об’єктів (screen objects). Ці об’єкти представляють собою набір функціоналу який зображений в певний момент часу на екрані. Оскільки в Wolfram Mathematica основним зображенням для роботи є новий текстовий документ (так званий notebook), то відповідно до цього в модуль буде включений клас під назвою NotebookScreen.

Якщо під час розробки ПЗ виникне ситуація що призведе до розширення функціоналу, таким чином, що в програмі з’явиться нове вікно з абсолютно новими елементами управління та їх властивостями, то не складе ніяких проблем розширення модуля задля більш ефективного тестового покриття. Такий підхід надає модулю можливості для гнучкості та масштабування при оновленні ПЗ, що тестується до нової версії. Відповідно до всіх створених класів екранних об’єктів варто створити абстрактний клас під назвою AbstractScreen, який безпосередньо і надає можливість для масштабування всієї системи. До даного класу варто віднести всі загальні властивості його класів-потомків. Якщо розглядати питання з точки зору екранних об’єктів, то це можуть бути, на приклад, час очікування реакції на натискання кнопки чи іншого елемента управління. Також це може бути час очікування безпосередньо вікна, але варто бути обережним і враховувати реальні часові можливості машини на якій відбувається процес тестування, оскільки може виникнути неприємна ситуація, коли закінчився час очікування вікна, а воно ще не встигло завантажитись через особливості операційної системи, або апаратне забезпечення машини. В такому випадку отримаємо виключення FindFailed, і відповідно провалений тест.

Екранні об’єкти працюють безпосередньо з елементами управління отже вони повинні мати механізм для розпізнавання, таким механізмом виступають засоби Sikuli API. Найбільш розповсюджені інструменти для розпізнавання зображень графічного інтерфейсу є класи Region та Pattern. Згідно з документацією до Sikuli API [11] клас Pattern описує шаблон для пошуку на екрані. Він може створюватись на основі зображення із диску або з Інтернет чина основі створеного скріншоту. Крім стандартних операцій пошуку і очікування, він підтримує основні процедури взаємодії з елементом управління так як клік, перетягування і т.д. Клас Region описує регіон екрану де буде вестись пошук. Це прекрасний інструмент придуманий розробниками Sikuli для того щоб локалізувати пошукову область і тим самим скоротити часові затрати на пошук зображення на екрані. Такий підхід є досить ефективним і широко

використовується під час пошуку зображення, якщо, звісно зарані відомо в якій частині екрану очікується знайти зображення. Додатково використовується клас Key, який допомагає виконувати операції з клавіатурою. Підтримується одночасне натискання певної комбінації клавіш, що може буди корисним і навіть необхідним у деяких випадках. Для прикладу в програмі Wolfram Mathematica щоб виконати написаний в документі код необхідно натиснути комбінацію Shift + Enter тож використання класу Key є просто необхідним в нашому випадку.

Серед утиліт представлених в пакеті utility використовуються засоби мови Java для доступу до зовнішніх ресурсів програми. Таку можливість надає клас Properties, що використовується в класі DataProperties з пакету utility. Він завантажує файли що знаходяться в папці на диску та забезпечує універсальний доступ до них. Даними файлами є заздалегідь підготовлені скріншоти, що місять зображення елементів управляння. Крім завантаження і доступу до файлів клас DataProperties надає доступ до файлу конфігурації де зберігається інформація що може бути змінювана, це переважно параметри програми такі як шлях до виконуваного файлу, для запуску програми, час очікування вікна та елемента управлінні (вони відрізняться через довше завантаження вікна).

Іншою утилітою є клас Driver який надає доступ до об’єкту екрану. Він використовує клас Screen із Sikuli API для отримання доступу до екрану, щоб надалі здійснювати пошук по ньому. Таким чином Driver надає доступ до екрану із будь якої точки програми. Крім того він реалізований по шаблону проектування Singleton, таким чином, щоб створювати єдиний об’єкт екрану при першому зверненні і повертати цей же об’єкти при всіх подальших зверненнях до даного класу. Реалізація такої поведінки зображена на рисунку 11 і виглядає наступним чином:

```java
package mathematica.utils;

import org.sikuli.script.Screen;

public class Driver {

    private static Screen driver;

    public synchronized static Screen getInstance() {
        if (driver == null){
            driver = new Screen();
        }
        return driver;
    }
}
```

Рис. 11 Реалізація шаблону проектування Singleton для класу Driver

Найвищою ланкою зв'язку між Sikuli та ПЗ, що тестується є безпосередньо тести. Для них створений загальний клас BaseTestCase, який містить в собі основні процедури тестування. До цих процедур належать: запуск ПЗ, завершення роботи ПЗ. У випадку з використання Wolfram Mathematica сюди також входить створення нового документа. Запуск програмного забезпечення відбувається за допомогою класу App, який надає можливості доступу до ПЗ встановленого на машині по шляху до виконуваного файла даного ПЗ.

Далі відповідно до функціоналу ПЗ створюються потомки базового класу для тестування. Оскільки тестовий модуль має на меті лише продемонструвати систему тестування, в ньому буде реалізований один клас-потомок від BaseTestCase з назвою ManipulateTest, який має на меті продемонструвати виконання тесту для функції Manipulate із Wolfram Language.

3.3.2 Використання Sikuli API

Перше використання засобів Sikuli зустрічаємо уже на першому кроці алгоритму – запуск програми тестування. Для цього слугує клас App та його метод open(). Приклад використання класу:

```
protected static Region runApp() throws FindFailed {
    App.open(appName);
```

```
        Region welcomeWindow = Driver.getInstance().wait(welcomeWindowP,
        windowTimeout);
        Region nb = createNotebook(welcomeWindow);
        return nb;
}
```

Даний метод ініціює запуск зображення за допомогою строчки appName, де зберігається шлях до виконуваного файлу Wolfram Mathematica. Далі відбувається очікування стартового вікна, зображеного на рисунку 12.

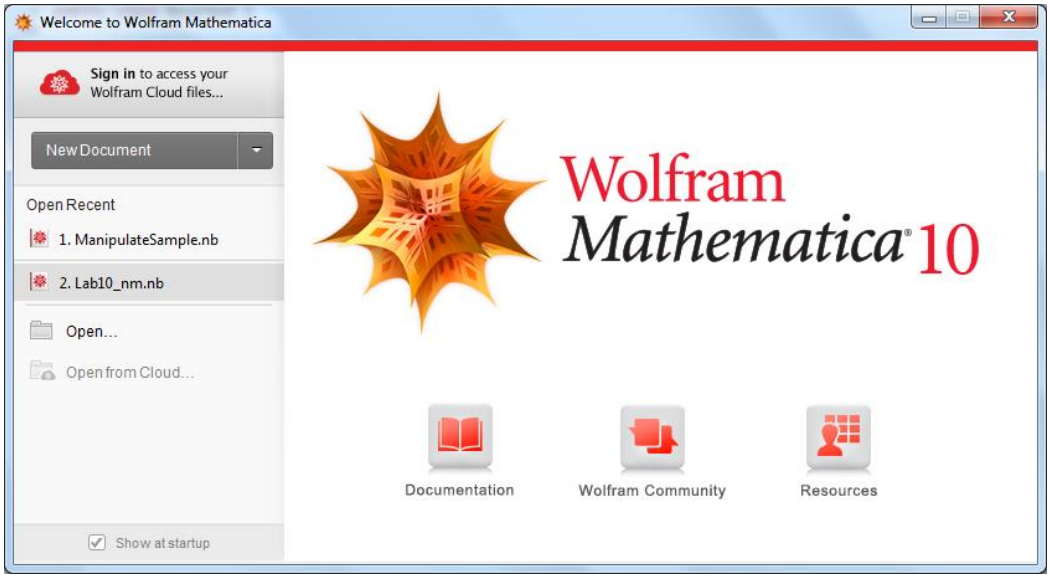


Рис. 12 Зображення стартового вікна Wolfram Mathematica.

Процедура очікування відбувається наступним чином: зображення для очікування зберігається у об’єкт Pattern під назвою welcomeWindowP. Далі за допомогою драйвера отримуємо доступ до екрану, де з’являється можливість викликати метод очікування wait(), який повертає знайдений об’єкт у формі регіону (Region welcomeWindow). Даний регіон пошуку відправляється до наступного методу, для того щоб під час пошуку певних елементів управління використовувати не весь екран, а тільки певну його частину. Тут же відбувається створення екранного об’єкта за допомогою виклику методу createNotebook(welcomeWindow). Код даного методу наведений нижче.

```
private static Region createNotebook(Region windowRegion) throws FindFailed {
        windowRegion.find(newDocument).click();
        Region notebookWindow = Driver.getInstance().wait(notebookWindowP);
        return notebookWindow;
}
```

По тій же схемі відбувається пошук кнопки з назвою newDocument зображеної на рисунку 13.

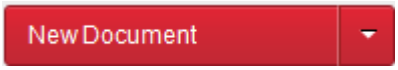


Рис. 13 Зображення кнопки створення нового документу.

Як тільки кнопка буде знайдена симулюємо клік по ній на допомогою методу click(). А далі очікуємо з’явлення нового вікна з виглядом на рисунку 8 і так же зберігаємо результат в регіон пошуку щоб передати його в подальше опрацювання. Нагадаємо, така процедура дозволяє значно скоротити час пошуку зображення на екрані.

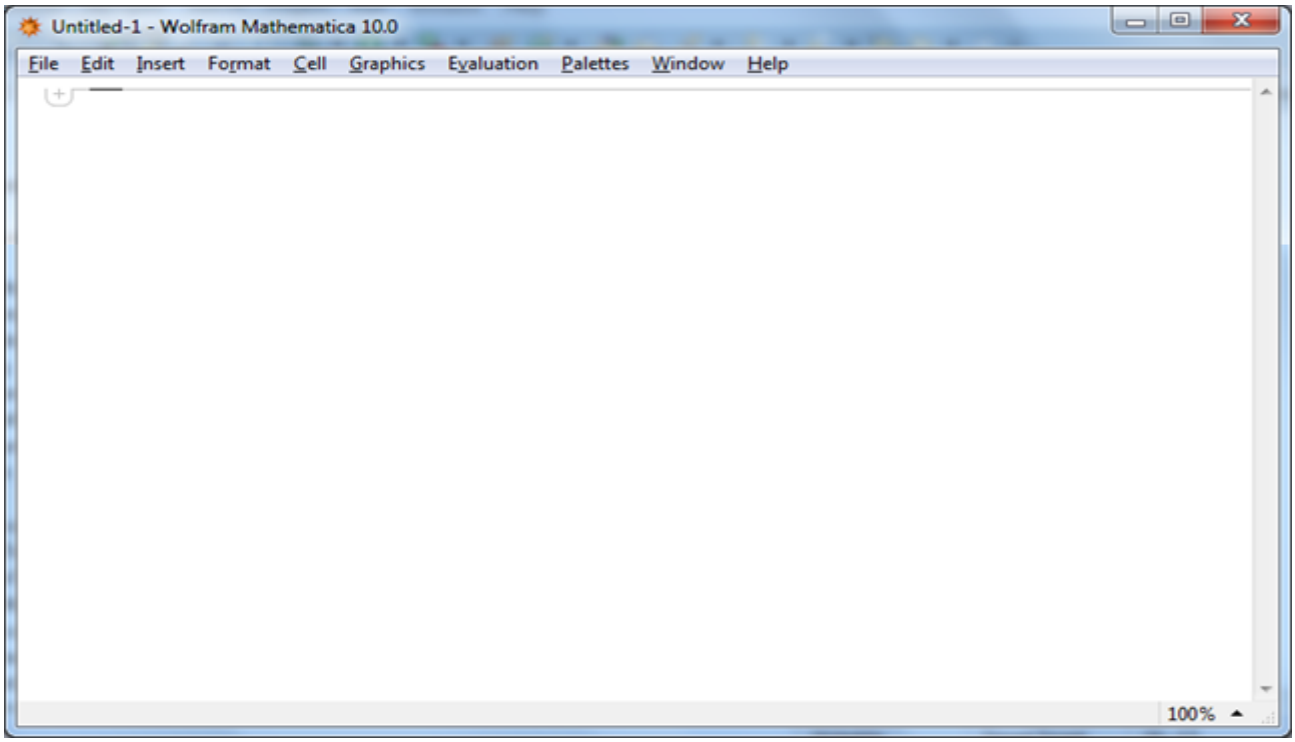


Рис. 14 Вигляд вікна нового документу.

Далі відбувається введення коду в вікно нового документу та його компіляція. Компіляція у Wolfram Mathematica звичайно відбувається за допомогою натискання комбінації клавіш Shift + Enter. Тому Sikuli містить засоби для роботи з клавіатурою. В приведеному нижче коді відбувається виклик стандартного методу type(), який служить засобом клавіатурного вводу інформації. Але перший виклик друкує в документі сам код, який надходить в клас ззовні, з файлу конфігурації. А другий виклик служить лише для отримання результатів введеного коду за допомогою комбінації клавіш Shift + Enter.

```
public void typeCodeAndCompile(String code) {
        notebook.type(code);
        notebook.type(Key.ENTER, KeyModifier.SHIFT);
}
```

Результатом виконання введеного коду є рисунок 9.

Операція знаходження слайдера подібна до описаних вище. Пошук патерна з рисунку 1 відбувається в регіоні отриманому на попередньому кроці. Коли слайдер знайдений його регіон також зберігається, адже далі буде відбуватися пошук повзунка в межах слайдера. Наведений нижче код описує переміщення знайденого повзунка вздовж слайдера.

```
public void moveSlider(Region slider, Pattern panelP) throws FindFailed,
InterruptedException {
        Region thumb = slider.find(sliderThumb);
        Region openAdds = slider.find(openAddsP);
        Settings.DelayAfterDrag = 1;
        Settings.MoveMouseDelay = 4;
        slider.dragDrop(thumb,openAdds);
        Settings.MoveMouseDelay = 0.5f;
        openAdds.click();
        Region panel = getDriver().find(panelP);
```

```
        animation(panel);
    }
```

Під час виконання даного коду ми приймаємо 2 аргументи: загальний регіон пошуку slider та шаблон пошуку – зображення панелі інструментів слайдера. Спочатку ми знаходимо слайдер і кнопку відкриття панелі інструментів. Далі використовуємо методи класу Settings для встановлення затримки на швидкість переміщення вказівника миші. Основним методом тут є slider.dragDrop(thumb,openAdds); який безпосередньо і виконує переміщення повзунка. Далі відбувається відкриття панелі інструментів, представленої на рисунку 15, і робота з нею за допомогою методу animation().

Рис. 15 Панель інструментів

Після виконання всіх дій виконується завершення роботи програми. Для цього треба спочатку закрити документ. Під час закриття з’явиться вікно з запитом на збереження нового документу. Оскільки це тестовий приклад. То збереження документу не є значущим при виконанні роботи, тож не будемо зберігати внесені зміни. Нарешті отримуємо завершальну стадію виконання програми – основне вікно виходу з програми, зображене на рисунку 16.

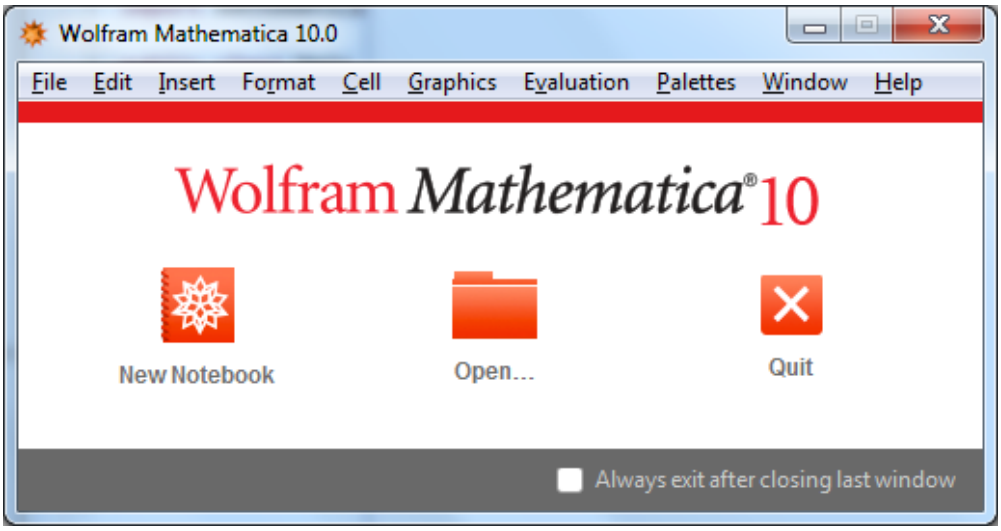


Рис. 16 Вікно завершення роботи Wolfram Mathematica

Завершальним моментом є знаходження кнопки виходу із програми. Воно відбувається за тією ж схемою, як і попередній пошук елементів.

3.4 Висновок

Розробка модуля розпізнавання відбувається відповідно до зарані розробленого тест-плану. При написанні програмного коду можна використовувати засоби тестування подібно до TestNG або JUnit, для більш ефективної та гнучкої організації тестів.

В розділі були наведені аргументи на користь використання мови програмування Java, виділені її найсильніші сторони та проведений детальний опис її можливостей. Також наведені всі деталі реалізації програмного продукту тестування GUI від архітектури побудови модуля до прикладів використання Sikuli. Також дані деякі рекомендації по спрощенню розробки та підтримки ПЗ, націленого на тестування GUI, які доцільно можуть бути використані задля уникнення поширених помилок при написанні програмного коду.

4. ОГЛЯД РЕЗУЛЬТАТІВ РОБОТИ МОДУЛЯ РОЗПІЗНАВАННЯ

4.1 Розгляд отриманих результатів роботи

Основне джерело звітності по результатах роботи модуля розпізнавання є побудований лог. Лог надає можливість відслідкувати всі дії, які виконуються програмою, їх час та спосіб виконання, таким чином отримуємо маємо можливість отримати детальний звіт про стан системи протягом виконання тесту. Лог містить в собі певну процедуру, таку як пошук зображення чи очікування вікна на екрані, час виконання даної процедури. Якщо процедура пов’язана із взаємодією з елементом управління, також додатково вказується тип взаємодії (клік лівою кнопкою миші - CLICK on L, клік правою кнопкою миші CLICK on R, TYPE “some text”), координати взаємодії та екран на якому відбулась взаємодія.

Основний логер створений розробниками Sikuli не є досить сильним та інформативним, тому він був перероблений таким чином, щоб задовольняти вищевказаним вимогам. Хоч логування не є сильною стороною Sikuli, її розробниками була надана можливість виправити цю помилку іншим розробникам. Даний функціонал надається класами Debug та Settings, які дозволяють налаштовувати інформативність логу, включати в лог власні повідомлення, надають доступ до існуючого логу повідомлень. Тож користуючись такою нагодою лог був перероблений таким чином, щоб його інформація детально звітувала про виконання тест-кейсів а не створювала звичайний список дій.

Розглянемо результат роботи модуля розпізнавання елементів GUI у вигляді представленого нижче логу.

[log][15:12:54] >>> Test start

[log][15:12:54] >>> App.open C:\Program Files\
Wolfram Research\Mathematica\10.0\Mathematica.exe(5124)

[log][15:12:54] >>> wait for welcome window

[log][15:13:11] >>> search "new document" button

[log][15:13:11] >>> CLICK on L(381,256)@S(0)[0,0 1366x768]

[log][15:13:11] >>> wait for new document window

[log][15:13:11] >>> TestCase: Manipulate actions

[log][15:13:11] >>> TYPE "Manipulate[Plot[A*Sin[f x + p], {x, 0, 6}],
{{f, 1, "Frequency"}, 1, 6}, {{p, 0, "Phase"}, 0, 2*Pi},
{{A, 0.2, "Amplitude"}, 0.2, 0.5}]"

[log][15:13:12] >>> compile the code

[log][15:13:12] >>> (Shift) TYPE "#ENTER."

[log][15:13:12] >>> wait for compilation result

[log][15:13:13] >>> set mouseMoveDelay = 4

[log][15:13:13] >>> search thumb of frequency

[log][15:13:13] >>> DRAG_DROP L(445,185)@S(0)[0,0 1366x768]

[log][15:13:17] >>> set mouseMoveDelay = 0.5

[log][15:13:17] >>> search "open adds" button

[log][15:13:17] >>> CLICK on L(650,185)@S(0)[0,0 1366x768]

[log][15:13:17] >>> search "reverse direction" button

[log][15:13:18] >>> CLICK on L(628,212)@S(0)[0,0 1366x768]

[log][15:13:18] >>> search "slider play" button

[log][15:13:18] >>> CLICK on L(547,212)@S(0)[0,0 1366x768]

[log][15:13:18] >>> wait for 3 sec

[log][15:13:21] >>> CLICK on L(547,212)@S(0)[0,0 1366x768]

[log][15:13:21] >>> set mouseMoveDelay = 4

[log][15:13:21] >>> search thumb of phase

[log][15:13:21] >>> DRAG_DROP L(445,238)@S(0)[0,0 1366x768]

[log][15:13:25] >>> set mouseMoveDelay = 0.5

[log][15:13:25] >>> search "open adds" button
[log][15:13:25] >>> CLICK on L(650,238)@S(0)[0,0 1366x768]
[log][15:13:25] >>> search "reverse direction" button
[log][15:13:26] >>> CLICK on L(628,263)@S(0)[0,0 1366x768]
[log][15:13:26] >>> search "slider play" button
[log][15:13:26] >>> CLICK on L(547,263)@S(0)[0,0 1366x768]
[log][15:13:26] >>> wait for 3 sec
[log][15:13:29] >>> CLICK on L(547,263)@S(0)[0,0 1366x768]
[log][15:13:29] >>> set mouseMoveDelay = 4
[log][15:13:29] >>> search thumb of amplitude
[log][15:13:30] >>> DRAG_DROP L(445,289)@S(0)[0,0 1366x768]
[log][15:13:34] >>> set mouseMoveDelay = 0.5
[log][15:13:34] >>> search "open adds" button
[log][15:13:34] >>> CLICK on L(650,289)@S(0)[0,0 1366x768]
[log][15:13:34] >>> search "reverse direction" button
[log][15:13:35] >>> CLICK on L(628,316)@S(0)[0,0 1366x768]
[log][15:13:35] >>> search "slider play" button
[log][15:13:35] >>> CLICK on L(547,316)@S(0)[0,0 1366x768]
[log][15:13:35] >>> wait for 3 sec
[log][15:13:38] >>> CLICK on L(547,316)@S(0)[0,0 1366x768]
[log][15:13:38] >>> finalize test
[log][15:13:38] >>> search "close notebook" button
[log][15:13:39] >>> CLICK on L(1044,52)@S(0)[0,0 1366x768]
[log][15:13:39] >>> wait save request window
[log][15:13:40] >>> search "don`t save" button
[log][15:13:40] >>> CLICK on L(722,385)@S(0)[0,0 1366x768]
[log][15:13:40] >>> wait exit window
[log][15:13:40] >>> search "quit" button for mathematica
[log][15:13:41] >>> CLICK on L(829,352)@S(0)[0,0 1366x768]
[log][15:13:41] >>> Tect end

Розглянемо детально принцип побудови даного логу. Початковий вигляд логу базувався тільки на повідомленнях із змістом: CLICK on L(628,263)@S(0)[0,0 1366x768] . таке представлення є досить незручним для читання і розбору результатів тестування. Тому бели прийняті наступні нововведення:

- додати вирази «wait for»;
- додати вирази «search»;
- додати іншу інформацію яка полегшує сприйняття результатів;
- показувати час виконання кожної дії.

Отже отриманий лог набуває форми прийнятної для надання вичерпної інформації про графічний інтерфейс програного продукту, який проходить тестування.

Таким чином розпізнавання елементів графічного інтерфейсу користувача надає можливість ефективно провести автоматизацію тестування GUI, забезпечує тестувальника всією необхідною інформацією та надає детальний звіт по працездатності графічного інтерфейсу програми яка проходить тестування.

4.2 Рекомендації по розробці програмного забезпечення орієнтованого на тестування GUI

Варто дотримуватися певних правил при реалізації програмного продукту орієнтованого на тестування шляхом розпізнавання елементів графічного інтерфейсу. Тож будуть дані певні рекомендації щодо цієї теми.

Використовувати концепцію побудовану на екранних об’єктах. Екранні об’єкти представляють собою набір функціоналу який зображений в певний момент часу на екрані. Якщо під час розробки ПЗ виникне ситуація що призведе до розширення функціоналу, таким чином, що в програмі з’явиться

нове вікно з абсолютно новими елементами управління та їх властивостями, то не складе ніяких проблем розширення модуля задля більш ефективного тестового покриття. Такий підхід надає модулю можливості для гнучкості та масштабування при оновленні ПЗ, що тестується до нової версії.

Відповідно до всіх створених класів екранних об’єктів варто створити абстрактний клас під назвою AbstractScreen, який безпосередньо і надає можливість для масштабування всієї системи. До даного класу варто віднести всі загальні властивості його класів-потомків. Якщо розглядати питання з точки зору екранних об’єктів, то це можуть бути, на приклад, час очікування реакції на натискання кнопки чи іншого елемента управління. Також це може бути час очікування безпосередньо вікна, але варто бути обережним і враховувати реальні часові можливості машини, на якій відбувається процес тестування, оскільки може виникнути неприємна ситуація, коли закінчився час очікування вікна, а воно ще не встигло завантажитись через особливості операційної системи, або апаратне забезпечення машини. В такому випадку отримаємо виключення FindFailed, і відповідно провалений тест.

Отримувати доступ до зовнішніх ресурсів програми, використовуючи клас DataProperties. Він завантажує файли що знаходяться в папці на диску та забезпечує універсальний доступ до них. Даними файлами є заздалегідь підготовлені скріншоти, що місять зображення елементів управляння. Крім завантаження і доступу до файлів клас DataProperties надає доступ до файлу конфігурації де зберігається інформація що може бути змінювана, це переважно параметри програми такі як шлях до виконуваного файлу, для запуску програми, час очікування вікна та елемента управлінні (вони відрізняться через довше завантаження вікна). Для цього слід користуватись стандартним для мови Java класом Properties.

Також варто включити клас Driver який надає доступ до об’єкту екрану. Він використовує клас Screen із Sikuli API для отримання доступу до екрану, щоб надалі здійснювати пошук по ньому. Таким чином Driver надає доступ до екрану із будь якої точки програми. Крім того він реалізований по шаблону

проектування Singleton, таким чином, щоб створювати єдиний об’єкт екрану при першому зверненні і повертати цей же об’єкти при всіх подальших зверненнях до даного класу.

Якщо постає питання організацію класів в програмі, то слід використовувати наведену на рисунку 4 ієрархію класів і відповідну організацію по пакетах. Використовується поділ на 3 пакети: screen, test та utility. Пакет screen демонструє логіку роботи програми. Його класи містять методи для роботи з усіма елементами графічного інтерфейсу над якими буде відбуватись процес тестування. Кожен клас відповідає за функціонал певного вікна і містить унікальний набір методів для роботи з елементами які включає в себе це вікно.

Пакет utility містить допоміжні засоби які спрощують та узагальнюють доступ до ресурсів необхідних до тестування. Ресурсами в даному випадку виступають скріншоти вікон і елементів управління графічного інтерфейсу. Доступ до зображень надається класом DataProperties. Ще один клас присутній в пакеті – Driver, він надає монопольний доступ до екрану з будь якої частини програми. Driver організований згідно шаблону проектування singleton, таким чином створюється тільки один об’єкт для доступу до екрану пристрою.

І нарешті пакет test створений для зберігання тестів. При використанні спеціалізованих засобів для тестування, як на приклад TestNG, це є особливо актуальним. Класи цього пакету містять методи для безпосереднього запуску програми та проведення тестування. Вони переважно використовують об’єкти із пакету screen для доступу до функціоналу програмного забезпечення.

Для ефективної і зручної роботи з проектом варто використовувати репозиторій Maven, який дозволяє збирати проект без особливих труднощів із завантаженням необхідних бібліотек.

І наостанок варто надати певні рекомендації по роботі із Sikuli.

1) *Використання осмислених імен зображень;*

Рис. 17 Рекомендація по назві зображень.

Осмислені імена набагато спростять роботу з підтримки тестового коду. Тому варто відразу називати зображення відповідно до їх функціональності, щоб уникнути значних труднощів при розширенні системи тестування.

2) *Використання одного і того ж зображення багатократно;*

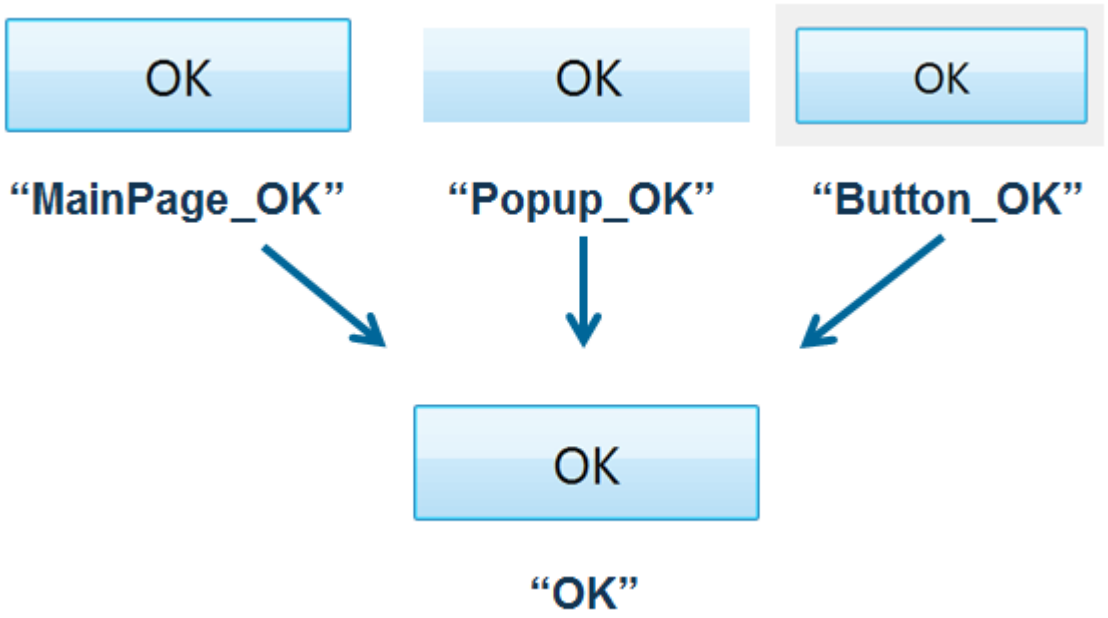


Рис. 18 Рекомендація по використанню кнопок.

Пункт, недотримання якого також веде до проблем з підтримкою системи тестування. Тест відбувається в межах одної операційної

системи, а отже варто потурбуватись про те, щоб в ньому брали участь тільки унікальні зображення.

3) *У випадку помилки FindFailed намагатись використати розпізнавання тексту.*

   Якщо елемент управляння дозволяє розпізнати на ньому певний текст то неодмінно варто цим скористатись при відловлювання виключення FindFailed. Адже, якщо при зміні графічного інтерфейсу змінився зовнішній вигляд елемента управляння. Його назва з високою ймовірністю залишається тією ж самою.

4.3 Напрямки подальшого розвитку досліджень

В перспективі рішення для модуля розпізнавання елементів графічного інтерфейсу користувача можливо розширити до фреймворку який буде більш детально здійснювати тестування. В основу логіки роботи фреймворку буде покладено алгоритм здатний до автоматичного аналізу роботи програми. На основі цього аналізу буде виданий більш детальний звіт про зміни у програмному забезпеченні. Також надана можливість будувати дерево графічного інтерфейсу користувача.

Доцільно буде створити структуру для зберігання інформації про елементи управління програмою. Така структура передбачає створення класу Control для детального опису кожного елемента управління. Об’єкт класу Control буде зберігати:

- зображення елементу управляння;
- регіон екрану, де він був знайдений востаннє;
- текст назви елементу управлянні (якщо такий існує);
- очікувана реакція на взаємодію з елементом управління.

Відповідно до цього варто також створити клас Window, який буде слугувати контейнером для елементів управління. Вся описана структура для зручності доступу може бути збережена в формат JSON. Такий варіант надасть

можливість вільно і швидко одержувати доступ до необхідної інформації, коли вона буде потрібна.

Отже основний алгоритм роботи буде здійснювати два проходи по шляху розпізнавання. Перший прохід буде відбуватись наступним чином: Запуск ПЗ, що підлягає тестуванню. Потім Ініціалізація всіх елементів управління і вікон зазначених у тест-плані. Далі для кожного елемента управління визначаємо його позицію на екрані, якщо виникли збої в пошуку певного елемента управління, все це заноситься в лог виконання тесту. Таким чином відбувається побудова файлу, який повністю зберігає структуру дерева графічного інтерфейсу.

Другий прохід починається з відновлення структури дерева графічного інтерфейсу шляхом зчитування інформації з файлу. Далі проходимо по цьому дереву і перевіряємо чи відповідає запис в файлі дійсності. А також перевіряємо очікувані результати після взаємодії з відповідним елементом управляння. Якщо пошук певного елементу управління в результаті дав виключення FindFailed, то при відловлюванні варто використовувати розпізнавання тексту даного елементу управляння. Адже ймовірно що невдалий пошук графічного елементу спричинила зміна зовнішнього вигляду елементу управління, в такому випадку досить ймовірно що назва елемента управління залишилась незмінною, тож в нагоді стає модуль розпізнавання тексту в Sikuli. Паралельно всі зміни і події будуть заноситись в лог. В кінці другого проходу знайдені зміни будуть записані у файл.

Особливу увагу отримає виведення інформації у вигляді логу подій. Лог може бути перероблений на власний, з блекджеком і шлюхами. Крім цього він може містити окремий функціонал який дозволить підсвічувати знайдені області на екрані.

Сама структура лога може бути змінена в кращу сторону за допомогою оформлення звіту у вигляді html-сторінки. Це дасть зможу швидко орієнтуватись у результатах і в зручному вигляді мати доступ до необхідної інформації. Паралельно з виводом символьної інформації, такої як час

виконання дії, назва дії, значення параметрів активного елемента управління і т.д.. буде виводитись графічна інформація з результатами пошуку зображення у вигляді регіону пошуку і виділеною областю максимального співпадіння регіону до шаблону пошуку. Також до лога варто додати свої власні події такі як виклик вікна чи назви тестів. Це додасть логу чіткості та зрозумілості і покращить його сприйняття тестувальниками.

4.4 Висновок

Отже результатом роботи є покращений лог, який надає можливості детально відслідкувати результати виконання тестів.

У даному розділі також були наведені практичні рекомендації по розробці програм орієнтованих на тестування графічного інтерфейсу користувача базованих на розпізнаванні елементів управління. Дотримання цих рекомендації є обов’язковим при розробці програмного забезпечення даного виду, оскільки вони містять як рекомендації по роботі з графічними об’єктами, так і рекомендації щодо архітектурної складової програми. Значно підвищується рівень масштабованості системи та спрощується процедура підтримки (зміни) тестів у зв’язку із зміною функціоналу.

В цілому ідея розпізнавання є досить перспективною для використання її у широкому виробництві. Тому були представлені напрямки подальшої роботи в цій галузі. Очевидно, що є досить велика площадка для роботи і подальших досліджень в області автоматизації тестування елементів GUI.

5. ОХОРОНА ПРАЦІ І БЕЗПЕКА В НС

Вступ

Робота з програмним забезпеченням, що розроблюється в даній дипломній роботі проводиться у жилому приміщенні з обладнаним необхідними інструментами робочим місцем. Основним елементом обладнання є ПК та встановленні на цьому ПК програмні додатки отже в цьому розділі необхідно розглянути питання охорони праці при роботі з ПК та проаналізувати умови середовища кімнати(мікроклімат, виробничий шум, пожежна безпека та ін.) на предмет відповідності існуючим санітарним нормам. У підсумку ми отримаємо відповідь щодо відповідності даного приміщення цим нормам.

5.1 Характеристика приміщення

Програмний продукт, що використовується в дипломній роботі, використовуватиметься в приміщенні, план якого приведено на рисунку 19.

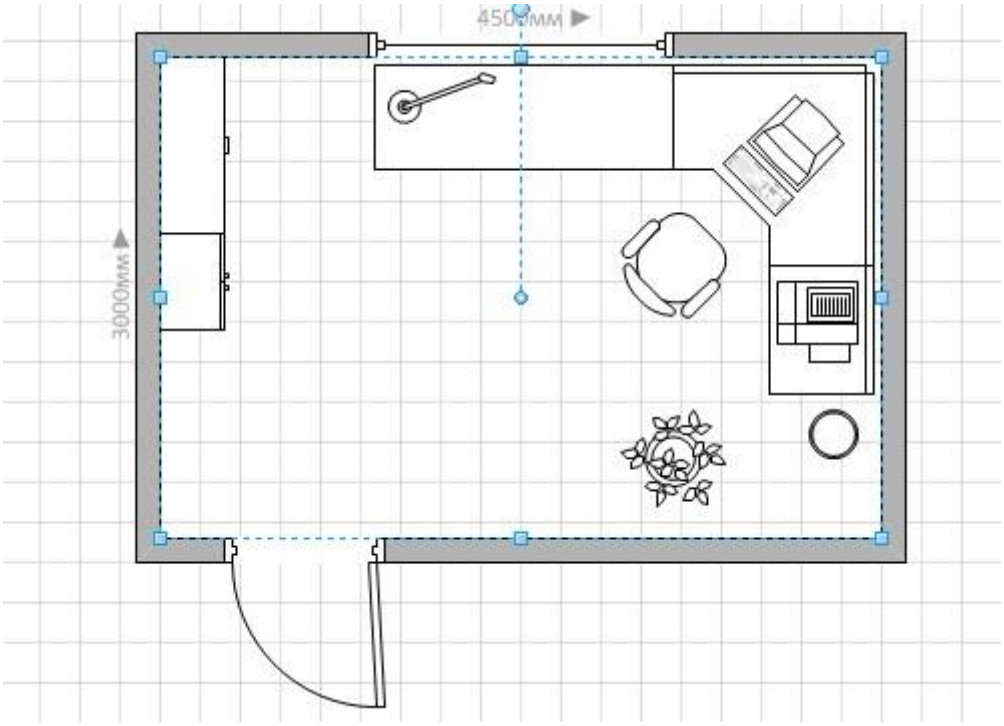


Рис. 19 План приміщення

Приміщення має одностороннє природне освітлення і загальне штучне освітлення. Стіни обклеєні світло-блакитними шпалерами, стеля оброблена білим пластиковим покриттям, підлога вкрита світло-коричневим ламінатом. У приміщенні відсутні сильні вібрації та шкідливі речовини. Склад повітря в нормі.

У кімнаті знаходиться ПК з двоядерним мікропроцесором Intel Core i7 та 21 дюймовим LCD монітором, БФП, а також меблі.

Приміщення має довжину 4.5 м, ширину 3 м, висоту стелі 3 м. Кількість робочих місць - одне. Приміщення знаходиться на третьому поверсі семиповерхової цегляної будівлі. Площа – 13,5 м^2, об'єм – 40,5 м^3. Виходячи з цього, отримаємо дані, наведені в таблиці 1. Нормативні значення згідно [12].

Табл. 3 Фактичні та нормативні значення параметрів приміщення

Параметр	Норма	Реальні параметри
Площа, S	не менше 6 м^2	13,5 м^2
Об’єм , V	не менше 15 м^3	40,5 м^3

Обладнання і організація робочого місця працюючих з ЕОМ мають забезпечувати відповідність конструкцій всіх елементів робочого місця. Висота робочої поверхні складає 80 см, через свою трикутну форму має великий простір для ніг - близько 100 см. Робочий стілець – сучасний стілець з можливістю обертання, регулятором висоти і відкидною спинкою. Відстань від екрану монітора до користувача складає 65-75 см.

Згідно [12], можна зробити висновок, що отримані характеристики приміщення та ергономіки робочого місця відповідають існуючим нормам та вимогам.

5.2 Мікрокліматичні умови

Оскільки основна частина роботи працівника є сидячою, тобто такою, що не вимагає фізичного навантаження, відносимо цю роботу до категорії 1А,

згідно [13]. Джерелами тепла в цьому приміщенні є люди, електроустаткування, освітлювальні прилади в темний час доби і система опалювання взимку. Оператором виділяється до 120 Ккал теплової енергії за годину. Оптимальні та фактичні значення параметрів мікроклімату приведені в таблиці 4.

Табл. 4 Оптимальні та фактичні параметри повітря робочої зони для категорії 1А

Період року	Параметр	Оптимальний	Фактичний
Теплий	Температура	23 – 25 °С	25 °С
	Вологість	40 – 60 %	45 %
	Швидкість повітря	≤ 0.1 м/с	
Холодний	Температура	22 – 24 °С	22 °С
	Вологість	40 – 60 %	55 %
	Швидкість повітря	≤ 0.1 м/с	

Для того, щоб температурні норми постійно витримувались проектом передбачено встановлення у приміщенні кондиціонеру, для збільшення температури повітря у холодний період року та, відповідно, зменшення у теплий період року, якщо це буде необхідно. Норми по відносній вологості та швидкості руху повітря витримуються.

Всі показники задовольняють вимогам зазначеним в [13] для робіт категорії 1А - легка і є задовільними для здоров'я людини.

5.3 Освітлення

Згідно [14] ця робота відноситься до $V_б$ (малої точності) розряду зорових робіт. Передбачається використання природного, штучного і змішаного освітлення.

Природне освітлення здійснюється за допомогою вікна, площа якого складає S = 1,9*2,2 = 4,18 м^2 та являється боковим освітленням.

У світильниках місцевого і загального освітлення використовуються лампи розжарювання потужністю 100 Вт із світловим потоком лампи = 1550лм.

Таким чином освітлення задовольняє встановленим нормам.

5.4 Шум і вібрація

Джерелами шуму в приміщенні є комп'ютер і БФП. Кулери комп'ютера є сучасними і мають низький рівень шуму, так само як і БФП. Згідно технічній документації шум обумовлений кулером в блоці живлення складає 20 дБ, кулером процесора - 35 дБ, загальний, - 35 дБ. Враховуючи незначний рівень шуму від персонального комп'ютера, малий рівень шуму від БФП, незначний рівень фонового шуму від іншого устаткування - сумарний рівень шумового забруднення приміщення не перевищує максимально допустимий рівень коригованої звукової потужності і складає не більше 50 дБ.

При роботі з персональним комп'ютером в робочому приміщенні значення характеристик вібрації на робочих місцях в загальному не перевищує допустимих значень.

5.5 Випромінювання

У приміщенні відсутні інфрачервоні, ультрафіолетові та та електромагнітні випромінювання, бо усі монітори ПК вироблені на основі рідкокристалічної матриці, підсвітка якої здійснюється неоновою лампою, що не має сильного електромагнітного випромінювання і сертифіковані в Україні.

5.6 Організація оптимального режиму праці та відпочинку

При організації праці, пов'язаної з використанням ПК, для збереження здоров'я працівника, запобігання професійних захворюванням і підтримки працездатності передбачаються перерви для відпочинку. Нетривалі перерви повинні передувати появі об'єктивних і суб'єктивних ознак стомлення й зниження працездатності. При виконанні робіт, що належать до різних видів трудової діяльності, за основну роботу з ПК слід вважати таку, що займає не менше 50% робочого часу.

Впродовж робочої зміни мають передбачатися:

- перерви для відпочинку і вживання їжі (обідні перерви);
- перерви для відпочинку й особистих потреб;

У всіх випадках, коли виробничі обставини не дозволяють застосувати регламентовані перерви, тривалість безперервної роботи з ПК не повинна перевищувати 4 години.

При 12-годинній робочій зміні регламентовані перерви повинні встановлюватися в перші 8 годин робота аналогічно перервам при 8-годинній робочій зміні, а протягом останніх 4-х годин роботи, незалежно від характеру трудової діяльності, через кожну годину тривалістю 15 хвилин.

Для зниження нервово-емоційного напруження, стомлення зорового аналізатора, поліпшення мозкового кровообігу, подолання несприятливих наслідків гіподинамії, запобігання втомі доцільно деякі перерви використовувати для виконання комплексу вправ, які наведені у [12].

5.7 Пожежна безпека

У приміщенні знаходяться горючі та важкогорючі рідини, тверді горючі та важкогорючі речовини і матеріали: столи із пластмаси, ПК, монітори, периферія та ін. Отже воно відноситься до категорії В – пожежонебезпечні.

Відповідно до правил розташування електроустановок, приміщення відноситься до П-IIа класу, через то що в приміщенні перебувають тверді та

спалювані речовини та матеріали.

5.7.1 Технічні рішення системи запобігання пожежі

Виникнення пожежі можливо у випадку:

- короткому замиканні в ланках електроживлення.
- порушення правил експлуатації обладнання, що може призвести до запалення або вибуху, які й спричинять пожежу.
- недотримання норм протипожежного захисту, зокрема куріння за межами спеціально відведених для цього місць.

Для запобігання виникнення пожежонебезпечної ситуації передбачається:

- якісна ізоляція усіх струмовідвідних провідників до робочих місць.
- чітке витримування всіх норм протипожежного захисту та правил експлуатації обладнання.
- забезпечення полегшеного теплового режиму.

5.7.2 Технічні рішення системи протипожежного захисту

Конструкція будинку виконана із залізобетонних плит, тобто його конструктивні елемент не спалювані. Будинок відноситься до не спалювані. Будинок відноситься до II ступеня вогнестійкості.

У випадку виникнення пожежі у приміщенні знаходиться вогнегасник (вуглекислотний вогнегасник ВВ-2, ємністю 3 літра згідно нормам) та дотримуються всі нормативні параметри евакуаційного виходу. Ширина коридору, висота до перекриття, ширина дверей у приміщенні та ін. відповідають нормам [15].

Таким чим он можемо зробити висновок що у приміщенні витримуються усі необхідні заходи пожежної безпеки.

5.8 Висновок

Аналіз умов праці в розглянутому робочому приміщенні показав, що умови праці з ЕОМ відповідають вимогам, оскільки площа та об’єм не менше нормативних значень, рівні шуму, вібрації і загазованості не перевищують нормативних обмежень. Ергономіка робочого місця і режим зорової роботи задовольняють встановленим вимогам, організація оптимального режиму праці та відпочинку сприяє зниженню втоми та підвищенню продуктивності робітника.

Для підтримання параметрів мікроклімату в приміщенні встановлено радіатор центральної водяної системи опалення, що складається з 4 секцій та кондиціонер. У приміщенні передбачені система запобігання пожежі та система протипожежного захисту.

ВИСНОВКИ

В результаті виконання даної роботи була здійснена розробка модуля розпізнавання, базованого на пошуку графічний елементів. Дана методологія може бути активно використана у автоматизації тестування графічного інтерфейсу користувача для будь якого ПЗ. Основною перевагою такого підходу до тестування є незалежність від платформи та технологій створення GUI.

Тема розпізнавання зображень є досить цікавою і представляє собою широке поле для подальших досліджень в галузі автоматизації тестування ПЗ. Тому в роботі були визначені ключові вимоги до засобу автоматизації тестування. На основі проведеного аналізу встановлено засіб автоматизації, що є найбільш оптимальним для виконання задачі розпізнавання елементів GUI.

Також було розглянуто деталі загального алгоритму пошуку зображень, розкриваються основні можливості обраного засобу по роботі із зображеннями, висвітлені ключові функції цієї системи та наведені практичні рекомендації по покращенню ефективності пошуку зображення на екрані.

Розробка модуля розпізнавання відбувається відповідно до зарані розробленого тест-плану. Для демонстрації роботи модуля розпізнавання був створений тест-план, а в якості системи тестування була обрана програма Wolfram Mathematica 10.

Далі були наведені всі деталі реалізації програмного продукту тестування GUI від архітектури побудови модуля до прикладів використання Sikuli.

Представлені практичні рекомендації по розробці програм орієнтованих на тестування графічного інтерфейсу користувача базованих на розпізнаванні елементів управління, а саме використання екранних об’єктів, формування окремої бази зображень та організація доступу до них. Було показано, що дотримання цих рекомендації при розробці програмного забезпечення даного виду значно підвищує рівень масштабованості системи та спрощує процедуру

підтримки ПЗ у зв’язку із зміною функціоналу. В роботі присутні як рекомендації по роботі з графічними об’єктами, так і рекомендації щодо архітектурної складової програми. Крім того присутні рекомендації з охорони праці.

Результатом роботи є готовий до використання модуль розпізнавання, який задовольняє вимогам поставленим до системи розпізнавання елементів GUI, та повністю відповідає до цілей наведених у роботі.

В цілому ідея розпізнавання є досить перспективною для використання її у широкому виробництві. Тому представлені напрямки подальшої роботи в цій галузі у вигляді розробки фреймворку тестування. Його засоби надаватимуть можливості формування прекрасного html - логу, підкріпленого графічними елементами та областю пошуку на екрані. Розширений функціонал автоматизації тестування, який дозволить тепер слідкувати за змінами в графічному інтерфейсі під час розробки ПЗ. Наведені думки свідчать про досить велику площадку для роботи і подальших досліджень в області автоматизації тестування елементів GUI.

Розпізнавання елементів графічного інтерфейсу з метою автоматизації тестування набирає все більших обертів в сучасному виробництві ПЗ, і має всі можливості для того щоб стати основним методом автоматизації тестування GUI.

ПЕРЕЛІК ПОСИЛАНЬ

1. Mohammad Rafi Automated Software Testing. A Study of State of Practice / Dudekula Mohammad Rafi & Kiran Moses // School of Computing Blekinge Institute of Technology – Sweden - 2010. – С. 5
2. Ranorex .NET Documentation – Режим доступу: http://www.ranorex.com/Documentation/Ranorex/ - Дата доступу: 04.06.1015.
3. Katam Reddy Picture-Driven Computing / Katam Reddy//Information Inc.,- 2010 - Bethesda, Maryland, USA.
4. Kenneth R. Virtual Network Computing/ Tristan Richardson , Kenneth R. Wood and Andy Hopper// Quentin Stafford-Fraser - 1998. – С. 43
5. T-Plan Robot Overview Documentation – Режим доступу: http://www.t-plan.com/robot/index.html - Дата доступу – 24.05.1015.
6. Template matching algorithm – Режим доступу: http://docs.opencv.org/doc/tutorials/imgproc/histograms/template_matching/template_matching.html?highlight=matchtemplate - Дата доступу – 23.05.1015.
7. Галатенко Д.В. Адаптивний фільтр для правильної локалізації підзображень, / Галатенко Д.В. // Системний аналіз та інформаційні технології : «САІТ-2015», 23–25 червня 2015, Київ, Україна : матеріали. – К. : НТУУ «КПІ», 2015. – С. 228.
8. Normalization in Template matching algorithm – Режим доступу: http://werner.yellowcouch.org/Papers/subimg/index.html - Дата доступу – 01.06.1015.
9. SAD – Sum of the Absolute Differences – Режим доступу: https://siddhantahuja.wordpress.com/tag/sum-of-absolute-differences-sad/ Дата доступу – 03.06.1015.

10. Wolfram Language Documentation Center Introduction to Manipulate – Режим доступу: https://reference.wolfram.com/language/tutorial/IntroductionToManipulate.html - Дата доступу – 03.06.1015.
11. SikuliX API documentation – Режим доступу: http://nightly.sikuli.de/docs/index.html Дата доступу – 05.06.1015.
12. Державні санітарні правила і норми роботи з візуальними дисплейними терміналами електронно-обчислювальних машин [Текст] ДСанПіН 3.3.2.007-98.
13. Санітарні норми мікроклімату виробничих приміщень [Текст] : ДСН 3.3.6.042-99. – К., 2000.- 16 с.
14. Природнє і штучне освітлення : ДБН В.2.5-28:2006. – [Чинний від 2006-10-01]. – К. : Міністерство будівництва, архітектури та житлово-комунального господарства України, 2006. – 68 с. – (Національні стандарти України).
15. Захист від пожежі. Пожежна безпека об’єктів будівництва. ДБН В.1.1.–7–2002. – К. : Держбуд України, 2003. – 36 с.