# Automation Testing for Order to Cash Process in Microsoft Dynamics 365

Arjun Vishnu Roy[1*], Shahid Ali[2]

[1] AGI Education Limited and 190 Great South Road Epsom, Auckland, New Zealand, Indian.
[2] AGI Education limited and 190 Great South Road Epsom, Auckland, New Zealand, New Zealand.

* Corresponding author. Email: shahid@agi.ac.nz
Manuscript submitted August 31, 2019; accepted October 31, 2019.
doi: 10.17706/jsw.14.11.548-558

**Abstract:** This discusses the software automation testing of Microsoft Dynamics (MSD) 365 in finance and operations modules. To perform automation testing, order to cash process was selected. The aim of this research is to build an effective testing framework for order to cash process in MSD 365. A test for order to cash process was designed to do the automation testing in MSD. In this research, Agile-based Scrum method was followed. Adoption of Agile methodology is beneficial for whole team and daily stand-up meetings in agile methodology helped in raising the issues. Daily meetings with the development team have helped in getting valuable feedback on changes that needs to be incorporated for improving the quality of automation testing script. This approach helped us in reducing the overall time for implementation the code and leading to increase work productivity. However, the main contribution of this research is the proposed solution to overcome some of the key challenges faced during the automation of web application. The second main contribution of this research is the automation testing for order to cash process in MSD 365 which could be beneficial for organizations. This will help organizations in reducing the effort of manual testing and the overall time needed for test execution.

**Key words:** Microsoft dynamics (MSD) 365, regression, automation testing.

## 1. Introduction

Technology is being growing rapidly over the years, the rapid growth in the technology and the large volume of information have led many organizations to adopt IT systems to manage their daily business activities. With the rise of competition in the market, a need was arisen to increase the sales of the organization as well as maintaining the existing customers base. As a result, most of the small and medium-sized enterprises were looking for IT solutions that can help them to manage their sales process effectively. Currently, two major web applications that cater to ERP based solutions are MSD 365 and Salesforce. Both applications have several capabilities like call scripting, automated routing, automatic-response and email marketing etc. However, MSD 365 has the edge over Salesforce when it comes to power Business Intelligence (BI), sales tracking, and reporting as these features are not available in Salesforce. In this research, Contoso has been chosen as a demo company in MSD 365 Enterprise Resource Planning (ERP). The company facilitates enterprise resource planning capabilities of MSD 365 for sales, warehouse management, purchase and inventory, etc. Company provides industrial solutions in different areas like Retail, Marketing, Customer service & Finance and Operation

etc. This research work was carried out to give a testing solution to improve product quality for Small and Medium Enterprises (SMEs), who are using MSD 365 for order to cash process to manage their business processes. The major challenge is that there is no ready-made solution available to perform automation testing for order to cash process in MSD 365. In order to cope with said challenge a testing framework is build that can be extended to small and medium-sized enterprises for testing the business process and thus ensuring that order to cash process are well managed.

This research paper is organized as follow: Section 2 focuses on the literature review of the automation testing. Section 3 is focused on the research methodology. Section 4 explains the architecture of the automation framework and provides discussion on the results of execution time reports. Finally, in section 5 conclusion to the research is provided.

## 2. Literature Review

Several techniques and methods of software automation testing have been reported in literature [4]. Currently, Scrum methodology is widely applied and most studied topic in software development research domain. In order to carry out this research, Scrum methodology is followed that has identified two key benefits namely increased performance and fault detection [1]. Scrum is flexible to in terms of testing that carried out in each iteration, which help in detecting the bugs in the early stages of the software development life cycle. Scrum was helpful for this research to improve the code quality and reduce the severity of faults at later stages. Scrum approach reduces the cost of fixing bugs as well as reduces the overall cost of the project. In addition, the daily stand-up meetings provide an opportunity to discuss the issues faced while performing automation testing. These meetings with the developers are helpful in getting valuable feedback on the changes that must be made on the automation test scripts so that an effective testing framework can be implemented. Implementing this methodology helped in improving the team collaboration and consequently increased the work productivity.

Scrum was compared with the traditional approaches in software development life cycle [2]. Explanation was provided in the contest how development and testing are carried out in parallel in Scrum compared to the waterfall approach in which testing only starts when the development is complete. In our research, the adopted approach is Scrum because the testing must go together with the development phases. The adopted approach is a major departure from the traditional approach.

Next major decision in the research was the selection of most suitable testing tool. There are several tools available in the market that can used to perform automation testing. Some of the most prominent test automation tools used by most organizations are Selenium and Quick Test Professional (QTP) [3]. Both tools support platforms such as android and IOS and can also be integrated to several to several plugins that are available in the market. However, in this research we implemented selenium web driver for performing automation testing. One of the main reasons for selecting this tool is that it is open source and provides OS support on multiple platforms like MAC OS X, Linux and Windows. However, QTP requires HP dedicated licensing which is expensive and runs only on Windows XP/7/8 (no other operating system is allowed).

For the implementation of the automation scripts, the design pattern adopted is Page Object Model (POM). Some of the pros of using POM is that all the web elements can be stored in an object repository that holds all the Web elements of each webpage. The POM uses page factory class for creating the object repository where all the web elements will be stored. All the methods and objects will be defined for each webpage that allows easier maintenance of code. Implementation of POM in this research has

helped in improving the reliability, reduction in code maintenance, allowing re-usability of the code and changes that only need to be done on the page factory class if the locator changes [6].

In this research, the framework selected for implementing the automating test scripts is TestNG. Some of the key benefits of using TestNG is that test cases can be easily parametrized, and it supports multiple annotations. Implementing TestNG framework in this research helped in generating test reports. These test reports provide a detailed analysis of the status of test execution. These tests include the number of test cases passed, skipped and failed in a test suite [7].

An approach was proposed for reducing overhead of running Selenium based load tests [9]. The performance overhead was reduced by sharing the browser instances between test user instances leading with the launch of many browser instances during the execution of a test. The results of this study showed that the proposed approach can increase the number of instances dramatically which are tested on the machine without overloading the load driver. Eventually this approach leads to the improvement of efficiency of their own Selenium based load tests, which is a great help for software practitioners.

Selenium configuration was analysed using different web browsers and time out strategies (waits) in order to test stability and the runtime performance [10]. There were hardware and software considerations as well in this research. Results of this research showed that HtmlUnit with Thread waits resulted in the lowest number of test failures and produced the optimum runtime on poor performance hardware. The outcome for this research lead to faster teaching application along with a recommendation to how configure Selenium applications to large electronic health record system like iTrust2 that operates in continuous integration.

During the research execution, several challenges were encountered which were related to automation testing. A quick look at the literature sources revealed that some of these challenges with specific reference to web-based applications are reported in [8]. Some of these include dynamically changing class names, control and synchronization issues between tool and application under test.

## 3. Research Methodology

This section describes the automation test plan adopted for Order to Cash process. After this a brief description is provided on how order to cash process is carried out in MSD365. Following this it also covers how POM architecture is implemented in this project and further java code used for this project is discussed in the proposed methodology for this project.

### 3.1 Automation Test Plan for Order to Cash Process in MSD365

This section outlines some of the key test plan components adopted for this research. The automation test plan for the research is summarized in Table 1.

Table 1. Components of the Test Plan

| S. No | Test Plan Components | Description |
|---|---|---|
| 1 | Tool | The tool that will be used for automating MSD 365 application is Selenium WebDriver |

| | | |
|---|---|---|
| 2 | Selection of Test scenarios & plan | Ten test scenarios are selected for performing automation and all these scenarios will run based on the defined priority |
| 3 | Total Time | Eight working days (64 hours) for developing the automation test scripts |
| 4 | Resources | One automation tester |
| 5 | Maintenance Effort | 15 hours for maintenance of automation scripts |
| 6 | Environment | Environment- IE browser, windows 10, Eclipse, Selenium libraries |
| 7 | Reports | TestNG will be used for report generation. Reports will be sent in HTML format |
| 8 | Return on investment | Time needed for running the automation scripts will be 5 minutes. Time needed for performing manual testing is more than 30 hours. Hence automation will save a lot of effort, time and money in the later stages of Software development life cycle. |

We have designed the automation test plan now we will proceed towards order to cash process in MSD365.

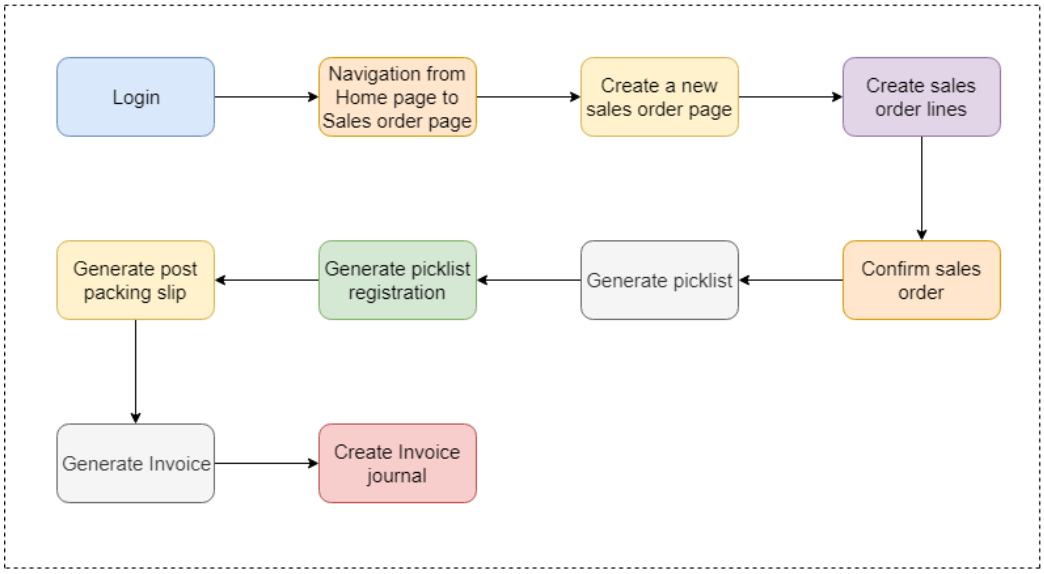


Fig. 1. Order to cash process in MSD.

## 3.2 Order to Cash Process in MSD365

In the MSD365 process named “order-to-cash”, the first step is logging into application by entering the valid credentials. After the login part is successful, then navigation process is done from the home page to the new sales order. Following this, a new sales order is created by entering the customer account. After the sales order creation, the sales order lines are updated by entering the item number and saving it. Then, a picklist is generated which is sent to the warehouse for picking and packing the products. Once picking and packing process is completed, the picklist register is updated. Then, the post packing slip is sent from the warehouse team to the customer service. On receiving the packing slip, the customer service team generates an invoice. After the invoicing process is complete, the invoice journal

is printed and sent to the customer. Fig. 1 depicts step-by-step how order to cash process is carried out in MSD 365.

### 3.3 Page Object Model (POM) Architecture for Order to Cash Process in MSD365

In this research, page object model was implemented for developing the automation scripts. In the architecture of the automation framework, there was a main TestNG class that had all the ten test scenarios of the order to cash process in MSD 365 application. The object repository held the page factory classes of all the web pages. In this architecture, all the page factory classes were imported into the TestNG main class for performing execution. During execution phase, all the test scenarios were run based on the priority set in the automation test scripts. After the completion of test execution, all reports were generated in HTML format. This is shown in Fig. 2.

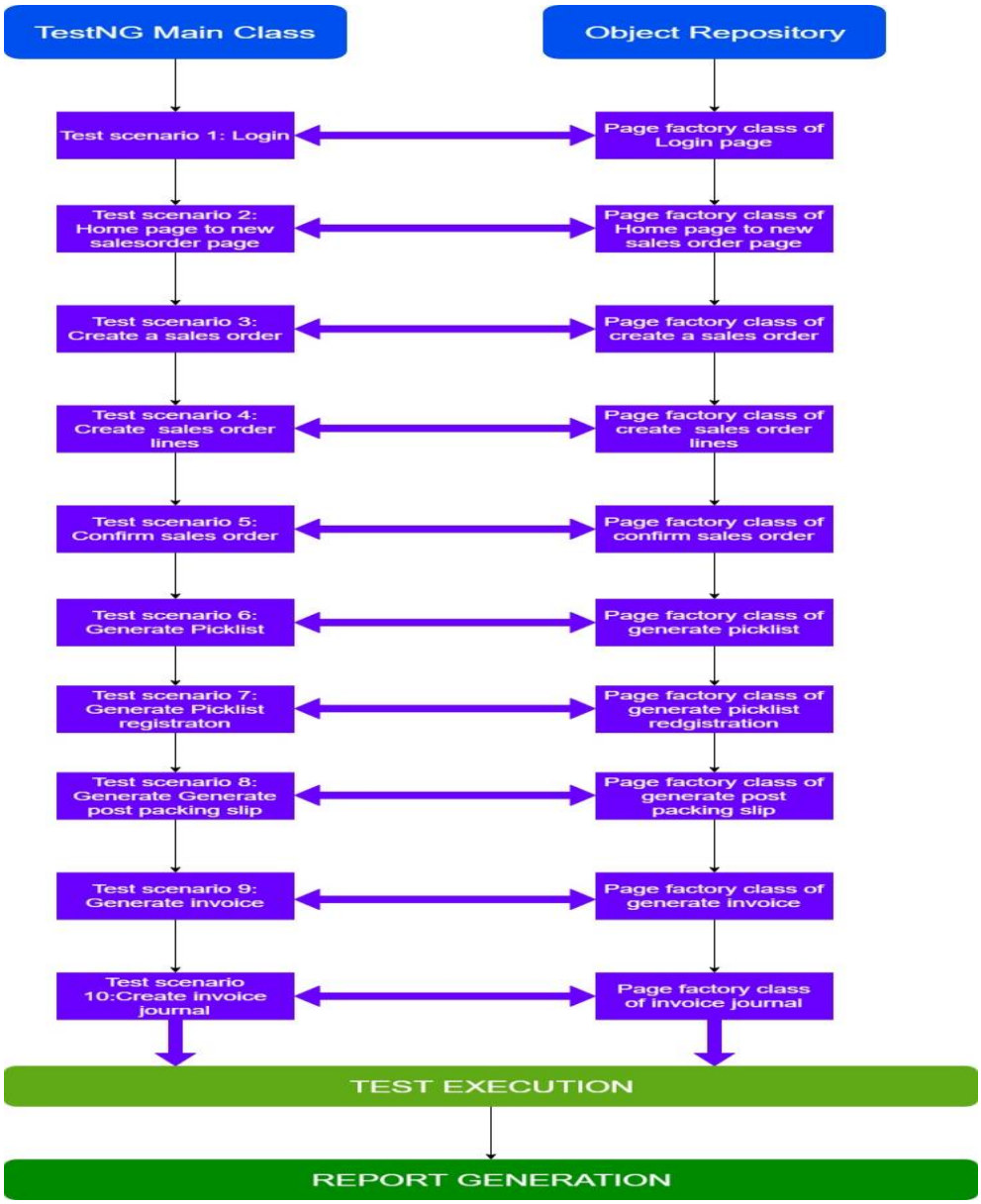


Fig. 2. POM architecture.

### 3.4 Java Code Snippets

This section presents some of the code snippets implemented in Eclipse for test execution. In this

section a brief explanation is given on how the main TestNG class were implemented. Following this a brief description is given of how the page objects and methods were managed using this framework.

### 3.4.1 Main testng class

This snippet depicts the main TestNG class for order to cash process in MSD 365. In the figure, annotation '@Test' shown in the picture explains all the test scenarios that are implemented in this research. Priority is set for all test scenarios so that it executes based on the values that are defined in it. The main method of each test shows how page factory classes are imported into the main TestNg class and how the sub methods are called in the main method of the class. In figure below 'login' is the main method and 'signin' is sub method to perform actions like sending username, password and click signin button. The screenshot of the main TestNG class is presented in Fig. 3.

```
@Test(priority=0, enabled=true)
public void login()
{
   Login_page signin= PageFactory.initElements(driver,Login_page.class);
   signin.User_name("vishnu@metcashpoc.onmicrosoft.com");
   signin.Next_button();
   signin.Password("Davidbilla123");
   signin.Signin_button();
   signin.stay_signed_in_button();
}
@Test(priority=1, enabled= true)
public void Homepage_to_New_SO_page()
{
    Page_objects.Homepage_to_New_SO_page Navigate=PageFactory.initElements(driver, Page_objects.Homepage_to_New_SO_page.class);
    Navigate.Hamburgericon();
    Navigate.Accounts_Receivable();
    Navigate.All_sales_order();
    Assert.assertTrue( Navigate.Newbutton.isDisplayed());
}
```

Fig. 3. Java code snippet depicting Main TestNG class.

### 3.4.2 Object repository

This java code snippet depicts how page factory class is used for storing the elements of the sales order lines webpage. In this class '@find by' is used for finding the elements in the web page. CSS selector "NAME " is used for locating the web element in the application. After locating the values, they are stored in a web element. The screenshot of the object repository is presented in Fig. 4.

### 3.4.3 Page object methods

This Java snippet code depicts how methods are implemented in the page factory class. In this method it explains various actions like 'moveToElement' which is used for moving the focus to a particular location, "WebDriverWait" which is a wait command that waits for the specified time defined before performing the next set of actions, "Click()" which performs the click function in the application and "sendkeys" which sends values in text field. The screenshot of page object methods is presented in Fig. 5.

```java
package Page_objects;

import org.openqa.selenium.WebDriver;

public class Salesorder_lines
{
WebDriver driver;
public Salesorder_lines(WebDriver driver)
{
    this.driver=driver;
}

@FindBy(how=How.NAME, using="SalesLine_ItemId")
WebElement Item_number;
@FindBy(how=How.NAME, using="SalesLine_SalesQty")
WebElement quantity;
@FindBy(how=How.NAME, using="SystemDefinedSaveButton")
 WebElement save_button;
```

Fig. 4. Java code snippet depicting object repository for Sales order lines

```java
public void Item_number(String item_number)
{
    Actions action=new Actions(driver);
    action.moveToElement(Item_number).build().perform();
    WebDriverWait wait=new WebDriverWait(driver, 20);
    wait.until(ExpectedConditions.elementToBeClickable(Item_number));
    Item_number.sendKeys(item_number);
    Reporter.log("Item number filled in successfully");
}
public void Quantity(String Quantity1)
{
    Actions action=new Actions(driver);
    action.moveToElement(quantity).build().perform();
    WebDriverWait wait=new WebDriverWait(driver, 20);
    wait.until(ExpectedConditions.elementToBeClickable(quantity));
    quantity.sendKeys(Quantity1);
    Reporter.log("Quantity filled in successfully");
}

public void Save_Button()
{
    Actions action=new Actions(driver);
    action.moveToElement(save_button).build().perform();
    WebDriverWait wait=new WebDriverWait(driver, 20);
    wait.until(ExpectedConditions.elementToBeClickable(save_button));
    save_button.click();
    Reporter.log("Save button clicked successfully");
}
```

Fig. 5. Java code snippet depicting methods for sales order lines.

## 4. Results

This section covers architecture of the automation framework that was developed for order to cash process in MSD 365. Later in this section test reports that were generated using this framework were also discussed. This includes test execution status report of all the test scenarios that were executed. Following this, a brief analysis is carried out on reporter logs that were generated using this framework. Finally, a discussion is made on the execution time reports which were analysed in detail.

### 4.1 Architecture of Automation Framework for Order to Cash Process in MSD365

In this research, an automation framework was developed to manage all the test scenarios of the order to cash process in MSD 365. This framework explains how architecture of this research was designed so that test execution can be managed effectively. Figure 6 explains how the framework was implemented in this research. In this framework, TestNG is the main class that holds all the ten functional test scenarios. The page factory class holds all the objects and methods of each webpage in the application. All the test case was run from the selenium WebDriver. During execution, the IE browser opens and starts executing the automation scripts on the MSD365 application. Once the execution is complete, all the test reports are generated in TestNG main class. The architecture of the automation framework is shown in Fig.6.

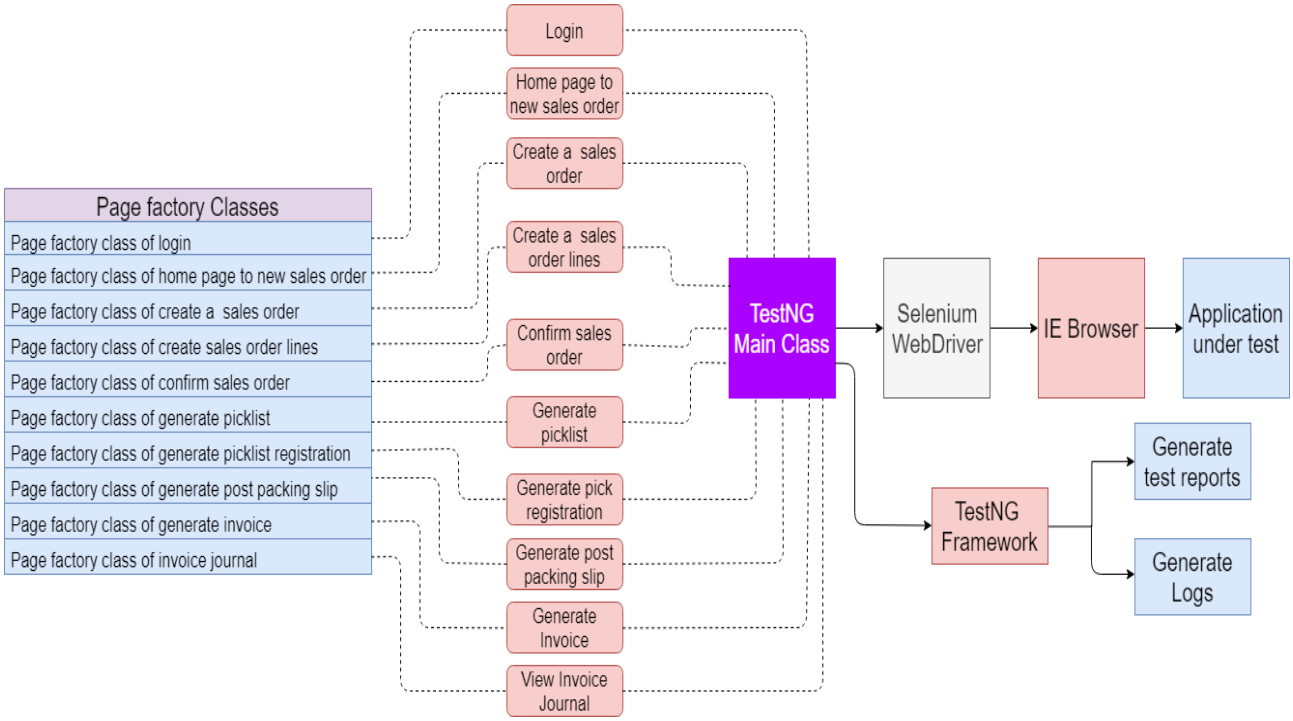


Fig. 6. Test automation framework.

## 4.2 Test Execution Status

In this research, a total of ten functional test scenarios were automated for order to cash process in MSD 365 and each test scenario in this process contained more than five functional test cases. The test execution status holds the information such as number of test cases passed, failed or skipped in a regression suite. It also displays the total time taken for executing the regression test suite. This figure depicts the status of test execution of all the ten functional test scenarios. From the execution results all the ten scenarios are passed, and the total execution time is 327892 (ms). The test execution status report is presented in Fig. 7.

| Test | # Passed | # Skipped | # Failed | Time (ms) | Included Groups | Excluded Groups |
|---|---|---|---|---|---|---|
| **Default suite** | | | | | | |
| Default test | 10 | 0 | 0 | 327,892 | | |

Fig. 7. Test execution status.

### 4.3 Log Report

In this research, logs report for logging the results of the test scenarios were used. The log report provides information about the test steps or any failures that occur during the test execution. This below figure depicts the logs of the test execution.The log report explain the various actions that were performed successfully in the web application for each test scenario.To exemplify,in the test scenario, 'Homepage to New SO_page' all the buttons like 'hamburgericon', 'Accounts receivable' and 'All sales order' were clicked successfully. Details of the test execution status report is presented in Fig.8.

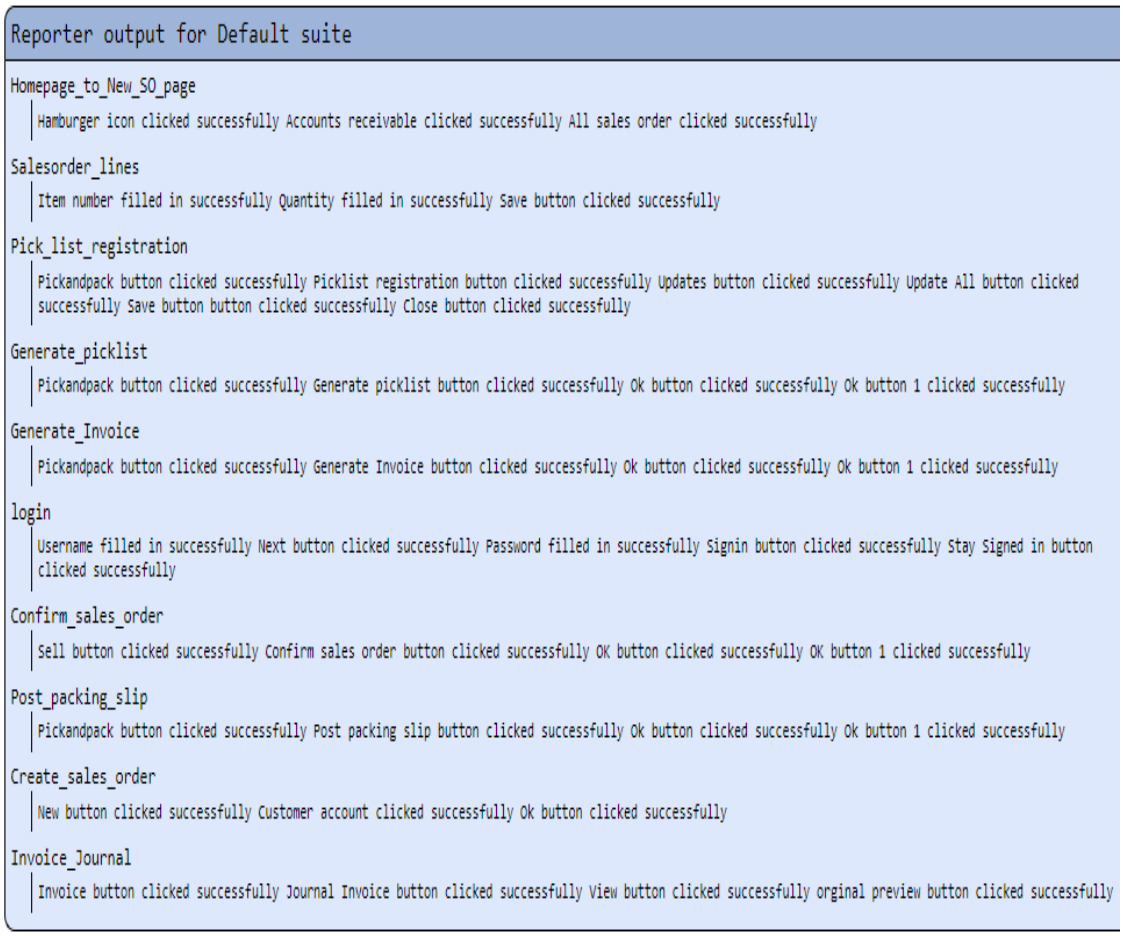


Fig. 8. Log reports.

### 4.4 Execution Time

In this research, execution time reports were generated using the TestNG framework for finding total time needed for test execution. The execution time report provides the information about all the test scenarios that were executed for order to cash process and the time taken for executing each scenario. Details of Execution time is presented in Fig. 9.

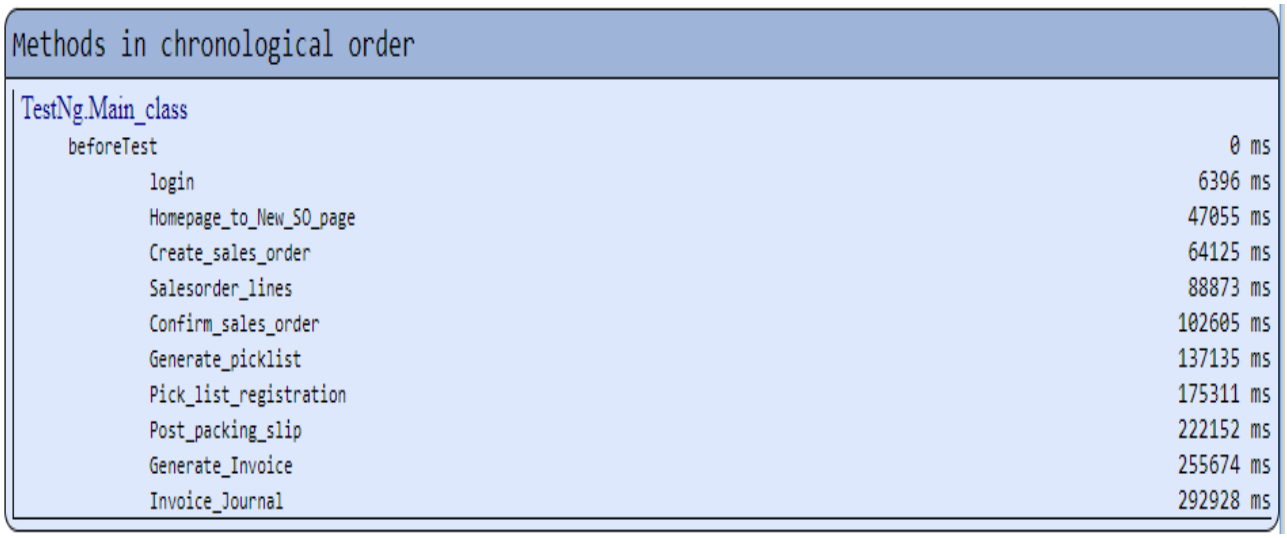


Fig. 9. Execution time.

### 4.5 Recommendations

This section describes some of the key recommendations for automation testing in MSD365 which is explained as follows.

- One of the key challenges faced while automating the MSD365 application was the dynamically changing web elements that failed the test cases execution. However, if static ID's could be implemented in the application, then it will be easier to execute the automation scripts and subsequently increase the test coverage.
- Implementing parametrization using XML/Data provider in the framework will lead to improving the quality of automation script as well as improving the reusability.

## 5 Conclusion

Building an effective testing framework design requires detailed planning and effort. In order to achieve this, the framework should be designed and developed accurately. In this research, an automation framework was developed to test order to cash process in MSD 365 finance and operations module using Selenium WebDriver. In order to test the order to cash process the proposed framework saved a lot of time that is needed for writing test cases. By adopting this framework, one can easily generate customized testing reports. These reports can be used to analyse the status of the test execution. Moreover, this framework can also be helpful for organizations who are planning to adopt MSD365 for managing sales order process. The proposed framework can also be helpful in reducing the overall time needed for test execution compared with manual testing which needs more time for executing the test cases.

## References

[1] Kumar, G. & Bhatia, P. (2012). Impact of agile methodology on software development process. *International Journal of Computer Technology and Electronics Engineering*, 2249-6343.

[2] Leau, Y. B., Loo, W. K., Tham, W. Y., & Tan, S. F. (2012). Software development life cycle AGILE vs traditional approaches. *Proceedings of the International Conference on Information and Network Technology*, *37(1)*, 162-167.

[3] Rahate, S. R., & Bhave, U. (2016). A survey on test automation. *International Journal of Innovative Research in Computer*, *4(6)*, 12361-12372.

[4] Rodrigues, A., & Dias-Neto, A. (2016). Relevance and impact of critical factors of success in software test automation lifecycle: A survey. *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing (SAST)*. ACM.

[5] Jamil, M. A., & Arif, M. (2016) Software testing techniques: A literature review. *Proceedings of the 6th International Conference on Information and Communication Technology for The Muslim World*, Malaysia.

[6] Nishmitha., G. C., & Phaneendra, H. D. (2016). Implementation of page object model in selenium automation. *International Journal of Advanced Research in Computer and Communication Engineering*, *5(5)*, 446-448.

[7] Gojare, S., Joshi, R., & Gaigaware, D. (2015). Analysis and design of selenium webdriver automation testing framework. *Procedia Computer Science*, *50(1)*, 341-346.

[8] Duan, Y., Edwards, J. S., & Xu, M. X. (2005). Web-based expert systems: Benefits and challenges. *Information and Management*, *42(6)*, 799-811.

[9] Shariff, S. M., Li, H., Bezemer, C. P., Hassan, A. E., Nguyen, T. H., & Flora, P. (2019, May). Improving the testing efficiency of selenium-based load tests. *Proceedings of the 14th International Workshop on Automation of Software Test* (pp. 14-20).

[10] Presler-Marshall, K., Horton, E., Heckman, S., & Stolee, K. T. (2019, May). Wait wait. No, tell me: Analyzing selenium configuration effects on test flakiness. *Proceedings of the 14th International Workshop on Automation of Software Test* (pp. 7-13). IEEE Press.

**Arjun Vishnu Roy** has completed his bachelors in electronics and communication engineering from 2010-2014 in India. Following this his worked as a software test engineer at Espina software solutions from 2014-2018. He completed his graduate diploma in software testing from AGI Education Limited, New Zealand. Currently he is pursuing his career as test analyst at Independent Advisory Services (IAS) Limited, New Zealand.

**Shahid Ali** is a senior lecturer and IT program leader at AGI Education Limited, Auckland, New Zealand. He has published number of research papers on ensemble learning. His expertise and research interests include machine learning, data mining, ensemble learning and knowledge discovery.