Performance Evaluation of Graphical User Interfaces in Java and C#

Hassan Bapeer Hassan
College of Medicine,
University of Duhok,
Duhok, Kurdistan Region, Iraq
hassan.bapeer@uod.ac.

Qusay Idrees Sarhan
Department of Computer Science,
College of Science, University of Duhok,
Duhok, Kurdistan Region, Iraq
qusay.sarhan@uod.ac.

***Abstract*—With the growing usage and migration of interactive applications and systems into today's environment, the significance of the graphical user interfaces (GUIs) increases as they act as the gates into using systems efficiently. Therefore, extensive efforts are spent to enhance the usability of the GUIs. However, most of the works focus on testing the functional properties of applications' GUIs rather than non-functional features such as performance. For this reason, it is worthwhile to assess the performance of various GUI components at runtime. In this paper, the most popular programming languages used to create GUI based applications namely, Java and C# were compared experimentally to evaluate their performance in terms of the creation and manipulation of GUI components/controls. The experimental results of this study, which is based on 32 testing scenarios, showed that Java outperformed C# in all test scenarios. This might be because of the "HotSpot" performance engine that Java uses. This study is useful for developers to get insights into the performance of different GUI components provided by different programming languages. Also, it helps them to choose the right programming language for their GUI based applications hence enhance the overall applications' performance and user satisfaction.**

Keywords— Graphical user interface (GUI), GUI programming, GUI testing, Performance evaluation.

I. INTRODUCTION

Graphical user interfaces (GUIs) are the gates to interactive software systems that deal with user's commands/requests. A GUI consists of five main parts: controls (also called widgets or components), forms (also called windows or frames), menu bars, layouts, and interactions [1]. Users send requests through the usage of the GUI to the application logic, which will be processed and replied with some results on the screen. With the rapid advances in technology, the end-users are expecting a better performance of the entire system [2]. Because the GUI takes a larger amount of overall application size, and its validity affects the correctness of the entire application [3]. Also, the GUI part of most software is continuously updated and modified more than the core logic itself. Moreover, The GUI is mostly used by the end-users. Thus, the users' observation of the entire system's performance depends strongly on the performance of the GUI [4][5]. Therefore, the GUI-level testing is an essential method of quality assurance for software systems; it is primarily used to find the errors the end-users face [6]. A common way to test GUIs is to use record and playback systems which are expensive and tedious. Another method is to write scripts using GUI testing libraries that allow tests to interact with GUI widgets. This approach also is a slow process as each test has to be manually written [7]. Profiling and coverage are also used to evaluate the performance of GUIs. They are based on dynamic evaluation which captures the software behaviors at runtime [5]. The more practical and efficient way is automatic unit testing. However, when the GUI level is evolved, it becomes a difficult and complex process [8]. As there is a direct connection between the GUI and the logical part of any software system. In some cases, a delay due to bugs, failures or longer computations might adversely affect the GUI response time. For these reasons, evaluating the performance of the GUI is vital to help developers maximize the quality of products provided to the end-users and keep them engaged. In this paper, the two most popular GUI programming languages namely, Java [9] and C# [10] have been experimentally evaluated with each other to calculate their GUI performance at runtime. The results of this paper will give developers some insights about their performance and help them to choose the effective GUI programming language. Besides, it will help enterprise companies and researchers to focus on enhancing and reducing the existing latency in their GUI components.

The rest of this paper is structured as follows: Section II presents briefly the related work for this study. Section III describes in detail the testing methodology used in this paper. The experimental results and outcomes are shown in Section IV. Finally, the conclusions and some possible future directions are provided in Section V.

II. RELATED WORK

Several studies such as [11-16] conducted general research on introducing, exploring, and evaluating the performance of several GUI frameworks and Application Programming Interfaces (APIs). The authors in [17] conducted a comparative study on the performance of procedure-oriented languages such as C and Fortran and object-oriented programming languages like C# and Java. Object-oriented programming languages outperformed procedure-oriented. In their study, they compared the business logic, not the GUI. However, the performance evaluation process of the GUIs is not an easy task due to the users' non-deterministic behavior [18]. The GUI testing could be conducted as automatic, partially automatic, or manual. Several papers introduced automatic GUI performance testing approaches. For instance, the authors in [4] assessed the performance of two Java GUI frameworks namely, Swing and Thinlet. They measured the latency, CPU, and memory consumption of events handling at run-time. The authors in [18] and [19] evaluated the performance of the Java GUI using capture and replay tools that record and replay the interactive sessions of an application several times. They also revealed that

978-1-7281-5249-3/20/$31.00 ©2020 IEEE

performance measurements produced by the capture and replay tools are different from the sessions conducted manually by users. Additionally, their automated approach helps in conducting a comparison of GUI performance over different software versions, which is called automated performance regression. The authors in [5] investigated different profiling approaches for Java graphical frameworks, namely SWT and JFace. The previously stated studies were essential in giving a useful explanation of various types of GUI frameworks, GUI testing approaches, and challenges facing GUI performance testing. Besides, theoretical background and evaluation metrics are taken from the cited papers. Nevertheless, none of the earlier published studies in the literature has examined experimentally the GUI performance between Java and C# at runtime. Therefore, this was the reason behind this study to be performed.

III. EVALUATION METHODOLOGY

Papers [19] and [20] were selected to adapt the evaluation methodology used in this study. It covers the evaluation conditions, evaluation scenarios, evaluation metrics, and software/hardware settings that were adopted and used to evaluate the performance of different GUI components; using small pieces of code corresponding to a specific GUI functionality.

A. Evaluation Conditions

Several evaluation conditions were taken into account in this research, as follows:

- Default configurations and settings of the integrated development environments (IDEs) were used.
- The same computer system with the same software/hardware specifications was used for the selected scenarios to be programmed and executed.
- All user programs (except the used IDEs) were closed before conducting the experiments and measurements.
- The Internet was switched off from the computer while evaluating the scenarios.
- Each of the scenarios was executed 1000 times; then, the average execution time has been taken for more accuracy.
- The execution time has been measured only for the execution of each code mentioned in the scenarios.

B. Evaluation Scenarios

The testing scenarios (32 scenarios chosen because they are frequently-used scenarios and form the basis for GUI-based software applications) proposed in this paper could be implemented in different programming structures, which may lead to different results. However, we have selected the scenarios having the same programming structures in both Java and C#. The performance of each GUI scenario in both languages is compared experimentally as presented in Table I

TABLE I. TESTING SCENARIOS

#	Scenario	C# Code	Java Code

1	Creating a new form of width and height of 500 and 300 respectively.	Form form = new Form();
		form.Size = new Size(500, 300);
		form.Show();
			JFrame form= new JFrame();
			form.setSize(500, 300);
			form.setVisible(true);

2	Creating a new form of width and height of i and i respectively. The value of Increases from 0 to 999.	Form form = new Form();
		form.Size = new Size(i, i);
		form.Show();
			JFrame form= new JFrame();
			form.setSize(i, i);
			form.setVisible(true);

3	Setting the size of a form to width and height of 500 and 300 respectively. It was reset back to width of 0 and height of 0.	form.Size = new Size(500, 300);	form.setSize(500, 300);
4	Adding a new panel to a form.	Panel p = new Panel(); form.Controls.Add(p);	JPanel p = new JPanel(); form.add(p);
5	Creating an empty combo box.	ComboBox cb= new ComboBox(); form.Controls.Add(cb);	JComboBox cb = new JComboBox(); form.add(cb);
6	Creating a new label.	Label label = new Label(); label.Text = "Test"; form.Controls.Add(label);	JLabel label = new JLabel ("Test"); form.add(label);
7	Creating a new button.	Button button = new Button(); button.Text = "Test"; form.Controls.Add(button);	JButton button = new JButton("Test"); form.add(button);
8	Creating a new checkbox.	CheckBox checkBox = new CheckBox(); checkBox.Text = "Test"; form.Controls.Add(checkBox);	JCheckBox checkBox = new JCheckBox("Test"); form.add(checkBox);
9	Creating a new radio button.	RadioButton radioButton = new RadioButton(); radioButton.Text = "Test"; form.Controls.Add(radioButton);	JRadioButton radioButton = new JRadioButton ("Test"); form.add(radioButton);

10	Creating a new list box containing five elements.	string[] items = new string[5] { "Test 1", "Test 2", "Test 3", "Test 4", "Test 5" };
		ListBox listBox = new ListBox();
		listBox.Items.AddRange(items);
		form.Controls.Add(listBox);
			String items[]= { "Test 1", "Test 2", "Test 3", "Test 4", "Test 5"};
			JList listBox = new JList(items);
			form.add(listBox);

11	Creating a new text field.	TextBox textBox = new TextBox(); textBox.Text = "Test"; form.Controls.Add(textBox);	JTextField textBox = new JTextField ("Test"); form.add(textBox);

12	Creating a new password field.	TextBox textBox = new TextBox(); textBox.Text = "Test"; textBox.PasswordChar = '*'; form.Controls.Add(textBox);	JPasswordField textBox = new JPasswordField ("Test"); form.add(textBox);
13	Reading the value of a text field. The variable is defined before measuring the execution time.	textBoxv = textBox.Text;	textBoxv = textBox.getText();
14	Creating vertical scroll bars.	VScrollBar vScroller = new VScrollBar(); form.Controls.Add(vScroller);	JScrollBar vScroller = new JScrollBar(); form.add(vScroller);

15	Creating a tabpane of two tabs.	TabControl tabControl = new TabControl();
		tabControl.Name = "TabControl";
		form.Controls.Add(tabControl);
		TabPage tabPage1 = new TabPage();
		tabPage1.Name = "Tab1";
		tabPage1.Text = "Tab one";
		tabControl.TabPages.Add(tabPage1);
		TabPage tabPage2 = new TabPage();
		tabPage2.Name = "Tab2";
		tabPage2.Text = "Tab two";
		tabControl.TabPages.Add(tabPage2);
			JTabbedPane tabControl = new
			JTabbedPane();
			JPanel panel1 = new JPanel(false);
			JLabel filler = new JLabel("Tab one");
			filler.setHorizontalAlignment(JLabel.CENTE
			R);
			panel1.add(filler);
			tabControl.addTab("Tab one", null,
			panel1,"Tab 1");
			JPanel panel2 = new JPanel(false);
			JLabel filler2 = new JLabel("Tab two");
			filler2.setHorizontalAlignment(JLabel.CENT
			ER);
			panel2.add(filler2);
			tabControl.addTab("Tab two", null,
			panel2,"Tab two");
			form.getContentPane().add(tabControl);

16	Adding an image to a form.	PictureBox pb = new PictureBox();
		pb.ImageLocation = "C:/test.png";
		pb.SizeMode = PictureBoxSizeMode.AutoSize;
		form.Controls.Add(pb);
			Toolkit t=Toolkit.getDefaultToolkit();
			Image pb=t.getImage("C://test.png");
			ImageIcon icon = new ImageIcon(pb);
			JLabel jLabel = new JLabel ("Test");
			form.add(jLabel);
			jLabel.setIcon(icon);

17	Maximize the form by minimizing it after each iteration.	form.WindowState = FormWindowState.Maximized;	form.setExtendedState(JFrame.MAXIMIZED _BOTH);
18	Minimize the from by Maximizing it after each iteration.	form.WindowState = FormWindowState.Minimized;	form.setState(JFrame.ICONIFIED);
19	Checking a checkbox. It was unchecked before each iteration.	checkBox.Checked = true;	checkBox.setSelected(true);
20	Checking a radio button. It was unchecked before each iteration.	radioButton.Checked = true;	radioButton.setSelected(true);
21	Unchecking a checkbox. It was checked before each iteration.	checkBox.Checked = false;	checkBox.setSelected(false);
22	Unchecking a radio button. It was checked before each iteration.	radioButton.Checked = false;	radioButton.setSelected(false);
23	Checking the value of a radio button. It was checked before each iteration.	if (radioButton.Checked) radioButton.Checked = false;	if(radioButton.isSelected()) radioButton.setSelected(false);
24	Checking the value of a check box. It was checked before each iteration.	if (checkBox.Checked) checkBox.Checked = false;	if(checkBox.isSelected()) checkBox.setSelected(false);

25	Creating a new text area with font type “Serif” and size of 16.	TextBox textArea = new TextBox();
		textArea.Text = "This is an editable TextArea.";
		textArea.Multiline = true;
		textArea.WordWrap = true;
		textArea.Font = new Font("Serif", 16);
		form.Controls.Add(textArea);
			JTextArea textArea = new JTextArea("This is
			an editable TextArea.");
			textArea.setLineWrap(true);
			textArea.setFont(new Font("Serif",
			Font.PLAIN, 16));
			form.add(textArea);

26	Creating a new menu bar containing one top item with a single drop down level.	MenuStrip menuStrip = new MenuStrip();
		ToolStripMenuItem tpmi = new
		ToolStripMenuItem();
		tpmi.Name = "file";
		tpmi.Text = "File";
		ToolStripMenuItem dropdown1 = new
		ToolStripMenuItem();
		dropdown1.Name = "exit";
		dropdown1.Text = "Exit";
		tpmi.DropDownItems.Add(dropdown1);
		menuStrip.Items.Add(tpmi);
		form.Controls.Add(menuStrip);
			JMenu menuStrip;
			JMenuItem level1;
			JMenuBar mb=new JMenuBar();
			menuStrip=new JMenu("File");
			level1=new JMenuItem("Exit");
			menuStrip.add(level1);
			mb.add(menuStrip);
			form.setJMenuBar(mb);

27	Creating a toggle button.	CheckBox toggleButton= new CheckBox();
		toggleButton.Text = "Test";
		toggleButton.Appearance = Appearance.Button;
		form.Controls.Add(toggleButton);
			JToggleButton toggleButton = new
			JToggleButton("Test");
			form.add(toggleButton);

28	Creating a horizontal spinnes.	TrackBar tb = new TrackBar();
		tb.Maximum = 2000;
		tb.Minimum = 1;
		tb.Value = 1996;
			JSlider tb = new JSlider(1, 2000, 1996);
			form.add(tb);

		form.Controls.Add(tb);	
29	Reading the value of a spinner.	a = tb.Value; // a is an int	a = tb.getValue(); // a is an int
30	Setting the value of the spinner to 1000;	tb.Value = 1000;	tb.setValue(1000);

31	Creating a table viewer of two records.	DataGridView dataView = new DataGridView();
		dataView.ColumnCount = 3;
		dataView.Columns[0].Name = "Product ID";
		dataView.Columns[1].Name = "Product Name";
		dataView.Columns[2].Name = "Product Price";
		string[] data = new string[] { "1", "Product 1", "1000" };
		dataView.Rows.Add(data);
		data = new string[] { "2", "Product 2", "2000" };
		dataView.Rows.Add(data);
		Panel p = new Panel();
		p.SetBounds(10, 10, 500, 300);
		p.Controls.Add(dataView);
		form.Controls.Add(p);
			String data[][]={ {"1","Product 1","1000"}, {"2","Product 2","2000"}};
			String column[]={"Product ID","Product Name","Product Price"};
			JTable dataView=new JTable(data,column);
			dataView.setBounds(10,10,500,300);
			JScrollPane p=new JScrollPane(dataView);
			form.add(p);

32	Creating a tree view with a root and two nodes.	TreeView treeView = new TreeView();
		TreeNode tNode;
		tNode = treeView.Nodes.Add("Root");
		treeView.Nodes[0].Nodes.Add("Vegetables");
		treeView.Nodes[0].Nodes.Add("Fruits");
		form.Controls.Add(treeView);
			DefaultMutableTreeNode treeView = new DefaultMutableTreeNode("Root");
			DefaultMutableTreeNode vegetableNode = new DefaultMutableTreeNode("Vegetables");
			DefaultMutableTreeNode fruitNode = new DefaultMutableTreeNode("Fruits");
			treeView.add(vegetableNode);
			treeView.add(fruitNode);
			JTree tree = new JTree(treeView);
			form.add(tree);

C. Evaluation Metrics

As mentioned before, the execution time (runtime) has been used as a metric to evaluate and compare experimentally the performance of each testing scenario. Thus, any scenario that requires less time to be executed is considered as the best one and outperforms the same scenario written in the other used language. For example, if scenario A (written in Java) requires less time to be executed compared to the same scenario (written in C#).

Then, scenario A (written in Java) is considered as the best one and outperforms the same scenario (written in C#).

D. Software/Hardware Setup

The hardware/software specifications that were used in this study are shown respectively in Table II and III.

TABLE II. SOFTWARE SPECIFICATION

	Software	Version
Java Test Applications	Java JDK	1.8.0_241
	NetBeans IDE	8.2
C# Test Applications	Microsoft Visual Studio Community 2019	16.4.2
	Microsoft .Net Framework	4.8.03761
Operating System	Microsoft Windows	10 (64 bit)

TABLE III. HARDWARE SPECIFICATION

	Hardware	Detail
Computer System	Model	HP EliteDesk 800 G1 TWR
	CPU Type	Intel ® Core™ i7-4790
	CPU Speed	3.60 GHz
	CPU Cores	4
	RAM	16 GB
	Rating (Windows Experience Index)	6.4

IV. EXPERIMENTAL RESULTS AND DISCUSSIONS

The results of the Java and C# GUI performance analysis for the selected scenarios are presented in Table IV.

TABLE IV. PERFORMANCE EVALUATION RESULTS

Scenarios	C# (ms)	Java (ms)
1	37.793	18.219
2	37.604	17.107
3	1.197	0.884
4	1.930	0.062
5	5.976	0.829
6	1.686	0.094
7	1.711	0.094
8	1.829	0.109
9	1.595	0.11
10	3.165	0.156
11	2.182	0.344
12	2.177	0.281
13	0.028	0.0007
14	1.566	0.25
15	6.107	0.234
16	13.240	0.109
17	4.410	2.379
18	4.829	1.394
19	0.554	0.016
20	0.832	0.015
21	0.553	0.016
22	0.558	0.015
23	0.580	0.014
24	0.557	0.013
25	2.934	0.422
26	139.749	1.844
27	1.668	0.109
28	1.857	0.141

29	0.012	0.0002
30	0.002	0.0004
31	4.749	0.919
32	4.885	0.323

From Table 3 (bold font describes the best performance while the plain font denotes the worst performance), it is evident that Java outperformed C# in all the 32 test scenarios. This might be due to the fact that the Java compiler uses a specialized performance engine called "HotSpot" to do a compilation of Java code into machine code. The C# compiler does not employ such performance engines. The "HotSpot" offers many advanced features [21-23] including: (a) it is mainly upgraded to speed-up code execution. Therefore, it is well-suited for GUI applications. (b) it starts with the codes that are used frequently then it continues with the rest of codes. Whereas the C# compiler compiles all of the code into machine code once when it is called at runtime. In other words, it does not give any importance to the codes that are used frequently. (c) it produces smaller Java bytecode which might lead to better performance.

V. CONCLUSIONS

The primary goal of this experimental analysis was to evaluate the performance of the most used GUI components provided by Java and C#. To eliminate potential latency caused by the user interactions; all the scenarios were tested in runtime and executed without user intervention. The experimental results of this study showed that Java outperformed C# in all the employed test scenarios. The possible reason for this is that Java uses a more advanced compiler compared to C#. This study will assist developers to understand and reduce the usage of the GUI components with higher latencies; especially for developing complex GUI applications. Moreover, it will benefit the novice developers in the selection process of the most suitable GUI programming language. Additionally, researchers could focus on decreasing the latency presented in the GUI components. Few possible future works are valid for this work: (a) increasing the test scenarios by using other GUI components and their properties that are not used in this work. (b) measuring the CPU usage and memory consumption of each GUI component. (c) using different programming structures for each test scenario. (d) performing the same test scenarios on other operating systems (e.g., Linux and Macintosh) to measure their performance.

REFERENCES

[1] J. Bishop and N. Horspool, Developing principles of GUI programming using views (Proceedings of the 35th SIGCSE technical symposium on Computer science education). Norfolk, Virginia, USA: Association for Computing Machinery, 2004, pp. 373–377.

[2] A. Ng, J. Lepinski, D. Wigdor, S. Sanders and P. Dietz, "Designing for low-latency direct-touch input", *Proceedings of the 25th annual ACM symposium on User interface software and technology - UIST '12*, 2012, pp. 453–464. Available: 10.1145/2380116.2380174

[3] Q. Xie, "Developing cost-effective model-based techniques for GUI testing", *Proceeding of the 28th international conference on Software engineering - ICSE '06*, 2006, pp. 997–1000. Available: 10.1145/1134285.1134473

[4] C. J. Howell, G. M. Kapfhammer, and R. S. Roos, An examination of the run-time performance of GUI creation frameworks (Proceedings of the 2nd international conference on Principles and practice of programming in Java). Kilkenny City, Ireland: Computer Science Press, Inc., 2003, pp. 171–176.

[5] N. Beierle, P. M. Kruse and T. E. J. Vos, "GUI-Profiling for Performance and Coverage Analysis," *2017 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)*, Toulouse, 2017, pp. 28-31.

[6] O. Valdes, "Finding the shortest path to reproduce a failure found by TESTAR", *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering - ESEC/FSE 2019*, 2019, pp. 1223–1225. Available: 10.1145/3338906.3342496

[7] P. Aho and T. Vos, "Challenges in Automated Testing Through Graphical User Interface", *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2018, pp. 118-121. Available: 10.1109/icstw.2018.00038

[8] S. Carino and J. Andrews, "Evaluating the Effect of Test Case Length on GUI Test Suite Performance", 2015 IEEE/ACM 10th International Workshop on Automation of Software Test, 2015, pp. 13–17. Available: 10.1109/ast.2015.10

[9] "javax.swing (Java Platform SE 7)", Docs.oracle.com, 2020. [Online]. Available: https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html.

[10] "C# docs - get started, tutorials, reference.", Docs.microsoft.com, 2020. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/csharp/.

[11] M. Jovic and M. Hauswirth, "Performance Testing of GUI Applications", 2010 Third International Conference on Software Testing, Verification, and Validation Workshops, 2010, pp. 247-251. Available: 10.1109/icstw.2010.27

[12] A. Memon and Q. Xie, "Studying the fault-detection effectiveness of GUI test cases for rapidly evolving software", IEEE Transactions on Software Engineering, vol. 31, no. 10, 2005, pp. 884-896. Available: 10.1109/tse.2005.117

[13] N. Cruz Quental, C. de Albuquerque Siebra, J. Peixoto Quintino, F. Florentin, F. Queda Bueno da Silva and A. de Medeiros Santos, "Automating GUI Response Time Measurements in Mobile and Web Applications", *2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST)*, 2019, pp. 35-41. Available: 10.1109/ast.2019.00011

[14] M. Jovic and M. Hauswirth, "Listener latency profiling: Measuring the perceptible performance of interactive Java applications", Science of Computer Programming, vol. 76, no. 11, 2011, pp. 1054-1072. Available: 10.1016/j.scico.2010.04.009

[15] Y. Endo, Z. Wang, J. Chen and M. Seltzer, "Using latency to evaluate interactive system performance", Proceedings of the second USENIX symposium on Operating systems design and implementation - OSDI '96, 1996, pp. 185-199. Available: 10.1145/238721.238775

[16] M. Jovic and M. Hauswirth, "Measuring the performance of interactive applications with listener latency profiling", Proceedings of the 6th international symposium on Principles and practice of programming in Java - PPPJ '08, 2008, pp. 137-146. Available: 10.1145/1411732.1411751

[17] K. Biswa, B. Jamatia, D. Choudhury, and P. Borah, "Comparative analysis of C, FORTRAN, C# and Java programming languages," Int J Comput Sci Inf Technol, vol. 7, no. 2, 2016, pp. 1004-7.

[18] M. Jovic, A. Adamoli, D. Zaparanuks, and M. Hauswirth, "Automating performance testing of interactive Java applications," presented at the Proceedings of the 5th Workshop on Automation of Software Test, Cape Town, South Africa, 2010, pp. 8–15. [Online]. Available: https://doi.org/10.1145/1808266.1808268

[19] A. Adamoli, D. Zaparanuks, M. Jovic and M. Hauswirth, "Automated GUI performance testing", Software Quality Journal, vol. 19, no. 4, 2011, pp. 801-839. Available: 10.1007/s11219-011-9135-x

[20] S. P. Ahuja and N. Mupparaju, "Performance Evaluation and Comparison of Distributed Messaging Using Message Oriented Middleware", Computer and Information Science, vol. 7. no. 4, 2014, pp. 9-20. Available: 10.5539/cis.v7n4p9

[21] "The Java HotSpot Performance Engine Architecture", Oracle.com, 2020. [Online]. Available: https://www.oracle.com/technetwork/java/whitepaper-135217.html. [Accessed: 11- Apr- 2020].

[22] M. Paleczny, C. Vick, and C. Click, "The java hotspot TM server compiler," in Proceedings of the 2001 Symposium on Java TM Virtual Machine Research and Technology Symposium, 2001, vol. 1, no. S 1. pp. 1-3.

[23] J. Singer, "JVM versus CLR: a comparative study," in Proceedings of the 2nd international conference on Principles and practice of programming in Java, 2003, pp. 167-169.