Eliminating Human Visual Judgment from Testing of Financial Charting Software+

Kwan Yong Sim*
Faculty of Engineering, Computing and Science,
Swinburne University of Technology Sarawak Campus, Kuching, Malaysia
Email: ksim@swinburne.edu.my

Chin S. Low
Nextwave Software, 68100 Sri Gombak, Malaysia
Email: cslow@nextwavesoft.com

Fei-Ching Kuo
Faculty of Information and Communication Technologies,
Swinburne University of Technology, Hawthorn, Australia
Email: dkuo@swin.edu.au

***Abstract*—Financial charting software is widely used in share, commodity and foreign currency exchange markets to visualize and analyze price movements. Its quality is critical because incorrect outputs may lead to wrong analysis and trading decisions, and consequently substantial financial losses. Human visual judgment is often required to test financial charting software because of the graphical complexity of software outputs and limited knowledge of expected outputs. Such approach is labour intensive and error-prone. In this paper, we propose an automated testing technique combining metamorphic testing, assertion checking and a novel data label extraction method to eliminate human visual judgment from testing financial charting software. We used this technique to test pre-release builds of a commercial Point and Figure charting software component, and demonstrated that the proposed technique can effectively detect actual faults in the software component. Further, we discuss how the technique can be extended to test other charting software components.**

***Index Terms*— Software Testing, Financial Charting Software, Metamorphic Testing, Assertion Checking**

I. INTRODUCTION

The quality of software tools used in financial markets is of outmost importance because millions of dollars may be at stake if there is any fault in the software tools. Financial charts have been widely used as the primary tool in technical analysis of price movement in share, commodity and foreign currency exchange (also known as *forex*) markets. Previous study by Taylor and Allen [30] found that over 90% of dealers in forex market use financial charts to perform technical analysis prior to making trading decisions. Gehrig and Menkhoff [17] further reported that financial charts are commonly used by forex dealers and fund managers to forecast short-term price movements in various financial markets.

The main function of financial charting software is to process a large amount of time-series price data and translate it into visual representations in form of charts, which are more meaningful to its users. For instance, Point and Figure chart, Renko chart, Kagi chart and Three-line-break chart are widely used to visually highlight major trends and turning points in share, commodity and forex prices. Such visual presentation provides the essential information needed for analysis and decision making in trading and investment. Therefore, financial charts are often integrated into the market analysis and trading software used by dealers, fund managers and retail clients in financial markets.

As a crucial tool in financial trading, any fault in the financial charting software components could incur financial losses to the users. Therefore, testing of financial charting software components used in market analysis and trading software is crucial to ensure that faults are discovered and eliminated before released to the end users.

However, testing of financial charting software components presents a few challenges. Due to the graphical nature of the software outputs, testers have to manually inspect the chart produced by the software as output and exercise their visual judgment to determine the correctness of the chart. They may have to manually construct a chart and spot out any difference between the constructed chart and the chart produced by the software. This approach is known as *manual oracle* [22]. This is a tedious and error-prone task for any non-trivial chart with a substantial number of data points in its outputs. To make it worse, the correct charting outputs are often unknown or cannot be derived easily. In this situation, testers will not be able to determine the correctness of the outputs produced by the charting component. In software

+ A preliminary version of this paper has been presented in the Proceedings of the 2nd International Conference on Computer Science and its Applications, 2009.

* Corresponding Author.

Manuscript received Month Day, Year; revised Month Day, Year; accepted Month Day, Year.

doi:10.4304/jsw.9.2.298-312

testing, this is known as the *oracle problem* [14]. Given a test case as input, an oracle is the mechanism to specify the expected output for the software or component under test [16]. Complete oracle (where expected output is known for every input) is not available for financial charts such as Point and Figure chart, Renko chart, Kagi chart and Three-Point-Break chart.

Test automation could be an effective solution to eliminate error-prone human visual judgment from testing of financial charting components. However, automatic testing normally requires a complete oracle so that every test output can be verified. In the absence of oracle, automatic pixel to pixel verification cannot be done unless there exist some "pseudo oracles" [14], in which multiple independently-developed implementations of the same chart are used to compare the charting outputs for a given input. Past study [21] suggested that pseudo oracles not only are expensive to deploy, but also may not be feasible or effective. This is because multiple implementations may not exist. Even if they do exist, they may have been created by the same group(s) of developers who are prone to making the same types of mistakes. As a result, the pseudo oracle may not be trustable.

In the absence of both complete oracle and trustable pseudo oracle, *metamorphic testing* [11] can be used as a reliable way not only to detect faults in software under test but also to generate follow up test cases from existing ones. In metamorphic testing, if input x produces an output $f(x)$, the necessary property (known as *metamorphic relation*) of the software under test can be used to generate a follow test case x' for which the output $f(x')$ can be determined or predicted based $f(x)$. If the output $f(x')$ is not as expected according to the metamorphic relation, then we can conclude that there exists a fault in the software under test. Therefore, metamorphic testing provides a reliable way to detect any fault that causes violation to the metamorphic relation without the presence of oracles. Furthermore, metamorphic testing does not require multiple-implementations to provide pseudo-oracle. Hence, it is less expensive to deploy in testing.

On the other hand, assertion checking [6][31] can also be used to verify whether the execution of a test case satisfies some expected and necessary properties even though test oracles (pseudo or non-pseudo) are not available. Properties such as correct program states, variable initialization as well as lower or upper bounds of variable value and program output can be used as assertion conditions. These assertion conditions must be satisfied for correct program implementation and execution. Even though the expected output is unknown, any violation to the assertion conditions implies the presence of faults in the software under test. Therefore, assertion checking can be used to test software with oracle problem. In an empirical study to compare the use of metamorphic testing and assertion checking, Zhang, Chan, Tse and Hu [34] suggested that even though metamorphic testing is more effective than assertion checking in fault detection, assertion checking is more efficient in terms of time and cost of implementation and can provide finer granularity in testing.

Takahashi [29] proposed a coordinate and projection-based approach to automate the verification for Graphical User Interface (GUI) objects in software such as Microsoft PowerPoint. In a separate study, Xie and Memon [32] proposed and examined the effectiveness of different types of oracles for graphic user interface testing. Even though these techniques can be automated to eliminate human visual judgment from testing GUI objects, they still require test oracles to be present for the software under test. A more recent study by Zacharias [33] on test case generation and reuse on GUI also assumed complete oracle is available.

Reading a huge amount of data on a chart and then verifying the chart's correctness is a very challenging task. Many modern charting software tools such as Microsoft Excel assist the user to directly determine the data value on the chart by labeling these values besides the data points. Extracting data labels can automate output verification because data label extraction simplifies output verification from graphical comparison to numerical comparison.

In short, the graphical natures of its outputs and the absence of oracle prevent financial charting software components from being tested automatically without requiring human visual judgment. In view of this, we conducted a case study on the oracle problem in Point and Figure chart in [25] and proposed an automatic testing technique that encompasses test case generation, execution and output verification for Point and Figure charting software component. Our proposed testing technique combines metamorphic testing and assertion checking. Experiments conducted on five pre-release builds of Nextwave Software WPF Point and Figure charting component have successfully detected different actual faults in the charting component tested.

In this paper, we extend the work in [25] and further make the following contributions:

1. Overall, we have eliminated subjective human visual judgment from testing of financial charting component that has the oracle problem. While test automation techniques have long been used to replace human visual judgment in testing of graphical user interface, these techniques assume that complete test oracle is available for testing [29][32]. However, test oracle is not available for many financial charts.
2. We introduced a data label extraction method to obtain the values of output data points from the financial charts. Data label extraction also allows charting output verification to be simplified from graphical comparison to numerical value comparisons, which can be easily automated. Hence, manual visual inspection of charting output is no longer required. Unlike the coordinate and projection-based method for GUI testing proposed by Takahashi [29], data label extraction is simpler and more straight-forward to implement because it does

not suffer from problems related to display resolution and zooming (resizing) of charting outputs.

3. We extend the automatic testing technique for financial charts proposed in [25] to explicitly combine metamorphic testing, assertion checking with our invented data label extraction method. Metamorphic testing has been shown to be more effective than assertion checking in terms of fault detection capability but its granularity of testing criteria is coarser than that of assertion checking [34]. Based on this observation, we propose to use both assertion checking and metamorphic testing in complementary to maximize fault detection capability. In addition to the metamorphic relations proposed in [25], three new metamorphic relations (MR3, MR5 and MR7) are proposed in this paper, of which MR3 has successfully detected faults in the Point and Figure chart under test.
4. We report experimental studies of a real-world financial chart software component (Point and Figure chart) to evaluate the effectiveness of the proposed technique. Through our experiment setup, we have demonstrated that testing process can be fully automated to eliminate human visual judgment. From the experiment results, we found that both assertion checking and metamorphic testing have detected different real faults in the financial charting components under test. In addition to the real-life financial time-series data used in [25], we introduce the randomized data series and compare its fault detection capability with the real-life financial time-series.
5. We present and discuss the details of the bugs and their detection by assertion conditions and metamorphic relations.
6. We observe that the effectiveness of metamorphic testing relies on the test case used for testing. From the experiment results, we found that test cases with randomized data series have better fault detection capability than real-life financial time-series data for the Point and Figure charting component under study.
7. We analyze and discuss the extendibility of the proposed testing technique to three other reversal charts, namely, Renko chart, Kagi chart and Three-line-break chart.

The remainder of this paper is organized as follows. Section II presents the background on metamorphic testing and assertion checking. Section III presents the Point and Figure chart and its algorithm as the test target in this study. Section IV explains the metamorphic relations and assertion conditions and data extraction method that work together to form our charting component testing technique. Section V presents the experiment settings and experiment results. Section VI discusses how the proposed technique can be extended to the other financial charts that do not have test oracles. Section VII presents limitation and future work. Section VIII concludes the paper.

II. Background

This section gives an overview of metamorphic testing, assertion checking, and how these testing techniques can be used to detect faults in the absence of oracle.

A. Metamorphic Testing

Metamorphic testing was first coined by Chan, Chen, Cheung, Lau and Yiu [8] as a new approach to generate the next test case from existing test case (especially one that has not revealed any fault). In metamorphic testing, metamorphic relation is first defined based on the relation between a set of test inputs and their corresponding outputs. The metamorphic relations can be identified from the necessary properties of the software under test. From a test case (called source test case) with unverifiable output, the metamorphic relation can be used to generate follow up test cases such that the outputs of the source test case and follow up test cases can be checked against the metamorphic relation. If the metamorphic relation is violated, then we know that the software under test is faulty. Therefore, metamorphic testing can detect faults in the software under test without requiring test oracles.

The following example illustrates how metamorphic testing can be used to test a program that has been written to compute the *cos(x)* function. When $x = 24°$ is used as the test input to the program and the program produces *0.9135* as the computed output. However, the correctness of this output cannot be verified because the expected correct output is unknown (in other words, the oracle is absence). However, we know that cos(x) observes the trigonometry identity of $cos(x) = -cos(x+180°)$. Using this necessary property of *cos(x)* as the metamorphic relation, we can detect a fault in this program whenever this metamorphic relation is violated. In this case, the program should produce *-0.9135* as output when $24°+180°$ (that is $204°$) is used as test input. In metamorphic testing, $24°$ serves as the source test case, while $204°$ serves as the follow up test case. If the program outputs for these two test cases are different in absolute value, then we can conclude that the software under test has a fault because the metamorphic relation is violated.

To formally define a metamorphic relation, let:

$\mathbf{I_1} = \{T_1, T_2, \ldots, T_k\}$ be a set of test cases as inputs to a function *f*, where $k \geq 1$. $\mathbf{I_1}$ is known as the *source test cases*.

$\mathbf{O_1} = \{f(T_1), f(T_2), \ldots, f(T_k)\}$ be the set of outputs produced by *f* corresponding to test cases in $\mathbf{I_1}$.

$\mathbf{S} = \{f(T_{s1}), f(T_{s2}), \ldots, f(T_{sm})\}$ be a subset of O_1 where $m \geq 0$.

$\mathbf{I_2} = \{T_{k+1}, T_{k+2}, \ldots, T_n\}$ be another set of test cases as inputs to *f*, where $n \geq k+1$. $\mathbf{I_2}$ is known as the follow up test cases.

$\mathbf{O_2} = \{f(T_{k+1}), f(T_{k+2}), \ldots, f(T_n)\}$ be the corresponding set of outputs for test cases in $\mathbf{I_2}$.

$\mathbf{R_I}(T_1, T_2, \ldots, T_k, f(T_{s1}), f(T_{s2}), \ldots, f(T_{sm}), T_{k+1}, T_{k+2}, \ldots, T_n)$ be a relation among $\mathbf{I_1}$, $\mathbf{S}$ and $\mathbf{I_2}$. $\mathbf{R_I}$ is known as the test input relation.

$\mathbf{R_O}(T_1, T_2, \ldots, T_n, f(T_1), f(T_2), \ldots, f(T_n))$ be the relation among $\mathbf{I_1}$, $\mathbf{I_2}$, $\mathbf{O_1}$ and $\mathbf{O_2}$. $\mathbf{R_O}$ is known as the test output relation.

To formally define a metamorphic relation, assume that there exists a relation $\mathbf{R_I}$ among $\mathbf{I_1}$, $\mathbf{S}$ and $\mathbf{I_2}$, and another relation $\mathbf{R_O}$ among $\mathbf{I_1}$, $\mathbf{I_2}$, $\mathbf{O_1}$ and $\mathbf{O_2}$ such that $\mathbf{R_O}$ must be satisfied whenever $\mathbf{R_I}$ is satisfied. The metamorphic relation (MR) can then be defined as:

MR: If $\mathbf{R_I}(T_1, T_2, \ldots, T_k, f(T_{s1}), f(T_{s2}), \ldots, f(T_{sm}), T_{k+1}, T_{k+2}, \ldots, T_n)$, then $\mathbf{R_O}(T_1, T_2, \ldots, T_n, f(T_1), f(T_2), \ldots, f(T_n))$.

Metamorphic relations can normally be sourced from stakeholders of the software under test who have the knowledge in the application domain. In addition, previous study [34] showed that software testers who have been briefly introduced to metamorphic testing are also able to identify metamorphic relations for software under test.

Once the **MR** has been identified, testers need to generate or select some arbitrary test cases as the source test cases, $\mathbf{I_1}$. The following procedures can then be used to conduct Metamorphic testing:

1. Let program P be the software under test that implement function *f*. Run program P with $\mathbf{I_1}$ as source test cases. Record the corresponding outputs $\mathbf{O_1}$.
2. Generate follow up test cases $\mathbf{I_2}$ using $\mathbf{R_I}$, $\mathbf{I_1}$, and $\mathbf{S}$.
3. Run program P using $\mathbf{I_2}$ as follow up test case. Record the corresponding outputs $\mathbf{O_2}$.
4. Check the test outputs in $\mathbf{O_1}$ and $\mathbf{O_2}$ against the relation $\mathbf{R_O}$.
5. If $\mathbf{R_O}$ is violated, then there exists fault(s) in program P. Otherwise, no fault is detected by test cases in $\mathbf{I_1}$ and $\mathbf{I_2}$.

Metamorphic testing has been proven useful and effective in testing applications in the absence of oracles. Successful deployment of metamorphic testing has been reported in many application domains such as numerical analysis [8], aviation software [20], numerical solution of partial differential equations [12], web search engines [35], image processing [27], die casting [26], machine learning [23], biomedical applications [13], power-aware software for wireless sensor networks [9], middleware-based applications [10] and healthcare simulation software [24]. Furthermore, metamorphic testing has been used to improve the testability of program components [5]. Preliminary study on Point and Figure chart also showed that metamorphic testing is effective in detecting faults in financial charting software [25].

B. Assertion Checking

In the absence of oracles, assertion checking verifies execution of a test case against some expected and necessary properties [2][7][34]. It is a property-based testing technique, where properties of software under test are identified as *assertion conditions*, which are logical expressions that evaluate to either *true* or *false*. An assertion condition must be satisfied (that is, evaluate to *true*) for correct implementation and execution of the software. If the assertion condition evaluates to *false*, the assertion is violated and it implies that the software under test is faulty. Therefore, assertion checking does not require a test oracle to detect faults in the software. Assertion checking can be done not only on the output of the software, but also intermediate program states and variable values. Normally, assertion checking is directly embedded in the code of the software developed. It is widely supported by popular programming platforms such as the Microsoft .Net and Java platforms.

To illustrate the use of assertion checking to test software without the presence of oracles, consider the same program that has been used in the previous subsection (Section II A) to compute the *cos(x)* function. Let $x = 24°$ be a test input to the program. As the expected output is unknown (in other words, the oracle is absence), the computed output cannot be verified. However, we can use the trigonometry property of sine function $-1 \leq cos(x) \leq 1$ as an assertion condition for assertion checking. We know that if the program is implemented correctly, then the program output must satisfy this assertion condition (that is, the assertion condition must evaluate to true). If the program output is not between *-1* and *1* (both inclusive), then the assertion condition will evaluate to false and cause an assertion error message to be prompted on most programming platforms. In that way, we will know that the software under test has a fault even though the expected output is unknown.

Assertion checking has been successfully deployed to analyze the state-based behaviors [7] and detect state-related errors in object-oriented program [19] as well as behavioral conflict in aspect-oriented software [18].

III. TEST TARGET

While evidences suggested that financial charts have been used for over 100 years [1], it is only in recent decades that financial charting software became widely accessible to institutions and retail traders through market analysis and trading software packages. Among the others, the Line chart, Bar chart, Candle-stick chart, Point and Figure chart, Renko chart, Kagi chart and Three-line-break chart are standard features available in these software packages. Interested readers are referred to [1] for detailed description of each financial chart.

The Line chart, Bar chart and Candle-stick chart can be easily verified as they are plotted directly from the input data points. However, charting outputs of the Point and Figure chart, Renko chart, Kagi chart and Three-line-break chart cannot be easily verified. These charts belong to a broad category of charts known as *reversal charts*, where a new point will only be plotted after price values in the input data points have changed by a significant amount pre-defined by the user. In effect, these charts filter small price fluctuations in the input data and plot only the major price moves in the charting outputs. In addition to the complicated charting algorithm, these financial charts are required to accept a large number of data points as inputs to facilitate meaningful analysis based on historical data. These factors contribute to the oracle problems in testing the reversal charts.

In this paper, we focus our study on the Point and Figure chart as the primary test target because it is the most difficult to test among the reversal charts. The Point and Figure chart is plotted based on two user-defined variables, known as *box size* and *reversal amount* (to be explained in the next paragraph). On the other hand, Renko chart is plotted based on box size alone, while Kagi chart is plotted based on minimum reversal alone. Similarly, three-line-break chart is plotted based on the number of lines which is equivalent to reversal amount in concept. In short, the combination and interaction of both the box size and the reversal amount make the outputs of the Point and Figure chart more difficult to verify compared to the other reversal charts. Therefore, we will design the testing technique with the Point and Figure chart as the test target. Subsequently, we will also examine and discuss the extendibility of the proposed testing technique to the other reversal charts.

Upward trends or increasing prices are represented as a vertical column of 'X's in the Point and Figure Chart. Similarly, downward trends or declining prices are displayed as a vertical column of 'O's adjacent to the column of 'X'. The chart is plotted based on the box size and the reversal amount. The box size is defined as the minimum amount of price movement before a figure ('X' or 'O') is plotted on the chart. In other words, a new figure ('X' or 'O') will not be plotted in the current column until the price has increased (or decreased) by more than the box size set by the user. On the other hand, a new column (reversal) will not be plotted until the price has been pulled back by the reversal amount multiplying the box size. It disregards the time required to produce such price movements.

Figure 1 shows a Line chart plotted based on the Dow Jones Industrial 30 Index (DJI30) daily closing price data from 2008 to 2009. This is plotted from more than 400 data points (daily closing prices) as charting inputs. The same data can be plotted into the Point and Figure Chart in Figure 2, with the box size set to 200 and a reversal amount of 3. In this case, the DJI30 index has to advance at least 200 points for an 'X' figure to be recorded in a column of 'X's. Conversely, it has to decline at least 200 points for an 'O' figure to be recorded in the column of 'O's on the chart. For a new column to be plotted, the DJI30 index has to reverse by at least 600 points (3×200 points). On the other hand, if the box size and the reversal amount are increased to 300 and 5 respectively as in Figure 3, a reversal of 1500 points (5×300 points) is required for a new column to be plotted. Hence, fewer columns are produced on the chart.

By comparing the Line chart in Figure 1 with Point and Figure charts in Figure 2 and Figure 3, it can be observed that the Point and Figure charts have less data points in the charting outputs. In effect, the Point and Figure chart filters minor price fluctuations in input financial time-series data in order to accentuate the major trends of price movement, and turning points [2].

The technique for constructing Point and Figure charts has remained substantially unchanged since the methodology was first illustrated by deVilliers [15]. To define the algorithm required to plot a Point and Figure Chart, let array p_i denote the price value for figure-i ('X' or 'O') plotted on the chart where $i=0, 1, 2\ldots$, and d_j denote the data point j of the financial time-series data input where $j = 0, 1, 2\ldots$. The algorithm for constructing the Point and Figure chart is outlined in Figure 4.

This algorithm is adapted from the Point and Figure Chart reversal algorithm as proposed by Archer and Bickford [1] with additional sub-steps for plotting 'X's and 'O's on the Point and Figure Chart. We noted that there exist other variants of algorithms to construct a Point and Figure chart. However, we will base our study on the algorithm outlined in Figure 4, which is used to develop the test target.

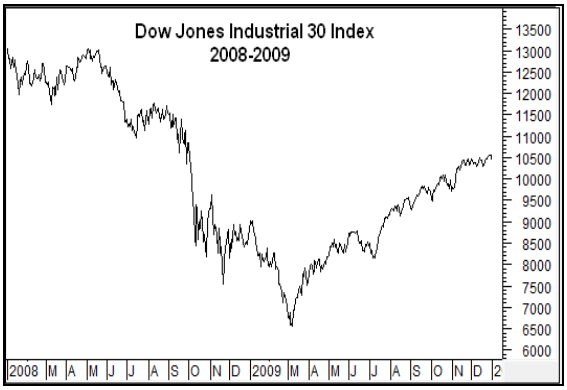

Figure 1. Line chart - Direct plotting of daily closing price data of Dow Jones Industry 30 Index from 2008 to 2009.

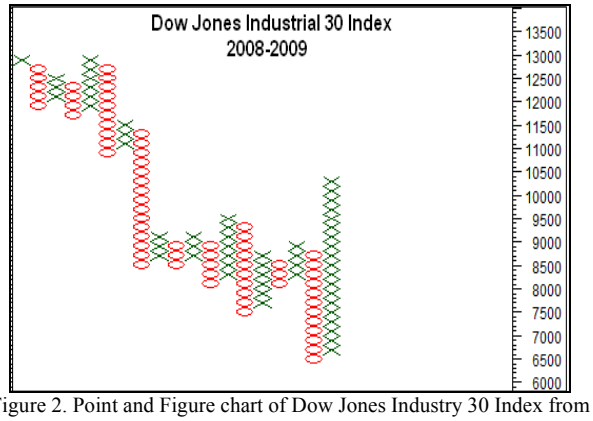

Figure 2. Point and Figure chart of Dow Jones Industry 30 Index from 2008 to 2009, with box size = 200 and reversal amount =3.

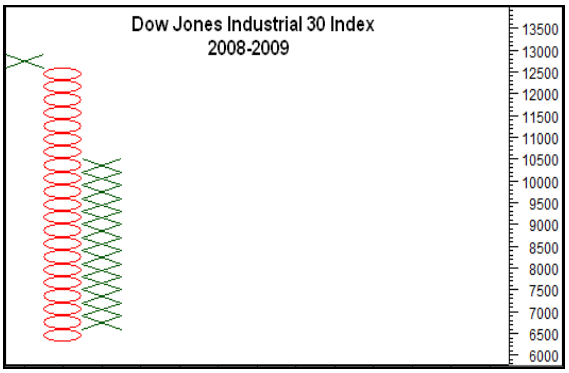

Figure 3. Point and Figure chart of Dow Jones Industry 30 Index from 2008 to 2009, with box size = 300 and reversal amount =5.

1. Initialize *boxSize* and *reversalAmount* variables.
2. Initialize *columnNumber=0*, *i=0, j=0*, $p_i = d_j$ and *direction = NULL*.
3. Increment *j*.
4. While *direction = NULL AND* $j \le$ *numberOfDataPoint*
   if $d_j \ge (p_i + boxSize)$, then
   Set *Direction = UP*.
   Plot 'X' at p_i at the current *columnNumber*
   While $p_i + boxSize \le d_j$
   Increment *i*, set $p_i = p_{i-1} + boxSize$
   Plot 'X' at p_i at the current *columnNumber*
   Endwhile
   Else if $d_j \le (p_i - boxSize)$, then
   Set *Direction = DOWN*.
   Plot 'O' at p_i at the current *columnNumber*
   While $p_i - boxSize \ge d_j$
   Increment *i*, set $p_i = p_{i-1} - boxSize$
   Plot 'O' at p_i at the current *columnNumber*
   Endwhile
   Endif
   Increment *j*.
   Endwhile
5. While $j \le$ *numberOfDataPoint*
   if *direction = UP*, then
   If $d_j - p_i \ge boxSize$ Then
   While $p_i + boxSize \le d_j$
   Increment *i*,set $p_i = p_{i-1} + boxSize$
   Plot 'X' at p_i at the current *columnNumber*
   Endwhile
   Else if $d_j \le (p_i - boxSize * reversalAmount)$, *Then*
   Increment *columnNumber*
   Set *Direction = DOWN*
   While $p_i - boxSize \ge d_j$
   Increment *i*, set $p_i = p_{i-1} - boxSize$
   Plot 'O' at p_i at the current *columnNumber*
   Endwhile
   Endif
   Else if *direction = DOWN*, then
   if $p_i - d_j \ge boxSize$ Then
   While $p_i - boxSize \ge d_j$
   Increment *i*, set $p_i = p_{i-1} - boxSize$
   Plot 'O' at p_i at the current *columnNumber*
   Endwhile
   Else if $d_j \ge (p_i + boxSize * reversalAmount)$, *Then*
   Increment *columnNumber*
   Set *Direction = UP*
   While $p_i + boxSize \le d_j$
   Increment *i*, set $p_i = p_{i-1} + boxSize$
   Plot 'X' at p_i at the current *columnNumber*
   Endwhile
   Endif
   Endif
   Increment *j*
   End while
6. Exit

Figure 4. The algorithm for constructing Point and Figure chart

IV. TESTING APPROACH

The testing approach proposed in this section combines metamorphic testing, assertion checking and our invented data label extraction method to eliminate human visual judgment from testing. Metamorphic testing can be applied without the source code of the software under test, while the assertion checking requires assertion conditions to be inserted into the source code, but the conditions are not generated based on the program. Therefore, our testing approach is a black-box testing approach [4] that is, by definition, independent of the source-code of the software under test. Upon detection of fault in the chart, then the source code will be examined for debugging.

A. Identifying Metamorphic Relations

The preliminary step in metamorphic testing is to identifying metamorphic relations. We propose seven metamorphic relations, MR1 to MR7, to test the Point and Figure charting software component in this study. Let f be the function that represents the Point and Figure charting software component. Let T_s and T_f denote the source test case and follow up test case, respectively, for each metamorphic relation. Furthermore, let $f(T_s)$ and $f(T_f)$ be the outputs produced by the software under test for T_s and T_f respectively. In line with the notations used in Figure 4, d_j denotes the data point j of the financial time-series data input ($j = 0, 1, 2$...).

The seven metamorphic relations and the method for generating follow up test cases from the metamorphic relations are outlined below.

1. Let T_s be the financial time-series data used as the source test case, execute the software to obtain its output, $f(T_s)$. Let T_f equal to $f(T_s)$.If the software is executed again with T_f using the same box size and reversal amount to obtain the output $f(T_f)$, then $f(T_f)$ must be equal to $f(T_s)$. This metamorphic relation is derived from the invariant property $f(T_s)=f^n(T_s)$, where $n>1$, for the Point and Figure Chart. In other words, if the output of a Point and Figure Chart is applied as the input to the Point and Figure Chart again as follow up test case, then the resulting output must be identical to the output of the source test case. Therefore, MR1 is defined as:
   MR1: If $T_f = f(T_s)$, then $f(T_f) = f(T_s)$
   To generate the test case for MR1, simply copy $f(T_s)$ and use it as the follow up test case, T_f. Note that both T_s and T_f must use the same box size and reversal amount.
2. Let $T_s = (d_0, d_1, \ldots, d_k)$ be the financial time-series data used as the source test case, where $d_{j+1} > d_j$ for all $0 \le j \le k-1$. If the follow up test case is generated by inserting a random value between every pair of adjacent data points in the source test case, such that the random value is between the data points before and after it, then the output of the follow up test case must be identical to the output of the source test case. This metamorphic relation is identified based on the algorithm to plot a new figure ('X' or 'O') where no new figure may be plotted if the price has not advanced or declined by more than the box size. In short, MR2 can be defined as:
   MR2: if $T_s=(d_0, d_1, \ldots, d_k)$, $T_f=(d_0, d_{(0,1)}, d_1, \ldots, d_{k-1}, d_{(k-1,k)}, d_k)$ where $k \ge 1 \mid d_j < d_{(j, j+1)} < d_{j+1}$, $0 \le j \le k-1$, then $f(T_f)=f(T_s)$.
   To generate the follow up test case for this metamorphic relation, insert a random value between every pair of adjacent data points in the source test case, such that the random value is between the data points before and after it. Note that both T_s and T_f must use the same box size and reversal amount.
3. Use the same box size and reversal amount for both the source test case and follow up test case. If the follow up test case is generated by deleting one data point that has a value between the data point before and after it, then the output of the follow up test case must be the same as the output of the source test case. Similar to MR2, this metamorphic relation is also identified based on the algorithm to plot a new figure ('X' or 'O') where no new figure may be plotted if the price has not advanced or declined by more than the box size. In short, MR3 can be defined as:
   MR3: if $T_s(d_0, \ldots, d_j, \ldots, d_k)$, $T_f(d_0, \ldots, d_{j-1}, d_{(j+1)}, \ldots, d_k)$ where $k \ge 1 \mid d_{j-1} < d_j < d_{j+1}$, $1 \le j \le k-1$, then $f(T_f)=f(T_s)$.
   To generate the follow up test case for this metamorphic relation, delete a data point from the source test case if the data point has a value between the data point before and after it. Note that both T_s and T_f must use the same box size and reversal amount.
4. If the follow up test case is identical to the source test case and the reversal amount is incremented by one, then the output of the follow up test case must have a smaller or the same number of columns as the output of the source test case. MR4 is defined as:
   MR4: if $T_f=T_s$, $reversalAmount_{Ts}=a$, $reversalAmount_{Tf}=a+1$, where $a>0$, then $columnNumber_{f(Ts)} \ge columnNumber_{f(Tf)}$.
   To generate the follow up test case for this metamorphic relation, simply create a copy of the source test case and use it as the follow up test case. Increment the reversal amount of the follow up test case by one.
5. If the follow up test case is identical to the source test case and the reversal amount is decremented by one, then the output of the follow up test case must have more or the same number of columns as the output of the source test case. MR5 is defined as:
   MR5: if $T_f=T_s$, $reversalAmount_{Ts}=a$, $reversalAmount_{Tf}=a-1$, where $a>1$, then $columnNumber_{f(Tf)} \ge columnNumber_{f(Ts)}$.
   To generate the follow up test case for this metamorphic relation, simply copy the source test case and use it as the follow up test case. Decrement the reversal amount of the follow up test case by one.
6. If the follow up test case is identical to the source test case but the box size is reduced by halve, then

the output of the follow up test case must have more or the same number of columns as the output of the source test case. MR6 is defined as below:
MR6: if $T_f=T_s$, $boxsize_{Ts} = b$, $boxsize_{Tf} = 0.5b$, $b>0$, then $columnNumber_{f(Tf)} \geq columnNumber_{f(Ts)}$.
To generate the follow up test case for this metamorphic relation, simply copy the source test case and use it as the follow up test case. Note that the box size of the follow up test case must be half of the box size of the source test case.

7. If the follow up test case is the same as the source test case but the box size is doubled, then the output of the follow up test case must have a smaller or equal number of columns as the output of the source test case. MR7 is defined as below:
MR7: if $T_f=T_s$, $boxsize_{Ts} = b$, $boxsize_{Tf} = 2b$, $b>0$, then $columnNumber_{f(Ts)} \geq columnNumber_{f(Tf)}$.
To generate the follow up test case for this metamorphic relation, simply copy the source test case to the follow up test case. Set the box size of the follow up test case to be double the box size of the source test case.

The seven metamorphic relations defined above serve two purposes in metamorphic testing. Firstly, the definition of each metamorphic relation can be used to automatically generate follow up test cases. Secondly, they serve as the references for output verifications in the metamorphic testing procedure. These follow up test cases generated using the metamorphic relations target the detection of faults in the Point and Figure chart that cause any violation to the metamorphic relations defined. It is worth noting that there could be other metamorphic relations that can be used to test Point and Figure chart as the seven metamorphic relations proposed above are not exhaustive.

B. Defining Assertion Conditions

A Point and Figure Chart is plotted by incrementally adding 'X' or 'O' onto an existing column or a new column. As described in the algorithm in Figure 4, the first figure ('X' or 'O') is plotted at the value of first input data point. The subsequent figure ('X' or 'O') in the same column on the chart is plotted by adding or subtracting the box size to/from the price value of the previous figure. Therefore, a necessary property for the Point and Figure Chart is that the value of the first figure ('X' or 'O') must be the same as the first input data point. Another necessary property is that the interval between two adjacent figures p_i and p_{i+1} ('X' or 'O') must match the value of the box size. Based on the knowledge of these necessary properties, assertion checking can be used to detect violation of these properties in the Point and Figure Chart software component. These properties can be defined as assertion conditions in (1) and (2).

$$\text{assert: } p_0 = d_0 \quad (1)$$

$$\text{assert: } |p_{i+1} - p_i| = box\ size \quad (2)$$

where p_i denotes the price value for figure-i ('X' or 'O') plotted on the chart ($i=0, 1, 2...$) and d_0 denotes the value of the first input data point. For output data with $k+1$ points, where $k > 0$, the assertion condition (2) must hold for $0 \leq i \leq k$, irrespective of the difference in value between adjacent data points in the time-series input data.

C. Data Label Extraction

After injecting the test input data into the charting component, the test outputs, which are the data labels of the figures ('X' or 'O') to be plotted on the Point and Figure Chart, will be extracted from the charting component and exported for output verifications. This approach allows output verification to be simplified from graphical comparison to numerical comparison, hence alleviating human visual judgment from testing.

Similar technique has been proposed in [29] where the coordinate data of screen output passed to the graphical API was exploited for testing of on Microsoft PowerPoint. However, coordinate data of screen output are subjected to the influence of display resolution and zooming of graphical outputs under test. Data label extraction does not suffer from this problem because it is the actual value of the data point to be plotted on the chart. Therefore, its value will not be affected by display resolution and zooming of graphical output under test.

V. Experiments

In this section, we evaluate the effectiveness of the proposed testing technique on the test target. First, we outline the set up of experiment to test five pre-release software builds of Nextwave Software WPF Point and Figure chart software component which was developed and built on Microsoft .NET Framework's Windows Presentation Foundation (WPF) graphical subsystem. Next, we report the results of using assertion conditions and metamorphic relations proposed in Section IV to test the charting component under test.

A. Setup

Figure 5 outlines the set up for automatic testing of the Point and Figure charting component. First, the financial time-series data are used as the source test case, T_s, for the Point and Figure charting component under test. The resulting output data, $f(T_s)$, is extracted from the chart's data label and verified by the assertion checker against the assertion conditions defined in Section IV B. For MR1, the output data, $f(T_s)$, will be fed into the test case generator to generate follow up test cases. For MR2 and MR3, the source test case T_s will be modified with respect to a data point d_j to generate a follow up test case. For MR4 to MR7, the source test case T_s can be reused as the follow up test case with a change in either the reversal amount or box size. Lastly, the output data correspond to the follow up test case is verified with both the output verifier (based on metamorphic relations) and assertion checker (based on assertion conditions) defined in Section IV A and Section IV B, respectively. Any violation detected in this process is recorded as a failure.

For each source test case, this process is repeated until all metamorphic relations have been covered at least once by the test case generator. A sample of screen capture of the corresponding charting component graphical output is shown in Figure 6.

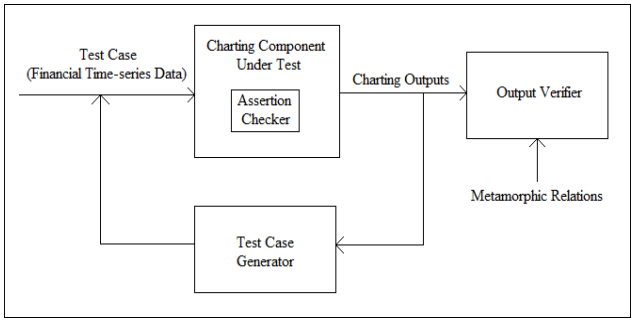

Figure 5. The experiment set up for testing of Point and Figure charting component.

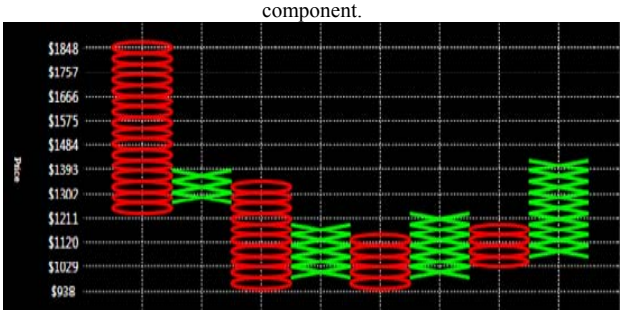

Figure 6. Screen capture of output from charting component for Seoul Composite Index (KS11) closing price from June 2008 to May 2009. The box size and the reversal amount are set to 40 and 3 respectively.

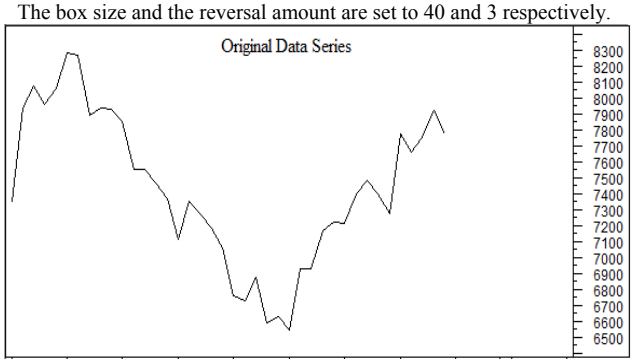

Figure 7. Line chart for the original data series that consist of 40 data points in chronological order.

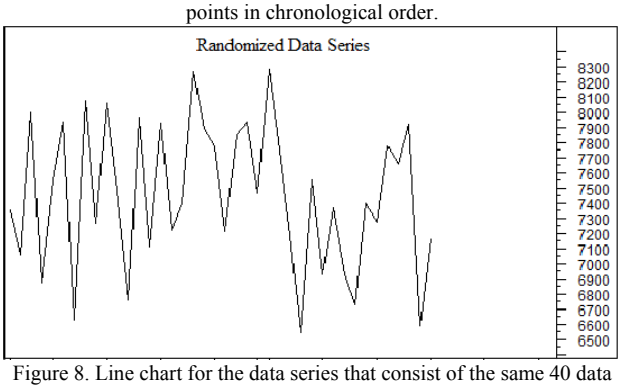

Figure 8. Line chart for the data series that consist of the same 40 data points in Figure 7, but in randomized order.

Table 1. Six data series used as source test cases in the experiments.

Series ID	Data Range		No. of Data Points	Box Size	Reversal Amount
	Min	Max			
DJI30	6547.05	6547.05	251	300	3
MSFT	15.15	28.93	251	1	3
KS11	938.75	1847.53	249	40	3
DJI30-R	6547.05	6547.05	251	300	3
MSFT-R	15.15	28.93	251	1	3
KS11-R	938.75	1847.53	249	40	3

Six data series have been prepared as source test cases for the experiments. Three out of six series are real-life financial time-series data of different data ranges, namely, the Dow Jones Industrial 30 Index (DJI30), Microsoft Corporation (MSFT) and South Korea's Seoul Composite Index (KS11). Each data series is constructed by taking the daily trading closing prices from 2 June 2008 to 29 May 2009, in chronological order.

From each of first three data series, a new data series is generated by randomly swapping the data points in the original data series, while keeping the number of data points, box size and reversal amount unchanged. The resulting new data series are named with a postfix "-R" (DJI30-R, MSFT-R and KS11-R) to indicate that the data have been randomized and are not in chronological order. The following example illustrates randomization of a series of 40 data points obtained from DJI30 data series used in our study:

Original data series = (8000.86, 7936.83, 8078.36, 7956.66, 8063.07, 8280.59, 8270.87, 7888.88, 7939.53, 7932.76, 7850.41, 7552.60, 7555.63, 7465.95, 7365.67, 7114.78, 7350.94, 7270.89, 7182.08, 7062.93, 6763.29, 6726.02, 6875.84, 6594.44, 6626.94, 6547.05, 6926.49, 6930.40, 7170.06, 7223.98, 7216.97, 7395.70, 7486.58, 7400.80, 7278.38, 7775.86, 7660.21, 7749.81, 7924.56, 7776.18)

Randomized data series = (7350.94, 7062.93, 8000.86, 6875.84, 7552.60, 7936.83, 6626.94, 8078.36, 7270.89, 8063.07, 7486.58, 6763.29, 7956.66, 7114.78, 7932.76, 7223.98, 7395.70, 8270.87, 7888.88, 7776.18, 7216.97, 7850.41, 7939.53, 7465.95, 8280.59, 7749.81, 7182.08, 6547.05, 7555.63, 6926.49, 7365.67, 6930.40, 6726.02, 7400.80, 7278.38, 7775.86, 7660.21, 7924.56, 6594.44, 7170.06)

For the original financial time-series data, price of a given day is usually close to the previous day price. Randomizing the data points is done with the aim to produce a new data series with larger price moves between subsequent data points as well as more turning points in prices. These can be observed in the Line charts for the original data series and randomized data series points in Figure 7 and Figure 8, respectively.

B. Experiment Results

Five pre-release builds of Nextwave Software WPF charting component have been used for the experiments. They are identified by version numbers v0.0.2, v0.0.3, v0.0.4, v0.0.5 and v0.0.6 respectively. The testing process as described in Section V-A has been repeated on each build for all the six input data series listed in Table 1.

As the test results for the DJI30 and the KS11 input data series are identical, they are combined and presented in Table 2. The test results for MSFT are presented separately in Table 3 because three additional violations were detected by MSFT in build v0.0.5 compared to DJI30 and KS11. Table 4 presents the test results for the three randomized data series (DJI30-R, MSFT-R and KS11-R).

Table 2. Test Result for DJI30 and KS11 data series

Build Version	Assertion Conditions		Metamorphic Relations						
	(1)	(2)	MR1	MR2	MR3	MR4	MR5	MR6	MR7
v0.0.2	Fail	Pass	Fail	Fail	Fail	Pass	Pass	Pass	Pass
v0.0.3	Pass	Fail	Fail	Fail	Fail	Pass	Pass	Pass	Pass
v0.0.4	Pass	Fail	Fail	Fail	Fail	Pass	Pass	Pass	Pass
v0.0.5	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass
v0.0.6	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass

Table 3. Test results for MSFT data series

Build Version	Assertion Conditions		Metamorphic Relations						
	(1)	(2)	MR1	MR2	MR3	MR4	MR5	MR6	MR7
v0.0.2	Fail	Pass	Fail	Fail	Fail	Pass	Pass	Pass	Pass
v0.0.3	Pass	Fail	Fail	Fail	Fail	Pass	Pass	Pass	Pass
v0.0.4	Pass	Fail	Fail	Fail	Fail	Pass	Pass	Pass	Pass
v0.0.5	Pass	Pass	Fail	Fail	Fail	Pass	Pass	Pass	Pass
v0.0.6	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass

Table 4. Test results for randomized data series: DJI30-R, MSFT-R and KS11-R

Build Version	Assertion Conditions		Metamorphic Relations						
	(1)	(2)	MR1	MR2	MR3	MR4	MR5	MR6	MR7
v0.0.2	Fail	Pass	Fail	Fail	Fail	Pass	Pass	Pass	Pass
v0.0.3	Pass	Fail	Fail	Fail	Fail	Pass	Pass	Pass	Pass
v0.0.4	Pass	Fail	Fail	Fail	Fail	Pass	Pass	Pass	Pass
v0.0.5	Pass	Pass	Fail	Fail	Fail	Pass	Pass	Pass	Pass
v0.0.6	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass	Pass

From the results in Table 2, Table 3 and Table 4, it can be observed that Build v0.0.2 violated assertion (1), MR1, MR2 and MR3 in the testing process for all six data sets. Build v0.0.3 contains bug fix for violation of assertion (1). However, the testing results show that even though violation of assertion (1) is no longer a problem, violation of assertion (2) has been detected in addition to MR1, MR2 and MR3. The same observation can be made to build v0.0.4 which contains bug fix for violation of assertion (2).

Subsequent build, version v0.0.5, which contains bug fix for violation of assertion (2) passed all assertion checking and metamorphic relation verifications for DJI30 and KS11 data sets. However, violations of MR1, MR2 and MR3 were detected for MSFT data series. This is an interesting observation because both DJI30 and KS11 data series did not trigger the violations of MR1, MR2 and MR3 in build v0.0.5. However, their randomized counterparts (DJI30-R and KS11-R) successfully triggered the violations of MR1, MR2 and MR3 in build v0.0.5.

This observation suggests that the randomized data series (DJI30-R and KS11-R) have better fault detection capability than their corresponding original time-series data (DJI30 and KS11). Further inspection on output data points found that, for the same number of input data points, randomized data series produces more output data points on Point and Figure chart compared to real-life financial time-series data. More precisely, randomization results in larger price moves (that is, more figures ('X' and 'O')) and more turning point in prices (that is, more reversals to be plotted) on the Point and Figure chart, which increases the likeliness to trigger violation in assertion conditions and metamorphic relations if faults do exist in the chart. Therefore, we recommend randomization of real-life financial time-series data for effective fault detection.

Finally, testing on build v0.0.6 which contains bug fix for violation of MR1, MR2 and MR3 passed all assertion checking and output verifications based on the seven metamorphic relations. Debugging details will be discussed in the next section.

In summary, it can be observed that MR4 to MR7 have detected no fault in any build v0.0.2 to v0.0.6 in our experiments. MR4 to MR7 are based on the necessary software properties related to the number of columns on a Point and Figure chart. A new column is plotted on the Point and Figure chart when price reverses by more than multiplication of two user-defined variables (that is, reversal amount multiplying the box size). Therefore, more reversals will create more columns on the chart, and vice versa. Our current stage of testing showing no violation of MR4 to MR7, which would suggest that either more different source test cases are required to detect this kind of fault, or the software contains no fault relating to the reversal properties. On the other hand, assertion condition (2) and MR1 to MR3 based on the necessary properties related to the box size and the data points in the input data series, violation of these assertion condition and metamorphic relations means that there

exist fault(s) relating to the processing of the user-defined variable (box size) and data points in the input data series. In complement to the mentioned properties, assertion condition (1) was shown effective in detecting an incorrect plotting of the first data point on the chart (one of the output variables).

It is important that our list of assertion conditions and metamorphic relations can cover all the possible input variables and output variables to achieve a more comprehensive testing. Assertion conditions (1) and (2) as well as MR1 to MR 7 are only used to demonstrate the effectiveness of our testing approach in absence of an oracle for charting software. Ideally, developers should start developing assertion conditions and metamorphic relations once a software specification is ready, so that they can keep these properties in mind and can regressively test the software using the same or more refined set of properties to detect as many faults in as early stage as possible.

B. Debugging

The violations of assertion conditions and metamorphic relations indicate the presence of faults in the build versions of the Point and Figure chart under test. In the experiments, testing was done on the earliest version first, followed by the later versions. Based on the violations of properties (assertion conditions and metamorphic relations) observed, the debugging process is performed to locate and fix the bugs related to these properties in the charting components that have potentially resulted in the faults. Below, we report all identified bugs at the completion of the testing process.

Bug 1: Omission error in the implementation of Step 4 of the Point and Figure chart algorithm.

Build versions: This bug is reported in build v0.0.2.

Description: While implementing Step 4 of the algorithm, plotting of the first figure ('X' or 'O') prior to entering the inner while loops was omitted by mistake, as shown in Figure 9.

Detection: This bug results in possible violations of Assertion Condition (1) and MR1, MR2 and MR3.

Bug 2: Misplace of "increment *i*" statement in the implementation of Step 4 of the Point and Figure chart algorithm.

Build versions: This bug is reported in build v0.0.3, v0.0.4, v0.0.5.

Bug Description: While implementing Step 4 of the algorithm, the "increment *i*" statement was misplaced after plotting of a figure ('X' or 'O'), as shown in Figure 10. The "increment *i*" statement is supposed to be placed before the plotting of a figure ('X' or 'O').

Detection: This bug results in possible violations of Assertion Condition (2) and MR1, MR2 and MR3.

Bug 3: Initialization errors in the implementation of Step 2 of the Point and Figure chart algorithm.

Build versions: This bug is reported in build v0.0.2, v0.0.3, v0.0.4 and v0.0.5.

Bug Description: While implementing Step 2 of the algorithm, variables *columnNumber*, *i* and *j* and are wrongly initialized to *1* instead *0*, as shown in Figure 11.

Detection: This bug results in possible violations of Assertion Condition (1) and (2) as well as MR1, MR2 and MR3.

Bug 4: Insertion error in the implementation of Step 4 of the Point and Figure chart algorithm.

Build versions: This bug is reported in build v0.0.5.

Bug Description: While implementing Step 4 of the algorithm, the "decrement i" statement was inserted after the while loop, as shown in Figure 12.

Detection: This bug results in possible violation of Assertion Condition (2) and MR1, MR2 and MR3.

From our discussion with the charting component developer, it was found that Bug 2 and Bug 4 were mistakenly induced into the Point and Figure chart component in the attempts to fix existing bugs. This is an example of classical case where a bug fix gives rise to new bugs.

While analyzing the relationship between the bugs identified and the assertion conditions and metamorphic relations, we realized that an identified bug may not be the only cause for violations of assertion conditions and metamorphic relations. Violations can be caused by multiple bugs. After a series of tests in this paper, we cannot guarantee to detect all bugs contributing to the violation. It can only be assured after we exhaustively test all the possible inputs and necessary properties. However, this is prohibitively expensive and infeasible as known in software testing. Under this limitation, the combination of assertion checking and metamorphic testing technique proposed in this study becomes more important to maximize the chance of fault detection and to reduce the time and cost of testing by eliminating subjective human visual judgment from the testing process.

VI. EXTENDIBILITY TO OTHER FINANCIAL CHARTS

While the testing technique proposed in Section IV is designed for Point and Figure chart, it can be extended and applied to test other financial charts that have oracle problems as well. The data label extraction method is independent of the type of chart. Therefore, it can also be applied to test other types of charts. In this section, we will discuss the extendibility of assertion conditions and metamorphic relations proposed in Section IV to three other reversal charts, namely, Renko chart, Kagi chart and Three-line-break chart.

A. Renko Chart

The Renko chart gets its name from renga, the Japanese word for bricks [1]. It is plotted based on brick size, which is equivalent to the box size in a Point and Figure chart. However, the Renko chart does not have the equivalent of reversal amount in the Point and Figure chart since the default reversal amount is always fixed to one brick. Hence, plotting of a Renko chart is only influenced by one user-defined variable, that is, brick size.

Figure 13 shows a Renko chart that corresponds to the Line chart on Dow Jones Industrial 30 Index in Figure 1. Based on the above knowledge of the Renko chart, it is

evident that assertion conditions (1) and (2) defined in Section IV can be reused to test the Renko chart. Furthermore, all metamorphic relations can be reused to test the Renko chart except MR4 and MR5 that require manipulation of reversal amount.

```
4.  While direction = NULL AND j ≤ numberOfDataPoint
      if d_j ≥ (p_i + boxSize), then
        Set Direction = UP.
        Plot 'X' at p_i at the current columnNumber (Omission error in implementation)
        While p_i+boxSize ≤ d_j
          Increment i, set p_i=p_i-1 + boxSize
          Plot 'X' at p_i at the current columnNumber
        Endwhile
      Else if d_j ≤ (p_i – boxSize), then
        Set Direction = DOWN.
        Plot 'O' at p_i at the current columnNumber (Omission error in implementation)
        While p_i – boxSize ≥ d_j
          Increment i, set p_i=p_i-1 – boxSize
          Plot 'O' at p_i at the current columnNumber
        Endwhile
      Endif
      Increment j.
    Endwhile
```

Figure 9. Omission error in the implementation of Step 4 of the Point and Figure chart algorithm

```
4.  While direction = NULL AND j ≤ numberOfDataPoint
      if d_j ≥ (p_i + boxSize), then
        Set Direction = UP.
        Plot 'X' at p_i at the current columnNumber
        While p_i+boxSize ≤ d_j
          Increment i, set p_i=p_i-1 + boxSize,
          Plot 'X' at p_i at the current columnNumber
          Increment i (misplace of Increment i in the implementation)
        Endwhile
      Else if d_j ≤ (p_i – boxSize), then
        Set Direction = DOWN.
        Plot 'O' at p_i at the current columnNumber (Omission error in implementation)
        While p_i – boxSize ≥ d_j
          Increment i, set p_i=p_i-1 – boxSize,
          Plot 'O' at p_i at the current columnNumber
          Increment i (misplace of Increment i in the implementation)
        Endwhile
        Endif
      Increment j.
    Endwhile
```

Figure 10. Misplace of "increment *i*" statement in the implementation of Step 4 of the Point and Figure chart algorithm

```
2.  Initialize columnNumber=0,columnNumber=1, i=0, i=1, j=0, j=1, p_i = d_j and direction = NULL.
    (variables columnNumber, i and j were initialized wrongly in the implementation)
```

Figure 11. Initialization errors in the implementation of Step 2 of the Point and Figure chart algorithm

```
4.  While direction = NULL AND j ≤ numberOfDataPoint
      if d_j ≥ (p_i + boxSize), then
        Set Direction = UP.
        Plot 'X' at p_i at the current columnNumber
        While p_i+boxSize ≤ d_j
          Increment i, set p_i=p_i-1 + boxSize
          Plot 'X' at p_i at the current columnNumber
        Endwhile
      Else if d_j ≤ (p_i – boxSize), then
        Set Direction = DOWN.
        Plot 'O' at p_i at the current columnNumber
        While p_i – boxSize ≥ d_j
          Increment i, set p_i=p_i-1 – boxSize
          Plot 'O' at p_i at the current columnNumber
        Endwhile
      Endif
      Increment j.
    Endwhile
    Decrement i. (Insertion of Decrement i statement in the Implementation)
```

Figure 12. Insertion error in the implementation of Step 4 of the Point and Figure chart algorithm

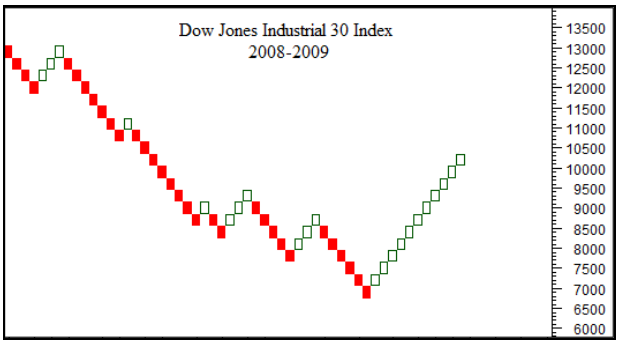

Figure 13. Renko chart with a brick size of 300.

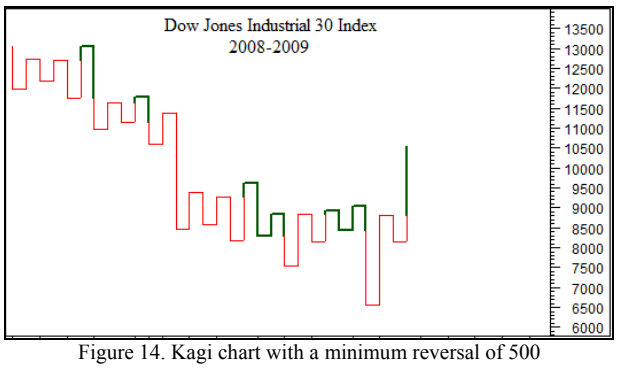

Figure 14. Kagi chart with a minimum reversal of 500

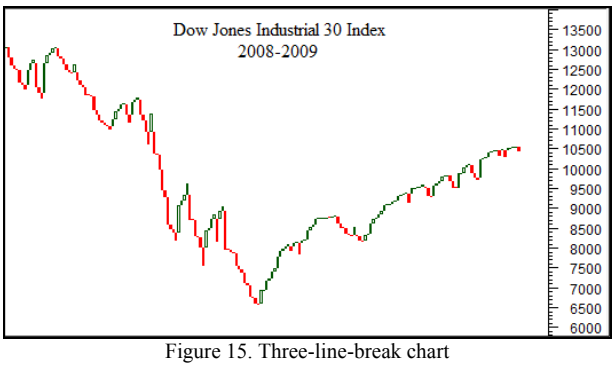

Figure 15. Three-line-break chart

B. Kagi Chart

Contrary to Renko chart, Kagi chart neither have a box size nor brick size. Kagi charts display a series of connecting vertical lines. If prices continue to move in the same direction, the vertical line is extended. Conversely, if prices reverse by a minimum reversal, a new Kagi line is then drawn in the opposite direction in a new column. Unlike a Point and Figure chart that requires the price to reverse by at least the reversal amount multiplying the box size to plot a new column, a Kagi chart only requires the price to reverse by the user-defined minimum reversal for the Kagi line to be plotted in a new column. Figure 14 shows a Kagi chart that corresponds to the Line chart on Dow Jones Industrial 30 Index in Figure 1.

Since the Kagi chart does not have a box size as in the Point and Figure chart, only assertion condition (1) can be reused to test the Kagi chart but not assertion condition (2). MR1, MR2 and MR3 can be reused without modification. A slight modification is required for MR4 and MR5. By replacing the reversal amount with the minimum reversal, MR4 and MR5 can also be used to test a Kagi chart. MR6 and MR7 cannot be reused to test a Kagi chart because they require manipulation of box size which does not exist in the Kagi chart.

C. Three-line Break Chart

Similar to the Kagi charts, a three-line break chart plots a series of vertical lines that are based on changes in prices. If price continue to move in the same direction exceeding the previous line, the line will be extended in the same direction (in a new column) by the amount of the price move. Therefore, the three-line-break chart does not require a box size. Typically, the price has to reverse by at least three lines for reversal to take place. This is the reason why this is named as Three-line-break chart. Figure 15 shows the Three-line-break chart corresponds to the Line chart on Dow Jones Industrial 30 Index in Figure 1.

As the three-line-break chart does not make use of box size, only assertion condition (1) can be reused for testing but not assertion condition (2). As for metamorphic relations, only MR1, MR2, and MR3 can be reused. MR4, MR5, MR6 and MR7 cannot be reused because they require manipulation of either box size or reversal amount.

VII. Limitations and Future Work

From the experiment results in Section V-B, we can observe that metamorphic relations that involve manipulation of box size and reversal amount (MR4, MR5, MR6 and MR7) have not detected any fault. It should be noted that in any stage, if no tests conducted can violate the assertion conditions and metamorphic relations defined, it does not mean that we have proved the correctness of the financial chart under test. This is because we have not been able to test all the possible inputs and all the possible necessary properties (known as a limitation of software testing). Six series of data points, two assertion conditions and seven metamorphic relations were used to demonstrate how our testing approach enables automatic testing of financial charts without relying on human visual judgment (error prone and labour intensive process).

It is worth noting that the assertion conditions and metamorphic relations identified in this paper are not exhaustive. Assertion conditions and metamorphic relations proposed here are merely some of the necessary properties of the financial chart under test. There are other possible assertion conditions and metamorphic relations that can be used to test Point and Figure charts. Determining the adequacy of assertion conditions and metamorphic relations is also a challenging problem in testing. On one hand, having more assertion conditions and metamorphic relations may increase the chance of fault detection. On the other hand, this will increase the cost of testing to a stage where there may not be sufficient resource to execute the tests related to the identified properties.

Selection of source test cases plays an important part in metamorphic testing. As observed in the experiment results, we noticed that randomized data series have

better fault detection effectiveness than the original financial time-series data. This suggests that fault detection effectiveness relies on the selection of source test case, which is the data series. Various types of data series could be explored in future studies to enhance the fault detection.

On the other hand, the data label extraction method used in this paper has effectively simplified the test output verification from graphical comparisons to numerical comparisons. However, this method assumes that the graphical APIs of the operating system is error free. Further, it is assumed that the pixel position of the charting screen output is calculated correctly. Even though data label is not affected by the charting screen output, correct screen output is still necessary to provide the correct visual display to the user. For example, suppose that the vertical axis of the chart has a range of 0 to 100 and occupies the length of 200 pixels. If the price to be plotted is 40, then the pixel position on the vertical axis should be calculated as (40/100)*200. This can be easily verified and tested because the expected output (test oracle) can be easily determined.

As for future work, we notice that despite its simplicity, the testing technique proposed for the Point and Figure chart can be easily extended to other financial charts. Beside financial charts, many other types of charts that have oracle problems can benefit from the testing technique proposed here. A generalized testing framework for charting software components can be developed for this purpose. In addition, we plan to study how metamorphic testing and assertion testing can benefit from new test case selection and generation techniques to improve fault detection effectiveness.

VIII. Conclusion

Despite the important role it plays in financial market trading, the quality of financial charts used in market analysis and trading software has been largely overlooked and taken for granted. Testing of financial charts is difficult due to the graphical complexity of its outputs and the oracle problem. Human visual judgment is often required to perform manual testing. This is both error-prone and labour intensive. We propose a new testing technique that combines metamorphic testing, assertion checking and a novel data label extraction method. Data label extraction allows charting output verification to be simplified from graphical comparison to numerical value comparisons, which can be easily automated. Hence, human visual judgment is no longer required in verifying charting output. The use of assertion checking and metamorphic testing has successfully alleviated the oracle problem in testing of Point and Figure chart. The deployment of test case generator, assertion checker and output verifier in the proposed technique allows the testing process to be fully automated.

To evaluate the effectiveness the proposed technique, we apply it to test five pre-release builds of Nextwave Software's Point and Figure charting component. Our experiment results show that the proposed testing technique has successfully detected actual faults in the Point and Figure charting component under test. From the experiment results, we observe that the effectiveness of metamorphic testing relies on the source test case used for testing. Specifically, we recommend randomization of real-life financial time-series data to improve fault detection for Point and Figure charts.

As the pilot study on testing of financial charts that have oracle problems, we believe that our work have made a significant contribution toward enhancement of quality of charts. As traders in the share, commodity and forex markets often rely on financial charts in making trading decisions, any fault in financial charting software could result in substantial financial losses. Therefore, it is critical that financial charting software is well tested before being deployed in live trading.

Acknowledgment

This project is supported by Malaysian Government MOHE FRGS (FRGS/2/2010/TK/SWIN/02/03) and an Australian Research Council Discovery Grant (ARC DP0984760).

References

[1] M. D. Archer and J. L. Bickford, *The Forex chartist companion: a visual approach to technical analysis*. John Wiley & Sons, Inc.: Hoboken, New Jersey, 2007.

[2] A. M. Alakeel, "An Algorithm for Efficient Assertions-Based Test Data Generation," *Journal of Software*, Vol. 5, no. 6, pp. 644-653, 2010.

[3] J. A. Anderson, "Point and figure charting: a computational methodology and trading rule performance in the S&P 500 futures market," *Int. Rev. of Financial Anal.*, Vol.17(1), pp. 198-217, 2008.

[4] B. Beizer, *Black-box testing. Techniques for Functional Testing of Software and Systems*, Wiley. 1995

[5] S. Beydeda, "Self-metamorphic-testing components," *Proceedings of the 30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, pp. 265−272, 2006.

[6] R. V. Binder, *Testing Object-Oriented Systems: Models, Patterns, and Tools*, Addison Wesley, 2000.

[7] L. C. Briand, M. D. Penta and Y. Labiche, "Assessing and improving state-based class testing: a series of experiments," *IEEE Trans Softw. Eng.*, Vol. 30 (11), pp. 770–793, 2004.

[8] F. T. Chan, T. Y. Chen, S. C. Cheung, M. F. Lau, S. M. Yiu, "Application of metamorphic testing in numerical analysis," *Proceedings of IASTED International Conference on Software Engineering*, pp. 191−197, 1998.

[9] W. K. Chan, T. Y. Chen, S. C. Cheung, T. H. Tse and Z. Zhang, "Towards the testing of power-aware software applications for wireless sensor networks," *Lect. Notes in Comput. Sci.*, Vol. 4498, pp. 84-99, Springer, Berlin, 2007.

[10] W. K. Chan, T. Y. Chen, H. Lu, T. H. Tse and S. S. Yau, "Integration testing of context-sensitive middleware-based applications: a metamorphic approach," *Int. J. of Softw. Eng. and Knowl. Eng.*, Vol. 16(5), pp. 677-703, 2006.

[11] T. Y. Chen, S. C. Cheung and S. Yiu, "Metamorphic testing: a new approach for generating next test cases," *Technical Report HKUST-CS98-01*, Department of Computer Science, Hong Kong University of Science and Technology, 1998.

[12] T. Y. Chen TY, J. Feng and T. H. Tse, "Metamorphic testing of programs on partial differential equations: a case study," *Proceedings of the 26th Annual International Computer Software and Applications Conference (COMPSAC'02)*, pp. 327-333, 2002.
[13] T. Y. Chen, J. W. K. Ho, H. Liu and X. Xie, "An innovative approach for testing bioinformatics programs using metamorphic testing," *BMC Bioinformatics*, Vol. 10(24), 2009.
[14] M. D. Davis and E. J. Weyuker, "Pseudo-oracles for non-testable programs," *Proceedings of the ACM '81 Conference*, pages 254-257, 1981.
[15] V. deVilliers, *The Point & Figure Method of Anticipating Stock Price Movements*, Windsor Books - Reprint of 1933 Edition, NY.
[16] M-C. Gaudel, "Testing can be formal, too," *Lect. Notes in Comput. Sci.*, Vol. 915, pp. 82–96, Springer-Verlag, Berlin, 1995.
[17] T. Gehrig and L. Menkhoff, "Extended evidence on the use of technical analysis in foreign exchange," *Int. J. of Finance & Econom*, Vol. 11(4), pp. 327-338, 2006.
[18] C. He, and L. Zheng, "Automatic Detection to the Behavioral Conflict in AOP Application Based on Design by Contract." *Journal of Software*, Vol. 6, No. 11, pp. 2255-2262, 2011.
[19] R. Helm, I. M. Holland and D. Gangopadhyay, "Contracts: specifying behavioral compositions in object-oriented systems," *ACM SIGPLAN Notices*, Vol. 25(10), pp. 169–180, 1990.
[20] S. Huang, M. Y. Ji, Z. W. Hui and Y. T. Duanmu, "Detecting Integer Bugs without Oracle Based on Metamorphic Testing Technique," *Applied Mechanics and Materials*, 121, pp. 1961-1965, 2012.
[21] J. Knight and N. Leveson, "An experimental evaluation of the assumption of independence in multi-version programming," *IEEE Trans Softw. Eng.*, Vol. 12(1), pp. 96-109, 1986.
[22] B. Marick, "When should a test be automated?" *Proceedings of the 11th International Software/Internet Quality Week*, 1998.
[23] C. Murphy, G. Kaiser, L. Hu and L. Wu, "Properties of machine learning applications for use in metamorphic testing," *Proceedings of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE'08)*, pp. 867–872, 2008.
[24] C. Murphy, M. S. Raunak, A. King, S. Chen, C. Imbriano, G. Kaiser, I. Lee, O. Sokolsky, L. Clark and L. Osterweil, "On effective testing of health care simulation software," *Proceedings of the 3rd Workshop on Software Engineering in Health Care*, pp. 40-47. ACM, 2011.
[25] K.Y. Sim, C.S. Low and F.-C. Kuo, "Automatic Testing of Financial Charting Software Component: A Case Study on Point and Figure Chart," *Proceedings of the 2nd International Conference on Computer Science and its Applications (CSA'09)*, pp. 177-183, Jeju, Korea, 2009.
[26] K.Y. Sim, W. K. S. Pao, and C. Lin. "Metamorphic testing using geometric interrogation technique and its application." *Proceedings of the 2nd International Conference of Electrical Engineering/Electronics, Computer, Telecommunications, and Information Technology (ECTI'05)*, pp. 91-95. 2005.
[27] K.Y. Sim, D. M. L. Wong, and T. Y. Hii, "Evaluating the Effectiveness of Metamorphic Testing on Edge Detection Programs," *International Journal of Innovation, Management and Technology*, Vol. 4(1), pp. 6-10, 2013.
[28] C. A. Sun, G. Wang, B. Mu, H. Liu, Z. Wang and T. Y. Chen, "A Metamorphic Relation-Based Approach to Testing Web Services Without Oracles," *International Journal of Web Services Research (IJWSR)*, Vol. 9(1), pp. 51-73, 2012
[29] J. Takahashi, "An automated oracle for verifying GUI objects," *ACM SIGSOFT Softw. Eng. Notes*, Vol. 26(4), pp. 83–88. 2001.
[30] M. P. Taylor and H. Allen, "The use of technical analysis in the foreign exchange market," *J. of Int. Money and Finance*, Vol. 11, pp. 304-314. 1992.
[31] R. N. Taylor, "Assertions in programming languages," *ACM SIGPLAN Notices*, Vol. 15(1), pp. 105−114, 1980.
[32] Q. Xie and A. M. Memon, "Designing and comparing automated test oracles for GUI-based software applications," *ACM Trans Softw. Eng. and Methodol.*, Vol. 16(1), Article No.4, 2007.
[33] B. Zacharias, "Test Case Generation and Reusing Test Cases for GUI Designed with HTML," *Journal of Software*, Vol. 7, No. 10, pp. 2269-2277, 2012.
[34] Z. Zhang, W. K. Chan, T. H. Tse and P. Hu, "An experimental study to compare the use of metamorphic testing and assertion checking," *J. of Softw. ISSN 1000-9825*, Vol. 20(10), pp. 2637–2654, 2009.
[35] Z. Q. Zhou, T. H. Tse, F.-C. Kuo, T. Y. Chen, "Automated functional testing of web search engines in the absence of an oracle," *Technical Report TR-2007-06*, Department of Computer Science, The University of Hong Kong, Hong Kong.

Kwan Yong Sim (member of IET, IEEE and IEEE Computer Society) received his Bachelor of Engineering in Electrical, Electronics and Systems from the National University of Malaysia in 1999 and the Master of Computer Science from the University of Malaya, Malaysia in 2001. Currently, he is a Senior Lecturer in the Faculty of Engineering, Swinburne University of Technology, Sarawak Campus, Malaysia. His current research interests include software testing and debugging.

Chin S. Low received his Bachelor of Engineering in Electrical, Electronics and Systems from the National University of Malaysia in 1999 and the Master of Computer Science from the University of Malaya, Malaysia in 2000. He is currently a technical lead for Nextwave Software, Malaysia. His current research interests include interactive multimedia and interface design.

Fei-Ching Kuo (member of IEEE Computer Society) received her Bachelor of Science Honours in Computer Science and PhD in Software Engineering, both from Swinburne University of Technology, Australia. She was a lecturer at University of Wollongong, Australia. She is currently a Senior Lecturer at Faculty of Information and Communication Technologies, Swinburne University of Technology, Australia. She is also the Program Committee Chair for the 10th International Conference on Quality Software 2010 (QSIC'10) and Guest Editor of a Special Issue for the Journal of Systems and Software, special issue for Software Practice and Experience, and special issue for International Journal of Software Engineering and Knowledge Engineering. Her current research interests include software analysis and testing.