Proposed Bilingual Model for Right to Left Language Applications

Farhan M Al Obisat
Computer and Information Technology Dept.
Tafila Technical University
Tafila, Jordan

Zaid T Alhalhouli
Computer and Information Technology Dept.
Tafila Technical University
Tafila, Jordan

Hazim S. AlRawashdeh
Department of Computer Science
Buraydah Private Colleges
Buraydah, Saudi Arabia

***Abstract*—Using right to left languages (RLL) in software programming requires switching the direction of many components in the interface. Preserving the original interface layout and only changing the language may result in different semantics or interpretations of the content. However, this aspect is often dismissing in the field. This research, therefore, proposes a Bilingual Model (BL) to check and correct the directions in social media applications. Moreover, test-driven development (TDD) For RLL, such as Arabic, is considered in the testing methodologies. Similarly, the bilingual analysis has to follow both the TDD and BL models.**

Keywords—software; testing; languages; right to left; development; application; bilingual; social media

I. Introduction

Test-driven development (TDD) comprises the major component of the values of the Agile Manifesto and the agile development drives from Extreme Programming (XP). However, TDD is not original. In fact, TDD was mention in the NASA Project Mercury, which was launch in the 1960s [1]. Some encouraging properties are reported to be attainable with the use of TDD. Moreover, while it is often considered a testing method, TDD is also design and development method where in tests already written before the code to ensure an error-free code.

In TDD, tests are added to the code. Then, this is restructured to achieve better internal structure as soon as the test is successfully passed. Usually, this process is iterated several times until all functions are fully verified to be well implementing.

Any software development process encompasses the following main activities:

1) problem analysis (specification),
2) Software design,
3) Software implementation,
4) Software testing,
5) Software maintenance, and
6) Software operation.

The TDD consists of the following six basic steps:

1) Writing a test meant for a part of functionality,
2) Running the tests to check whether the output of the new test would fail,
3) Writing codes aims to pass the tests,
4) Running the test to check whether they pass or not,
5) Code rewriting, and
6) Running the tests to check if the rewriting did not alter the external behavior [2].

The first step is known as test writing, and it includes writing a code for the purpose of testing the function or functionality of the tasks. The second stage is performed to confirm whether the test is working correctly (this means that the test should not fail at this point as functionality has yet to be implemented). When the test passes at this phase, the test is incorrect and it needs rewriting and validation. The third step involves writing the code into short segments so that it can effectively pass the test. Finally, all the tests must be run to confirm whether any desired functionality has been implemented. The internal structure of the code should be further improved through rewriting when all tests pass [3].

At this point, researchers perform the TDD test first. To answer the question as to why this must be done first, one should consider the key benefits behind adopting this model. The advantages of the TDD test are listed below.

1) TDD test can capture the intent of the developer or domain expert (e.g., about RTL languages)
2) It allows thinking about program design.
3) It ensures that the tests are written (and real)
4) It provides a higher quality code and runs faster because it has fewer integration problems.
5) The Ping Pong Pair Programming-style TDD leads to better distribution of knowledge in the team and reduces the "truck factor" (worst case scenario).

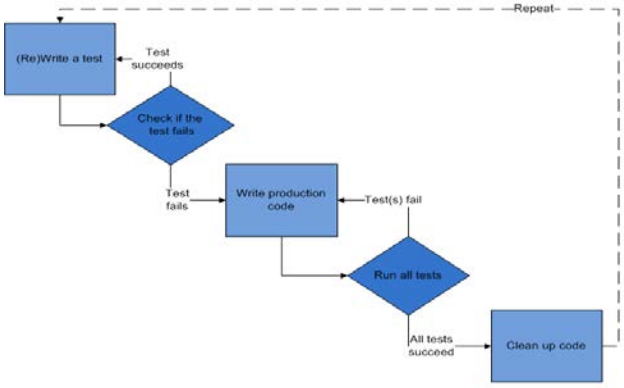

Fig. 1. Test-driven development cycle diagram [4]

TDDs are mostly coded for left to right languages with no consideration for other languages' directions, such as a right to left languages (RLL) or top-down languages, to name a few. In this work, researchers establish the many issues to be used when developing software for RLL applications.

As mentioned earlier, TDD is a software development procedure consisting of short development cycles that are repeated. At first, the developer creates an (initially failing) automated test case, which delineates a targeted improvement (or a new function). Then, the developer creates the minimal number of codes to pass that test. Finally, the new code is refractor to acceptable standards [4].

Software testing is an activity that assesses the features of a code and guarantees that the resulting code meets the essential output. Accurately finding all the errors (semantic or syntaxes) in the program is a difficult task. The choice of a correct approach at the right time will result in an efficient and effective software testing procedure [5].

"Software testing" refers to a process of the running code with the goal of finding errors. This method adapts test case designs into execution steps that are well-planned. This leads to the design and creation of successful software. The goal of software testing is to uncover the possible errors that may be present in the software. Thus, the main goal of a test case is to generate a set of tests with the highest likely hood of uncovering the errors. Also, software testing ensures that the computer code does what it must do. This is similar to a destructive process of identifying errors. The purposes of software testing may include reliability estimation, validation, quality assurance, or verification [6].

Software testing also assesses codes with the purpose of checking out errors within it, Software testing is a method that aims to evaluate a capability or attribute of a code or product and determine whether it satisfies quality requirements. Software testing is similarly employed to test the code for other factors use to assess software quality, such as usability, reliability, maintainability, integrity, capability, efficiency, security, portability, compatibility, and so on [7].

Software production includes developing codes according to assured requirements. Software testing is performed to validate and verify whether the code has been designed to satisfy these specifications [8]. Software testing for RLL apps has yet to be completely investigated by researchers in this field. Hence, this paper tries to establish a software testing approach for RT languages.

In the next sections the paper clarified the problem statement in details such as why bidirectional is important, and why the developers need to test their applications before pilot any developed software. Also, the research described test cases from software Facebook and Bocketcode. In the last section the Proposed Bilingual Model (BL Model) where discussed in detail.

II. Problem Statement

All the TDDs are mostly coded for a left to right languages with no consideration for other language directions, such as RLL or top-down languages, to name a few. In this study, the work team will focus in TDD for RTL such as Arabic, Hebrew, Farsi, Urdu, and more. Right to left (RTL) text is supported in widespread consumer software. Often, this support is explicitly enabled. Thus, mixing RTL with the left to right (bidirectional) text is necessary.

There are many user interfaces (UI) points to consider in dealing with RTL languages. These components involve the following:

- Arrow direction
- Forms
- Text fields
- Dropdown fields (list/menu/jump menu)
- Scrollbars
- Data entry fields
- Checkbox fields
- Radio buttons
- Bulleted and numbered lists
- Buttons
- Labels
- Pocket Code: Bricks and formulas
- Facebook (direction of the post, i.e., arrow)

A. *Why perform tests for RTL features?*

Several reasons exist as to why tests must be executed for RTL features. These reasons are listed below.

- Developers often have little knowledge about how RTL languages are rendered.
- Later changes in the source code (i.e., refactoring) can result in layout problems in other languages, especially RTL, that are already solved in earlier versions.
- Tests are important in documenting coding decisions relevant to these other languages.
- Automatic tests allow quick locating problems and also hedging against reoccurring bugs (i.e., regression tests).

III. An Experience Report (Pocket Code, Facebook, and Scratch)

Pocket Code is a software use to create applications and games especially for students dealing with school works through their smart phones, Figure 2 Snapshot from Pocket code with the bidirectional feature.

Interface for mathematical operation in Pocket Code where the places of x and y are in the left of the interface but should be in the right position as in Figure 3 Interface and direction of the text. User interface for Pocket Code with mathematical sin() function and the places of x and y are on the left of the interface but should be in the right position figure 4.

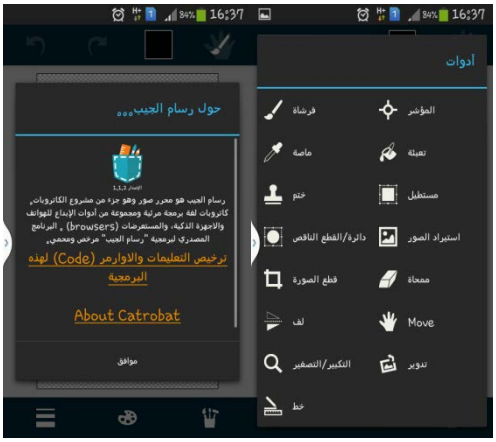

Fig. 2. Snapshot from Pocket code with bidirectional feature

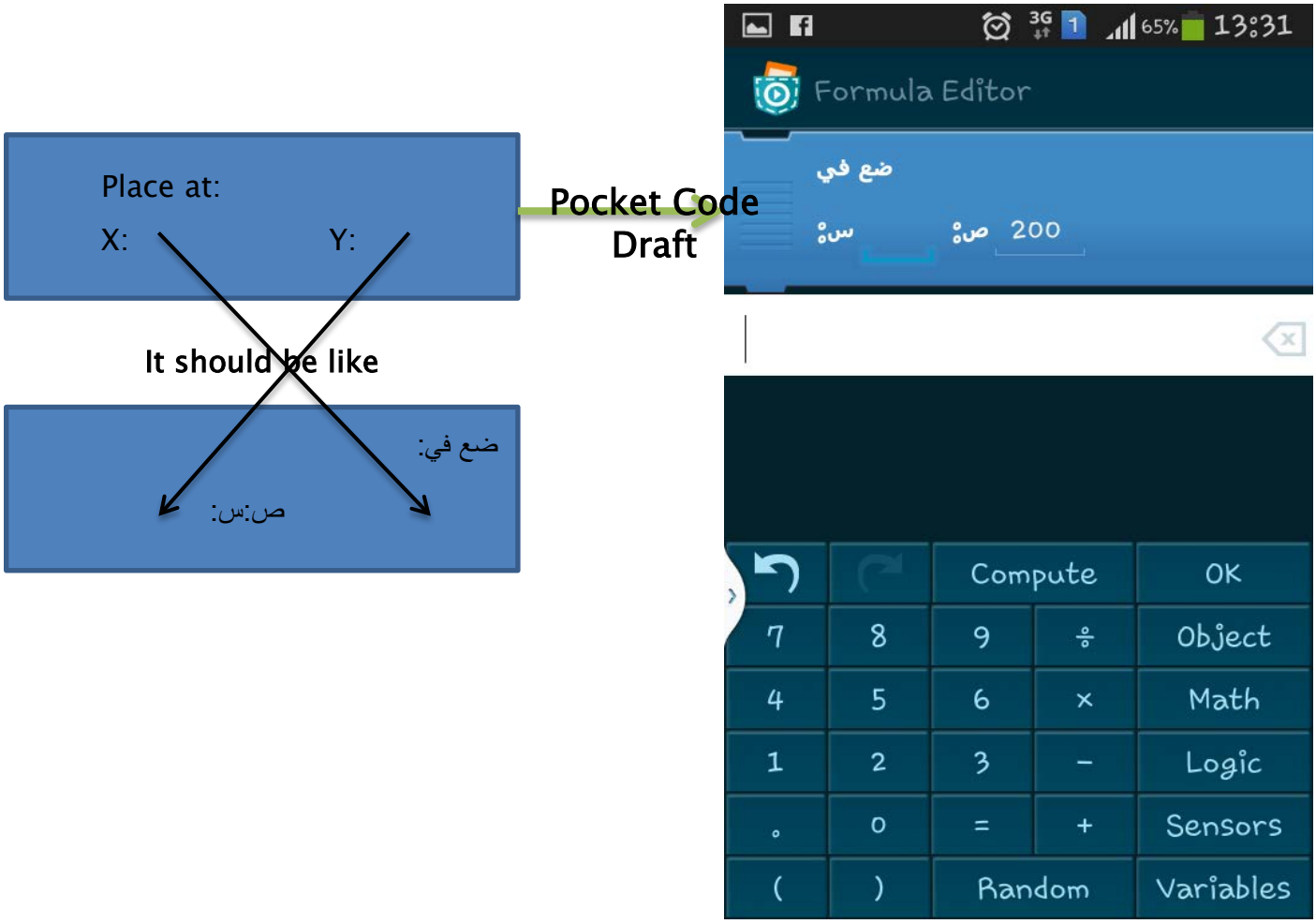

Fig. 3. Interface and direction of the text

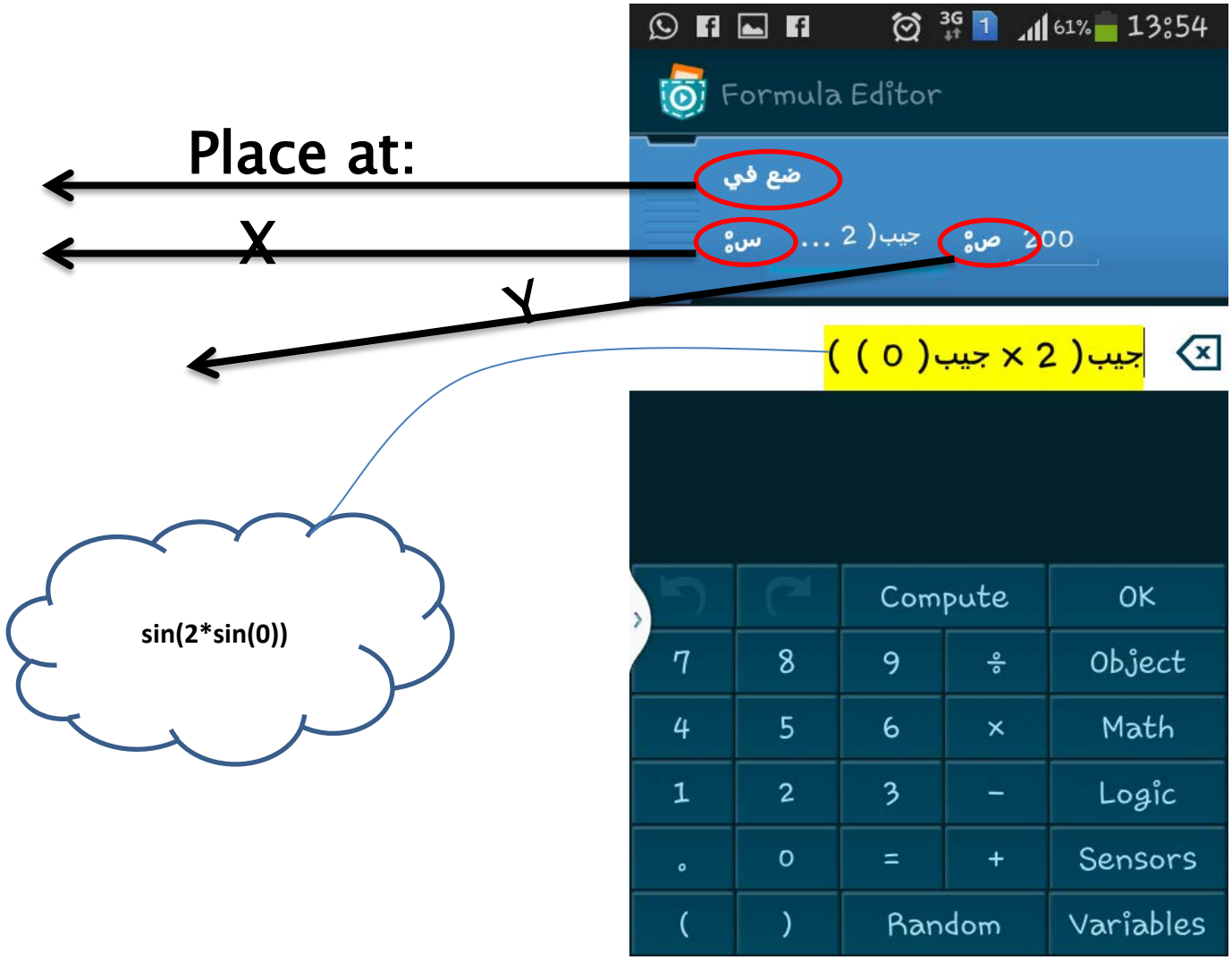

Fig. 4. Interface with mathematical operation

In Figure 5, Pocket Code accepts the equation as it appears in the snapshot from the Pocket Code where there is just one parenthesis opened, and the number of closed ones is three; the places of x and y are on the left of the interface but should be in the right position.

Fig. 5. Pocket Code equation

In figure 6, we can see that the direction of the arrow is wrong for RTL languages, and the position of the photo for the left snapshot should be on the right of the post.

Fig. 6. Snapshots from Facebook

IV. Proposed Bilingual Model (BL Model)

In the previous section, the study discussed the RTL problem, which is one of the strongest problems faced by researchers in this area. As shown in Figure 7, the problems appear in the directions of the profile image, sender and receiver information, and the publishing space for users. All these problems come from using different languages that also have different directions. Accordingly, researchers tried to construct a new model to solve this problem. Figure 8 clarifies the main components and steps or sequences of the processes that reduce the bidirectional problems in social media.

Fig. 7. Users posts in Social Media applications before applying the BL Model

The new model consists of nine phases as shown in Figure 8. The first and second phases are related to the language for both applications and user input so it is important to determine whether the languages used in this input, are compatible with the application language. In turn, this can help determine the correctness of the direction of the post. Accordingly, the test of the language compatibility will be conducted in the next phase. If the test succeeds, then no change is required, and the code is clean; otherwise, if the test fails, then there is a problem that must be investigated.

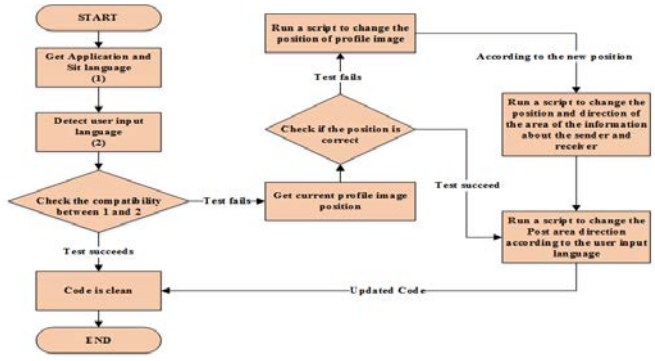

Fig. 8. Proposed Bilingual Model (BL) for social media applications

If a problem appears in the compatibility of the languages, the model must obtain the current location of the profile image and its details. In this stage, the model will retrieve the following information: Division width (DW), image width (IW), offset top (X), and offset left (Y) (see Figure 9). The model checks the validity of the position of the profile image according to the retrieved information. In case the location of the image is incorrect, the model will apply two different scripts: the first points the new location and the second changes the direction of the post area according to the user input language. Also, the model will proceed to apply the second script directly if the test succeeds. In the end, the interfaces of social media applications are supposed to appear, as shown in Figure 9, 10, after applying the proposed BL Model.

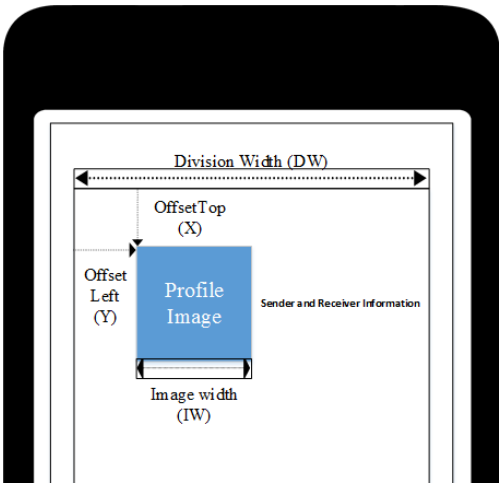

Fig. 9. Profile image details in social media applications

Fig. 10. User posts in social media applications after applying the BL Model

V. Future Works

Researchers and software developers in general do not consider testing for RLL applications. This work established new issues concerning RLL applications, which should be considered when testing the newly developed applications. This research work also presents the proposed BL model, which requires more testing and validation. Our future goal is to examine these issues to develop additional techniques to test the RTL or bidirectional language applications.

VI. Conclusion

The goal of this research is to establish and consider TDD issues related to RLL apps. Researchers have also shown that there are many points that should be considered when

developing software for RTL or bidirectional languages. In this work, we discuss cases from real working software, such as Pocket Code and Facebook, in which research considered forms, text fields, drop-down fields (list/menu/jump menu), scrollbars, arrow direction, data entry fields, checkbox fields, radio buttons, bulleted and numbered lists, buttons, photo positions, labels, and places of all objects in the interface.

REFERENCES

[1] http://projekter.aau.dk/projekter/files/204129305/Report_swd903e13_.pdf (2014)

[2] Shrivastava DP, Jain RC. Metrics for Test Case Design in Test Driven Development. International Journal of Computer Theory and Engineering. 2010 Dec 1;2(6):952.

[3] Kumar S, Bansal S. Comparative study of test driven development with traditional techniques. IntJ Soft Comput Eng (IJSCE). 2013;3(1):2231-307. http://en.wikipedia.org/wiki/Test-driven_development.

[4] Khan ME. Different forms of software testing techniques for finding errors. International Journal of Computer Science Issues. 2010;7(3):11-6.

[5] Thakare S, Chavan S, Chawan PM. Software Testing Strategies and Techniques. International Journal of Emerging Technology and Advanced Engineering. 2012.

[6] Sawant AA, Bari PH, Chawan PM. Software testing techniques and strategies. International Journal of Engineering Research and Applications (IJERA). 2012 May;2(3):980-6.

[7] Batra S. Improving Quality using testing strategies. Journal of Global Research in Computer Science. 2011 Jul 7;2(6):113-7.