Migrating ExcelForms to the .NET Framework

Flora Mei Ling Tan       Margaret M. Burnett
ftan@eecs.oregonstate.edu
June 09, 2006

A technical report submitted to the
School of Electrical Engineering and Computer Science
Oregon State University

Technical Report # CS06-60-02

Presented: June 09, 2006
Copyright © 2006 Flora Mei Ling Tan

Permission is hereby granted, free of charge, to any person obtaining a copy of this
document, to deal in the document without restriction, including the rights to copy,
modify, publish, distribute, and/or sell copies, provided the above copyright notice is
included in all copies.
ABSTRACT

ExcelForms is a front end Excel-based application that supports Forms/3, a research language based on the spreadsheet paradigm, end-user software engineering features. The old implementation of ExcelForms performed poorly, and was considered unstable, not robust, and not scalable enough for our users’ needs. This project addresses these issues by implementing ExcelForms on the .NET framework.
ACKNOWLEDGEMENTS

First and foremost, I would like to thank Dr. Margaret Burnett, my major professor, for all her help and guidance on writing this technical report. Also, special thanks to Dr. Rajeev Pandey and Dr. Carlos Jensen for graciously agreeing to be on my committee. Lastly, I want to thank Andrew Stucky for helping with the ExcelForms .NET implementation and other Forms/3 group members who have supported our work.
# TABLE OF CONTENTS

1  Introduction .................................................................................................................. 1  
   1.1  Problem with the Old Implementation.................................................................. 1  
   1.2  Solution ............................................................................................................... 4  
      1.2.1  VBA vs. VB .NET................................................................................... 5  
      1.2.2  Windows Forms Controls ....................................................................... 6  
   1.3  Organization of this Document........................................................................... 6  
2  Background and Related Work .................................................................................... 8  
   2.1.  Forms/3.............................................................................................................. 8  
      2.1.1  What You See Is What You Test ......................................................... 8  
      2.1.2  Fault Localization............................................................................... 9  
      2.1.3  Help Me Test....................................................................................... 9  
      2.1.4  Assertions............................................................................................ 9  
      2.1.5  Surprise-Explain-Reward ................................................................. 9  
      2.1.6  Regions ............................................................................................. 10  
   2.2.  ExcelForms........................................................................................................ 10  
   2.3.  Definitions, Acronyms, and Abbreviations ...................................................... 10  
3  Exploring Other Solutions............................................................................................ 12  
   3.1  Shapes and Comments in Excel......................................................................... 12  
   3.2  Active Template Library ................................................................................. 13  
   3.3  Piccolo ............................................................................................................. 14  
4  System Requirements Specifications............................................................................ 16  
   4.1  Introduction ...................................................................................................... 16  
      4.1.1  Scope of the System Specification................................................... 16  
      4.1.2  Purpose ............................................................................................. 16  
      4.1.3  Definitions, Acronyms, and Abbreviations ..................................... 16  
      4.1.4  References....................................................................................... 16  
   4.2  Overall Requirements...................................................................................... 16  
      4.2.1  Architectural Requirements and Constraints ................................ 17  
      4.2.2  System Perspective ............................................................................. 17
6.2.1 ExcelForms-Specific Messages ......................................................... 28
6.2.2 Connection User Interface ................................................................. 30
6.3 User Interface Design................................................................................ 31
6.3.1 Floating decision box ........................................................................ 31
6.3.2 Image Decision Box ......................................................................... 33
6.3.3 Tooltips............................................................................................. 34
6.3.4 Dashed borders.................................................................................. 34
6.4 WYSIWYT User Interface ........................................................................ 35
6.5 Fault Localization User Interface ............................................................... 35
6.6 HMT User Interface ................................................................................ 36
6.7 Assertions User Interface ........................................................................ 36
6.8 Explanations User Interface....................................................................... 36
6.9 Arrows User Interface ............................................................................... 37
7 Implementation and Deployment ...................................................................... 38
7.1 Implementation ......................................................................................... 38
7.1.1 Communications ............................................................................... 38
7.1.2 Floating Decision Box....................................................................... 38
7.1.3 Image Decision Box ......................................................................... 39
7.1.4 Dashed Borders.................................................................................. 39
7.1.5 Tooltips............................................................................................. 40
7.1.6 Toolbars ............................................................................................ 41
7.1.7 Extended Worksheet Conversion....................................................... 42
7.2 Performance Tests..................................................................................... 43
7.3 Deployment .............................................................................................. 45
7.3.1 Developer’s version........................................................................... 45
7.3.2 Release version.................................................................................. 46
7.4 Maintenance.............................................................................................. 46
7.4.1 CVS .................................................................................................. 46
7.4.2 Jira .................................................................................................... 47
7.4.3 Wiki .................................................................................................. 47
8 Conclusion and Future Work ................................................................. 48
References ........................................................................................................ 49
LIST OF TABLES

Table 1: Loading a spreadsheet with 1000 formula cells. ................................................. 3
Table 2: Loading a spreadsheet with 250 formula cells. .................................................... 4
Table 3: Windows Forms Controls vs. OLE Objects ............................................................. 6
Table 4: Comparing the performance of 900 ATL and OLE Objects. ............................. 13
Table 5: Messages specific to ExcelForms ................................................................. 29
Table 6: ExcelForms .NET vs. ExcelForms VBA ............................................................ 44
LIST OF FIGURES

Figure 1: The old ExcelForms implementation. .......................................................... 2
Figure 2: Shape and comments. ................................................................................ 12
Figure 3: An example program using Piccolo with zooming capability. ............... 14
Figure 4: Integrate Piccolo with Excel. ................................................................. 15
Figure 5: The three top nodes are alternative UI components. Only one can be used at a time. .......................................................... 17
Figure 6: Project Schedule ..................................................................................... 26
Figure 7: The developer’s version ......................................................................... 30
Figure 8: The release version ............................................................................... 31
Figure 9: Floating decision box ............................................................................. 32
Figure 10: Twin boxes with check-mark and X-mark ............................................. 32
Figure 11: Check-mark and X-mark button on the toolbar ..................................... 32
Figure 12: Check-mark and X-mark pop-up menu ................................................ 33
Figure 13: Image decision boxes and dashed borders ........................................... 34
Figure 14: Testedness Overview progress bar ....................................................... 35
Figure 15: Fault Localization ............................................................................... 36
Figure 16: Fault Likelihood Overview progress bar .............................................. 36
Figure 17: Show or hide arrows button on the toolbar .......................................... 37
Figure 18: Arrows between cells .......................................................................... 37
Figure 19: Floating decision box, image decision box and dashed borders ............ 39
Figure 20: The shaded borders are simply implemented using Cell properties ...... 40
Figure 21: Setting the tooltip text in XLBox ......................................................... 40
Figure 22: ExcelForms toolbars .......................................................................... 41
Figure 23: The toolbar button event code for the Enable WYSIWYT button ....... 42
Figure 24: Extended worksheet conversion ......................................................... 43
Figure 25: Spreadsheet load times ...................................................................... 45
1 Introduction

This project is about migrating ExcelForms, a user interface that supports the Forms/3 end-user software engineering features in Excel, from VBA to the .NET framework. The old ExcelForms had issues with poor performance and unpredictable crashes. As a result, the old ExcelForms was unstable, not robust, and not scalable enough for our users’ needs. It was necessary that we find a solution to resolve these problems because it was hindering the studies conducted by the research group. We explored different technologies while maintaining compatibility with the Excel framework. Our findings led us to implement ExcelForms on the .NET framework.

1.1 Problem with the Old Implementation

The main problem is the poor performance of the old ExcelForms implementation, as shown in Figure 1, when users test and debug a large spreadsheet with many formula cells. The two issues with the old implementation that we identified and wished to address were the robustness and the scalability of the system. The lack of robustness in ExcelForms has caused endless frustration among developers and end users. For the developers, the use of Visual Basic for Applications (VBA) to embed OLE Objects in the large spreadsheet has caused ExcelForms to crash frequently. In addition, our VBA development environment was unstable and it often hindered our work to debug the problem. The lack of robustness for end users meant that ExcelForms was unresponsive on the screen, and the end users did not know how get ExcelForms to successfully run again. As a result, the developers ended up providing on-site support every time there was a study session. In terms of scalability, loading a large spreadsheet often resulted in ExcelForms crashing or suspending indefinitely. The goal we are ultimately aiming for is to allow end users to load spreadsheets from their daily work in ExcelForms.

We conducted several tests on timings to isolate the cause of the performance problem. Based on our findings, we explored different solutions within the Excel
framework to resolve the problem. After discarding various alternative solutions, we finally decided that porting the old implementation to the .NET framework would overcome the robustness and scalability issues. Once the decision was made, we began migrating ExcelForms to the .NET framework. The details of this process are as follows.

To standardize our test environment, we ran all the time tests on *burnett-server*. The server has the following hardware specifications: Intel Xeon 3.40GHz CPU with 2GB RAM. The *burnett-server* machine runs on the Windows 2003 Server operating system. It has Excel 2003 and Visual Studio Tools for Office (VSTO) 2005 installed. These were exploratory; we first tested one of each. In Chapter 7, we followed a more rigorous methodology on the performance test of ExcelForms with VBA/Excel 2003 vs. ExcelForms with VSTO 2005/Excel 2003 on the .NET framework. Our
investigations revealed three source of difficulty: OLE Objects, screen updating, and tooltips.

Our time tests revealed showed that having OLE Objects embedded in the spreadsheet as a user interface device was the main culprit for the slowdown of the ExcelForms’ performance. Each OLE Object represents a decision box displayed in each formula cell. We tested the ExcelForms VBA implementation by loading of a spreadsheet with 1000 formula cells with and without OLE Objects. Table 1 indicates that it took four times longer to load a spreadsheet with OLE Objects as decision boxes.

**Table 1: Loading a spreadsheet with 1000 formula cells.**

<table>
<thead>
<tr>
<th>Library</th>
<th>Time (Minutes)</th>
<th>Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>OLE Objects with <em>ScreenUpdating</em> set to false.</td>
<td>22:00</td>
<td>Although all the decision boxes are painted at once, excessive flickering occurred on the screen.</td>
</tr>
<tr>
<td>No OLE Objects with <em>ScreenUpdating</em> set to false.</td>
<td>7:00</td>
<td>Huge compromise on the user interface and excessive flickering on the screen.</td>
</tr>
</tbody>
</table>

*ScreenUpdating* is a property in Excel that controls the painting of screen display while a macro is running [15]. In other words, with *ScreenUpdating*, we can control the frequency of UI devices or changes to the cell values to be painted on the screen. In Table 2, we found that having the property turned off until all decision boxes are created before painting them on the screen significantly reduces the slowdown in performance. However, this caused excessive screen flickering occurring on the screen, due to the programmatic logic of how ExcelForms processes the batches of messages received from the Forms/3 engine.
Because the built-in tooltips did not support the system’s requirements, tooltips were manually created in the old ExcelForms implementation. Although tooltips did not cause a performance bottleneck, they were irritating to the users because they did not exhibit the normal behavior users expect. This was due in part to some deficiencies in the built-in event reporting mechanism, and some mouse events were coded by hand instead. In one example, tooltips tended to freeze at one spot long after a user has mouse over a decision box.

### 1.2 Solution

This section discusses the reasons to replace the ExcelForms VBA implementation with the .NET version. Although some of the problems described can be resolved with VBA, we decided to port ExcelForms to the .NET framework. With the .NET framework, we hope to resolve the issues of instability, unpredictable crashes, and robustness plaguing our VBA implementation. The .NET framework also promises potential growth and benefits in customizing Microsoft Office application and it is actively advocated by Microsoft. In contrast, VBA development seems rather stagnant and its future is uncertain. Lastly, the .NET development tool, Visual Studio, offers a superior programming, testing, and debugging environment compared to the Visual Basic editor.
1.2.1 VBA vs. VB .NET

Visual Basic for Applications (VBA) was created to allow users to automate simple tasks in Excel using macros, create user defined functions in spreadsheets, and make use of ActiveX Controls. A popular task in Excel is when users record macros by demonstrations, which causes the VBA code to be generated automatically. Since VBA is a subset of the Visual Basic programming language, these macros may be used to call Excel built-in functions or made available as user defined functions. However, VBA solutions are intended for customizing standalone spreadsheets by novice programmers and there is hardly any support for concerns such as robustness, performance, scalability, and latest technological improvements.

In 2003, Microsoft introduced Visual Studio Tools for Office (VSTO) for developing Microsoft Office applications on the .NET platform. This presented an alternative solution to developing Excel-based applications using the .NET framework instead of VBA. The latest VSTO 2005 is a rich integrated development environment (IDE) tool meant for professional software developers to develop more complex and enterprise level Office applications. VSTO also provides a better programming environment for software developers to develop and debug their Excel application. Software developers can now leverage the .NET framework with its associated class libraries, technologies and languages to develop a more sophisticated Excel application.

As we transition from VBA to .NET, we hope to reap the benefits of a more stable, robust, and maintainable Excel application. One of the advantages of using the .NET framework is the support of various programming languages such as Visual Basic or C#. These languages support the Object Oriented Programming (OOP) paradigm which is not always the case with VBA. In having a better OOP design and implementation, we predicted that using VSTO would make it easier to talk to the Excel Object Model layer. The Excel Object Model is based on how Excel is implemented with its objects and their hierarchical relationships. Besides that, the .NET framework provides support for multi-threading [3] which VBA does not.
feature supposedly improves the responsiveness and the performance of an Excel application.

### 1.2.2 Windows Forms Controls

We tested the performance of creating forms controls versus OLE Objects on the .NET platform. To be complete, we also included tests on the performance of creating OLE Objects in VBA. As shown in Table 3, the performance of creating OLE Objects is slightly better on the .NET platform compared to VBA. In general, forms controls performed poorly against OLE Objects, and because of this, we cannot afford to have forms control in every formula cell. Instead, we decided to have a single forms control that would appear every time a formula cell is active. We call this UI device a floating decision box because it follows wherever the user selects a formula cell on the spreadsheet. The floating decision box is described with greater detail in Section 6.

<table>
<thead>
<tr>
<th>Number of Decision Boxes (row x col)</th>
<th>Time (Minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Forms Controls (VB .NET)</td>
</tr>
<tr>
<td>50 (5 x 10)</td>
<td>0:02</td>
</tr>
<tr>
<td>100 (10 x 10)</td>
<td>0:05</td>
</tr>
<tr>
<td>500 (10 x 50)</td>
<td>1:24</td>
</tr>
<tr>
<td>1000 (20 x 50)</td>
<td>6:35</td>
</tr>
</tbody>
</table>

### 1.3 Organization of this Document

This document describes how we arrived at and implemented the solution, including both design changes and porting from VBA to VB .NET. In Section 2, we describe the systems involved in this work. Section 3 – 5 presents the arguments, potential
solutions and plan for the system. Section 6 and 7 discuss the design and implementation, and Section 8 presents the conclusion and future work.
2 Background and Related Work

This section presents the necessary technical background information with emphasis on the Forms/3 front-end Excel framework. The definitions, acronyms and abbreviations are also listed below.

2.1. Forms/3

Forms/3 is a functional programming research language based on the spreadsheet paradigm [5]. Like the Excel engine, it evaluates and computes values defined by spreadsheet cells and their formulas. Forms/3 supports similar spreadsheet data types such as integers, real numbers, strings, and Booleans as well as additional ones not supported by Excel such as graphical types.

The Forms/3 engine is implemented in Common LISP. There are two graphical user interfaces that connect to the Forms/3 engine via sockets. The first interface is called JavaForms which is developed in Java. The second interface is ExcelForms which is implemented using the Excel framework. The benefit of having a self-contained Forms/3 engine is that we can easily switch to newer graphical user interface development environments and frameworks without having to re-write the spreadsheet evaluation engine.

Forms/3 has been used as a testbed for end-user software engineering features, and these features are the ones of most interest to us for experimentation in an Excel-like setting [6]. The following briefly describes the end-user software engineering features of Forms/3.

2.1.1 What You See Is What You Test

What You See Is What You Test (WYSIWYT) is a methodology that helps end users test their spreadsheets by providing visual feedback about the test coverage based on dataflow test adequacy criterion [6]. In a spreadsheet, untested cells with non-constant
formulas have red borders. As the end user incrementally test the spreadsheet by placing a checkmark in a formula cell’s decision box, the border colors will eventually progress to blue. The end user’s testing decision reflects the “testedness” of cells in a spreadsheet.

### 2.1.2 Fault Localization

Fault localization is a technique meant to help the user locate spreadsheet errors and visually indicates the areas where the faults are likely to be found [17] with interior cell colors. The darker the cell colors the more likelihood the faults. The three different algorithms used by Fault Localization are Test Count, Blocking and Nearest Consumers.

### 2.1.3 Help Me Test

Help Me Test (HMT) is a feature that helps users tests their spreadsheets by automatically suggesting testing inputs [11] [12].

### 2.1.4 Assertions

Assertions in the form of preconditions, postconditions, and invariants have helped professional programmers maintain and debug software, because these assertions can continuously attend to the entire program, reasoning about the properties the programmers expect of their program logic, and about interactions between different sections of the program [6]. The approach to assertions pioneered by our research group [7] is aimed to provide the same advantages to end-user programmers. These assertions are Boolean expressions about spreadsheet cell values.

### 2.1.5 Surprise-Explain-Reward

The Surprise-Explain-Reward (SER) strategy [6] makes use of software engineering devices, such as decision boxes, to inform end users of the benefits, costs, and risks of these devices and of potential faults in the spreadsheets. The aim is to help end users can make informed choices on the appropriate steps to take. SER uses the element of surprise to arouse the curiosity of end users. Should the end users become interested,
the system follows up with explanations, currently appearing in Tooltips, and, potentially, rewards.

### 2.1.6 Regions

Large spreadsheets usually contain repeated formulas with different cell references. By grouping these replicated formulas into regions, WYSIWYT is able to recognize similar cells and share the testedness of one cell with all other cells in the same region [13].

### 2.2. ExcelForms

The initial version of ExcelForms [9] was developed using Excel with Visual Basic for Application (VBA). ExcelForms visually supports the following features: WYSIWYT, Fault Localization, Help Me Test, and Assertions. Regions are not currently supported by ExcelForms and will be considered for future work.

At the time of the initial implementation, the .NET framework for Excel had just been recently launched by Microsoft. Since .NET was a fairly new and unknown technology, it was deemed better to continue developing Excel with a more mature technology such as VBA.

### 2.3. Definitions, Acronyms, and Abbreviations

The following terms are frequently used throughout this report.

1. **.NET Framework** – A development and execution environment that allows different programming languages and libraries to work together to create Windows-based application [19].

2. **ActiveX Control** – An applet using ActiveX technologies, with full access to the Windows operating system. ActiveX controls are usually executed within a web browser or Microsoft Office document [9].

3. **CVS** – An open source version control tool.

4. **Excel application** – An Excel spreadsheet customized with embedded macros.
5. **GUI** – Graphical User Interface.


7. **Jira** – An open source bug tracking tool.

8. **OLE Object** – Represents an ActiveX control on a worksheet.

9. **OOP** – Object Oriented Programming

10. **Socket** – Consists of a host and a port for communication.

11. **VBA** – Visual Basic for Applications is a programming language very similar to Visual Basic and it is used in Microsoft Office applications.


14. **Wiki** – A server software that allows users to create and edit Web content page using any Web browser.

15. **Windows Forms Control** – An object (Example: buttons, labels, etc) that is contained in the worksheet.
3 Exploring Other Solutions

This section discusses the alternative solutions we explored to counter the problem described in Section 1.1. It also lists the reasons why we did not select these alternative solutions.

3.1 Shapes and Comments in Excel

Shapes are objects drawn on the Excel worksheet, and they are located on a special drawing layer above the cells on the worksheet [4]. Each shape represents an object in the drawing layer, such as an OLE object or picture. Macros can be assigned to shapes and executed when a user clicks on the shapes. Shapes are considered to be a more lightweight UI device compared to OLE Objects. Comments are objects that display the cell information in an Excel worksheet. Each comment object represents a cell comment. In Figure 2, shapes are used to represent the decision boxes and comments are used to display the appropriate information related to the decision boxes.

We discarded the idea of using shapes as decision boxes because they do not support mouse events other than the single mouse click. Thus, there is no support for tooltips to appear on the shapes themselves. We discarded the idea of comments for information about decision boxes because comments could only be attached to the cells rather than the decision boxes. A comment would appear when the mouse hover.
event occurs on the selected cell, especially around the cell border. A comment would not appear when a mouse hovers within the decision box.

### 3.2 Active Template Library

Active Template Library (ATL) is a set of template-based C++ classes that allows users to create small and fast Component Object Model (COM) objects such as ActiveX controls [2]. In other words, ATL would allow us to create our own decision boxes equipped with mouse events and, possibly, tooltips. However, ATL requires a steeper learning curve and a considerable understanding of the COM fundamentals. Before pursuing deeper into the ATL implementation, we decided to test the performance of creating these ATL objects on a spreadsheet. We modified an existing ATL example program that created a polygon object with basic mouse clicks events. We populate the 900 polygon objects (30 row x 30 column) onto the spreadsheet and timed them. As shown in Table 4, it took 17-18 seconds to create the ATL objects. This was only 3 seconds better than the best time it took to create 900 OLE Objects on a spreadsheet. Overall, the performance was not significant enough to convince us to switch to ATL.

<table>
<thead>
<tr>
<th>Library</th>
<th>Time (Minutes)</th>
<th>Issues</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATL</td>
<td>0:17 – 0:18</td>
<td>The ATL mouse event handling is harder to detect in smaller sized polygon.</td>
</tr>
<tr>
<td>OLE Objects with ScreenUpdating set to false.</td>
<td>0:20 – 0:21</td>
<td>Although all the decision boxes are painted at once, excessive flickering occurred on the screen.</td>
</tr>
<tr>
<td>OLE Objects with ScreenUpdating set to true.</td>
<td>0:37</td>
<td>The decision boxes are painted one by one on the screen.</td>
</tr>
</tbody>
</table>
3.3 Piccolo

Piccolo is a UI development toolkit that supports 2D structured graphics programs and Zoomable User Interfaces [16]. It is an ongoing software research product developed by the University of Maryland. With Piccolo, a user can easily create 2D graphics objects, allow these objects to be animated, rotated, and scaled, and support mouse events. The Zoomable User Interfaces allows a user to zoom in for more detailed information, and to zoom out for an overview. All interfaces or graphic objects need to be placed in a Piccolo canvas, called the PCanvas, so that they may be viewed and interacted with by the user. Figure 3 shows an example Piccolo program on the .NET platform.

![Figure 3: An example program using Piccolo with zooming capability.](image-url)
In Figure 4, we successfully integrated Piccolo with Excel using VSTO. If we reduced the size of the `PCanvas`, then the graphic objects inside the canvas would represent a decision box. However, using Piccolo to mimic a decision box had some major issues. Firstly, while the graphics objects inside the `PCanvas` supports mouse events, the `PCanvas` itself does not. Tooltips would need to encompass more than the Piccolo canvas to display the appropriate explanations. Even if tooltips were limited to within the Piccolo canvas, the graphic objects themselves have no support of tooltips. Tooltips would need to be manually created from text labels instead. Next, the `PCanvas` does not support the transparent background property. Cell values or cell borders in a worksheet would be potentially covered by the Piccolo canvas. Lastly, moving the `PCanvas` around as a floating decision box would not be an easy task as it would involve calculation of the system coordinates that needs to be fairly accurate.
4 System Requirements Specifications

4.1 Introduction

This section provides the System Requirement Specifications for ExcelForms under the .NET implementation. These requirements are based on the IEEE Std 830-1993 requirements document standard [1].

4.1.1 Scope of the System Specification

The aspect of the system being specified here consists solely of the graphical user interface of the ExcelForms version of end-user software engineering. It will display and support the Forms/3 end-user software engineering features in the Excel framework.

4.1.2 Purpose

The ExcelForms front end supports the rich features of Forms/3 end-user software engineering in the Excel framework. Doing so allows us to study whether end users, who are already familiar with Excel, can use the ExcelForms front end to test and debug their spreadsheets seamlessly. It also encourages long-term studies to be conducted at our collaborators’ site in a practical manner.

4.1.3 Definitions, Acronyms, and Abbreviations

Refer to Section 2.3 for all definitions of terms, acronyms and abbreviations.

4.1.4 References

Refer to the References section at the end of this document.

4.2 Overall Requirements

The system being described is the front end graphical user interface of the Forms/3 end-user software engineering features. It implements the Excel framework and communicates with the Forms/3 engine via sockets. The ExcelForms front end is
packaged with the Forms/3 engine, and will be used for short-term and long-term studies. The ExcelForms front end, together with the Forms/3 engine, will be deployed in computer labs in schools and universities.

4.2.1 Architectural Requirements and Constraints

The ExcelForms front end communicates with the Forms/3 engine through a series of request and response socket messages. Upon receiving the response messages, the ExcelForms front end will then display the appropriate UI devices in Excel. The ExcelForms front end under no circumstances does any logic about testing, debugging, assertions, or formula correctness. This must all be handled by the Forms/3 engine.

4.2.2 System Perspective

The following diagram, Figure 5, depicts the interoperability of components within Forms/3. Note that only one front-end GUI can communicate with the Forms/3 engine at a single time. It is a dedicated single client and server architecture.

![Diagram](image)

Figure 5: The three top nodes are alternative UI components. Only one can be used at a time.

4.2.2.1 System Interfaces

The ExcelForms front end shall liaise with the Forms/3 engine using a standard message format.
4.2.2.2 **Versions of the System**

There shall be a developer’s version and a release version of the ExcelForms front end. The developer’s version will be used for in-house development. All features are available to the developers. The ExcelForms front end is able to connect locally or remotely to the Forms/3 engine. The release version shall be distributed to non-developers at collaborators’ sites. Not all features will be made available to the non-developers. These features will depend on the specific needs of the end user. The ExcelForms front end can only connect locally to the Forms/3 engine and, therefore, the Forms/3 engine is required to run on Windows XP.

4.2.2.3 **Software Platforms**

The ExcelForms front end shall require Microsoft Excel 2003 and Windows XP for its implementation. In addition, the Forms/3 engine is required to run on a Common LISP server that supports different operating system platforms such as Linux, UNIX and Windows XP.

4.2.2.4 **Communication Interfaces**

The ExcelForms front end shall communicate via TCP/IP sockets with the Forms/3 engine.

4.2.3 **User Characteristics**

There are four categories of users. They are short-term subjects, long-term subjects, collaborators, and graduate researchers. Short-term and long-term subjects are considered end users of the ExcelForms front end. Collaborators do not do development, but they may install the ExcelForms front end together with the Forms/3 engine, and turn certain features on and off. Graduate researchers are considered developers of the ExcelForms front end.

Studies that are held for 2-4 hours consist of short-term subjects. These subjects are typically undergraduates who major in business and are familiar with spreadsheets.
There is another group of short-term subjects who are high-school students also. They may have no prior knowledge of spreadsheet software.

Studies or education programs that are held from 1-6 months are considered long term subjects. These subjects are usually Mathematics and Science teachers who install the ExcelForms front end and the Forms/3 engine in their respective schools.

Lastly, the graduate researchers are the ones who are responsible for developing, distributing, deploying, and maintaining the ExcelForms front end. The ExcelForms front end and the Forms/3 engine will be installed at Oregon State University, University of Nebraska at Lincoln, Drexel University and other education institutions.

4.2.4 Constraints
The ExcelForms front end shall need security access privileges to run on Windows machines. Permission is usually granted by the Windows or Network Administrator. If the packaged system is deployed at the collaborators’ sites, the ExcelForms release version is distributed together with the Forms/3 engine as described in Section 4.2.2.2. The release version requires the Forms/3 engine to run locally on Windows XP.

4.2.5 Assumptions and Dependencies
The ExcelForms front end shall run on Microsoft Office 2003 version and the operating system is Windows XP.

4.3 Specific Requirements

4.3.1 External Interfaces
1. The ExcelForms front end shall communicate with Excel via the .NET framework.
2. The ExcelForms front end shall communicate with the host machine via sockets.
4.3.2 Features

The features are closely patterned after the system requirements in [9] because the functionality of the ExcelForms front end remains mostly unchanged even if the implementation method has changed.

4.3.2.1 Communications

1. The ExcelForms front end shall send and receive internal messages between Excel and the Forms/3 engine.
2. These messages must be in the standard interface format as used by other Forms/3 components. The ExcelForms front end shall validate and parse these messages.
3. The communication session must be initiated by the user of the ExcelForms front end.
4. The ExcelForms front end shall run on a single machine where the Forms/3 engine resides locally in the same machine. The communication session is through sockets.
5. Alternatively, the ExcelForms front end shall run on one machine and connect remotely to the Forms/3 engine on another machine. Both machines communicate via remote socket connection.

4.3.2.2 Basic Algorithms

1. The ExcelForms front end shall support the following features:
   a. The testedness information shall be calculated in the Forms/3 engine and displayed on the front end in each Excel worksheet.
      i. The ExcelForms front end shall provide an input device to allow users make testing decisions. For example, a checkbox.
      ii. The ExcelForms front end shall display the testedness information.
      iii. The ExcelForms front end will update the testedness information when a user edits a cell’s formula.
iv. The ExcelForms front end will update the testedness information after the user makes a testing decision.

v. A CRG will be constructed for the worksheet to support WYSIWYG.

b. Fault Localization shall be displayed within the spreadsheet.

c. Help Me Test information shall be displayed for each workbook.
   i. HMT is triggered by the user selecting a device within Excel.
   ii. A request is sent to the Forms/3 engine for test case generation. The response is then output to the ExcelForms front end.

2. The list of Excel features the ExcelForms front end shall or shall not support:
   a. The ExcelForms front end shall work with absolute and relative cell references.
   b. The ExcelForms front end shall work with cell designation by range and list. For example, (A1:A3) or (A1,A2,A3).
   c. The ExcelForms front end must not change the Excel spreadsheet language.
   d. The ExcelForms front end need not work with multiple worksheets.

3. The ExcelForms front end must not modify or interfere with commonly used Excel features:
   a. Charting
   b. Cell-locking
   c. Validation (Excel’s built-in assertion mechanism)
   d. Auditing (Excel’s built-in Arrows)
   e. Open/Closing files
   f. Quit
   g. Undo (should work between testedness updates)

4. The ExcelForms front end may be incompatible with the Excel constructs:
   a. Add-ins
   b. Macros
c. Circular References

d. Undo between testedness actions

4.3.2.3 Advanced Features

1. The ExcelForms front end shall link cells with Arrows to indicate testedness information.
2. The ExcelForms front end shall provide some form of explanation for each customized input device. For example, tooltips.

4.3.2.4 General Release

1. The ExcelForms front end shall have a developer version and a release version.
2. The release version can be distributed to others for collaboration and demonstration.
3. The installation shall be documented to help users install and run the ExcelForms front end.
4. All non-functional and performance constraints are met.
5. The ExcelForms front end shall support the evaluation of multiple workbooks and worksheets.
6. The ExcelForms front end is robust and handles exceptions or errors gracefully.

4.3.3 Performance Requirements

1. The ExcelForms front end and the Forms/3 engine shall support a single client and server communication at any time.
2. The ExcelForms front end shall perform no worse than the prior version of ExcelForms VBA. For example, in Table 6, spreadsheets with less than 1000 cells must load in under 3 minutes.
4.3.4 Design Constraints

Since Microsoft Excel 2003 is the chosen spreadsheet software for our implementation, the ExcelForms front end shall adhere closely to the Excel Object Model. Furthermore, the ExcelForms front end shall run solely on Windows XP.

4.3.5 System Software Attributes

4.3.5.1 Reliability

1. The ExcelForms front end shall be responsive to user actions.
2. The ExcelForms front end performance shall perform in a stable and robust manner.

4.3.5.2 Security

1. The ExcelForms front end shall adopt the .NET security standard.

4.3.5.3 Maintainability

1. The ExcelForms front end shall support future versions of Microsoft Office application.
2. The ExcelForms front end may not be backward compatible with earlier versions of Microsoft Excel.
3. Code shall be well-documented and modular or flexible enough to support any changes to the ExcelForms front end.
4. CVS, a version control tool, shall be used to keep track of changes to the code.
5. Any documentation about the ExcelForms front end shall be published or stored on the Forms/3 Wiki website.
6. Jira, a bug-tracking tool, shall be used to track and resolve software defects in the ExcelForms front end.

4.3.5.4 Portability

1. The Forms/3 engine shall run on Linux, UNIX or Windows XP operating systems.
2. When distributing the ExcelForms front end and the Forms/3 engine to our long-term subjects or collaborators, the packaged system shall run on Windows XP entirely.
5 Project Schedule and Risks

This section describes the project plan of the ExcelForms front end in terms of schedule, resource and risks.

5.1 Schedule

5.1.1 Preliminary Work

We started some preliminary work during summer 2005 to lay the groundwork for this project. The VSTO Beta 1 and VSTO Beta 2 versions were installed and evaluated so that we would be familiar with the development tool. We wrote sample codes to deploy and run an Excel application embedded with OLE Objects and Forms Controls separately. Timings were taken to gauge the performance of the programs. Our goal was to ensure we could embed UI devices in spreadsheet without compromising on the performance of the Excel application. Also, we took a feature of ExcelForms and implemented the feature using VB .NET. We wanted an estimate of how easy or how difficult it would be to port the entire ExcelForms application to the .NET platform. During the migration process, we explored alternative solutions to replace some functionality by VBA that are no longer supported by VB .NET. Overall, we benefited from this exercise because it helped ease our transition to the .NET platform.

5.1.2 Project Plan

The following Figure 6 describes the project plan and schedule of the ExcelForms front end.
Figure 6: Project Schedule
5.2 Resource

We allocated 2 people to work on the entire duration of the project. We have a full-time graduate researcher working 20 hours per week and an undergrad student researcher working 10 hours per week.

5.3 Risks

No project is without risks. We identified several factors that may influence the outcome of the project. The new technology switch requires considerable re-learning of skills. Firstly, the developers will need to transition their programming skills from VBA to VB .NET. For example, some VB code syntax became obsolete and is no longer supported by VB .NET. Secondly, the developers are required to learn how to use VSTO as their new development environment. Since VSTO is relatively new tool, we were prepared to encounter problems with installation, building, debugging or deployment of the ExcelForms front end. Lastly, we hope that implementing the Excel Object Model on the .NET framework will provide us a more robust and stable system. Since this is the first time we are implementing a .NET framework solution, we do not have any historical data or previous project experience to ensure the success of the project.

We decided to take several steps to mitigate these risks and prevent any risk from materializing [10]. We ensured we had available resources from Microsoft via website links, emails, seminar and technical support. We kept our knowledge base in a central repository on our Forms/3 wiki. In addition, we documented our efforts in trying out new .NET libraries or code samples. In the case of the VSTO, we evaluated and experimented early on with the beta 1 and beta 2 versions of the tool. Through incremental learning, we would then be able to port our codes easily and with familiarity when VSTO 2005 was officially launched.
6 Design

This section describes the high-level design of the ExcelForms front end based on the overall architecture, its communications, and its user interface.

6.1 Overall Architecture

Refer to Figure 5 for the overall architecture.

6.2 Communications

The Forms/3 engine is a self-contained spreadsheet evaluation engine that consists of analytical functions which are LISP-based. The Forms/3 engine runs as a socket server that listens on the default port 49152, and awaits requests from the front-end systems which are, in fact, socket-based clients. To communicate with the Forms/3 engine, the ExcelForms .NET version must be implemented as a socket client using the VB .NET socket library.

6.2.1 ExcelForms-Specific Messages

ExcelForms uses the Forms/3 messages. In addition, the following ExcelForms-specific messages [9] in Table 5 are supported.
Table 5: Messages specific to ExcelForms

<table>
<thead>
<tr>
<th>Message Name</th>
<th>Description</th>
<th>Direction</th>
</tr>
</thead>
<tbody>
<tr>
<td>End-of-Processing</td>
<td>The last message to be sent to ExcelForms as part of the response messages from the Forms/3 engine.</td>
<td>From Forms/3 engine to ExcelForms.</td>
</tr>
<tr>
<td>Gui-Update-Workbook</td>
<td>The message is sent when the user loads a workbook. The message consists of workbook name, worksheets names, the worksheet ranges, and the row, column, formula, and value of each cell.</td>
<td>From ExcelForms to Forms/3 engine.</td>
</tr>
<tr>
<td>Make-Workbook-From-List</td>
<td>The message is sent to the Forms/3 engine every time the user changes the workbook. The message consists of formula cells and the values of their dependents.</td>
<td>From ExcelForms to Forms/3 engine.</td>
</tr>
<tr>
<td>NameRegion</td>
<td>The message identifies how cells are groups into regions on the front end.</td>
<td>From Forms/3 engine to ExcelForms.</td>
</tr>
<tr>
<td>Request-Evaluation</td>
<td>The message specifies an expression to the front end for evaluation.</td>
<td>From ExcelForms to Forms/3 engine.</td>
</tr>
<tr>
<td>Return-Evaluation</td>
<td>The message returns the result of a request-evaluation call.</td>
<td>From Forms/3 engine to ExcelForms.</td>
</tr>
<tr>
<td>Start-of-Processing</td>
<td>The first message to be sent to ExcelForms as part of the response messages from the Forms/3 engine.</td>
<td>From Forms/3 engine to ExcelForms.</td>
</tr>
</tbody>
</table>
6.2.2 Connection User Interface

There are two user interfaces in ExcelForms to connect to the Forms/3 engine. They are the developer’s version and the release version. In Figure 7, the user or the developer is able to connect via sockets to any local or remote machine that has the Forms/3 engine running. The user can also specify the port number which the Forms/3 engine is hosting. Note that the Forms/3 engine will need to be manually started by the user or the developer.

For the release version, as shown in Figure 8, the end user is only allowed to connect via socket to the local machine, and the port number is fixed to 49152. The restriction is placed to discourage end users from connecting to remote machines due to security issues, and the fact that system administrators may not allow permission to access other machines. Nevertheless, it is easier for the end user run the ExcelForms front end in this version. With a single click on the button, the ExcelForms front end automatically starts and connects itself to the Forms/3 engine.
6.3 User Interface Design

This section describes the user interface guidelines and the user interface features of ExcelForms. The discussion is on the behavior and appearance of the ExcelForms front end with the main focus being on changes to the UI design in the .NET implementation. To achieve the performance and robustness objectives, we modified the Creelan and Fisher 2004 UI as follows.

6.3.1 Floating decision box

We decided to use a single decision box that would move or float around the spreadsheet, as shown in Figure 9, whenever a user selects a formula cell. When the user clicks on the floating decision box, a set of twin boxes with check-mark and X-mark will appear, as in Figure 10, to allow the user to check off either of the twin boxes when testing the spreadsheet. The check-mark is blue in color while the X-mark is red. In Figure 11 and Figure 12, the user can also check-mark or X-mark a decision box by selecting the toolbar button or right-click on the menu.
### Grades To Date

<table>
<thead>
<tr>
<th></th>
<th>Robin</th>
<th>May</th>
<th>Joe</th>
<th>Annie</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homework1</td>
<td>65</td>
<td>89</td>
<td>98</td>
<td>83</td>
<td>84</td>
</tr>
<tr>
<td>Homework2</td>
<td>54</td>
<td>78</td>
<td>67</td>
<td>81</td>
<td>66</td>
</tr>
<tr>
<td>Grade to Date</td>
<td>60</td>
<td>84</td>
<td>83</td>
<td>82</td>
<td>75</td>
</tr>
<tr>
<td>Passing</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Percent Passing</td>
<td>2</td>
<td>0.75</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 9: Floating decision box**

### Grades To Date

<table>
<thead>
<tr>
<th></th>
<th>Robin</th>
<th>May</th>
<th>Joe</th>
<th>Annie</th>
<th>Average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homework1</td>
<td>65</td>
<td>89</td>
<td>98</td>
<td>83</td>
<td>84</td>
</tr>
<tr>
<td>Homework2</td>
<td>54</td>
<td>78</td>
<td>67</td>
<td>81</td>
<td>66</td>
</tr>
<tr>
<td>Grade to Date</td>
<td>60</td>
<td>84</td>
<td>83</td>
<td>82</td>
<td>75</td>
</tr>
<tr>
<td>Passing</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Percent Passing</td>
<td>2</td>
<td>0.75</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 10: Twin boxes with check-mark and X-mark**

**Figure 11: Check-mark and X-mark button on the toolbar.**
The floating decision box design improves the overall performance of the ExcelForms front end when loading a big spreadsheet with many formula cells since there is no need to create a decision box for every formula cell. With a single floating decision box, the goal is for ExcelForms to respond immediately to any testing decision made by the user as the number of formula cells increases.

The floating decision box is controlled by the study policy on the Forms/3 engine. The study policy allows or disallows features from appearing on the front-end systems. To enable the floating decision box feature, the `FloatingCheckbox` flag should be set to `true`. Since this policy eliminates having every decision box in every formula cell, the `Checkbox` flag must also be set to `nil`. Thus, we can use either with the floating decision box or the regular ones (one per cell) until we have conducted enough user testing to be sure whether to switch permanently to the floating decision box.

### 6.3.2 Image Decision Box

After a user checks off the floating decision box in a particular formula cell, an image decision box is displayed in the formula cell. The image decision box cannot be interacted with, but it reminds the user the testing decision he or she has made throughout the spreadsheet as shown in Figure 13. In a big spreadsheet, however, as
the user checks off the decision box, more image decision boxes are created and the overall performance of the ExcelForms front end will be affected. Still, it is bounded by the number of decisions the user has made, rather than by the number they are allowed to make, as was the case of regular (not floating) decision boxes.

The image decision box feature is enabled by setting the study policy `DecisionImage` to `true`. This policy is dependent on having the floating decision box feature enabled also.

### 6.3.3 Tooltips

In the ExcelForms VBA implementation, tooltips were implemented using the `Windows.Forms.Labels` with an OLE Object as a wrapper. Mouse hovers and mouse click events were poorly implemented by code because of limited event handling methods that the OLE Object supports. As a result, tooltips were buggy and showed inconsistent mouse events behavior. In the new ExcelForms, we want to use tooltips that are provided by the `Windows.Forms.ToolTip` library. By using the default Windows tooltips, we need not explicitly implement our own mouse events beyond what is the next action that should occur once an event is raised.

### 6.3.4 Dashed borders

To differentiate the cells where a test checkmark could make progress as opposed to having a regular decision box with a question mark, the cell borders on the bottom and
rightmost have dashed colored borders. The cell border colors are displayed on the bottom and rightmost borders of each cell. Figure 13 shows an example of dashed borders. The dashed border feature is enabled by setting the study policy \textit{DashedBorder} to \textit{true}.

6.4 WYSIWYT User Interface

The WYSIWYT visual UI devices consist of the floating decision box, image decision boxes, cell border colors, and the testedness progress bar. The floating decision box, the image decision boxes, and the cell border colors are summarized in Section 6.3.1, Section 6.3.2, and Section 6.3.4 respectively.

The Testedness Overview progress bar is an indicator of the percent testedness in the spreadsheet. In Figure 14, the progress bar is represented by a series of toolbar buttons. Blue represents the percentage of tested cells while red represents the percentage of untested cells. We would rather implement a forms control progress bar on the toolbar instead of a chain of buttons but the progress bar is not supported by the toolbar class.

![Figure 14: Testedness Overview progress bar](image)

6.5 Fault Localization User Interface

Fault Localization differentiates faulty and correct cells in a spreadsheet [18] using visualization techniques. When a user X-marks a suspect value in a cell, fault localization is triggered and the interior of the suspect cell and its dependencies are shaded from yellow to orange. However, the border colors of the cells affected remain unchanged as shown in Figure 15.
The Fault Likelihood Overview progress bar is an indicator of faulty cells most likely to be present in the spreadsheet. In Figure 16, the progress bar is represented by a series of toolbar buttons. The varying shades represent the percentage of bug likelihood cells of each shade.

6.6 HMT User Interface

Refer to Creelan and Fisher 2004 for the HMT user interface design.

6.7 Assertions User Interface

Refer to Creelan and Fisher 2004 for the assertions user interface design.

6.8 Explanations User Interface

Tooltips are used in the explanation part of SER. When a user hovers his or her mouse over a UI device, a tooltip appears and may give suggestions to the user on what actions to perform next. These tooltips belong to the Windows.Forms.ToolTip class and, therefore, they have common behaviors expected from standard Windows tooltips. Not all UI devices support tooltips though. The UI devices that currently support tooltips are decision boxes, decision images and progress bars. Tooltips support for Arrows and cell borders are being considered in the future.
6.9 Arrows User Interface

Figure 17 and Figure 18 show an example of how the dataflow arrows work. A user selects a formula cell and clicks on *show/hide arrows* button to display or hide the arrows between cells. Refer to Creelan and Fisher 2004 for more details on the arrows user interface design.

**Figure 17:** Show or hide arrows button on the toolbar.

<table>
<thead>
<tr>
<th>Grades To Date</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Robin</td>
<td>May</td>
<td>Joe</td>
<td>Annie</td>
</tr>
<tr>
<td>Homework1</td>
<td>65</td>
<td>89</td>
<td>96</td>
<td>83</td>
</tr>
<tr>
<td>Homework2</td>
<td>54</td>
<td>78</td>
<td>67</td>
<td>81</td>
</tr>
<tr>
<td>Grade to Date</td>
<td>60</td>
<td>84</td>
<td>83</td>
<td>82</td>
</tr>
<tr>
<td>Passing</td>
<td>9</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Percent Passing</td>
<td>0.75</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 18:** Arrows between cells.
7 Implementation and Deployment

7.1 Implementation

This section highlights the ExcelForms components affected when transitioning from VBA to VB .NET. The features that required major code changes were the socket code, the UI devices, and the worksheet conversion.

7.1.1 Communications

ExcelForms implements a synchronous socket client using the TCPSocket class provided by .NET library. This implementation of sockets is different from the VBA implementation where an event is automatically raised every time a message is received. There is no such event provided by TCPSocket. Our socket client is a single-threaded process that sends a single message request to the Forms/3 engine and waits for the response messages from the backend. The backend sends multiple messages through the socket traffic. The socket client continuously loops to receive these messages and stops when it receives the End-of-Processing message. There is a message queue in ExcelForms that caches the received messages, and soon after, the MsgHandler processes the messages that are waiting in the queue.

7.1.2 Floating Decision Box

The user must set the FloatingCheckbox policy to true in the Forms/3 engine to enable the floating decision box feature. In ExcelForms, the floating decision box, shown in Figure 19, is represented by XLBox.

A limitation is that the Excel display needs to be 100% in order for the forms control to work properly. If Excel is not at 100%, should a user attempt to click on the floating decision box, a warning pop-up box is displayed. To avoid this problem, use sensible font size.
7.1.3 Image Decision Box

The user must set the DecisionImage policy to true in the Forms/3 engine to enable the image decision box feature. This policy requires having the floating decision box feature enabled also. The image decision box, in Figure 19, is represented by XLBox. It is also limited by the 100% view display in Excel.

7.1.4 Dashed Borders

The dashed border property is determined from the study policy in the Forms/3 engine. The Dashboard policy is set to true to enable the display of cell with dashed borders on the Excel front end, as shown in Figure 19. Only formula cells are allowed to have dashed borders.

In ExcelForms, the cell border properties in the XLC4Cell class are set as shown in Figure 20. Notice that the bottom and right cell borders are the only borders affected in each formula cell.
7.1.5 Tooltips

Tooltips on decision boxes are instantiated from the ToolTip class. This class provides the solution to implement the standard Windows tooltips with its default mouse hover events. Both the floating decision box and the image decision box no longer use the XLTooltipable class which was the old way of implementing tooltips and mouse events by hand. Figure 21 shows the code snippet to set the tooltips.

```vba
Private WithEvents tooltipManager As _
    System.Windows.Forms.ToolTip
    ...
    tooltipManager = xltooltip
    ...
    tooltipManager.SetToolTip(Me, tooltipText)
    ...
```

Figure 20: The shaded borders are simply implemented using Cell properties.

Figure 21: Setting the tooltip text in XLBox.
7.1.6 Toolbars

The ExcelForms toolbars, Figure 22, consists of the Connection toolbar, the Testedness toolbar, the Fault Likelihood toolbar and the Assertions toolbar. The Connection toolbar allows ExcelForms to connect to the Forms/3 engine. The Testedness toolbar and the Fault Likelihood toolbar display the progress bars which are a chain of toolbar buttons. The Assertions toolbar performs assertions on the spreadsheet.

![ExcelForms toolbars](image)

**Figure 22: ExcelForms toolbars**

The toolbar button events are handled differently in VB .NET. The `onAction` property no longer works in VB .NET as it does in VBA. A method must be created to support the mouse click event. Figure 23 shows the code snippet of how the toolbar button event is coded for the `Enable WYSIWYT` button on the Connection toolbar. The same concept applies for event handling of all toolbar buttons.
7.1.7 Extended Worksheet Conversion

There are two Excel worksheet classes supported by VSTO [8]. The first is the `Microsoft.Office.Interop.Excel.Worksheet` class which handles native Excel worksheets belonging to the Excel Object Model. Second, the `Microsoft.Office.Tools.Excel.Worksheet` class handles embedded forms controls in the Excel worksheet. It is also called the extended worksheet because it supports forms controls and their events. This is a big advantage, but the disadvantage is that the extended worksheet supports limited classes under the Excel Object Model. Our design requires having forms controls and full access to the classes in the Excel Object Model.

Because of this, we convert the native Excel worksheet to the extended worksheet to supports forms controls every time a user loads a spreadsheet. Figure 24 shows the function that does the conversion. A weakness of this method is the use of `this.RuntimeCallback` to get the required `IRuntimeServiceProvider` instance that is associated to the original ExcelForms workbook. When a user loads a workbook or spreadsheet, the extended worksheet reference retrieved is resolved in the context of

```vbnet
Private WithEvents cbEnabled As CommandBarButton

Private Sub cbEnabled_Click(ByVal Ctrl As CommandBarButton, _
    ByRef CancelDefault As Boolean) _
    Handles cbEnabled.Click
    If cbEnabled.Caption = "Enable WYSIWYT" Then
        SystemControl.StartSystem()
    Else
        SystemControl.StopSystem()
    End If
End Sub
```

Figure 23: The toolbar button event code for the Enable WYSIWYT button.
the original workbook, not the newly created workbook. This explains why forms controls are being added to the original worksheets instead of the new worksheets. There is no known solution to retrieve the IRuntimeServiceProvider instance associated with the new workbook. A workaround we did was to copy the worksheets from the new workbook to the original workbook. Any changes made by the user will be reflected and saved in both workbooks.

```vbnet
Public Function CreateViewForWorksheet(ByVal workSheet As _
    Me.VBProject.VBComponents.Item(1)
    Dim hostprovider As IHostItemProvider
    hostprovider = 
        RuntimeCallback.GetService(GetType(IHostItemProvider))
    WorkSheetView = _
            RuntimeCallback, _
            workSheet.CodeName, _
            Constants.vbNull, _
            workSheet.Name)
    Return WorkSheetView
End Function
```

**Figure 24: Extended worksheet conversion**

### 7.2 Performance Tests

We ran several time tests on the performance of loading a spreadsheet with varying number of formula cells on the ExcelForms .NET implementation vs. the ExcelForms VBA implementation. We ran the tests on *burnett-server* with the following hardware specifications: Intel Xeon 3.40GHz CPU with 2GB RAM. The *burnett-server* machine has Windows 2003 Server, Excel 2003, and Visual Studio Tools for Office (VSTO) 2005 installed. Our gathering techniques include running the time tests 50 times, and calculating the mean and standard deviation. The new ExcelForms solution was
implemented on the .NET framework with VSTO 2005/Excel 2003, and had the floating decision box. The old ExcelForms was implemented with VBA/Excel 2003, and had regular decision boxes. We compared the timings of both the implementations as shown in Table 6 and Figure 25. Overall, the ExcelForms .NET implementation with its floating decision box performed better than the ExcelForms VBA implementation with its regular decision boxes. When loading smaller spreadsheets, the differences in timings are minimal. But as the number of formula cells increases, the ExcelForms VBA implementation performed poorly by taking twice the time to load a spreadsheet as compared with the ExcelForms .NET implementation.

<table>
<thead>
<tr>
<th>Number of Formula Cells</th>
<th>Mean Time in Minutes (Standard deviation)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>ExcelForms .NET</strong></td>
</tr>
<tr>
<td></td>
<td>(Standard deviation)</td>
</tr>
<tr>
<td>50</td>
<td>0:13 (5.83E-06)</td>
</tr>
<tr>
<td>100</td>
<td>0:20 (7.00E-06)</td>
</tr>
<tr>
<td>250</td>
<td>0:34 (1.09E-05)</td>
</tr>
<tr>
<td>1000</td>
<td>2:27 (2.76E-05)</td>
</tr>
</tbody>
</table>
7.3 Deployment

There are two ways to deploy the ExcelForms .NET implementation. First, we deploy the ExcelForms front end from within VSTO. This approach is meant for developers who want to build and run their application in one integrated development environment. The second approach is to package and deploy a release version for our short-term subjects and long-term subjects. This version can run as it is by using Excel.

7.3.1 Developer’s version

Here are the steps to deploy the ExcelForms developer’s version:

1. Start up VSTO. Open the project file ExcelForms.sln. Double clicking on the file will also start up VSTO automatically.
2. Ensure the Release flag in Globals2 class is set to False.
3. Go to the Debug menu. Select Start Without Debugging. VSTO will build and deploy the ExcelForms.
4. ExcelForms is successfully deployed. In VSTO, the Trust Assemblies Local project property is default to true. This property allows Excel to fire up and...
load ExcelForms and its assemblies without restrictions in security permission. Bear in mind that the Windows account user will need sufficient permission to run any .NET applications on the Windows machine.

### 7.3.2 Release version

First, we build the ExcelForms release version:

1. Start up VSTO. Open the project file `ExcelForms.sln`. Double clicking on the file will also start up VSTO automatically.
2. Ensure the `Release` flag in `Globals2` class is set to `True`.
3. Go to the `Build` menu. Select `Configuration Manager`. A pop-up box will appear.
4. In the `Configuration` column for the ExcelForms project, select `Release` in the drop-down menu. Click on the `Close` button.
5. Go to the `Build` menu. Select `Build Solution`. The following files are created: `ExcelForms.dll`, `ExcelForms.pdb`, and `ExcelForms.xls`. The first two files are called the assembly files.

Here are the steps to deploy the release version:

1. Copy the three files in a new folder.
2. Get the Windows Administrator to grant full permission [14] on the folder to run the assembly files.

### 7.4 Maintenance

To manage and maintain the ExcelForms front end, we use the following tools.

#### 7.4.1 CVS

The ExcelForms source code is stored in an open source version control tool called CVS. A CVS repository module called `ExcelFormsNET` is created to distinguish the new implementation from the old one. Developers can easily check out the module, modify the code and commit the changes.
7.4.2 Jira

Jira, a bug tracking tool, is used to track and monitor any defects in ExcelForms. Feature improvements for ExcelForms can also be logged into in Jira.

7.4.3 Wiki

Documents such as technical reports, user guide, how-to demos, and implementation resources articles are stored in the Wiki. The Wiki is used as a central repository for any documentation pertaining to ExcelForms. It is very convenient to update the articles on the fly and to store downloadable documents.
8 Conclusion and Future Work

Software development for Microsoft Office Applications on .NET platform is a fairly new and emerging technology. There are still many improvements to be made, issues to resolve, and ongoing development happening on the .NET framework for Office Applications. As Microsoft moves further away from VBA development for Office Applications, then so too must we. We hope, in our efforts of migrating ExcelForms to the .NET platform, we would have a more stable and robust system to support our end users. The ExcelForms front end should perform efficiently and promote end-user software development, testing, and debugging of spreadsheet.

Future work of the ExcelForms front end should include a better explanations component, progress bar implementation on the toolbars, UI support for regions, and a multi-threaded implementation. An improved explanation component is in the early brainstorming stage. Some of the suggested UI devices to use for explanations are side panel display, multi-level tooltips, and tooltips for arrows and cell borders. Secondly, we want to implement progress bars on the WYSIWYG and Fault Likelihood toolbars because the current indicator, with its chain of buttons, looks rather ugly. ExcelForms support for regions are non-existent because they are currently broken on the Forms/3 engine. Once this feature is redesigned and improved upon on the backend, the ExcelForms front end will need to be developed to accommodate regions with the proper UI devices. Lastly, we would like to implement multi-threads for the communications aspect of the ExcelForms. If we could separate the socket traffic from the UI display using different threads, we hope that the ExcelForms front end would be more efficient in processing the messages and painting the Excel screen. Furthermore, the ExcelForms front end would be more interactive and responsive to the end user.
References


