Component Programming

Min Jie Hu
Tairawhiti Polytechnic
PO Box 640, Gisborne, NZ
min@tairawhiti.ac.nz

Abstract

This paper discusses the possibility of teaching component programming after introducing object-oriented programming at a tertiary level. A field usability testing method is applied to create and test the component. The research discovered that the basic object-oriented concepts such as inheritance, polymorphism and override are seamless as a whole to the re-use of components that are created by VB.NET and C#.

Keywords: Component, DLL, Field usability testing.

1 Introduction

A component is a reusable piece of software in binary form, which has an extension file name like .dll, .ocx, or .exe. MS Windows supports the COM (Component Object Model) technology to enable components to communicate with each other and to link together to build applications. This means we can re-build a single component to update a whole application and we do not need to build the whole application again.

After teaching object-oriented (O-O) programming, the author introduced component programming to create DLL and its application for DipICT Level 6 at Tairawhiti Polytechnic by using MS VB 6.0. However, the student's assignment with DLL, worked perfectly on the student's computer before being handed in for assessment. But, it does not directly work on the tutor's computer until the DLL is registered to the Window registry at the computer where it will be accessed. Otherwise, it has to be accessed through the same computer that created it. This is because there is a problem with COM, which is called "DLL Hell".

The research firstly tried to find a method of removing the shadow of DLLhell and to then find a means of continuing to teach component programming by changing to .NET. Next, we chose a suitable method for this research, and then conducted the research of component programming by .NET and finally, this paper discusses the research findings.

This quality assured paper appeared at the *19th Annual Conference of the National Advisory Committee on Computing Qualifications* (NACCQ 2006), Wellington, New Zealand. Samuel Mann and Noel Bridgeman (Eds).

2 Literature Review

The literature review started from a trial to find out what happens to DLL, what other research had done for component programming, and what we needed to do in order to improve our teaching. It is more efficient to retrieve articles through the Internet. This is especially, when searching for the best source of peer-reviewed material about component programming, such as Microsoft Corporation on line library.

2.1 DLL

Webopedia (2005) describes the definition of DLL as short for Dynamic Link Library, a library of executable functions or data that can be used by a Windows application. Some DLLs are provided with Windows, while others are written for applications. A significant reason to create DLL is to let a single DLL component be shared by several applications at the same time. Thus, the research on how to design and use shared components has been raised in education and industry.

Miller (1999) conducted a research of DLLs by MS VC++ 6.0. Mgama (2002) also demonstrated how to create a DLL by VC++ and add it to Windows Installer. Since Visual Basic (VB) is gradually becoming more and more popular, Boondog (1998) has successfully linked its MS VB 5.0 application to its own DLL component created by MS VC 5.0. This finding tells us that we can now use VB to access the DLLs created by VC++, Delphi, or Borland C++.

However, the inside story (Wong, 1998) about DLL hell makes us hesitate to use and teach DLL. Microsoft provides DLLs for both Windows and its own applications. These DLLs contain groups of functions instead of implementing them in each application separately. One application may need to install a new version of the common DLL component shared by multiple applications. If this new DLL is not backward compatible with the old one, the rest of applications that depended on the old DLL might not work with this new DLL. Wikipedia (2006) defines the difficulties of managing MS DLLs as DLL hell. These problems mainly refer to conflicts between DLL versions, difficulty in getting required DLLs etc.

Microsoft (Anderson, 2000) released a DLL Universal Problem Solver (DUPS) packaged together with the approaches of Windows File Protection and Private DLLs, to temporally resolve the DLL compatibility

problems. It could not solve all the DLL hell problems until Microsoft introduced its .NET framework in 2001.

2.2 .NET and Component

In .NET, the notion of side-by-side (D'Souza, Whalen, and Wilson, 1999), Pratschner, 2001) makes it possible to install and run different versions of components on the same computer. The building blocks in .NET are called assemblies, which equate to DLLs, exactly known as "logic DLLs". A principle guideline in .NET is to create and use isolated components. This means each component can only be accessed by one application. It is also called application-private assembly. The .NET encourages users to create isolated application through application-private assembles. Therefore, the application will not be affected by the changes, which are made to the system by other applications. This means that it won't raise any DLL hell issues when using .NET to teach student component programming.

Groh (2002) demonstrated how to create a VB.NET DLL component and tested it by VB.NET. He appreciated that component-based programming promoted the basic concepts of O-O programming (e.g. class, properties, methods, and events) to a high level. He also demonstrated how to use a COM component from VB.NET. While Mead (2003) introduced how to use VB.NET to build a custom control component, Poth (2003) presents a way to create a DLL component first and then sets it up as a custom control component in the toolbox.

How are COM (Microsoft, 2005) and .NET related? Both of them are complementary development technologies. However, Microsoft recommends developers use .NET rather than COM for new development. COM is a feature of Windows and .NET is an attribute of new applications. The .NET provides bi-directional integration with COM. This means we can use COM from .NET (Gunderloy, 2001) as well as call .NET components from COM. (Gunderloy, 2002). Utley (2001) also demonstrated how to use MS ActiveX controls from VB.NET application.

Unfortunately, there is no research paper either to show how to use different .NET languages to create DLL components or how to apply the core O-O techniques (such as inheritance, polymorphism and override) to reuse existing components and to create 100% backward compatible component by different languages. Therefore, the aim of this research is to integrate O-O techniques to component programming with different languages in .NET.

For the purpose of improving teaching of O-O programming, we firstly chose VB.NET instead of VB 6.0 so that no more DLL hell issues could be raised after students completed their components. Now, the most important thing was to select a robust research methodology and to conduct the research at Tairawhiti Polytechnic.

3 Methodology

3.1 Choosing a Research Methodology

Georigi, Pauls, Rochel, Weth, and Fielden, (2005) classified the research methods utilised in NZ into quantitative, qualitative, and mixed methods. The combined mixed method, research aims more to solve the problem rather than work with predefined methods. Esteves and Pastor (2004) argued that collecting different data by different methods from different sources provides a wider range of coverage. It is also called triangulation (Chenail, 1997).

Firstly, we looked to the laboratory experiments approach, to conduct the creating and testing of VB.NET component and its interactions with other language in .NET. However, this approach uses quantitative, analytical, techniques with a small number of isolated variables, which may over-simplify the experimental situation from the real world.

On the other hand, field experiments are an extension of laboratory experiments into the real world situation. It is preferred over laboratory experiments, as these often focus on specific and even narrower issues (Good, Campbell, Lynch and Wright, 1989). But Wixon et. al (2002) suggested that field experiments are mostly conducted within large organizations for long-term product design improvements. It yields immense amounts of data, which need deep, complex methods and techniques to analyse and present them. Therefore, the discount field research, "usability testing" method (Kantner and Keirnan, 2003), is regularly used in short-term data collection projects.

3.2 Use Field Usability Testing

Field usability testing adapts the methodology of laboratory testing by conducting the sessions in the participants' own computer system environment. It is more suitable for problem identification than for performance measures or quantitative comparisons. Kantner, Sova, and Rosenbaum, (2003) argued that when users use their own data with their own equipment in field usability testing, they collect more in-depth feedback about the product itself because there are no distractions from simulated data. Therefore, field usability testing was conducted at Tairawhiti Polytechnic environment. The same case for original VB 6.0 DLL is adapted to this research.

The first goal of field usability testing is to evaluate different versions of VB.NET DLL components working side-by-side at the same computer and to make sure that one isolated application is working well with all the application-private assembles. We used VB.NET to create a DLL component first and then to create an application to test the component. Next, we created a new version of the component and tested it with the same application.

Then field usability testing was used to evaluate how different computer language in .NET interacts with each other in component programming. We used C# to create a DLL component to replace the DLL created by VB.NET and used the same VB.NET application to test this new component.

Once basic O-O class is implemented in component programming, core O-O techniques of inheritance, polymorphism and override also need to be tested by different language in .NET.

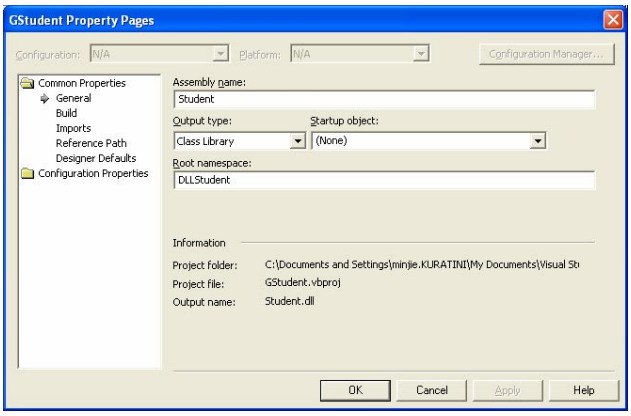
A research diary (Hughes, 2000) is used to keep a records of the field usability testing research. It accumulates knowledge and records the history of the research, the idea and its process. The diary normally includes four parts: reflection, plan, action and observation. In component programming, they are equate to "analyse, design, implementation and testing". It is also easy to generate hands-on teaching resources from the research diary.

4 Findings

The following testing adopts O-O examples from our textbook (Schneider, 2003). It's likely to find the seamless transition from teaching O-O programming to component programming. Also, it tries to find possibilities to reuse components made by different languages. All the testing was done at both tutor's office and student's lab.

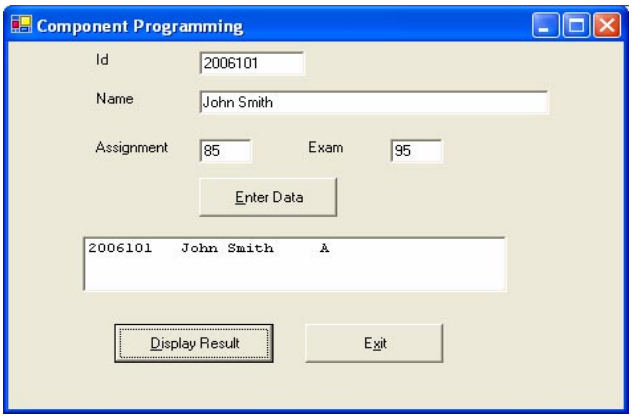
4.1 Create DLL Components by VB.NET

Firstly, we used a traditional O-O class code (class, properties, and methods) (refer Appendix List 1) in the VB.NET component module. We re-named the project property settings (e.g. change Assembly name to Student; Root namespace to DllStudent, see Figure 1), in order to build a component named "Student.dll".



(Figure 1 Project property setting for VB.NET component)

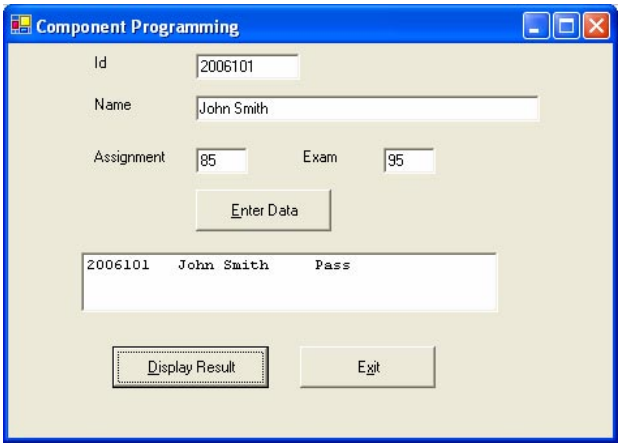
Next, we tested this DLL with a VB.NET application (refer appendix list 2). The application needs to add reference to the "Student.dll" and a statement such as "`Imports DLLStudent`" (namespace) to use the component. We found that the result (see Figure 2) showed that this DLL worked together with the application.



(Figure 2, Result from application-private VB.NET component)

To update this application, we built a new component with a different function (refer Appendix List 3). We also found that the second version of component must be built in the same project property settings (Assembly name: Student; Root namespace: DllStudent). Therefore, we get the second "Student.dll" with the same name in different folder.

So far, we can easily update the application by copying-and- pasting the new DLL to the same application's bin folder, and then double clicking on the exe file there. A different result (see Figure 3) told us that the application has been updated without being rebuilt with the new component. There is no need to register the new DLL to windows registry.



(Figure 3, Testing result from new updated C# component)

We also discovered that it is simple to recover the original Grade Letter function by copying the first DLL back to the application's bin folder to replace the second one. The same results came out as Figure 2, vice versa.

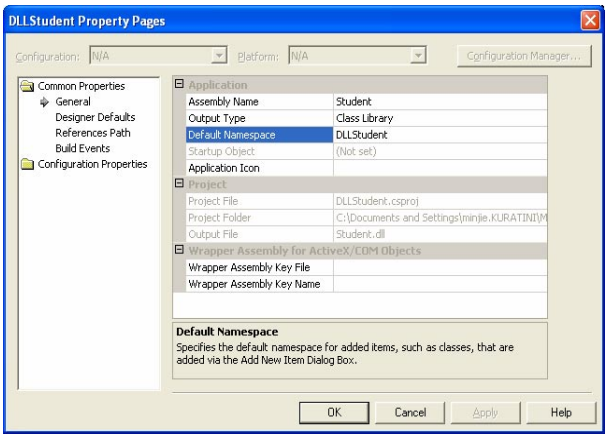
From this testing, we found that it is a seamless transition from O-O programming to component programming. The re-use principle can be presented to our students at compiled component level rather than only at class code level as in traditional O-O programming. All the

components are able to be compatible to the application program if they have used the same name and same namespace. We can develop component and application at the same time. Updating can be easily done by copying-and-pasting. There is neither register requirement for DLL nor rebuild needs for application. We can recover any of the history function if needed. It is certain that components created by students can be assessed at any lab with Windows XP or Windows 2000 plus .NET framework in the Tairawhiti Polytechnic.

4.2 Create DLL Components by C#

We used C# to test whether components could be built by another language and were able to work together with the same application created by VB.NET since our students have already studied C# in previous programme (DipICT Level 5).

After translating the first VB.NET class code (refer Appendix List 1) into C#, we made the same settings (Assembly Name: Student; Default Namespace: DllStudent, see Figure 4) for the new C# component project. Another new "Student.dll" component was built from C#. Copying and pasting this new DLL to a previous VB.NET application, which has been updated to a Pass/Fail Grade. While running the application, the result presents the same Grade Letter from C# component as that from VB.NET component (see Figure 2).



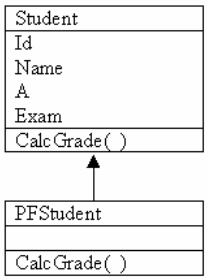
(Figure 4. Project property setting for C# component)

Similarly, to update this application by another C# component with a Pass/Fail Grade function, we translated the second VB.NET class (refer Appendix List 3) into C#. We built the second C# component under the same settings (see Figure 4). Then, we used the new DLL to update the application. The result showed that the Pass/Fail Grade from second C# component is the same as that from second VB.NET Component (see Figure 3). There is no need to rebuild the VB.NET application with the new C# component. Once the old VB.NET component is copied back, the application turns back to its former state.

The most important discovery from this experiment is that we can use a different language in .NET to create components. Our students can choose any language they like to complete their tasks. It integrates their learning not only of C# and VB.NET language but also O-O and components. It is also possible to change the way of development from having many applications sharing one public component, to making immense amounts of different private components in any language, available to one application to choose in various situations.

4.3 Create inheriting component by C#

We successfully transferred the class (properties and methods) from pure O-O programming to component programming. A preferable way of introducing the inheritance, polymorphism and override into component programming, is by adopting another example from our textbook (Schneider, 2003). The hierarchical relationship of the two classes "Student and PFStudent" is presented in Figure 5.

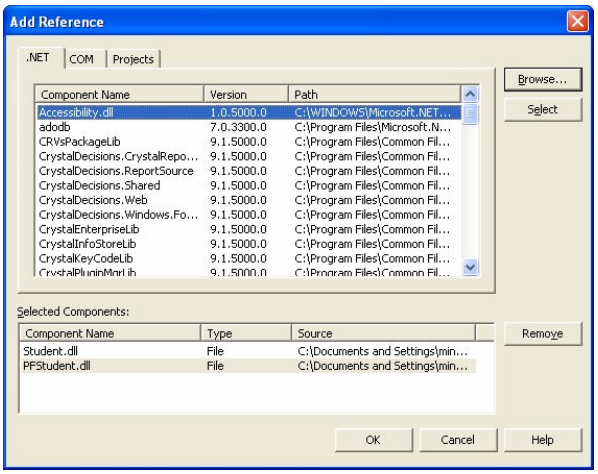


(Figure 5 hierarchical relationships of two classes)

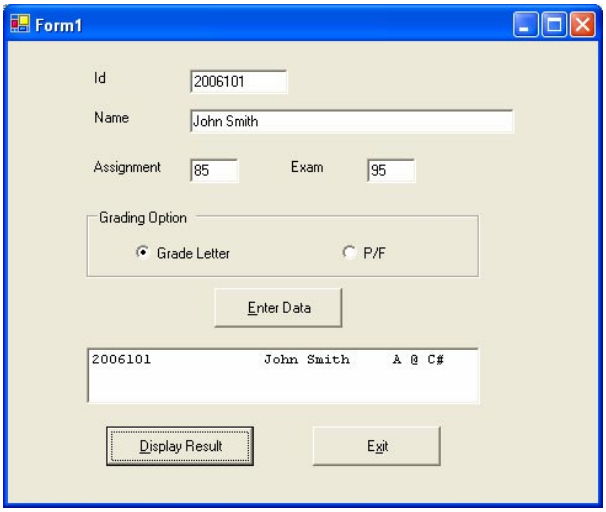
C++ has been used in O-O programming for many years. We began with the new C family member, C#, to test the component programming for inheritance, polymorphism and override.

The parent class "Student" (refer Appendix List 4) was firstly built as component "Student.dll" after the project property had been set up as Student, and DLLStudent. The child class "PFStudent" (refer Appendix List 5) was set as PFStudent, and Inherit Student. The child class was then referenced to its parent "Student.dll" component. Two DLLs "Student.dll" and "PFStudent.dll" were built from the child class.

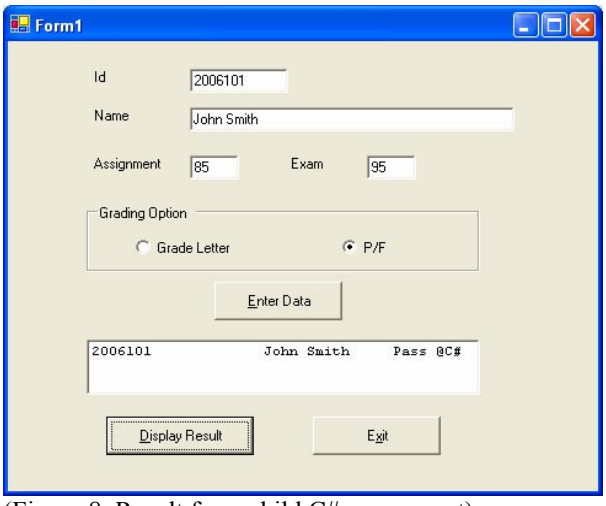
Next, we created a VB.NET inheritance testing application (refer Appendix List 6) to test the DLLs by setting a reference to both components from the child class (see Figure 6). The application results now shows both Letter Grade and Pass/Fail Grade from C# component (see Figure 7 and 8).



(Figure 6, Set reference to child component)



(Figure 7. Result from parent C# component)



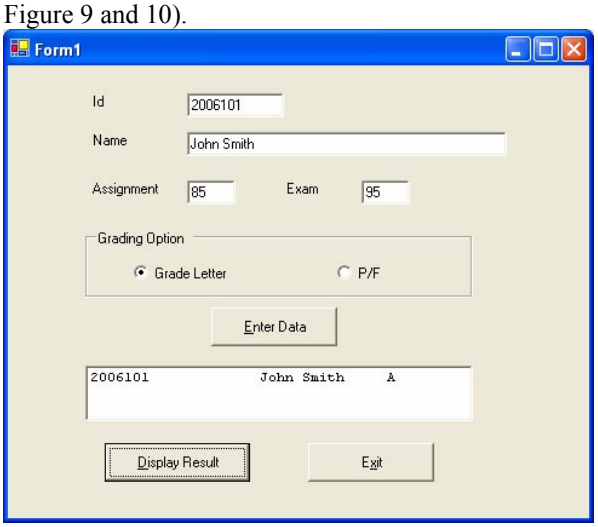
(Figure 8. Result from child C# component)

Inheritance, polymorphism and override can be used not only at O-O code level but also at the component level. We can make a new version of the child component by re-using the whole parent component. This child component is 100% backward compatible to its parent component. No more DLL hell can be caused by .NET components.

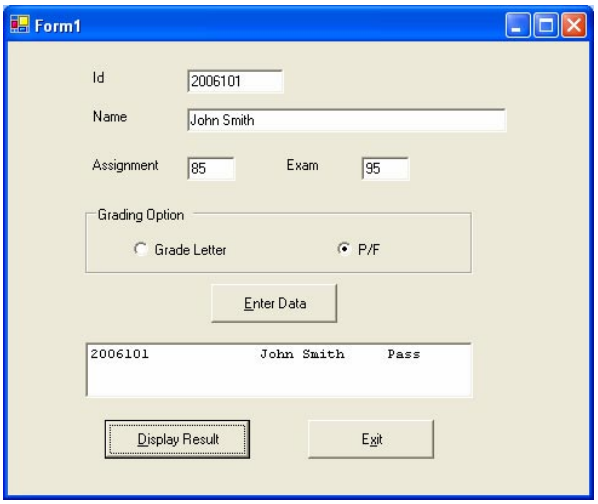
Unfortunately, we cannot build abstract base class and implement its children classes separately. They have to be built together to create only one component. Even though they contain inheritance, polymorphism and override at O-O code level, it is still considered as an individual component, which is similar to that created from one class, as discussed in the first DLL case.

4.4 Create Inheriting Components by VB.NET

To test the same theory with VB.NET using both parent (refer Appendix List 7) and child (refer Appendix List 8) class, we still need to keep the project setting as the previous C# components. Similar to C#, the two components ("Student.dll" and "PFstudent.dll") have been made available by VB.NET in the child's bin folder. After using these two new components to replace the old C# components, the following results show both Letter Grade and Pass/Fail Grade from VB.NET component (see Figure 9 and 10).



(Figure 9. Result from parent VB.NET component)



(Figure 10. Result from child VB.NET component)

We concluded that both C# and VB.NET can be used for inheritance, polymorphism and override in component programming.

4.5 Create Parent Component by VB.NET and Child Component by C#

We tried to mix components created by different languages in order to find out if one would work for the other and vice versa. To create a C# child component with VB.NET parent component, we formed a shortcut by re-using an existing C# child component project, referencing to the VB.NET parent component. From the child project we have received two components, "Stdent.dll" (by VB.NET) and "PFStudent" (by C#). We copied and pasted both DLLs to the inheritance testing application, to replace its old components. It showed Letter Grade from the parent VB.NET component (see Figure 9), and Pass/Fail Grade from the child C# component (see Figure 8).

C# can re-use the VB.NET component to create a new component. The two components created by different languages are able to work together. Even though, we cannot use the code of two different languages for one object-orient program, we can still integrate them together at the component level. It implements the author's idea (Hu, 2005) of integrating computing education (ICE) in component programming.

4.6 Create Parent Component by C# and Child Component by VB.NET

To make a VB.NET child component from a C# parent component, we referenced the child project to the parent. From the child project we also received two components, "Stdent.dll" (by C#) and "PFStudent.dll" (by VB.NET). Following a similar method as 4.5, we used the new DLLs to replace old components. It shows Letter Grade from C# parent component (see Figure 7), and Pass/Fail Grade from child VB.NET component (see Figure 10).

Once again we found that VB.NET can complete what C# does in component programming. It allows the component to be re-used in order to create a new backwards-compatible component. This means that even though they are created from different languages, they can still be re-used.

5 Conclusion

The purpose for conducting this research was to improve the teaching of O-O programming. Literature Review explored what has been done and found out what was needed for this research. Methodology tells us how to conduct this experiment. Analysing the findings, lead to the necessary guidelines for our teaching. This is the "why", "what", and "how" of this research.

O-O programming allows a smooth transition to component programming. The O-O concepts and re-usable principles are seamlessly applied at component level rather than at programming code level. The core O-O techniques such as inheritance, polymorphism and override can be successfully applied to component programming in order to re-use existing components and create 100% backward compatible component from different languages.

Different computer languages can also interact with each other at component level. This allows us to use .NET to write a program with a component in one language and an application in another. We can also develop a project with others using different languages to create different components at the same time.

Students can choose the language of their choice to build a class into component. All the components can be compatible to the application if they have used the same name and namespace. Updating and recovery are easy because there is no requirement to register DLL, nor is it needed to rebuild the application. Components created by students can be applied at any computer with a .NET framework.

Since there is no more DLL hell coming from .NET, it is possible to make many different private components using different languages available to one application chosen in different situations. This means we are able to download or purchase different components to update our program without reinstalling the entire program again.

In summary, teaching component programming is the right choice after O-O programming. Changing to .NET is the first step to improve our O-O teaching. Component programming integrates O-O techniques and different computer languages and promotes the re-use concept to a high level. It is a cool practice of "ICE". This research is based on practice and is aimed at the theory to guide our teaching. Through the cycle of practice-and-theory, we are on the right track towards achieving our goal of best practice.

6 References

Anderson, R. (2000): The end of DLL hell, Microsoft Corporation, http://msdn.microsoft.com/library/en-us/dnsetup/html/dlldanger1.asp

Boondog, P. (1998): Programming custom hardware in Visual Basic, http://www.boondog.com/Ctutorials Cdlltutor Cdlltutor.htm

Chenail, R. (1997): Keeping things plumb in qualitative research, The qualitative report, Vol. 3, No. 3, Sept.

D'Souza, D., Whalen, BJ and Wilson, P. (1999): Implement side-by-side component sharing in application (expanded), Microsoft Corporation, http://msdn.microsoft.com/library/en-us/dnsetup/html/sidebyside.asp

Esteves, J. and Pastor, J. (2004): Using a multi method approach to research enterprise systems implementations, Electronic journal of Business research methods, pp69-82, Vol 2, Issue 2.

Georigi, C., Pauls, S., Rochel, R., Weth, A. and Fielden, K. (2005): A review of research on managing

information technology in New Zealand, Bulletin of Applied computing and information technology, Vol, 3, Issue 2, NACCQ, July

Good, M., Campbell, R., Lynch, G. and Wright, P. (1989): Experience with contextual field research, proceedings of CHI '89 Human factors in computing systems, ACM, pp21-24

Groh, M. (2002): Creating components in .NET, Microsoft Corporation, http://msdn.microsoft.com/library/en-us/dndotnet/html/componentsnet.asp

Gunderloy, M. (2001): Calling COM components from .NET clients, Microsoft Corporation, http://msdn.microsoft.com/library/en-us/dndotnet/html/callcomcomp.asp

Gunderloy, M. (2002): Calling a .NET component from a COM component, Microsoft Corporation, http://msdn.microsoft.com/library/en-us/dndotnet/html/callnetfrcom.asp

Hu, M.J. (2005): ICE: Integrated computing education, an individual integrated computing teaching experience, The 18th NACCQ proceedings, pp 183-188.

Hughes, I. (2000): How to keep a research diary, Action research e-reports, and the university of Sydney

Kantner, L. and Keirnan, T. (2003): Field research in commercial product development, UPA 2003 proceedings, Usability professionals' association

Kantner, L., Sova, D. and Rosenbaum, S. (2003): Alternative methods for field usability research, SIGDOC 2003 proceedings, pp 68-72.

Mead, G. (2003): Simple steps in VB.NET building a custom control, http://www.devcity.net/Articles/87/1/ssteps_customcontrol.aspx

Mgama (2002): MSI custom action DLL, http://www.codeproject.com/tips/msicustomaction.asp

Miller, K. (1999): Creating and using DLLs, http://www.flipcode.com/articles/article_creatingdlls-pf.shtml

Microsoft (2005): COM: Component object model technologies, Microsoft Corporation, http://www.microsoft.com/com/default.mspx

Poth, G. (2003): How to make a DLL with VB.NET standard edition, http://www.devcity.net/Articles/111/1/vbnet_se_dll.aspx

Pratschner, S. (2001): Simplifying development and solving DLL hell with the .NET framework, Microsoft Corporation, http://msdn.microsoft.com/library/en-us/dndotnet/html/dplywithnet.asp

Schneider, D. (2003): An introduction to programming using Visual Basic .NET, 5th edition, pp528-564, Prentice Hall

Utley, C. (2001): Using ActiveX controls with windows forms in visual studio .NET, Microsoft Corporation, http://msdn.microsoft.com/library/en-us/dndotnet/html/actxctrlswinforms.asp

Webopedia (2005): What is DLL, A word definition from Webopedia Computer Dictionary, http://www.webopedia.com/TERM/D/DLL.html

Wikipedia (2006): DLL hell, The free encyclopedia, http://en.wikipedia.org/wiki/DLL_hell

Wixon, D., Ramey, J., Holtzblatt, K., Beyer, H., Hackos, J., Rosenbaum, S., Page, C., Laakso, S. and Laakso, K. (2002): Usability in practice: Field methods evolution and revolution, CHI 2002, pp880-884, ACM

Wong, F. (1998): DLL hell, the inside story, Desaware Inc., http://www.desaware.com/tech/dllhell.aspx