



Fostering Computing Education in NZ

Bulletin of Applied Computing and Information Technology

Refereed Article A3:

06:01

Dec 08/Jan 09

One Step Ahead of Teaching Object-Oriented Programming

Min-Jie Hu
Tairawhiti Polytechnic, New Zealand
(min@tairawhiti.ac.nz)

Hu, M-J. (2008/2009, Dec/Jan), One Step Ahead of Teaching Object-oriented Programming. *Bulletin of Applied Computing and Information Technology* Vol. 6, Issue 1. ISSN 1176-4120. Retrieved January 19, 2009 from http://www.naccq.ac.nz/bacit/0601/2008Hu_OOP.htm

ABSTRACT

Teaching programming is hard. Teaching object-oriented programming (OOP) is even harder. It is often an obstacle to persuade students to believe in the advantages of OOP when they study the course. Often the reuse of code on sub-procedures is more preferable to students rather than that on class inheritance and overriding. It can be difficult to understand how OOP is related to the reusable, maintainable and extendible ability of software. This paper discusses the possibility of introducing component-oriented programming (COP) after OOP. A framework of Computing Research Methods (CRM) is utilised to design the course for the integration of OOP and COP through spiral cycles of the framework. Concept-first and Pair Programming approaches are also employed. Some recommendations are made on how the teaching might be improved in order to bridge the gap to further courses.

Keywords

Object-oriented programming, OOP, Component-oriented Programming, COP, DLL.

1. INTRODUCTION

Teaching programming is hard. Teaching object-oriented programming (OOP) is even harder. It is often an obstacle to persuade students to believe in the advantages of OOP when they study the course. Often the reuse of code on sub-procedures is more preferable to students rather than that on class inheritance and overriding. It can be difficult to understand how OOP is related to the reusable, maintainable and extendible ability of software.

One of the significant promises (Sparks, Benner, & Faris, 1996) of OOP is to reuse code. Class reuse is often by inheritance. A set of classes, which is named as framework, is designed to work together to solve a problem. A framework (Johnson, 1992) is a reusable design of a program, which is created by experts in a particular domain and then used by non-experts. A particular application is created as concrete subclasses from the abstract classes in the framework. Thus, framework reuse is considered to lead to strong gains in productivity, quality, consistency, portability and maintainability.

Design patterns (Gamma, Helm, Johnson, & Vlissides, 1994; Schmid, 1996) are utilised as a guideline on the construction and documentation of frameworks. But they are more abstract

than frameworks, which are micro-architectures that are applied to a cross-domain design problem. Using design patterns to achieve high levels of design is too abstract because of no actual code.

Component-based software engineering (CBSE) has brought the notion of reusable software components and component-oriented programming (COP) into widespread use within both the commercial software and education communities (Parrish, Hale, Dixon, & Hale, 2000; Spalter, 2002; Liu & Cunningham, 2004). With CBSE, software can be developed by gluing prefabricated components together like assembling a "stereo system" (Siddiqui, 2000, Wikipedia, 2007a). Using reusable concrete components to build up a framework is more realistic and acceptable. Once students understand the reuse of components, it will be easy for them to build reusable frameworks under the guide of design patterns in the future. Since objects and components are like brothers from the same family (Taylor, 1998), the goal of this research is focused on how to help students understand the integration of OOP and COP.

At Tairawhiti Polytechnic, OOP is taught at the second year course DipICT Level 6. After teaching OOP, the author introduced COP to let student build DLL components and create applications by using MS VB 6.0. However, students are frustrated that their assignment with DLL, which worked perfectly on student's home computer, does not directly work on the school's computer until the DLL is registered to the Windows registry. This is because there is a problem with COM, which is called "DLL Hell" (Anderson, 2000).

This paper reviews the current literature in the field to get rid of the shadow of DLL hell first, and then discusses how to introduce COP after OOP. Concept-first, Pair Programming and a framework of Computing Research Methods (CRM) are employed to set up a framework to teach the integration of OOP and COP. Finally, some recommendations are made on how the teaching might be improved in order to bridge the gap to further courses.

2. LITERATURE REVIEW

2.1 Components and DLL

"A component is a modular and replaceable part of a system that encapsulates its contents" (Arlow & Neustadt, 2005). It is a reusable piece of software in binary form, which has an extension file name like .dll, .ocx, or exe. 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 for creating DLL is to let a single DLL component to be shared by several applications at the same time. Thus, research on how to design and use shared components was initiated in education and industry.

Miller (1999) conducted research on creating and using DLLs in MS VC++ 6.0. Mgama (2002) also demonstrates how to create a DLL in VC++ and add it to Windows Installer. Since Visual Basic (VB) is gradually becoming more and more popular, Boondog (1998) has successfully linked the MS VB 5.0 application to a DLL component created by MS VC 5.0. The finding also indicated that VB could access 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 for COP. Microsoft provides DLLs for both Windows and its own applications. 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.

2.2 Assembly and Namespace

In .NET (Wikipedia, 2007b), assembly is a partially compiled code library. It equates to DLL, exactly known as "logic DLL". A namespace (Wikipedia, 2007c) is a context in which a group of one or more identifiers might exist. An identifier defined in one namespace is independent

of the same identifier declared in another namespace. This means an assembly is identified by both assembly name and namespace. On the other hand, two assemblies can be compatible when they have the same assembly name and the same namespace.

The notion of side-by-side (D'Souza, Whalen, & Wilson, 1999; Pratschner, 2001) makes it possible to install and run different versions of components on the same computer. A principle guideline in .NET is to create and use isolated components. This means one 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 assemblies. Therefore, the applications will be no longer affected by the DLL changes from other applications. .NET will not raise any DLL hell issues when using it to teach COP.

Furthermore, Groh (2002) demonstrated how to create and test a DLL component in VB.NET. He appreciated that COP promoted the basic concepts of OOP (e.g. class, properties, methods, and events) to a high level.

3. METHODOLOGY

Georgi, Pauls, Rochel, Weth and Fielden (2005) classified research methods into quantitative, qualitative, and mixed methods. The combined research method aims to best 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).

3.1 Concepts-first and UML

Many educators (like Zhu & Zhou 2003) emphasised OO concepts-first and then OOP when they noticed the problem that students still code their C++ programs in a non-object-oriented style after OOP. Similarly, a conceptual approach (Madsen, Ron, Thorup, & Torgersen, 1998) was successful in teaching OOP. The sequence of teaching begins with the conceptual framework of object-orientation, then OOP, and finally OOAD.

Bennedsen and Caspersen (2004) developed a model-driven object-first approach in their teaching of OOP. The conceptual model is expressed in Unified Modelling Language (UML). Then it is followed by a coding pattern for the translation from the UML to code. Xu (2004) also uses UML to allow students to visually present outcomes of OO analysis and plan the process of code implementation. Recent research (Ritzhaupt & Zucker, 2006) reported success in teaching OOP using VB.NET. It also argued that UML class diagrams help students to understand OOP better. The undividable relationship of UML and OOP is similar to that of Explanation and Mechanism (Soloway, 1986).

3.2 Pair Programming

Many educators are also interested in adopting pair programming practices in the teaching programming (Greca, Jovanvic, & Harris, 2003; Hanks, 2004; Hanks, McDowell, Draper, & Krnjic, 2004; Baheti, Williams, Gehringer, & Stotts, 2002). With pair programming, two students work at one computer. One student plays the role as the driver while the other is acting as the navigator of their programming. Hanks et al. (2004) found that the paired students are more confident in their work and more satisfied with the programming than the frustrated solo students. Baheti et al. also applied both team and pair approaches to their OOP.

3.3 Framework for CRM

Many educators (Masters, 2000; Mahony & Poulos 2004) applied a spiral of cycles of action strategies for both teaching improvement and quality assurance. Recently, a group of educators (Holz, et al. 2006) have successfully utilised a cyclical framework for computing research methods (CRM) as a teaching device at different levels for the transition from novice to expert.

This framework is widely applied to teach both research and non-research courses. It adopts action research cycle to computing discipline. The cycle to facilitate teaching is described as:

- A. "What do we want to achieve?" (Observe),
- B. "Where does the data come from?" (Reflect), "How to collect? Where to collect?" (Plan),
- C. "What do we do with the data?" (Act),
- D. "Have we achieved our goal?" (Observe).

Therefore, we firstly employ the Concepts-first approach and UML as a vehicle for the effective problem solving in COP. Then we choose the combination of team and pair programming to organise our teaching practice. The framework of CRM is applied as a guideline of our course design for the integration of OOP and COP.

4. COURSE DESIGN

4.1 Team and Pair for DLLs from a Single Class

Following the framework of CRM (Holz, et al. 2006), (A) the goal of the cycle is designed to let students evaluate whether one application program works well with many private assemblies. (B) The assemblies will be made by the Student Class, which is adapted from our textbook (Schneider, 2003) with the same property and method name. (C) The students are divided into two teams. Both teams build up DLLs with the same assembly name and namespace, but different functions. (D) One member from each team is chosen to form a pair to test both DLLs using one application. The testing results provide the answer towards the goal.

In the first cycle, (A) students aimed to find out whether one VB.NET application can be updated by many DLLs made by VB.NET. (B) The component "Student.dll" is based on the UML (see Figure 1). (C) Two student teams build by VB.NET with the same project settings (see Figure 2). One team implements CalcGrade () with function for "Letter Grade". The other codes CalcGrade () with function for "Pass/Fail Grade". (D) A member from each team is chosen to form a pair. They test the DLLs from each group using the same application to find out whether the DLLs can be replaced by each other.

The part of the Application 1 code in VB.NET, which was modified from textbook, is listed in Figure 3. Both namespace and assembly name are needed for the Class here unless the Imports statement is used for namespace (see Figure 6).

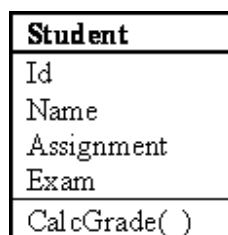


Figure 1. Student Class in UML

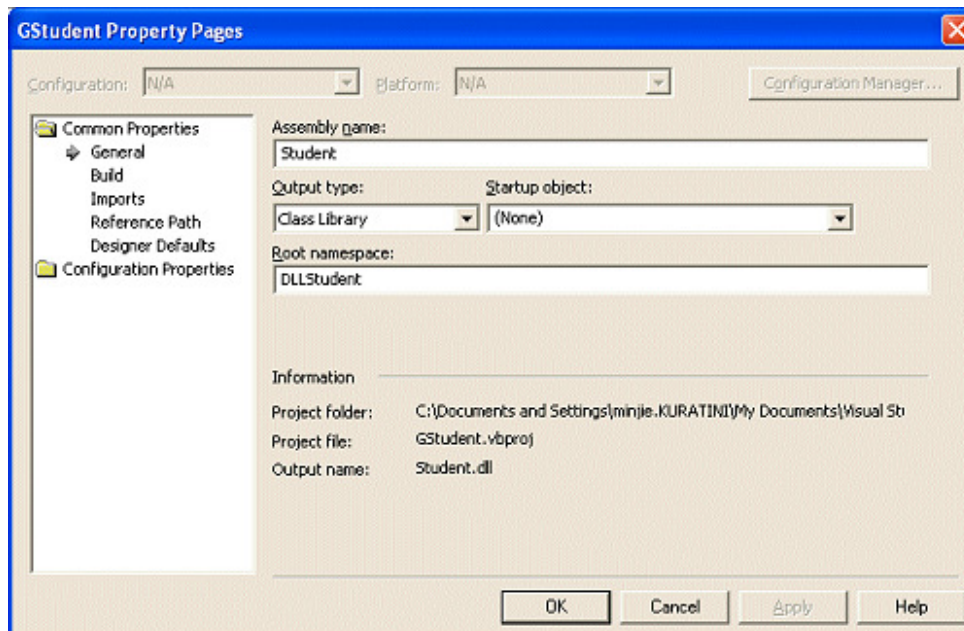


Figure 2. Project property settings

```
Dim pupil As DLLStudent.Student

    pupil = New DLLStudent.Student

'code for object omitted here
```

Figure 3. Part of test application code in VB.NET

After the first cycle, the pair should find the VB.NET application could be updated by different DLLs from VB.NET. The second cycle is to (A) find out whether the VB.NET application can be updated by many DLLs made by C#. (B) The UML model is still the same, but the language has changed to C#. (C) Both teams are required to create DLLs with different functions by C# under the same project settings. (D) The pair of students from different groups are to test their new DLLs to find out if the DLL from C# works with the application by VB.NET and also to see if the DLLs can be replaced by each other. The setting of Teams and Pairs is listed in Table 1.

Table 1. Team and Pair Setting for Application 1

Application 1	Letter Grade Team	Pass/Fail Grade Team
Pair Test 1	Student.dll (VB.NET)	Student.dll (VB.NET)
Pair Test 2	Student.dll (C#)	Student.dll (C#)

4.2 Team and Pair for DLLs from Parent-child Classes

After the transition from the basic Class (properties and methods) of OOP to COP, the further course is designed to let students reuse the DLL components. They are going to use the existing DLL to generate a new DLL, which needs to be backward-compatible to its original one. Can different languages still work together with COP?

A more preferable way of introducing the inheritance, polymorphism and override into COP is by adopting another example from our textbook (Schneider, 2003). The hierarchical relationship of the two classes "Student and PFStudent" is presented in Figure 4.

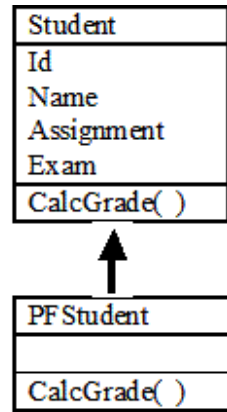


Figure 4. Hierarchical Relationships of Parent-Child Class

Both Parent and Child DLLs will be built separately. Their class code in both VB.NET and C#, which was changed from textbook, is listed in Figure 5.

```

Public Class Student
'code for property is omitted here
Overridable Function CalcGrade() As String
'code for function as Letter omitted
End Function
End Class
Public Class PFStudent
'Set referece to Student.dll
Inherits DLLStudent.Student
Overrides Function CalcGrade() As String
'code for function as Pass/Fail omitted
End Function
End Class
namespace DLLStudent //parent class
{ public class Student : System.ComponentModel.Component
{// code for properties is omitted here
public virtual string CalcGrade()
{//method for Letter Grade omitted}}}
using DLLStudent; //parent namespace
namespace InheritStudent//child namespace
{public class PFStudent : DLLStudent.Student // inherit
{public override string CalcGrade()
{//method for Pass/Fail Grade omitted }}}
```

Figure 5. Part of Parent-Child Class Code in VB.NET and C#

The Application 2 code in VB.NET, which was changed from textbook, is listed in Figure 6.

```

Imports DLLStudent 'parent namespace
Imports InheritStudent 'child namespace
Private Sub btnEnter_Click(..)..
Dim pupil As Student
If RadioButton1.Checked Then
pupil = New Student
Else
pupil = New PFStudent
End If
'code for object omitted here
End Sub
```

Figure 6. Code for Test Parent/Child DLLs in VB

The teams still create DLLs while the pair is going to test the DLLs using the application. The new setting of Teams and Pairs is listed in Table 2.

The third cycle is going to (A) reuse the existing DLL to generate a new DLL in the same language. (B) The new DLL is built from the Child Class, which has assured backward-compatibility to its Parent class. The second application program is adapted to fit both Parent and Child DLLs. (C) Student teams are built up by their favourite computer language. One team is composed of students who like VB.NET. The other team is composed of those who prefer C#. Every team needs to create two components "Student.dll" (CalcGrade () with function for "Letter Grade") and "PFStudent.dll" (CalcGrade () with function for "Pass/Fail Grade") with the same project settings. (D) The pair is set up again with one from each team to test whether both Child (inherited from its Parent DLL in the same language) and Parent DLLs work in the same application.]

Table 2. Team and Pair Setting for Application 2

Application 2		VB.NET Team	C# Team
Pair Test 3	Parent:	Student.dll (VB.NET)	Student.dll (C#)
	Child:	PFStudent.dll (VB.NET)	PFStudent.dll(C#)
Pair Test 4	Parent:	Student.dll (C#)	Student.dll (VB.NET)
	Child:	PFStudent.dll (VB.NET)	PFStudent.dll (C#)

After the achievement of reusing DLLs, similarly, the last cycle of this framework is to (A) find out whether reusing can be applied in a mixture of different languages. (B) Students still use the same Class UML model and application. (C) Two teams swap the Parent DLL. Then the VB.NET team is going to generate a Child DLL by VB from a Parent DLL made by C# while the C# team creates a Child DLL by C# from a Parent DLL made by VB. (D) The pair will work together again to test the mixture of Parent and Child DLL made by different languages.

5. EXPERIMENTAL RESULTS

5.1 Build and Test Two Replaceable DLLs

With the first cycle to (A) update an application by different DLLs made by VB.NET, (B) we introduce the UML and provide an example of the class and application code in VB.NET. (C) Students experiment with building up their first component (Student.dll) as a team. Two students from different teams carry two DLLs with different functions: "Letter Grade" and "Pass/Fail Grade". (D) Each pair of students recognised that the application must be referenced to the DLL (see Figure 7) during their first trial.

Reference is no longer required afterwards when the application is updated by another DLL. Nor do they need to register the new DLL component to the Windows registry. The simplest way to update the application is just by copying-and-pasting the new DLL to its bin folder. The test results from two DLLs are shown in Figure 8.

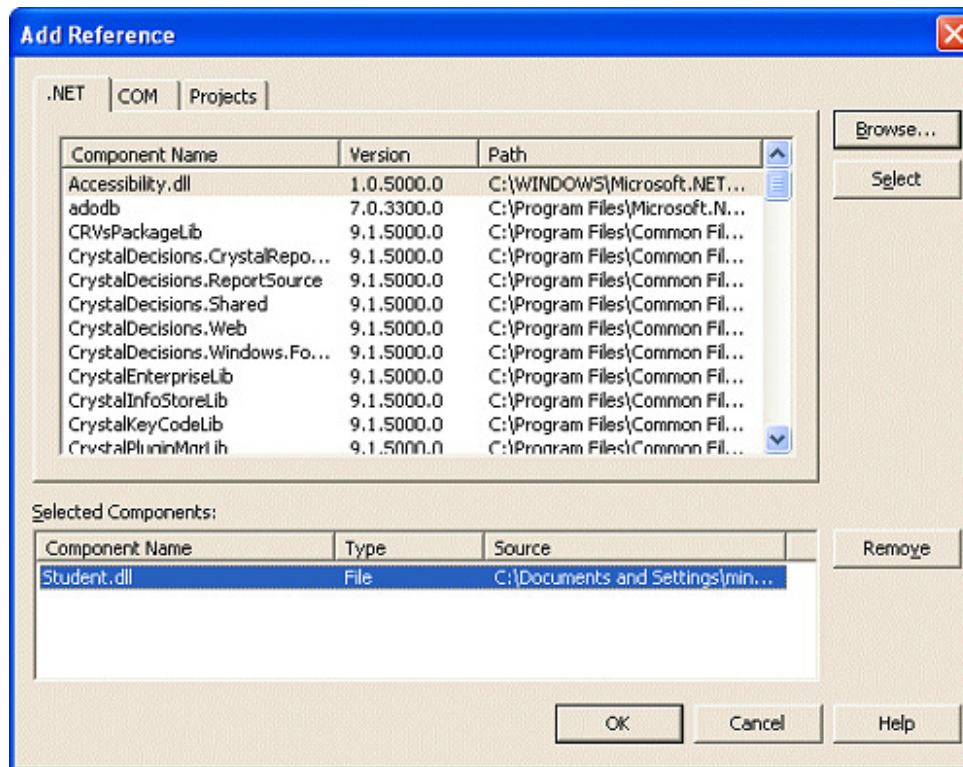


Figure 7. Add Reference to DLL for application

From building and testing DLLs within team and pair, students learned that even using the same name, DLLs could be implemented to provide different functions. These DLLs can be used to update the same application if they have the same assembly name and namespace.

We concluded that one application could employ many different DLL components for different purposes; one application is reusable to many DLLs. It helps students to understand the reuse principle at component level rather than only at class code level. They can also easily update or recover any of the history functions if needed.

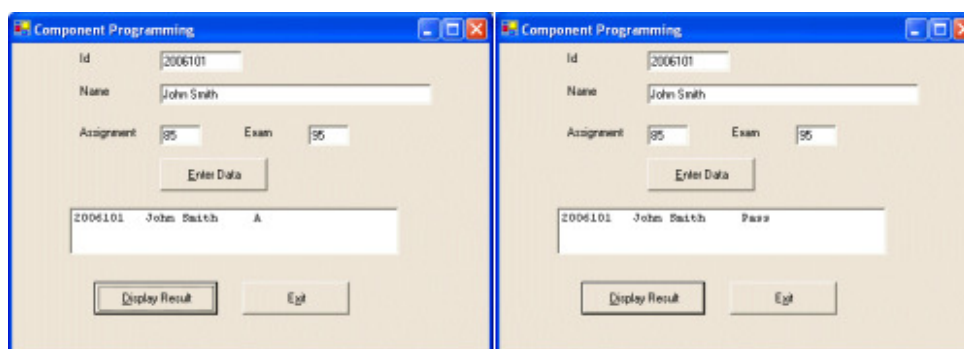


Figure 8. Test Results from two DLLs (Zoom: [Left](#), [Right](#))

Through observation of lab tasks and marking of assignments, we find that students are engaged in the DLL components. None of them attempted to avoid using OO method to implement their tasks. They appreciated the role that OO plays in COP. For building DLL, students need to learn OO. With the success of COP, they understand OOP in a further step.

Students are also keen to find if they could use different languages to build the DLLs. Therefore, the second cycle of teaching framework follows the first findings.

(A) To find out whether DLLs made by C# can be applied to update the same VB.NET application, (B) students are required to translate the same UML model by C#. With the same project setting, two teams have built up two different versions of "Student.dll" again using C#. (C) By copying-and-pasting the new DLL to update the application, (D) the pair of

students found the same results as previous using DLLs from VB.NET (see Figure 8). They also found there is no need to rebuild the VB.NET application with a new C# component. Once a previous VB.NET DLL is copied back, the application turns back to its former state. Therefore, students can choose the language they like to create DLL components with different functions. These DLLs can be used to update the same application easily.

5.2 Build and Test both Parent and Child DLLs

So far, students understand that the application can be updated to a new function by replacing the DLL. The further step for students to learn is how the replacement could be compatible to its original function by the integration of OOP and COP. Thus, the third cycle of this framework focus on how to (A) reuse the existing DLL and generate a new compatible DLL. It starts with using the same language for both old and new DLLs. (B) Students join the team with their preferred language VB.NET or C#. They learned how to build a Child DLL by inheriting the function from its Parent DLL with one language they utilised. Each team obtained two DLLs, "Student.dll (Parent DLL) and "PFstudent.dll" (Child DLL) from the Child DLL project when it referenced to its Parent DLL component.

(C) The pairs are set up again from the teams. They test each member's Parent-Child DLLs on one application, which has reference to the DLLs. The results of the testing are as shown in Figure 9. Similarly, with copying-and-pasting, they replace one member's DLLs by the other member's. They received the same results as previous.

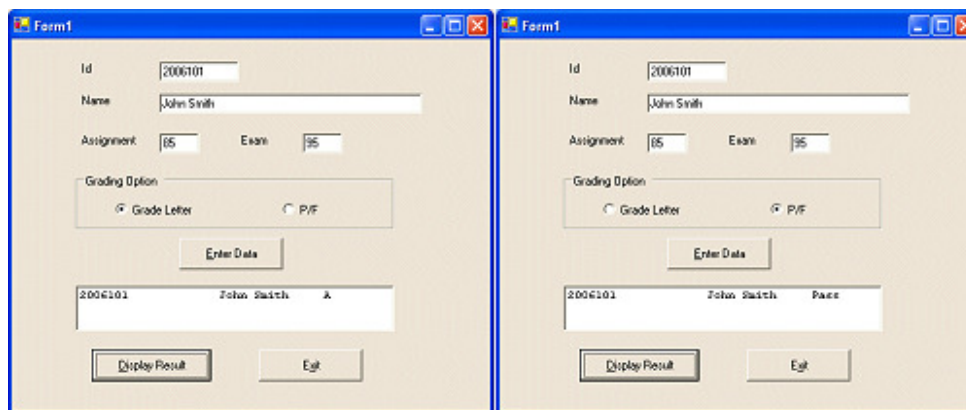


Figure 9. Result from Parent-child DLL (Zoom: [Left](#), [Right](#))

(D) Students learned that inheritance, polymorphism and override can be used not only at OO code level but also at component level. They can make a new version of child component by reusing the parent component. This child component is 100% backward-compatible to its parent component. This is because the child component was made up by including both Parent and Child DLL. No more DLL hell can be caused by the Parent-Child .NET components.

Similarly, students are keen to know whether the Parent-child DLLs are language independent. The goal of the fourth cycle is to (A) reuse the Parent DLL to build a Child DLL by different languages. (B) Two teams swap their Parent DLLs. Each team still use the same language to build a new Child DLL from the Parent DLL, which was built by the other team in a different language. Thus, the VB team got Parent DLL by C#, Child DLL by VB.NET; C# team had Parent DLL by VB.NET, Child DLL by C#.

(C) The pair of students uses their mixed Parent-Child DLLs to update the previous application. They are pleased to see the previous way of copying-and-pasting is still applied to update the application by these DLLs from mixed languages.

(D) The students found out that two DLL components created by different languages are able to work together. Even though students cannot use the code of two different languages for one OOP, they are still able to integrate different languages together at component level. DLL can be reused to create a new backward-compatible component with language-independence. It also language-independently updates the application by simply copying-

and-pasting.

6. CONCLUSION

The purpose of conducting this research is to improve the teaching of OOP. Based on the approaches of concepts-first and mixture of team and pair programming, the spiral cycles of framework actively sweep into an integrated course design. The teaching experience indicates that it is a smooth transition from OOP to COP. The O-O concepts and reusable 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 COP. It reuses existing components and creates 100% backward-compatible components from different languages.

"Seeing is believing". Through several cycles of building and reusing DLL components, students experienced the O-O reuse from concept to practice in a concrete format. Many years of teaching in COP proves that it helps students to understand the different reuse between sub-procedure and Class. It also motivates students to learn and use OOP to solve problems for the purpose of component reuse. COP fills the gap between OOP and design patterns with frameworks. After COP, it is then the time to introduce design patterns to guide students through how to organise components to build a framework for a certain domain. Otherwise, it is too abstract for students to understand the high-level knowledge and design reuse only from the OOP code and static UML map.

Since different computer languages can also interact with each other at component level, this allows students to use .NET to write a component in one language and an application in another. They can also develop a project with others using different languages to create different components at the same time.

All the DLLs can be compatible if they have used the same name and namespace. Updating and recovering are easy because there is no requirement to register the DLLs, nor does it need to rebuild with the application. DLLs created by students can be used on any computer with .NET framework.

Students are no longer frustrated by DLL hell. They have now gained a better understanding of why or why not they have to restart their computer after downloading a new component to update existing program or installing a new program. Students are also interested to build up their own applications through downloading or purchasing third party components. They are even more confident to make many different components, which can be built by any language, available to one application to choose in different situations.

In summary, teaching COP is a right choice after OOP. Using .NET is the first step towards the improvement of teaching OOP. The framework of CRM is the guideline for the course design. The approaches that were best practiced in OOP such as Concepts-first approach and UML, and team and pair programming, are still effectively applied in COP. COP integrates OO techniques and different computer languages. It promotes the reuse from OOP to a high concrete level. COP makes it easier for students to understand the reuse in OOP and stimulates their motivation to study OOP and use it to solve problems. Students gain much more confidence in OOP by the language independence of building components. COP is presented as a consolidated step ahead of teaching OOP, which bridges the gape between OOP and student's further study in terms of OOAD, design patterns and framework.

7. ACKNOWLEDGEMENTS

The author would like to thank Dr. Donald Joyce and two anonymous reviewers for their comments, which have significantly contributed to the revision of the paper.

REFERENCES

Anderson, R. (2000). The end of DLL hell. Retrieved December 12, 2005, from <http://msdn.microsoft.com/library/en-us/dnsetup/html/dlldanger1.asp>

- Arlow, J. & Neustadt, I. (2005). UML 2 and the unified process: Practical object-oriented analysis and design (2nd ed.). Addison-Wesley.
- Baheti, P., Williams, L., Gehringer, E. & Stotts, D. (2002). Exploring pair programming in distributed object-oriented team projects. Paper presented at the OOPSLA Educator's Symposium, Seattle, WA.
- Bennedsen, J. & Caspersen, M. (2004). Teaching object-oriented programming - Towards teaching a systematic programming process. Paper presented at the ECOOP04 Conference, Oslo, Norway.
- Boondog. (1998). Programming custom hardware in Visual Basic. Retrieved November 19, 2005, from <http://www.boondog.com/Ctutorials%5Cdlltutor%5Cdlltutor.htm>
- Chenail, R. (1997). Keeping things plumb in qualitative research. *The Qualitative Report*, 3 (3)
- D'Souza, D., Whalen, B. J. & Wilson, P. (1999). Implementing side-by-side component sharing in applications (expanded). Retrieved December 5, 2005, from <http://msdn.microsoft.com/library/en-us/dnsetup/html/sidebyside.asp>
- Esteves, J. & Pastor, J. (2004). Using a multimethod approach to research enterprise systems implementations. *Electronic Journal of Business Research Methods*, 2(2), 69-82.
- Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.
- Georgi, C., Pauls, S., Rochel, R., Weth, A. & Fielden, K. (2005). A review of research on managing information technology in New Zealand. *Bulletin of Applied Computing and Information Technology*, 3(2).
- Greca, A., Jovanovic, V., & Harris, J. (2003). Enhancing learning success in the introductory programming course. Paper presented at the 33rd ASEE/IEEE Frontiers in Education Conference, Denver.
- Groh, M. (2002). Creating components in .NET. Retrieved November 19, 2005, from <http://msdn.microsoft.com/library/en-us/dndotnet/html/componentsnet.asp>
- Hanks, B. (2004). Distributed pair programming: An empirical study. Retrieved December 10, 2005, from <http://repositories.cdlib.org/cgi/viewcontent.cgi?article=1713&context=postprints>
- Hanks, B., McDowell, C., Draper, D., & Krnjic, M. (2004). Program quality with pair programming in CS1. Paper presented at the ITiCSE'04 Conference, Leeds, UK.
- Holz, H., Applin, A., Haberman, B., Joyce, D., Purchase, H., & Reed, C. (2006). Research methods in computing: What are they and how should we teach them? *SIGCSE Bulletin*, 38(4), 96-114.
- Johnson, R. (1992). Documenting frameworks using patterns. Paper presented at the OOPSLA'92 Conference, Vancouver, Canada.
- Liu, Y. & Cunningham, C. (2004). BoxScript: A component-oriented language for teaching. Paper presented at the 43rd ACM Southeast Conference, Kennesaw, GA.
- Madsen, O., Ron, H., Thorup, K., & Torgersen, M. (1998). A conceptual approach to teaching object-orientation to C programmers. Retrieved December 1, 2005, from <http://www.daimi.au.dk/~olm/PUB/teachingootocprg.pdf>
- Mahony, M. & Poulos, A. (2004). Strengthening the nexus between teaching and learning through increased attention to feedback to students: A research-led teaching approach. Paper presented at the Australian Association for Research in Education Conference, Melbourne, Australia.
- Masters, J. (2000). The history of action research. Retrieved December 1, 2005, from <http://www2.fhs.usyd.edu.au/arow/arer/003.htm>
- Mgama (2002). MSI custom action DLL. Retrieved November 11, 2005, from <http://www.codeproject.com/tips/msicustomaction.asp>
- Miller, K. (1999). Creating and using DLLs. Retrieved November 19, 2005, from http://www.flipcode.com/articles/article_creatingdlls-pf.shtml
- Parrish, A., Hale, D., Dixon, B., & Hale, J. (2000). A case study approach to teaching component based software engineering. *Proceedings of the 13th Conference on Software Engineering Education & Training* (pp 140-150). Washington, DC: IEEE Computer Society.
- Pratschner, S. (2001). Simplifying development and solving DLL hell with the .NET framework Retrieved December 1, 2005, from <http://msdn.microsoft.com/library/en-us/dndotnet/html/dplywithnet.asp>
- Ritzhaupt, A. & Zucker, R. (2006). Teaching object-oriented programming concepts using Visual Basic.NET, *Journal of Information Systems Education*, 17(2), 163-169.
- Schmid, H. (1996). Design patterns for constructing the hot spots of a manufacturing

- framework. *Journal of Object-Oriented Programming*, 9(3), 25-37
- Schneider, D. (2003). *An introduction to programming using Visual Basic .NET* (5th ed.). Prentice Hall.
- Siddiqui, F. (2000). *Component based software engineering*. Retrieved December 1, 2005, from <http://www.smb.uklinux.net/reusability/>
- Soloway, E. (1986). Learning to program = Learning to construct mechanisms and explanations. *Communications of the ACM*, 29(9), 850-858.
- Spalter, A. (2002). Problems with using components in educational software. Paper presented at the International Conference on Computer Graphics and Interactive Techniques, San Antonio, TX.
- Sparks, S., Benner, K. & Faris, C. (1996). Managing object-oriented framework reuse. *IEEE Computer*, 29 (9), 52-61.
- Taylor, D. (1998). Are objects obsolete? *Component Strategies*, 1(1), 16-17.
- Webopedia. (2005). What is DLL? Retrieved December 1, 2005, from <http://www.webopedia.com/TERM/D/DLL.html>
- Wikipedia. (2006). DLL hell. Retrieved March 15, 2006, from http://en.wikipedia.org/wiki/DLL_hell
- Wikipedia. (2007a). Software componentry. Retrieved May 15, 2007, from http://en.wikipedia.org/wiki/Software_componentry
- Wikipedia. (2007b). .NET assembly. Retrieved May 15, 2007, from http://en.wikipedia.org/.NET_assembly
- Wikipedia. (2007c). Namespace (computer science). Retrieved May 15, 2007, from http://en.wikipedia.org/wiki/Namespace_%28computer_science%29
- Wong, F. (1998). DLL hell, the inside story. Retrieved December 1, 2005, from <http://www.desaware.com/tech/dllhell.aspx>
- Xu, T. (2004). Programming concepts and methods for enhanced student learning. Retrieved December 1, 2005, from <http://www.cs.ubc.ca/wccce/Program03/papers/TongXu/TongXu.htm>
- Zhu, H. & Zhou, M. (2003). Methodology first and language second: A way to teach object-oriented programming. Paper presented at the OOPSLA'03 Conference, Anaheim, CA.

Copyright © 2008-2009 Min-jie Hu

The author(s) assign to NACCQ and educational non-profit institutions a non-exclusive licence to use this document for personal use and in courses of instruction provided that the article is used in full and this copyright statement is reproduced. The author(s) also grant a non-exclusive licence to NACCQ to publish this document in full on the World Wide Web (prime sites and mirrors) and in printed form within the Bulletin of Applied Computing and Information Technology. Authors retain their individual intellectual property rights.

Copyright © 2008-2009 NACCQ, Krassie Petrova and Michael Verhaart (Eds.). An Open Access Journal, DOAJ # 11764120.