Visual Based Software Construction: Visual Requirement Authoring Tool and Visual Program Generator

DENG-JYI CHEN*, MING-JYH TSAI*, CHUNG-YUAN HUANG**
*) Department of Computer Science
National Chiao Tung University
1001 Ta Hsueh Road, Hsinchu, Taiwan, 300, ROC
**) Department of Computer Science and Information Engineering
Chang Gung University
259 Wen-Hwa 1st Road, Tao-Yuan, Taiwan, 333, ROC
djchen@csie.nctu.edu.tw, mjtsai@csie.nctu.edu.tw, gis89802@gmail.com

Abstract: - Most software development errors are caused by incorrect or ambiguous specifications gathered during the requirement elicitation and analysis phase. For the past two decades, both academic researchers and software engineers have been seeking better software requirement analysis and representation approaches in order to overcome this problem. Another equally important issue in software engineering is reducing the cost of developing and maintaining application software. Here we propose a visual based software construction approach that uses multimedia requirement scenarios created by a visual requirement authoring tool to depict application software requirements. It also supports a visual program generator that allows system developers to generate target application systems as specified in multimedia requirement scenarios. The target application software can be generated based on the function binding feature provided in the visual program generator. Evidence is presented showing that application software development and maintenance costs can be reduced by applying the proposed software construction approach.

Key-words: Visual representation, multimedia requirement scenario (MRS), visual program generator (VPG), multimedia, visual requirement authoring tool (VRAT), user requirements.

1. Introduction

An important phase in the software life cycle focuses on eliciting requirements from users [24, 28]. Gathering requirement information is considered a difficult task. Users and system developers generally have no common terminology or domain knowledge to share when establishing software development requirements. It is widely recognized that the form of requirement representation can affect communication between software end users and designers. The most common requirement representation form (text-based narration) usually leads to misunderstandings and irrelevant or ambiguous information being shared. Text documentation created during the requirement gathering process is often difficult to comprehend. To address some of these problems, there are an increasing number of innovative approaches that use visual (multimedia or animated) formats to represent software requirement scenarios [5, 19, 20, 21, 22].

Multimedia technology has played an important role in modern computing environments because it offers more natural and user-friendly interactions with automated systems. This is especially true for systems that utilize graphical, icon, or window-based input and output. Multimedia technology also facilitates "reuse" [8, 15, 16], since basic components and animation can be reused for several scenarios. Therefore, multimedia technology is frequently used to graphically display software requirement specifications (SRSs).

Since documents can only display the static graphical appearance of SRSs, there is a need for explanatory text and flow charts that still fail to truly simulate dynamic specification behavior. Clear explanations of dynamic behavior often result in excessive amounts of text and flow charts that fail to simplify or clarify input.

In the most commonly used process, a user interface (UI) programmer implements the SRS using software development tools such as Borland C++ Builder [10, 12] or Microsoft Visual C++ [11, 14] to write the corresponding functional program and to implement the SRS for the application system. This method makes a UI programmer's workload heavier during the software development phase because the programmer not only has to write the functional program but also has to implement the SRS; this increases system development costs. SRS changes require the UI programmer to rewrite code; this increases system maintenance costs.

In this paper, we propose a visual based software construction approach that uses a multimedia requirement scenario (MRS) created by a visual requirement authoring tool (VRAT) to depict application software requirements, provide a more natural means of communication between

users and facilitators, and elicit early user feedback. System developers can use the confirmed MRS as input for binding scene actors with their corresponding software components or external execution files to generate target application software. Thus, application software development and maintenance costs can be reduced

2. Related Work

There are several commercially available authoring tools for visual requirement specification, including Microsoft Visio [17], Microsoft PowerPoint [23], Apple Keynote, and Macromedia's Flash [26], or Dreamweaver. A comparison of their features is presented in Table 1. As shown, they all allow adjustments to component position and size via dragging and dropping, but considerable differences exist for several other features. Macromedia Flash uses script programs to create interactive relationships, making it unsuitable for users who don't have programming experience. Furthermore, the four tools require significant effort for modifying and maintaining script programs. Although these tools enable requirement facilitators to author visual requirement specifications, they cannot be used as input data for generating application software.

Table 1 Comparison of features among three visual representation authoring tools and the proposed approach.

Item for Comparison	Authoring Tools			
	Microsoft Visio	Microsoft PowerPoint	Macromedia Flash	Proposed approach
Provides galleries of multimedia components or icons.	No	Yes	Yes	Yes
WYSIWYG approach to visual representation authoring.	Yes	Yes	Yes	Yes
Supports visual-based control operations (e.g., switch-case, loop, if-then-else).	No	Yes	Yes	Yes
Allows for interactive action without writing script programs.	No	No	No	Yes
Supports various scene patterns for creating specification requirements.	No	No	No	Yes

Some software development packages (e.g., Borland C++ Builder [10, 12], Borland JBuilder, Eclipse [7], and Microsoft Studio.Net [11, 14]) have easy-to-use design tools for creating labels, data-entry fields, list boxes, check boxes, combo boxes, buttons, and other UI controls (or actors) when assembling visual UI scenes. However, users must write code in languages such as C++, BASIC, or Java to implement the actions. Therefore, their use is limited to individuals who have programming backgrounds (Table 2).

Table 2. A comparison of software development tools with the proposed approach.

Item for Comparison	Development Tools			
	Borland C++ Builder	Microsoft Visual C++	Eclipse	Proposed approach
Provides many UI controls.	Yes	Yes	Yes	Yes
Provides galleries of multimedia components or icons.	No	No	No	Yes
WYSIWYG approach to component authoring.	Yes	Yes	Yes	Yes
Supports visual-based control operations (e.g., switch-case, loop, if-then-else).	No	No	No	Yes
Allows for interactive action without writing script programs.	No	No	No	Yes
Supports various UI patterns for creating MRSs and generating target application systems.	No	No	No	Yes

Unified Modeling Languages (UMLs) are currently in widespread use for developing application software systems. Although UMLs enable users to define system frameworks for generating source code via sequence, use case, and class diagrams (among other methods), they lack features that support UI program generation. The proposed VRAT was purposefully designed in response to this limitation. Rational Rose [29], Together [3], and PowerDesigner [25] do not support features for GUI program generation (Table 3).

Table 3. A comparison of CASE tools with the proposed approach.

Item for Comparison	CASE Tools			
	Rational Rose	Together	PowerDesigner	Proposed approach
Supports features for scene authoring.	No	No	No	Yes
Supports scenario definition and visual-based control operations (e.g., switch-case, loop, if-then-else) for creating interactive scenarios.	No	No	No	Yes
Supports automatic scene program generation.	No	No	No	Yes
Supports various scene patterns for creating multimedia-based presentations.	No	No	No	Yes

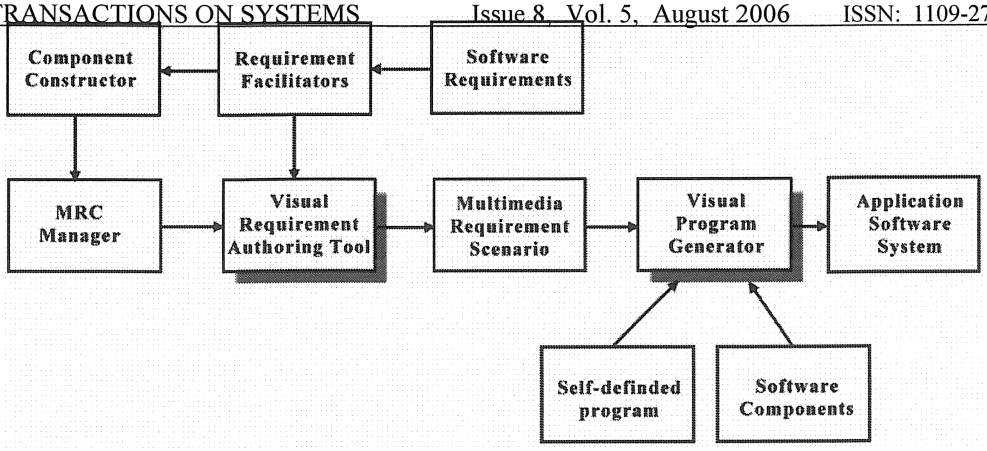

Fig. 1. The visual based software construction model.

3. Visual Based Software Construction Model (Fig. 1)

The framework for this model has two major subsystems: the visual requirement authoring tool (used to create the MRS) and the visual program generator (used to generate target application software according to the VRAT-generated MRS). The following terms will be used in this discussion:

- *Software requirements:* It includes three distinct levels -business requirements, user requirements, and functional requirements. In addition, every system has an assortment of non-functional requirements.
- *Requirement facilitator:* an individual who uses the VRAT to create the MRS.
- *Component constructor:* a tool for making new multimedia reusable components (MRCs) that are used to represent basic elements in the authoring tool (text, video, image, animation, etc.).
- *MRC Manager:* a database management system that provides an interface for adding or deleting MRCs and for retrieving existing MRCs.
- *Visual Requirement Authoring Tool (VRAT)*: a multimedia authoring tool.
- *Multimedia Requirement Scenario:* requirement specifications represented in a visual format generated by the VRAT.
- *Visual Program Generator (VPG):* used to bind MRS actors with their underlying functional programs (existing library functions) and a system developer's self-defined program to generate an application system.
- *Software components:* existing library functions written by programmers. These are usually written as part of a dynamic linking library.

Requirement facilitators use existing MRCs in conjunction with the VRAT to produce an MRS. In the absence of an existing MRC, requirement facilitators can use the component constructor to produce a new one, then add it to the MRC manager for further use. A MRS is produced when the authoring process is completed. The MRS is then used as input for the VPG, which uses a) the function binding system to bind the software component and b) the developer self-defined functional program (according to the MRS) to generate the target application system.

4. Visual Requirement Authoring Tool and Visual Program Generator

4.1 System architecture

The system architecture of the Visual Software Construction Framework is depicted in Fig. 2. The system architecture includes the VRAT (top left of the figure) and VPG. The VPG consists of two systems: a function binding system (bottom left) and a program generating system (top right).

System operation procedures consist of five steps:

1. Requirement facilitators select a scene background for actor layout, and decide on how many scenes will be required for the complete MRS.

2. For each scene, new actors are added, deleted, or copied using the scene authoring tool. The requirement facilitator chooses appropriate MRCs from the MRC database for actor settings.

3. Actor interaction (or scenario) settings are created by using the interaction authoring tool to execute dynamic behavior interaction simulations. Requirement facilitators can preview the authoring result (MRS) using the player system. Changes are made by repeating steps 1, 2 and 3 until the MRS is satisfied.

4. Actor binding with corresponding software components is performed. A system developer can choose appropriate software components (usually in *.dll format) from an application function library. If the desired component is not available, then the system developer must define, compile, integrate, and implement the function before binding can occur.

5. In the current version, application system code is compiled using Borland C++ Builder [8, 10] to generate execution files.

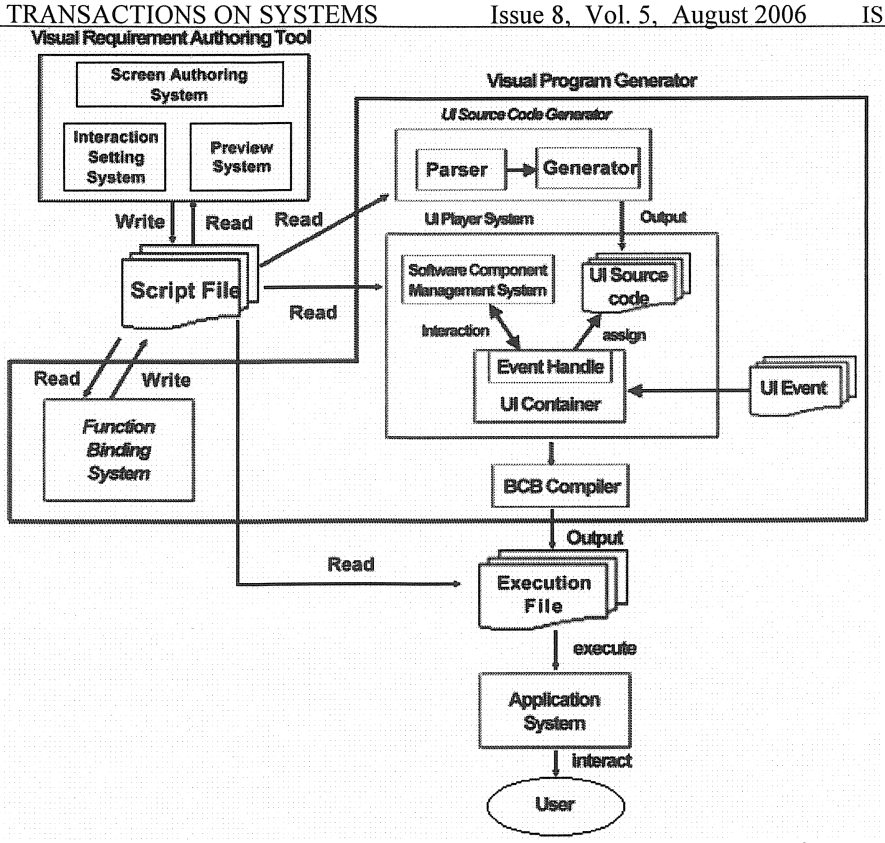

Fig. 2. System architecture of the visual based software construction

4.2 Visual Requirement Authoring Tool (VRAT)

The main VRAT goal is to assist in the layout of multimedia requirement scenarios for target application systems without writing text-based programs. This also allows for faster feedback from end-users. The VRAT contains three systems (Fig. 2):

1. *Scene authoring system.* Every effort was made to support multimedia formats in order to provide users with a large number of actor layout options. Primary purposes are authoring scenes, achieving interaction between scenes, and defining interactions among actors—in other words, creating an MRS for each scene. In addition to text and button actors, we made it possible for facilitators to provide multimedia, animation, audio, video, and image actors to enrich scene representation. The scene authoring tool makes it possible for the corresponding script language program to be automatically generated without any coding effort on the part of the GUI programmer.

2. *Interaction setting system* (also known as the *scenario setting system*). This system is responsible for interactive definitions, time and space relationships, and dynamic behavior among actors. Our proposed system specifically supports such scenarios as prelude, finale, connection, scene switching, URL connections, and interactive actor relationships.

3. *Preview system.* As its name implies, this system is used to preview visual requirement scenarios created by the scene and interaction authoring systems. It interprets time and space relationships and scenario definitions among actors according to script programs generated by the interaction setting system. A scene is downloaded by the player according to the script file prior to user interaction according to interactive relationship and dynamic behavior records in the script file. Users can then decide if the scenario meets their requirements.

The VRAT is best viewed as a multimedia authoring tool consisting of script language, a graphical user interface (GUI) for interconnecting MRC, and animation. It is used to visually capture a requirement and play it back to the user. The VRAT allows for the conversion of textual requirements to the MRS, which consists of three elements:

- Scene — a container for actors or a space for actor placement. Each scene may contain several actors and scenario definitions.
- Actors — also known as MRCs, the basic MRS unit. MRCs are encapsulated as object-oriented paradigms and designed in a standardized format.
- Scenario — sometimes referred to as a use case. It is a useful way to understand time and space relationships among scene actors.

4.3 Visual Program Generator (VPG)

The main purpose of the visual program generator (VPG) is to reduce the cost of application system development and maintenance. It uses the confirmed MRS as function binding system input for binding each actor with its corresponding application software function, external executable file, or self-defined program. The final VPG step is

generating the target application system code. The VPG has four subsystems:

1. Function Binding System

The purpose of this system is to *directly* embed the software component into the MRC (or actor). Once a programmer designs a software component, the function binding system embeds it in an actor and formulates the interactive relationship, which is subsequently written into the script files. The three types of binding programs are:

- Matching software components with a system. After defining software components and interfaces, the system can load and execute matches. Once software components and actors (e.g., text, animation, and buttons) are matched in a scene, a mouse click on an actor triggers an interaction event that allows the system to load the component.
- External execution files. The system allows for the binding of external execution files and scene actors. Here a mouse click triggers an interaction event that runs the execution file.
- Self-defined program. In the absence of a software component or external execution file for binding actor interactions, the system developer must provide a self-defined program. This feature gives system developers considerable flexibility.

2. UI Source Code Generator

Since the function binding system is not a program compiler environment, system developer-defined programs cannot be added to the application system. The UI source code generator was developed for that purpose. The generator requires script files to operate properly.

3. UI Player System

To play back the results of UI authoring, the player system uses a script interpretation mechanism, a resource communication mechanism to retrieve multimedia files, and a scenario decoding mechanism for time and space presentations of scenes and actors. The three parts of the UI player system are: a) UI source code generated by the UI source code generator; b) a software component management system for dispatching messages among components loaded into the system; and c) a UI container for displaying and playing UI backdrops, actor images, animation, video, and audio, plus UI information processing and displacement (Fig. 2). The UI container interacts with the software component management system and UI source code via the UI event handler.

4. Compile Environment

System developer-defined programs can be added to the UI system and compiled to produce target application system execution files. The current version of the proposed system uses Borland C++ Builder to compile programs. UI execution entails reading script files to acquire the UI, interaction data, and other information. The UI is displayed in a manner that allows for user interaction.

5. Visual Based Software Construction Assessment

In addition to creating MRSs (UI specifications), software development tools such as Borland C++ Builder and Microsoft Visual C++ can also generate application systems. We compared the proposed approach with these two tools in terms of time spent in system development and maintenance.

5.1 Purpose

An example of a digital camera application system was used to determine if our proposed visual based software construction approach can help designers who are accustomed to using such tools as Borland C++ Builder and Microsoft Visual C++ to save time when performing development and maintenance tasks.

5.2 Participants

Study participants were 40 students enrolled in a software engineering class in a Taiwanese graduate school. The students' majors were computer science (30), electrical engineering (6), information management (2), and other computer-related departments (2). Participants all had at least three years' experience in software programming-related areas.

5.3 Design

All participants were asked to use either Borland C++ Builder or Microsoft Visual C++ to develop and maintain the digital camera application system described in Appendix A, then use our proposed approach to perform the same task. All students had prior knowledge of the two commercially available software development tools and received extensive tutoring in the use of the VRAT and function binding system. The task was divided into system development and maintenance sub-tasks. The sheet shown in Appendix B was used to collect data on the participants' efforts.

5.4 Data and Analysis

5.4.1 Preferred software development tool

Approximately two-thirds (26) of the students chose Borland C++ Builder 5.0 (BCB5). The other 14 chose Microsoft Visual C++ 6.0 (MVC6) for this assignment.

5.4.2 Development Time

As shown in Fig. 3, it took the 26 participants who chose to work with BCB5 an average of 460 minutes to complete all development tasks, compared to 377 minutes using the proposed approach. The 83-minute difference represents a time savings of 18%.

Data on time spent developing the digital

camera application system using MVC6 are shown in Fig. 4. The average time was 447 minutes, compared to 375 minutes when they used the proposed approach. The 72-minute difference represents a time savings of 16.1%.

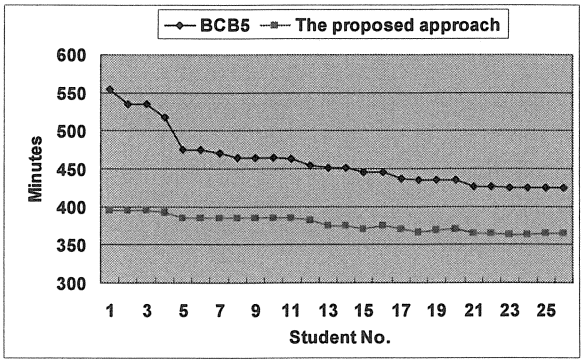

Fig. 3. Time spent in system development using BCB5 and the proposed approach.

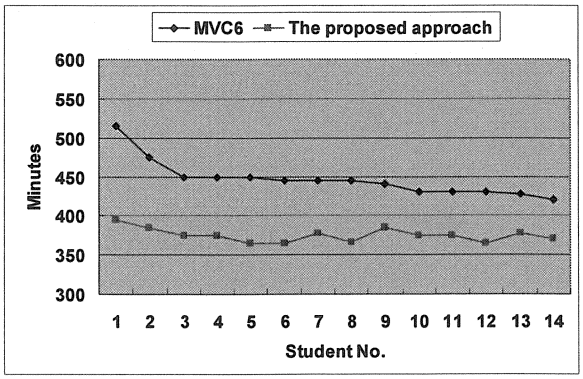

Fig. 4. Time spent in system development using MVC6 and the proposed approach.

The combined results shown in Fig. 5 indicate that the participants needed an average of 455 minutes to develop the assigned system using their preferred software development tool and an average of 376 minutes using the proposed approach—an average savings of 79 minutes, or 17.4%.

5.4.3 Maintenance Time

Data on time spent maintaining the digital camera application system using BCB5 are shown in Fig. 6. It shows that the average time spent was 162 minutes, compared to 144 minutes using the proposed approach—a savings of 18 minutes, or 11.1%. The same data for students who chose MVC6 are shown in Fig. 7. Here the average difference was 18 minutes (163 using MVC6 versus 145 using the proposed approach), or 11%.

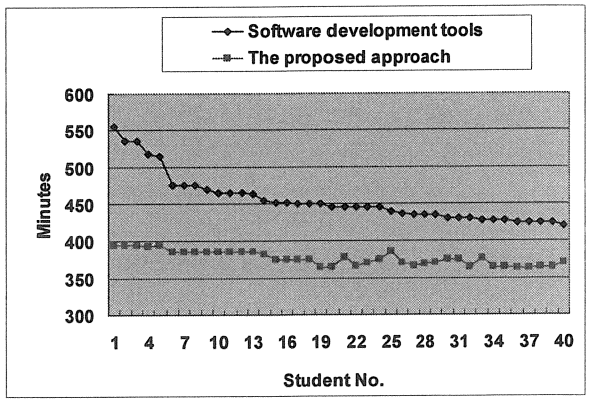

Fig. 5. Time spent in system development using various software development tools and the proposed approach.

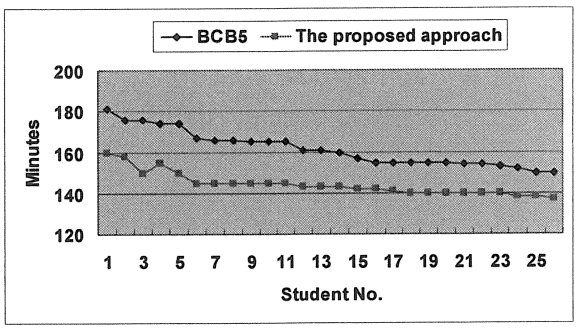

Fig. 6. Time spent in system maintenance using BCB5 and the proposed approach.

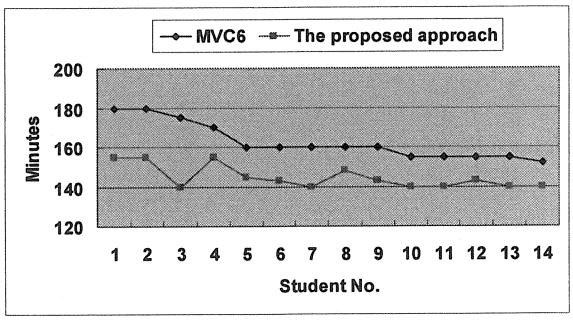

Fig. 7. Time spent in system maintenance using MVC6 and the proposed approach.

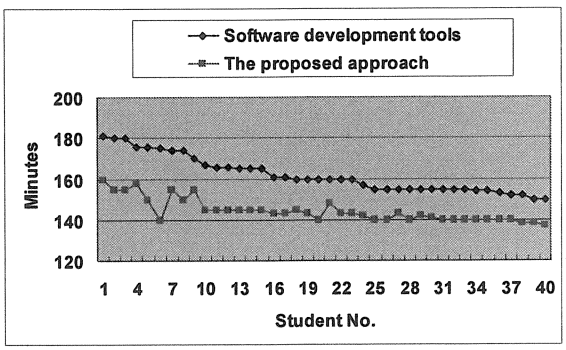

Fig. 8. Time spent in system maintenance using software development tools and the proposed approach.

The combined data in Fig. 8 show that the participants needed an average of 162 minutes for system maintenance using one of the two software development tools and an average of 144 minutes

using the approach described in this paper—a difference of 18 minutes, or 11.1%.

6. Conclusions

The proposed approach described in this paper has the following benefits:

- It replaces large amounts of text documentation.
- It allows facilitators to author MRSs without writing text-based programs.
- It provides a more natural means of communication between users and requirement facilitators.
- It allows for early user feedback that is more expressive in describing user requirements.
- It supports various scene patterns for reuse by requirement facilitators.
- It allows confirmed MRSs to be reused as input data for generating target application systems, freeing UI programmers from the need to repeatedly implement UI requirement specifications.
- It provides window-based operations for system developers to bind actors with corresponding software components without requiring them to write text programs.
- Its use reduces the amount of time required for application software development and maintenance.

Appendix A: Developing and Maintaining a Digital Camera Application System

To compare the effort required to use the visual based software construction approach with Borland C++ Builder and Microsoft Visual C++, a digital camera application system assignment was split into two phases.

A.1 Developing a Digital Camera Application System

The two user interfaces (UIs) in the digital camera application system are the Main (Fig. A-1) and Snapshot screens (Fig. A-2). The system enters the Snapshot screen when a user clicks on the PC Camera icon on the Main screen. To return to the Main screen, the user must click on the Back icon on the Snapshot screen.

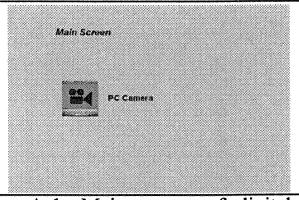

Fig. A-1. Main screen of digital camera application system.

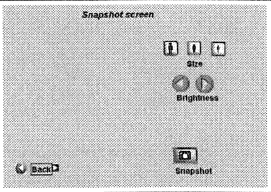

Fig. A-2. Snapshot screen of digital camera application system.

The Main screen has two text actors: a "main screen" and "PC Camera" actor. The only image actor on the Main screen is. The Snapshot screen has four text actors: "Size," "Brightness," "Snapshot," and "Back." The seven image actors on the Snapshot screen are , , , , , , . The image actors control four features:

- Size: Large, medium, and small size are respectively represented by , , and .
- Brightness: Image brightness can be increased by clicking on the icon and decreased by clicking on the icon.
- Snapshot: To take a picture, the user clicks on the icon.
- Back: To return to the main screen, the user clicks on the icon.

A.2 Maintaining the Digital Camera Application System

We added three functions to the digital camera application system:

- A Help function on the Main screen that links the user with a Website for assistance. This requires adding one text actor (Help) and one image actor ().
- A Contrast Adjustment function on the Snapshot screen to increase or decrease image contrast. This requires adding one text actor (Contrast) and two image actors (and).
- A Save Image function on the Snapshot screen, which requires adding one image actor ( ).

Depictions of complete application system UIs are shown in Fig. A-3 - A-4.

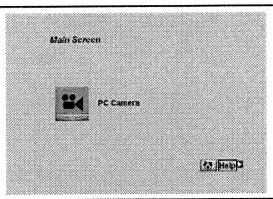

Fig. A-3. Complete Main screen of digital camera application system.

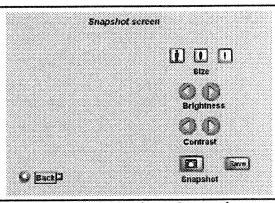

Fig. A-4. Complete Snapshot screen of digital camera application system.

Appendix B

Name:

Task	Method	
	Software development tools	Proposed approach
Time spent in system development (minutes)		
Time spent in system maintenance (minutes)		

Which software development tools did you use to develop and maintain the target UI application software system?

- □ Borland C++ Builder 5.0
- □ Microsoft Visual C++ 6.0

References

[1]. Astesiano, E. and Reggio, G., Knowledge Structuring and Representation in Requirement Specification, *Proceeding of the 14th International Conference on Software Engineering and Knowledge Engineering*, Ischia, Italy, 2002.

[2]. Ben Shneiderman, Catherine Plaisant, *Designing the User Interface*, Pearson/AW, 2005.

[3]. Borland international Co., *Borland Together Edition for Visual,* Borland International Co., 2004

[4]. Browne, G.J. and Rogich M.B., an Empirical Investigation of User Requirements Elicitation: Comparing the Effectiveness of Prompting Techniques, *Journal of Management Information Systems*, Vol. 17, No. 4, 2001, pp. 223-249.

[5]. Chen, D.J., Chen, W.C., Kavi, K.M., Visual requirement representation, *Journal of Systems and Software*, 61, 2002, pp. 129-143.

[6]. Chen, D.J., Huang, S.K., Interface of Reusable Software Components, *Journal of Object-Oriented Programming*, Vol. 5, No. 8, 1993, pp 42-53.

[7]. David Carlson, *Eclipse Distilled*, 2005, AW/Person.

[8]. Freeman, P., A Perspective on Reusability, In tutorial: Software reusability, *IEEE Computer Society Press,* 1987 pp. 2-8.

[9]. Grady Booch, James Rumbaugh, Ivar Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1999.

[10]. Herbert Schildt, Greg Guntle, *Borland C++ Builder: The Complete Reference*, McGraw-Hill, New York, NY, 2001.

[11]. Horton, Ivor, *Beginning Visual C++*, Wrox, Chicago, Illinois, 1998.

[12]. Jarrod Hollingworth, Paul Gustavson, Bob Swart, Mark Cashman, *Borland C++ Builder 6 Developer's Guide*, SAMS, Indianapolis, Indiana, 2002.

[13]. Jirotka, M., Heath, C. Luff, P., Ethnography by video for requirements capture, *Proceeding of the 2nd International Symposium on Requirements Engineering*, April 27-29, York, England, 1995, pp. 190-191

[14]. Kate Gregory, *Microsoft Visual C++ .NET 2003*, SAMS, Indianapolis, Indiana, 2003.

[15]. Lenz, M., Schmid, H.A., Wolf, P.W., Software reuse through building blocks, *IEEE Software*, Vol. 4, No. 4, 1987, pp.34-42.

[16]. Massont, P., van Lamsweerde, A., Analogical reuse of requirements frameworks, *Proceeding of the 3rd International Symposium on Requirements Engineering,* July 6-10, Annapolis, Maryland, 1997, pp. 26-37.

[17]. Mark H. Walker and Nanette Ea, Mark H. Walker, Nanette Eaton, *Microsoft Office Visio 2003 Inside Out*, Microsoft press, 2003.

[18]. Miller, E.; Kado, M.; Hirakawa, M.; Ichikawa, T.; HI-VISUAL as a user-customizable visual programming environment, *Proceeding of the 11th IEEE International Symposium on visual languages*, 1995, pp. 107-113.

[19]. Ohnishi, A., A Visual Software Requirements Definition Method, *Proceeding of the First International Conference on Requirements Engineering*, April 18-22, Colorado Springs, Colorado, 1994, pp.194-201.

[20]. Ohnishi, A., Audio-visual Software Requirements Specification, *Proceeding of the IEEE International Workshop on Multimedia Software Engineering Conference (MSE'98)*, April 20-21, Kyoto, Japan, 1998, pp. 26 – 33.

[21]. Ohnishi, A., Tokuda, N., Visual Software Requirements Definition Environment, *Proceeding of the 21st International Computer Software and Application Conference (COMPSAC)*, IEEE, New York, August 13-15, 1997, pp.624-629.

[22]. Ohnishi, A., VRDL: A Visual Software Requirements Language, *Transaction of the SDPS*. Vol. 3, No. 3, 1999, pp. 43-52.

[23]. Patrice-Anne Rutledge, *Special Edition Using Microsoft Office PowerPoint*, Que, 2003.

[24]. Pressman, R.S., *Software Engineering: a Practitioner's Approach*. McGraw-Hill, New York, NY, 2005.

[25]. Powersoft/Sybase, *Sybase PowerBuilder 9.5,* 2002, Sybase

[26]. Robert Reinhardt, Snow Dowd, *Macromedia Flash 8 Bible*, Hungry Minds, 2006.

[27]. Silva, P. P. de, Griffiths, T., Paton, N. W., Generating User Interface Code in a Model Based User Interface Development Environment, *Proceeding of the International Working Conference on Advanced Visual Interfaces (AVI2000)*, Palermo, Italy, 2000, pp. 1555-1600.

[28]. Sommerville, Ian, *Software Engineering*, Addison Wesley, New York, NY, 2004.

[29]. Terry Quatrani, *Visual Modeling with Rational Rose 2002 and UML*, 2002, AW/Person