Block-Based Approach for End-User Software Development

Article *in* Asian Journal of Information Technology · June 2011

DOI: 10.3923/ajit.2011.249.258

1 author:

Abdullah Mohd Zin
National University of Malaysia
149 PUBLICATIONS **914** CITATIONS

SEE PROFILE

Asian Journal of Information Technology 10 (6): 249-258, 2011
ISSN: 1682-3915

Block-Based Approach for End-User Software Development

Abdullah Mohd Zin
Programming and Software Technology Research Group,
Faculty of Information Science and Technology, Universiti Kebangsaan Malaysia, Bangi, Malaysia

Abstract: End-user programming refers to programming activities carried out by end users. These end-users can include teachers, accountants, scientists, engineers, parents and all other people who are not trained as programmers. End-user programming is now getting more popular. It was projected that the number of end-user programmers in the US is about 55 million compared to 2.75 million professional programmers. In order to support end-user programming, a number of programming systems have been developed which can be divided into the following categories: Application-specific languages, programming by example, visual programming and natural programming. This study discusses the concept of a new programming system to support end-user programming called the block-based programming system. In this programming environment, end-users can develop applications by integrated programming blocks that have been developed by block developers.

Key words: End-user programming, component-based software development, block-based programming, super-enduser, Malaysia

INTRODUCTION

Computers in the old days were not as accessible as they are today. Software at that time was not built for the general public. Now a days however, software is being used by people from all walks of life. Various kinds of application software are presently available to do most tasks imaginable such as listening to music, writing professional documents, scheduling projects and so on.

In an effort to meet the needs of diverse users, software developers normally include myriads of features which often results in bloated software. End-users usually have little choice but to accept what is offered even though they will probably never use most of the provided features. Some software however, provide features whereby end-users are able to write programs in order to extend, modify or customize certain aspects like appearance or behaviour. This type of programming activities are called end-user programming.

End-user programming is a term that refers to computer programming carried out by end users who do not necessarily have a background in writing programs in conventional programming languages (Goodell, 1998). Initially, the first group of end-user programmers were engineers and scientists who used some programming languages such as FORTRAN to write small programs. Nowadays, end-users programmers include teachers (Wiedenbeck, 2005), children (Petre and Blackwell, 2007), interaction designers (Myers *et al*., 2008) and scientists (Segal, 2007). It is also possible for this list to be expanded to include accountants, parents and all other people who are not trained as programmers. There is much interest amongst end-users to be able to write their own programs. For example, it is estimated that there are 55 million end-user programmers in the US, compared to 2.75 million professional programmers (Paterno *et al*., 2003).

There are various end-user programming techniques from simple but limited metaphors to more complex and powerful techniques for advanced end-user programmers. One of the earliest attempt for end-user programming is spreadsheet macro that enables end-users to write some simple programs to do some specific numerical calculation tasks. Now a days, there are a lot of end-user programming techniques and tools that have been developed.

Problem formulation and solution: Most of the end-user programming tools are developed by different group of people and for different purposes. Each of these tools employs different method and programming style. Therefore, it is not easy for users to move from one application to another.

Furthermore, many of these tools still require a small amount of coding. Although, this type of coding may be very simple and straight forward for professional programmers, it can still be a difficult task for end-users. Small errors in coding can cause a huge amount of errors. For example, it was reported that in 1980s mistakes made in writing macros for Lotus 123 spreadsheet by some Florida contractors to prepare for tenders has caused them to lose a large amount of money (Harrisson, 2004).

In order to solve this problem, the researchers are proposing a general purpose, adaptive and programming-

less end-user programming system. By having this system, all end-users applications can be developed by using a single programming system. In this programming system, all facilities and functions needed by end-users will be provided in the form of programming blocks. End-users will be able to develop an application simply by integrating relevant programming blocks. Thus, the researchers propose that this programming system to be called Block-Based Programming System (BBPS).

RELATED WORK

The idea of BBPS approach is derived from the combination of Component-Based Software Development approach (CBSD) and End-User Programming (EUP) approach.

Component-based software development: Since, the early 1990's, Component-Based software technology has become an increasingly popular approach to facilitate the development of evolving systems as it promised to address some of the problems of object-oriented development technologies (Harrisson, 2004).

A component can be defined as a software module with a published interface which offers a set of related services; it must be delivered in a ready to use form. Missed schedules, exceeded budgets and defective products are the major crises in current software development. CBSD is a hopeful solution to the current software crisis and has gained considerable attention in the software industry.

The objective of this technology is to take elements from a collection of reusable software components and build applications by simply plugging them together. Hence, the main aim of this technology is to produce high-quality software systems with shorter and more cost-effective development cycles. By reconfiguring components, adapting existing components or introducing new components it is hoped that applications could be adapted to changing requirements of real-world software systems more easily and address the problems of object-oriented development approaches (Cheung, 2004).

In order to support CBSD, a number of component models have been proposed such as CORBA (OMG, 1996), COM (Rogerson, 1997), JavaBeans and more recently Enterprise JavaBeans (Monson-Haefel, 2000) and the CORBA Component Model. Furthermore, a number of component-based development environments have been made available such as Delphi, Visual Studio, Visual Age and Eclipse.

End-user development: End-User Development (EUD) means the active participation of end-users in the software development process. According to Costabile *et al.* (2004) it has been argued that EUD can help to develop a software that meets the actual requirement of the user.

The range of active user participation in the software development process can range from providing information about requirements, use cases and tasks including participatory design to end-user programming. Terms such as mutual development, co-development and participatory design refer to activities in which end-users are involved in a system design but may or may not be involved in the actual coding (Costabile *et al.* 2009).

Extreme Programming (XP) is an example of a software development methodology that supports active end-users participation. In XP, requirements are expressed as scenarios or user stories. Once the scenarios have been developed, they are implemented as a series of tasks. Programmers work closely with users to develop and test the task (Beck, 1999).

End-user programming implies that some programming tasks that are traditionally performed by professional programmers are transferred to the users. The phrase end-user programming was first popularized by Nardi (1993) in her investigations into spreadsheet use in office workplaces.

End-user software engineering (Ko *et al.*, 2008) is a new concept that involves systematic and disciplined software engineering activities that address software quality issues but these activities are secondary to the goal that the program is helping to achieve. Current research in end-user software engineering is to develop a software engineering methodology that incorporate software engineering activities into users' existing workflow without requiring people to substantially change the nature of their work or their priorities. For example, rather than expecting end-users to do thorough testing, tools can be used to provide feedback about software quality as the user edits the program. Approaches like these allow users to stay focused on their primary goals while still achieving software quality.

End user programming systems: End-user programming is made possible by the availability of new programming systems. Generally, these systems fall into the following categories: application-specific languages, programming by example, visual programming and natural programming. Application specific language is a type of language that can be used by end-users in a very specific domain. An example of this type of system was developed by Prahofer *et al.* (2006) to be used in automotion domain. In

this system, a new domain-specific language called Monaco (Modelling Notation for Automation Control) serves as a basis for building control flow programs. The language is similar to Statecharts but adopts an imperative notation that is much closer to the perception of the domain experts. A compiler-generator produces the Monaco compiler to read Monaco source code programs and generate parse tree object models, Monaco CodeDOM. A Monaco CodeDOM is then executed by the Monaco virtual machine (Monaco VM). End-users of the automation software system is the individual machine operators whose basic task is handling and supervising the machine operations, usually with the help of an electronic control panel. Providing all the machine parameter settings and defining an operation sequence for a very specific manufacturing task usually requires deep knowledge of the domain and due to its complexity, still represents a great programming challenge. In this system, end-users do the programming task by interacting with an Eclipse-based integrated development environment (Monaco IDE).

Other application specific end-user programming systems include LabVIEW and Mathematica. LabVIEW is an end-users graphical programming environment used by engineers and scientists to develop sophisticated measurement, test and control systems. It provides integration with hardware devices and provides hundreds of built-in libraries for advanced analysis and data visualization. The programming environment provided by LabVIEW is known as graphical dataflow programming environment. End-user programmers need to define execution flow in code by creating diagrams that show how data moves between functions (known as Virtual Instruments or VIs). Mathematica is an end-user computational software that enables scientists and engineers to write programs to solve problems related to mathematical computing. The programming language provided by Mathematica is simple and easy to be used by end-users since, Mathematica is supported by some mathematical function libraries and tools such as matrix and data manipulation tools, symbolic computation tool, 2D and 3D data and function visualization and animation tools, solvers for systems of equations, numeric and symbolic tools for discrete and continuous calculus and multivariate statistical libraries.

Programming by Example (PBE) or Programming by Demonstration (PBD) is a programming system that allow a user to demonstrate the desired program by going through the steps. The simplest form of PBD is macros such as those found in Microsoft Office. Macros allow the user to record a sequence of fixed actions and replay these actions by using a single mouse click. A more powerful PBD systems may contain conditional constructs, variables and iteration that can combine the ease of use of macros with the expressiveness of programming (Lau and Weld, 1999). Some of the programming systems use AI techniques to try to automatically generalize the program from the user's examples. Examples of this type of systems are SMARTedit (Lau *et al*., 2003), Sheepdog (Lau *et al*., 2004) and GoScripter (Little *et al*., 2007). SMARTedit is a text editor that uses PBD to automate repetitive text-editing tasks. For example, when reformatting text copied and pasted from the web into a document, one can demonstrate how to reformat the first line or two of text and the system learns how to reformat the remaining lines. Sheepdog is a PBD system for learning to automate Windows-based system administration tasks. For example, based on several demonstrations of experts fixing the configuration of a windows laptop in different network environments, the system produces a procedure that could apply the correct settings. CoScripter is a PBD system for capturing and sharing scripts to automate common web tasks.

Visual programming focus on using graphics to make the process of programming easier. One of the popular tools for visual programming is the Microsoft Visual Basic. Research are currently carried out to develop simpler visual programming tools for end-users. One example of the system is Kodu (Maclaurin, 2011) that is a designed specifically for young children to learn through independent exploration. The simplified programming model provided by Kodu lowers the barrier of entry for new programmers. To make it more interesting, Kodu is integrated in a real-time 3D gaming environment with intuitive user interface and graphical production values.

Natural programming is a type of programming systems that faithfully representing nature or life which implies it works in the way people would expect. Current research in natural programming aims towards developing general principles, methods and programming languages and environment designs that will significantly reduce the amount of learning and effort needed to write programs for people who are not professional programmers. An example of a natural programming system is called HANDS (Myers *et al*., 2004) that uses an event-based language that features a new model of computation, provides queries and aggregate operators that match the way non-programmers express problem solutions. HAND also has high visibility of program data and includes domain-specific features for the creation of interactive animations and simulations. A study that was carried out shows that 10 years old children were able to learn the HANDS system during a 3 h session and then, used it to

solve programming problems. The study also found that children using the full-featured version of HANDS performed significantly better than their peers who used a version modified to be more like typical programming systems.

End-users programming behaviours: A number of empirical studies of end-user programming behaviour have been reported. Several studies have made uses of the EUSES spreadsheet corpus (Fisher and Rothermel, 2005) to examine the behaviors of end-user spreadsheet developers; this corpus contains 4, 498 artifacts. Bogart *et al.* (2008) studied the CoScripter web scripting environment. The most interesting study so far was carried out by Dinmore and Boylls (2010) about the end-user programming behaviour when using Yahoo! Pipes. Yahoo! Pipes is a web-based, visual programming environment introduced by Yahoo! in the year 2007 to enable users to rewire the web. Pipes is a dataflow system in which the data is sourced from the web (RSS feeds, web pages, raw data) and flows through an interconnected set of modules that act upon it, ultimately producing some result. The result of the study shows that end-users employ only a small number of the programming options that Pipes offers to them and typically compose them in pipes consisting of only three or four modules. Further, while they are able to create more complex, multi-branch pipelines, they strongly favor simple, straight-through pipes.

Another study was carried out by Blackwell and Morrison (2010) to compare the behaviour of end-user programmers and professional programmers. This study involved monitoring the programming behaviour of an end-user who was a healthcare professional in customizing an electronic patient record system for a hospital. In order to explore and contrast the different ways that the same situation was conceived by an end-user programmer and by a professional programmer, a professional programmer was also employed as a quasi-experimental participant.

The main theme of the findings is that end-users have users too. The end-user programmer receives feature requests from the other hospital staff with proposals or recommendations for further customisation. Thus end-user programmer's job is not confined to programming alone but also in business process analysis, user interface design and many other aspects of software engineering. Another finding indicates that software tools that are normally used by professional programmers may not be suitable for end-user programmers. The third finding is that end-user programmers are not intrested in the technical programming details but more concern with ensuring that the system works. Thus it is important that any end-user programming systems to make a clear distinction between the system model that must be understood by end-users and the underlying technical behaviour of the system. In the psychology of programming community, this has been described as the black box inside the glass box.

BLOCK-BASED PROGRAMMING

Block-based programming system has been proposed as a general purpose, adaptive and programming-less end-user programming system. The idea of this programming system is derived from the following objectives:

- The programming system must be general purpose which implies that it is not limited to a specific programming domain
- It must be programming-less which implies that end-user programmers are not required to write programming codes in order to develop applications
- It must support the idea of end-user software engineering which implies that it must address the software quality issue
- Similar to most of other end-user programming system, application development must be based on components or modules
- The components or modules must be simple to be used. It should also must be adaptive and customizable
- It must make a clear distinction between tasks of end-user programmers and tasks of professional programmers

In order to achieve these objective, the block-based programming system can be described as follows:

- It should support software development in many problem domain
- Many blocks will be made available for each problem domain
- Each block supports a certain task or function
- End-users are allowed to customize blocks and to build applications adapting to their needs
- Application software development can be done by integrating these blocks

In this study, the concept of block-based programming system have been shown in Fig. 1.

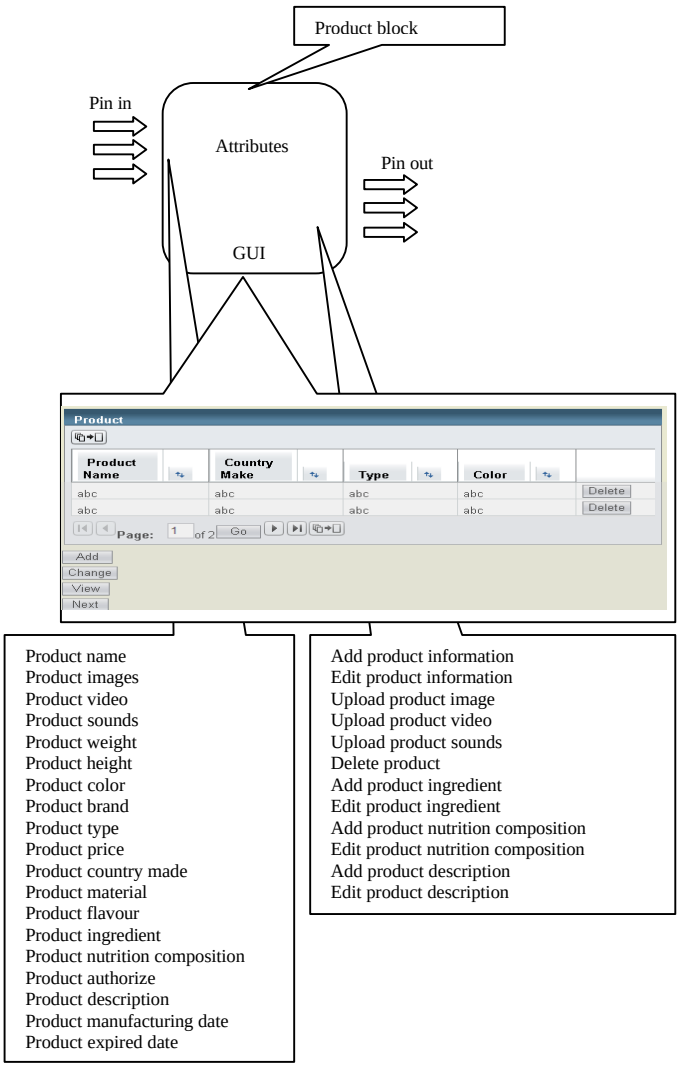

Fig. 1: Block description

Blocks: The term programming block (or simply, a block) refers to a software component which is easy to reuse, highly composable, customizable and configurable. Block contains four elements: attributes, GUI elements, behaviors or methods and interfacing (Pin in/Pin out).

Each block is designed and created to support a specific task. Blocks can be divided into two categories: domain specific blocks and general-purpose blocks. Domain specific blocks refer to blocks that are developed for a specific application domain.

General purpose blocks can be used for any type of applications. Figure 1 is an example of a block called Product block which is a block that being use to develop an e-commerce application. Users are allowed to customize this block by changing fonts, colors, attributes and behaviors and even can make minor change on GUI.

Software developers: Software developers are persons responsible to design the software as well as writing the programs. In BBSD, there are two types of software developers. They are known as:

- The block developers are professional programmers who are responsible for blocks development. The block developers need to have a very strong technical expertise in order to be able to produce a very high quality and reliable blocks
- The application developers are the end-users (such as parents, teachers, managers administrators, engineers and scientists) who are responsible for selecting suitable blocks, customizing them and integrate them to develop an application. The application developer need to focus more on the application domain as well as the system model of the proposed application

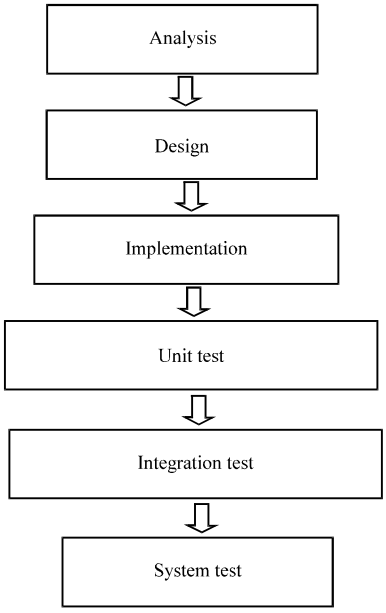

Fig. 2: Conventional process model

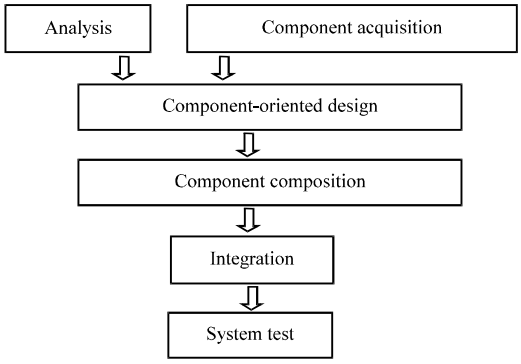

Fig. 3: Component-based software process model

Process model: Comparison between conventional process model, component-based software process model and BBP process model is shown in Fig. 2.

In the conventional approach (Fig. 2), a programmer or a programming team will be given the task to do the requirement analysis. Based on the analysis, researcher will continue to do the design, coding and testing.

In component-based software process model (Fig. 3), a programmer or a programming team needs to do the requirement analysis. Based on this analysis, software components are acquired. If any of the components are not available, they must be developed in-house. The programmer then proceed to combine these components into a complete application, followed by a system testing. In block-based programming system (Fig. 4), there will be two groups of people working independently.

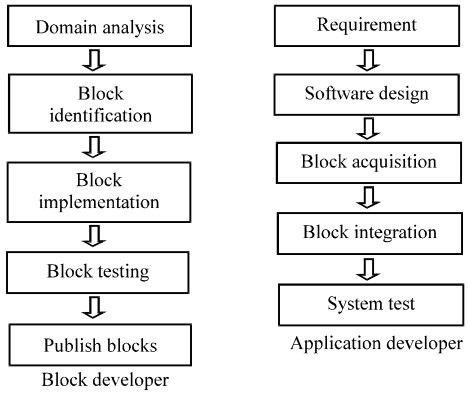

Fig. 4: Block based programming system process model

Block developers develop the blocks for a specific domain and publish these blocks. Application developers are end-users who will acquire blocks and will integrate these blocks to form an application.

Block identification methodology: This methodology helps block developers to identify blocks that are needed for a specific domain. The researchers propose a Block Modeling Language (BML) which is basically based on UML. Similar to UML, use cases will be used for domain analysis. As an example, Fig. 5 shows the use case diagram for the domain of online shopping.

From the analysis, several different types of blocks for online-shop had been identified as shown in Fig. 6. These blocks are product block, shoppingcart block, payment block, shipping block, advertisement block, blog block, email block, manage product block, database block, serverconnection block and websetup block.

Blocks implementation: The implementation refers to the process of designing, coding and testing a block. Ideally blocks could be implemented by using any programming languages.

However, in the real world, some programming languages are better suited for implementing blocks than the other. The researchers found that Java is currently the best programming language for implementing blocks since, it is based on virtual machine that is supported by most of the computing platforms.

Blocks customization: An application developer can customize blocks according to the requirement of the application. This customization can be done by changing the value of the attributes of the blocks. Some blocks may allow application developers to customize some of the behaviour of the blocks.

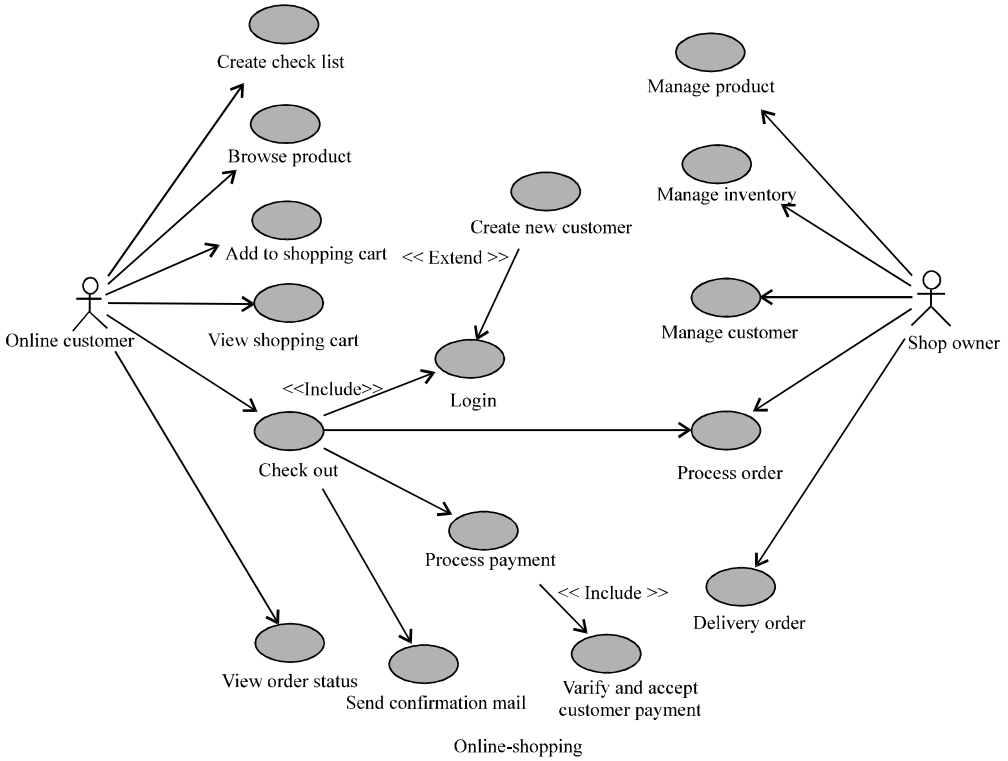

Fig. 5: Domain analysis for online shopping

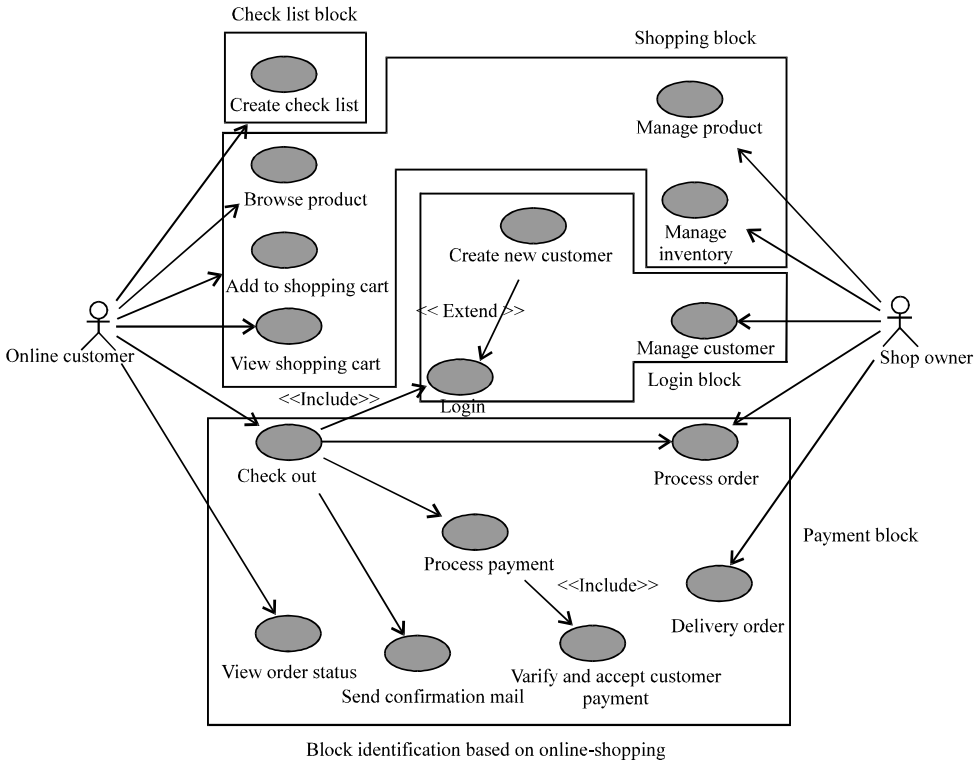

Fig. 6: Blocks identification

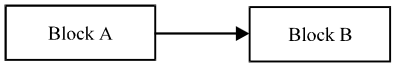

Fig. 7: Block integration environment (Sequential ordering)

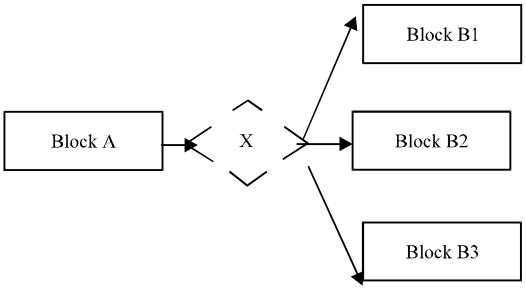

Fig. 8: Block integration environment (Deterministic selection)

Blocks integration: Block integration is the process of integrating blocks into applications. Blocks can be integrated by using selectors or combinators. The researchers proposed that there should be four types of selectors for integrating blocks.

Sequential ordering: Sequential ordering implies that after block-A has been executed, block-B will be executed.

Deterministic selection: After block-A is executed, block-Bn will be executed based on the condition X.

Random selection: After block-A is executed, one of block-Bn will be selected at random. This type of selection is important in order to ensure that a certain degree of non-determinism can be provided.

Repeat: Block A will be repeated if condition X.

Tool support: There are a number of software tools needed to support block-based programming environment. The researchers have identified two important tools:

Blocks creation tools: Blocks can be developed by using any of the currently available programming tool. One possible tool Netbeans IDE 5.0 written completely by using Java Beans concept and methodology. However, there is a need for special purpose block creation tool to be provided to help block developers to create blocks properly. This tool will also be provided with a suitable testing mechanism to ensure that the created blocks can work together with other blocks.

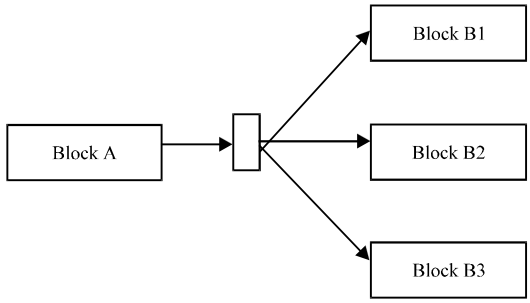

Fig. 9: Block integration environment (Random selection)

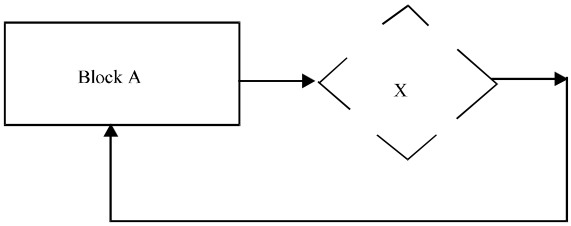

Fig. 10: Block integration environment (Repeat)

Blocks integration tools: There are a number of ways that blocks can be integrated. One approach is to use a similar approach to visual programming where blocks can be integrated by using drag-and-drop mechanism as shown in Fig. 7-10. Another approach for blocks integration is the story telling approach. While visual IDE may be considered easier to use by most programmers, story-telling approach may be more attractive to non-programmers.

CONCLUSION

This study describes a general purpose, adaptive and programming-less end-user programming system. This system is called block-based programming system since, it is based on the concept of programming blocks. Through this system, end-user programmers will be able to compose, adapt or customize applications to meet their specific requirements. It is hoped that the availability of this programming system will enable end-user programmers to develop high quality software applications.

A number of the research need to be carried out in order to support block-based programming system. Some of the researches are in progress and will be reported later. The researchers can divide the research into three main areas. The first area is to develop methodologies that can support the use and implementation of block-based programming. One of the important methodologies is the block identification methodology that will ensure that all of the blocks that are needed for a particular domain can

be identified and developed. The second area is to develop blocks for some application domains. Two particular domains are the development of blocks for developing e-commerce applications and blocks for developing courseware. The third one is to develop some software tools to support this programming approach. Both tools that have been identified are currently being developed.

REFERENCES

Beck, K., 1999. Extreme Programming Explained: Embrace Change. Addison-Wesley, Boston.

Blackwell, A.F. and C. Morrison, 2010. A logical mind, not a programming mind: Psychology of a professional end-user. Proceedings of the 22nd Annual Workshop of Psychology of Programming Interset Group, Sept. 19-21, University Carlos III of Madrid.

Bogart, C., M. Burnett, A. Cypher and C. Scaffidi, 2008. End-user programming in the wild: A field study of CoScripter scripts. Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing, Sept. 15-19, Herrsching am Ammersee, Germany, pp: 39-46.

Cheung, H.W., 2004. The impact of component-based technology on the role of user in traditional software development. 21st Comput. Sci. Seminar.

Costabile, M.F., D. Fogli, P. Mussio and A. Piccinno, 2004. Software environments for end-user development and tailoring. Psychnol. J., 2: 99-122.

Costabile, M.F., P. Mussio, L.P. Provenza and A. Piccinno, 2009. Supporting end users to be co-designers of their tools. End User Dev., 5735: 70-85.

Dinmore, M.D. and C.C. Boylls, 2010. Empirically-observed end-user programming behaviors in Yahoo! pipes. Proceedings of the 22nd Annual Workshop of Psychology of Programming, Sept. 19-21, Madrid.

Fisher, M. and G. Rothermel, 2005. The EUSES spreadsheet corpus: A shared resource for supporting experimentation with spreadsheet dependability mechanisms. Proceedings of the 1st Workshop on End-User Software Engineering, May 21st, Saint Louis, Missouri, USA.

Goodell, H., 1998. End user programming. Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, September 1998, Dallas, Texas, pp: 215-222.

Harrisson, W., 2004. The dangers of end-user programming. IEEE Software, 21: 5-7.

Ko, A.J., R. Abraham, L. Beckwith, A. Blackwell and M. Burnett *et al.*, 2008. The state of the art in end-user software engineering. ACM Comput. Surveys, 43: 1-44.

Lau, T., L. Bergman, V. Castelli and D. Oblinger, 2004. Sheepdog: Learning procedures for technical support. Proceedings of the IUI, Jan. 13, ACM Press.

Lau, T., S.A. Wolfman, P. Domingos and D.S. Weld, 2003. Programming by demonstration using version space algebra. Mach. Learn., 53: 111-156.

Lau, T.A. and D.S. Weld, 1999. Programming by demonstration: An inductive learning formulation. Proceedings of the 4th International Conference on Intelligent User Interface, Oct. 8, Redondo Beach, CA.

Little, G., T.A. Lau, A. Cypher, J. Lin, E.M. Haber and E. Kandogan, 2007. Koala: Capture, share, automate, personalize business processes on the web. Proceedings of the CHI 2007 Collaboration at Work, April 28-May 3, San Jose, CA, USA., pp: 943-946.

Maclaurin, M., 2011. The design of kodu: A tiny visual programming language for children on the Xbox 360. Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, Jan. 26-28, San, Diego, CA, USA.

Monson-Haefel, R., 2000. Enterprise JavaBeans. 2nd Edn., O'Reilly and Associates, California, Pages: 472.

Myers, B., S.Y. Park, Y. Nakano, G. Mueller and A. Ko, 2008. How designers design and program interactive behaviors. Proceedings of the IEEE Symposium On Visual Languages and Human-Centric Computing, Sept. 15-18, Herrsching Am Ammersee, Germany, pp: 177-184.

Myers, B.A., J.F. Pane and A. Ko, 2004. Natural programming languages and environments. Commun. ACM, Special Issue End-User Dev., 47: 47-52.

Nardi, B.A., 1993. A Small Matter of Programming: Perspectives on End User Computing. MA:The Mit Press, Cambridge, USA.

OMG, 1996. The Common Object Request Broker: Architecture and Specification. Object Management Group, Salt Lake City, UT, USA., Pages: 177.

Paterno, F., M. Klann and V. Wulf, 2003. Research agenda and roadmap for EUD. IST-2001-37470, EUD-Net Network of Excellence.

Petre, M. and A.F. Blackwell, 2007. Children as unwitting end-user programmers. Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, Sept. 23-27, IEEE Computer Society, pp: 239-242.

Prahofer, H., D. Hurnaus and H. Mossenbock, 2006. Building end-user programming systems based on a domain-specific language. Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling, Oct. 22, Portland, Oregon, USA.

Rogerson, D., 1997. Inside COM: Microsoft's Component Object Model. Microsoft Press, Bellevue, Washington, Pages: 376.

Segal, J., 2007. Some problems of professional end user developers. Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, Sept. 23-27, Idaho, USA., pp: 111-118.

Wiedenbeck, S., 2005. Facilitators and inhibitors of end-user development by teachers in a school environment. Proceedings of the Visual Languages and Human-Centric Computing, Sept. 20-24, IEEE Computer Society Washington, DC, USA., pp: 215-222.