International Journal of Soft Computing 2 (1): 199-207, 2007

Un-match: Object-Oriented Software Design Method and Visual Programming

[1]Bill Yu-Fan Cheng, [2]Jonathan Jiin-Tian Chyou and [3]Jung-Jung Liu
[1]Department of Information Management, Hsiuping Institute of Technology, Taiwan
[2]Department of Management Information Systems, National Chengchi University, Taiwan
[3]Department of Computer Science and Information Technology,
National Taichung Institute of Technology

Abstract: Software development methods (SDMs) are valuable but not a panacea. If selected and used correctly, they can provide great benefits; if regarded as magic solutions or otherwise misused, they can prove to be an expensive failure. Recently in software engineering practice, a systematic mismatch phenomenon between Object-oriented Software Design Method (OOSDM) and Visual Programming (VP) has been observed, but this mismatch phenomenon has been strangely neglected by academia. Therefore, this paper focuses on the suitability of OOSDM, while its purpose is to argue the un-match between OOSDM and VP from the viewpoint that "a blueprint must be followable." Moreover, this paper also presents a series of in-depth interviews conducted to validate the correctness of the proposed arguments.

Key words: Contingent software development, mismatch phenomenon, suitability of software development method, object-oriented software design method, visual programming

INTRODUCTION

Software Development Methods (SDMs) consist, at least in principle, of both a development process and a set of software models. The development process is SDMs' suggestion on what steps to take in undertaking analysis, design, implementation and testing. The software models are the tools that developers use to express requirements, designs and test cases. It has been established that the proper use of a SDM appropriate to a problem can significantly increase productivity and quality[1].

SDMs are valuable but not a panacea. If selected and used correctly, they can provide great benefits; if regarded as magic solutions or otherwise misused, they can prove to be an expensive failure[2-4]. Selection of SDMs requires study of SDMs' suitability with the software development project's features. Unfortunately, most SDMs today lack clear descriptions of suitable situations and as developers must often choose which one to use with little or no information it is not surprising that many misuses of SDMs have occurred[3,5,6].

Related works: Given its importance in software engineering, the problem of misuse of SDMs has been brought to the attention of the software engineering community. Over the past few decades a great deal of research effort has been expended on making the right selection of SDMs and can be classified into four areas:

The first is research on SDMs' suitability, such as[7-12]. This type of research clarifies specific SDMs' suitable situations. Take Land's research[10] for example. Land examines the relationship between the development process and organizational environment. He found that both the waterfall process and spiral process fit in stable organizations, but the prototyping process fits in changeable organizations.

The second area is research on SDMs' selection factors, such as that conducted by Alexander and Davis[13]; DeLone and McLean[14]; Doke and Swanson[15]; Hardgrave[16]; Havelka and Lee[17]; Siau[18]. This type of research identifies critical factors that determine either the selection of SDMs or the adoption of specific SDMs. Hardgrave's research[16], for example, identifies eighteen factors that determine whether to adopt a prototyping process.

The third is research on selection supporting tools, such as[19-22]. This type of research develops tools to assist in determining either the selection of SDMs or the adoption of specific SDMs. Take the research of Hardgrave *et al.*[21] for example. Based on eighteen factors found by Hardgrave[16], the research develops a decision supporting tool to aid determine whether to adopt a prototyping process.

Corresponding Author: Bill Yu-Fan Cheng, Department of Information Management, Hsiuping Institute of Technology, No. 30, Lane 674, Sec. 1, Changnan Rd., Nantou City 54056, Taiwan

Finally, is the research of meta-methods, such as[23,26,6]. This type of research assumes that a customized SDM should be constructed to meet a particular software development project's needs. Accordingly, the authors propose systematic methods to guide the development of SDMs. Tolvanen's research[6], for example, proposes an experience- based incremental meta-method.

Research question: Within a very short period of time, both Object-Oriented (OO) Software Design Method (OOSDM) and Visual Programming (VP) have become more widely used in practice. Additionally, in recent years a systematic mismatch phenomenon between OOSDM and VP has been noted in software engineering practice, but this mismatch phenomenon has been strangely neglected by academia. As a result, this paper focuses on the suitability of OOSDM, while its purpose is to argue the un-match between OOSDM and VP from the viewpoint that "a blueprint must be followable."

Paper organization: The paper is organized into six sections. Section two and three explain separately the essential features of OOSDM and VP which serves as evidence of the argument. Based on this evidence, section four argues the un-match between OOSDM and VP. The fifth section presents a series of in-depth interviews conducted to validate the correctness of the proposed arguments. The final section concludes this study and proposes some research directions.

MATERIALS AND METHODS

Software design methods are special SDMs that focus on design activities and OOSDMs are one type of widely used software design methods that employs the object viewpoint. From the object viewpoint, the software is viewed as a collection of cooperative objects. These objects must be responsible for themselves and must collaborate with one another so as to realize each and every software function. Further, these objects can be classified by their similarities and differences, whilst all objects that are classified into a particular class, have the same attributes and methods, are said to be instances of that class. Based on the object viewpoint, the processes of OOSDMs can be defined briefly as procedures to identify and design objects in the software, built to meet the requirement specifications.

Although all OOSDMs fit this brief process definition and were very similar, they contained a number of differences among them. For this reason, this paper limits the following discussion to the most popular function-driven OOSDMs that are now called use-case-driven OOSDMs, such as OOSE, OOTC, OPF and RUP.

Design process: For each software function, the developers should develop some usage scenarios. This design activity can help developers to understand.

- What objects are parts of a software function.
- Their attributes and methods.
- Static relationships among them.
- How they realize the software function through their interaction.

Once the developers have developed all of the usage scenarios, that is to say all objects of software have been identified and designed, the developers should unify and classify these objects so as to form an ideal blueprint. Finally, the developers should adapt the ideal blueprint to the non-assumed implementation environments, methods, techniques and tools such as relational database management systems (RDBMS), non-OO programming languages and existing class libraries.

Design models: Good, clear software models play an essential part in developing software. Such software models have several important functions that include.

- Acting as an aid to clear thinking
- Acting as an aid to externalize design ideas.
- Allowing precise communication between members of the development team.
- Encouraging end users to sketch their needs clearly.
- Acting as an aid in testing
- Acting an aid in maintenance.[27,28] For these reasons, most SDMs are model-centric methods and OOSDMs are not exception. The following are the function-driven OOSDMs most commonly used as design-focused software models (design models)[27,28].
- Class diagram, used to illustrate the classes in the software, attributes of the classes, methods on the classes and the various kinds of static relationships that exist among these classes.
- Object interaction diagram, used to show a number of objects and the messages that are passed between these objects within a particular software function.
- State diagram, used to display all the possible states that a particular class can get into and how the class' state changes as a result of events that reach the class.
- Activity diagram, used to explain the process logic of a particular class method.
- Pseudo code, a substitute for activity diagram, is also used to describe the process logic of a particular class method.
- Decision table, used to clarify the decision logic of a particular class method.

Visual programming: The very first programmers had to work at the lowest possible level, writing programs as sequences of bits. Since then the development of programming techniques has aimed to make the programming task easier so that people with less training should be able to produce programs which work correctly, as quickly as possible. VP is one current aspect of that development[29].

VP is a programming method that allows developers to graphically construct software. Compared with traditional textual programming methods, VP provides a more efficient and easier way of producing software. Several VP paradigms already exist and the features provided by different VP paradigms may vary greatly. The Visual Basic-like VP paradigm is supported by many popular software development tools, such as Microsoft Visual Studio, Borland Delphi, Borland JBuilder and Sybase Power Builder. In fact, the Visual Basic-like VP paradigm is widely used in data-centric business software that includes point of sales systems and accounting information systems. In this study, the Visual Basic-like VP (VP) paradigm will be the focus.

VP reveals six distinguishing characteristics. Firstly, VP supports primarily the development of client/server, data-centric business software. The database runs on top of a shrink-wrapped RDBMS package. In contrast, the client programs (programs) are either custom-built window programs or ASP.NET-style web programs, both developed using VP.

Secondly, VP-developed programs consist of form modules, each form comprising COTS (Commercial-Of-The-Shelf) components. Moreover, there are five very common types of forms: splash form, main form, about form, function-specific primary form and function-specific secondary form. Typically, a VP-developed program is almost completely made up of these common forms.

Thirdly, the heart of VP consists of component libraries and code generators. Through the reuse of COTS components, VP enables developers to dramatically reduce the amount of time and code required to write a program. In addition, VP provides WYSIWYG code generators such as form editor and report editor, where the developers merely fill in forms, drag and drop icons, or click buttons to automatically generate most codes (structured codes) and thus developers only need to hand-write some codes (unstructured codes).

Fourthly, a component library designed especially for VP is a special OO class library that follows a certain component specification, such as OMG CCM component specification, Microsoft. NET component specification and Borland VCL component specification. Any component library must be installed on a VP tool and the installed components will appear on the component palette systematically. Of course, developers are not limited to using the components that ship with a VP tool. In fact, developers always add certain customized or third-party components to the VP tool for some reasons. Currently, Microsoft. Net Framework Class Library and Borland VCL Library are the most popular component libraries. Both libraries provide components and functionality that allow developers to build forms, to make reports and to access databases.

Fifthly, VP supports unstructured codes through the event-driven writing method. In event-driven writing the developers identify the events (such as a user action or a change in focus) that the program must handle and write event-handlers to respond to the events.

Finally, in VP the developers always write some non-event-handling codes. Typical reasons given include reducing code redundancy, simplifying complex event-handlers, decreasing memory requirements, simplifying data passing between forms and facilitating code reuse in the future.

It should be concluded, based on the characteristics outlined above that: (a) VP is a form- oriented, COTS components-based and event-driven programming method; (b) VP is an iterative and incremental process that is organized around five primary activities.

- Creating a new form.
- Adding a component to a form.
- Setting a component property.
- Writing an event-handler.
- Writing some non-event-handling codes.

Arguing the un-match between OOSDM and VP: Argument Viewpoint: The design activities specify the best solution to a problem and produce a blueprint to describe how it is to be organized. Implementation follows design. The implementation activities realize a system in accordance with the blueprint[30]. If the blueprint does not exist or if it is of poor quality, it should be no surprise when a less-than-acceptable system is produced. Such a system often is poorly organized and becomes a nightmare to support later. On the other hand, a good blueprint can improve system readability and reliability, reduce system complexity and cost and increase job satisfaction for developers[28].

Obviously, the blueprint is the most important input of the implementation activities and the blueprint must be followable by all. In other words, all design information required by the implementation activities must be specified in the blueprint and all design information specified in the blueprint must be used by at least one implementation activity. From this viewpoint, we can examine the followability of an OO software blueprint (OO blueprint) with VP and then determine that OOSDM either matches or un-matches with VP.

Why OOSDM does not match VP: The OOSDM is rooted in both building from scratch and one-line-at-a-time coding. Contrary to the OOSDM, VP is rooted in visually assembling from existing components. Taken in this light, OOSDM does not match VP. In the following, we discuss the unmatched reasons in more detail.

Reason one: A class is not always a component. In VP, a component is a special class but a class is not always a component. There are three differences between a component and common class. First, component developers need to follow additional restrictions. Take the VCL components for example[31].

- A component must derive directly or indirectly from the TComponent class.
- A component must function in any situation, without preconditions.
- A component as well as its component editors and property editors all have to be registered and installed
- A component has to support the visual operations of software developers at the design-time.

Secondly, in addition to the name, attributes, methods, states and relationships such as are standard parts of a class that are specified in the OO blueprint, a component involves additional parts that are used to support the visual operations of software developers at the design-time. Such additional parts of components include

- One identity icon.
- Several properties.
- Several events.
- Several property editors.
- Several component editors.
- One on-line help.

Finally, a component needs to be carefully generalized to enable use in a variety of contexts. In order to maximize the reusability, a component is completely different from common class. The design of a component must be based on the requirements of the specific application domain (a work product of domain analysis) rather than the requirements of specific software (a work product of software analysis). For the differences mentioned above there is no doubt that the design information of classes that are specified in the OO blueprint, are valueless in the development of VP's components.

Reason two: To buy a component rather than to build a component. Today, a new type of division of labor in the software industry has emerged-a new division between the development of components and software. The component providers develop and market a set of components in a particular application domain. The software developers, on the other hand, assemble software with COTS components. VP is a COTS components-based, form-oriented and event-driven programming method, as we stated earlier. In fact, VP is no other than the most widely used software assembly method. In VP, generally speaking, the developers are willing to buy COTS components from the component market (ship with the VP tool or obtained from third-party vendors) rather than custom build components from scratch. Viewed in this light.

- The design information of components (classes) that are specified in the OO blueprint are unnecessary.
- The custom components- (classes-) based design information of software that are specified in the OO blueprint are valueless to the development of COTS components-based software (the programs of client/server business software).

Reason three: A relational database is not an OO database. The relational database is a table-oriented database and is viewed as a collection of normalized tables. The RDBMS enables users to define, manipulate and maintain the relational database, providing controlled access to this relational database. Although Object-Oriented Database Management Systems (OODBMSs) are now available, time has proved that relational databases still excel for many application domains and data-centric business software is not an exception. A relational database is not an OO database, thus does not support the following six kinds of OO concepts.

- The object identifier.
- The complex attribute.
- The class method.
- The many-to-many relationship.
- The aggregation relationship.
- The inheritance relationship.

Consequently, the design information of persistent classes that are specified in the OO blueprint, are only partially valuable to the development of a relational database. Further, they cannot be directly implemented in the relational database, but rather have to adapt to the needs and constraints of the relational database. For these reasons, we have no reason to develop an OO design for a relational database.

Reason four: None that is needed is there. In the three reasons stated above, we argue the un-match between OOSDM and VP based on the content of the OO blueprint. Now, we turn to argue the un-match between OOSDM and VP based on the design information requirements of VP. According to the characteristics of VP stated earlier, we can recognize ten kinds of design information which are valuable to VP:

- The form-oriented functional decomposition: The information about how a software function is partitioned into form modules.
- The form flows: The information about the focus-shifts that exist among forms.
- The form structure: The information about the components that assemble the form and the instance relationships that exist among these components.
- The form states: The information about the states that a form may get into and the events and actions that can cause the form change to a new state.
- The form layout: The information about the actual appearance of form that the end-users will see during the operation of the software.
- The report structure: The information about the components that assemble the report and the instance relationships that exist among these components.
- The report layout: The information about the actual appearance of the report that the end-users will see during the operation of the software.
- The property setting: The information about the component properties that need to be set and the values that should be set in these component properties.
- The specification of the event-handler: The information about the design of the event-handler, such as the process logic and the decision logic.
- The specification of non-event-handling codes: The information about the design of non-event-handling codes, such as the process logic and the decision logic.

It is obvious that none of such design information appeared in the OO blueprint. That is to say, the OO blueprint is also valueless to VP, from the perspective of the design information requirements of VP.

As we said earlier, the VP-developed software consists of a "relational database" and several "programs". The relational database runs on top of a RDBMS. The programs, in contrast, are visually assembled from "components". Furthermore, the reasons mentioned above also clearly indicated.

- The OO blueprint is only partially valuable to the development of a relational database.
- The OO blueprint is valueless to the development of components and programs.
- The OO blueprint lacks all of the design information that is valuable to VP.

We therefore conclude that the OO blueprint is unfollowable (all that is there is unnecessary) and incomplete (none that is needed is there) for the VP, namely, OOSDM does not match VP.

Why mapping strategy is unworkable: Jacobson et al. stated, "The implementation is straightforward in the chosen programming language. An object-oriented language is preferable since all important concepts used in OOSE are directly mapped onto these languages. If the language is not object-oriented, some deviations must be made. However, an object-oriented structure is entirely possible even for systems implemented in non-oo languages"[31]. (p. 257)

Rumbaugh et al. stated, "Object-oriented concepts provide an excellent basis for modeling hierarchical, network, relational and object-oriented DBMS. Object models permit developers to think about a problem at a high, abstract level and yet rest assured that the resulting design can be easily and practically implemented. The following simple rules enable designers to convert an object model to relational DBMS tables"[32]. (p. 388-389).

These two quotations come from OO literature, clearly showing us that an underlying belief behind the OOSDM is that the OO blueprint is easy to implement in every implementation method. Even if the implementation method is not OO, the OO blueprint still can be indirectly implementable by the use of mapping strategy. The said mapping consists of applying a systematic procedure to adapt (restructure, augment, reduce or otherwise) the OO blueprint to the needs and constraints of the target implementation method.

In fact, the decision of whether to develop an OO blueprint for the software that must be implemented using a non-OO implementation method requires study of the possibility of mapping with the implementation method's features. Here and now, several non-OO implementation methods have obtained their mapping procedures, such as RDBMS and non-OO programming language. However, unfortunately today we lack the systematic procedure to adapt the OO blueprint to VP.

In addition, even if the mapping strategy is workable in VP, it will also be the same as other non-OO implementation methods in that a great quantity of unnecessary design effort will prohibit us from using the OOSDM.

Why method framework strategy is unworkable: Jacobson *et al.* stated, "First, it [RUP] is a framework. It has to be tailored to a number of variables: the size of the system in work, the domain in which that system is to function, the complexity of the system and the experience, skill, or process level of the project organization and its people"[33]. (p. 416)

Kruchten (2000) stated, "You will have to configure and implement it. To configure the Rational Unified Process means to adapt the process product to the needs and constraints of the adopting organization. To implement the Rational Unified Process in a software development organization means to change the organization's practice so that it routinely and successfully uses the Rational Unified Process in whole or in part"[34]. (p. 249)

These two quotations come from OO literature and point out that:

- In order to improve suitability, new generation OOSDMs, such as OPF and RUP, leave method and turn to method framework.
- Method framework is a parent method that can be tailored to derive a project-specific method.
- Method framework is very difficult to understand and use. In other words, method framework is of high suitability but low usability.
- New generation OOSDMs recognize that no single method is appropriate for all situations, however method framework implies itself as a universal solution and can be adapted to every and any development project.

For the sake of heightening the level of suitability, a method framework must do its utmost to generalize its process and maximize its software models. However, such suitability improvement activities will lower the level of usability. That is to say, suitability conflicts with usability. It is impossible to have no limits in heightening the level of suitability of the method framework, thus it must try to maintain the equilibrium of suitability and usability.

Current OO method frameworks, such as OPF and RUP, are not generalized to take leave of the OO paradigm. A project-specific method that is derived from an OO method framework will also be the same as a traditional OOSDM, providing both an OO design process that is used to identify, design and use objects, as well as a set of OO design models that are use to express design. For this reason, like all traditional OOSDMs, OO method frameworks do not match VP.

Choosing between OOSDM and VP: Engineering development tasks are of several kinds. One of the most significant distinctions separates routine from innovative development. Routine development involves solving familiar problems, reusing large portions of prior solutions. Innovative development, on the other hand, involves finding novel solutions to unfamiliar problems[35]. Compared with innovative development, routine development provides a more efficient and easy way of producing an artifact.

Today, business information processing is the single largest software application area. The software in this area restructures existing data in away that facilitates business operations or management decision making. Such data-centric business software has made considerable progress towards routine development. Well-designed VP tools and a very large set of high-quality COTS components have been developed. These assistants enable developers to create high-quality and low-cost software within very short time periods.

For data-centric business software, it is true that developers can always opt to use OOSDM to custom build from scratch rather than use VP to visually assemble with existing components. In most cases however, the arguments from quality, cost and time weigh against this alternative. Instead, the only option presented to developers is to choose which VP tool and COTS components to use.

Research validation: Software development is a complex activity involving human thinking and interpersonal communication. The human thinking is impossible to observe in any direct empirical way and the interpersonal communication is difficult to perform experimental manipulation on. Therefore, in software engineering the validation of research results is often an inherently difficult activity. Several empirical researchers[36-38] have found that most software engineering research lacks rigorous validation of their results.

An in-depth interview is a qualitative research strategy that allows person-to-person discussion. It can lead to increased insight into people's thoughts, feelings and experiences on important issues. Thus, we conducted face-to-face unstructured, in-depth interviews to collect experiences from VP professionals, so as to validate the correctness of the research results.

Participants: The participants in this validation were thirty-seven senior software engineers from nine small- to mid-size companies in Taiwan. All were familiar with OOSDM and all have five or more years usage experience

in VP. All were college graduates and major in information systems or computer science. Twenty-six have bachelor's degree and eleven have master's degree. Nine of the thirty- seven were woman.

Instrument: A validation instrument was developed drawing on an OO blueprint that appeared in OO literature[39]. The blueprint specified all the design information of a particular software system: the video store administration system for a chain of video rental stores. There were three reasons to use the blueprint.

- As the blueprint comes from OO literature, doubt about the correctness of it can be reduced.
- The software described in the blueprint was a typical data-centric business software system.
- The blueprint was an uncommonly detailed OO blueprint in OO literature.

The instrument is involved in six organizational functions that include circulation management, purchase management, inventory control, customer management, human resources management and decision analysis. Further, the instrument consists of following design models.

- One use case diagram (the only analysis model for explaining the requirements).
- Several object interaction diagrams.
- One class diagram.
- Several state diagrams.
- Several class specifications.
- One screen flow diagram.
- several screen layouts.

Process: First, under the interviewers' guidance, the participants were asked to read and understand the validation instrument (an OO blueprint). Then, the interviewers encouraged the participants to talk at length about the following topics:

- The usage experiences of VP.
- The usage experiences of OOSDM.
- Which aspects of the design information contained in the OO blueprint are valuable to VP?
- Whether any design information valuable to VP did not appear in the OO blueprint.

Furthermore, we analyzed the data to obtain the results. Finally, the results were reviewed and corrected by the participants.

RESULTS

Many qualitative analysis techniques have been developed, but analyzing qualitative data is still a difficult activity[40]. In quantitative analysis, numbers and what they stand for are the material of analysis. By contrast, qualitative analysis deals in words and is guided by fewer universal rules and standardized procedures than statistical analysis[41]. We study the data and look for themes, commonalities and patterns to try to make sense of the data. The following results (some tentative insights) were obtained:

First, VP is a practical method to implement software. All participants believed the type of data-centric business software, such as the software described in the instrument, can be implemented using purely VP and RDBMS.

Secondly, generally VP is purely an existing component-based implementation method. All participants disclosed that when they were the beginning users of a certain VP tool, they developed some components to enhance the component libraries that ship with the VP tool. However, as time went on, the number of new components being developed became fewer and fewer and the component development activities stopped in a short time. Additionally, two participants said that, their companies develop and maintain in-house component libraries and that their companies develop new software based purely on those libraries.

Thirdly, an OO blueprint is partially valuable to the development of a relational database. All participants indicated that each persistent class in class diagram can be mapped to a RDBMS table. Twenty-one participants point out that a data-related state diagram illustrates all the possible states that a particular persistent class can get into, so this kind of state diagrams is valuable to the development of relational databases. Nevertheless, all participants asserted no reason to develop an OO design for a relational database.

Fourthly, an OO blueprint is valueless to VP. All participants indicated that the design models within the instrument provide two kinds of design information.

- The design information of the custom classes.
- The design information of the software based on those custom classes. Such design information is not valuable to the existing purely component-based VP.

Finally, an OO blueprint lacks much of the design information that is valuable to VP. The participants pointed out several aspects of design information that did not appear in the instrument and which are valuable to VP. Such design information include.

- The form-oriented functional decomposition.
- The form structure.
- The form states.
- The property setting.
- The specification of event-handler.
- The specification of non-event-handling codes.

The results must be treated with caution because the validation was exploratory in nature and because of the small sample size. Nevertheless, the results suggest enough weighted evidence to support the correctness of the proposed arguments.

CONCLUSION

The purpose of this paper is to argue the un-match between OOSDM and VP from the viewpoint that "a blueprint must be followable." In the first place, we explained separately the essential features of OOSDM and VP which serves as evidence of the argument. Next, we inferred four reasons from the evidence to argue the un-match between OOSDM and VP. Additionally, we continued to argue that the mapping strategy and the method framework strategy are unworkable in VP. Finally, we presented the empirical validation of research results.

An argumentative research involves advocacy or persuasion. The researchers take a stand on an issue and defend it against opposing points of view. In such research, if the evidence is adequate and the reasoning is valid, then the conclusion is reliable[42-44]. This research has all such elements of a good argumentative research and moreover, the results of the empirical validation also support the research results. Therefore the argument that OOSDM un-match VP is reliable.

While VP poses unique challenges to software engineering, it does not require that we revisit each and every precept of theory and practice. To be sure, some outdated notions of development process need to be revised substantially and certain software engineering practices need to be adjusted. Further research may include a large-scale survey of software engineering requirements of VP. In addition, traditional software engineering techniques also have to adapt to VP. Such techniques include design processes, modeling tools, testing techniques, measurement methods and so on.

REFERENCES

1. Boehm, B.W., M.H. Penedo, E.D. Stuckle, R.D. Williams and A.B. Pyster, 1984. A software development environment for improving productivity. IEEE Computer, 17: 30-44.
2. Dorfman, M., 1990. System and Software Requirements Engineering. In R. Thayer and M. Dofrman (Eds.), Tutorial: System and Software Requirements Engineering, Los Alamitos, CA: IEEE Computer Society Press, pp: 4-16.
3. Fitzgerald, B., 1996. Formalised systems development methodologies: A critical perspective. Inform. Sys. J., 6: 3-23.
4. Sommerville, I., 2001. Software engineering. Boston: Addison-Wesley.
5. Avison, D.E. and G. Fitzgerald, 2003. Where now for development methodologies? Communications of the ACM, 46; 79-82.
6. Tolvanen, J.P., 1998. Incremental method engineering with modeling tools: Theoretical principles and empirical evidence. Unpublished doctoral dissertation, University of Jyvaskyla, Jyvaskyla, Finland.
7. Agarwal, R., A.P. Sinha and M. Tanniru, 1996. Cognitive fit in requirements modeling: A study of object and process methodologies. J. Management Inform. Sys., 13: 137-162.
8. Cooprider, J.G. and J.C. Henderson, 1991. Technology-process fit: Perspectives on achieving prototyping effectiveness. Journal of Management Information Systems, 7: 67-87.
9. Howard, G.S., T. Bodnovich, T. Janicki, J. Liegle, S. Klein, P. Albert and D. Cannon, 1999. The efficacy of matching information systems development methodologies with application characteristics: An empirical study. J. Sys. Software, 45: 177-195.
10. Land, F., 1998. A contingency based approach to requirements elicitation and systems development. J. Sys. Software, 40: 3-6.
11. Perez, G., K. El Emam and N.H. Madhavji, 1996. Evaluating the congruence of a software process model in a given environment. Proceedings of the 4th International Conference on the Software Process, Brighton, U.K., pp: 49-62.
12. Van Slooten, K.V. and B. Schoonhoven, 1996. Contingent information systems development. J. Sys. Software, 33: 153-161.
13. Alexander, L.C. and A.M. Davis, 1991. Criteria for selecting software process models. Proceedings of the 15th Annual International Computer Software and Applications Conference, Tokyo, pp: 521-528.
14. DeLone, W. and E.R. McLean 1992. Information systems success: The quest for the dependent variable. Information Systems Research, 3: 60-95.
15. Doke, E.R. and N.E. Swanson, 1995. Decision variables for selecting prototyping in information systems development: A delphi study of MIS managers. Information and Management, 29: 173-182.

16. Hardgrave, B.C., 1995. When to prototype: Decision variables used in industry. Information and Software Technology, 37: 113-118.
17. Havelka, D. and S. Lee, 2002. Critical success factors for information requirements gathering. Information Strategy: The Executive's Journal, 8: 36-46.
18. Siau, K., 2004. Informational and computational equivalence in comparing information modeling methods. J. Database Management, 15: 73-86.
19. Arinze, B., 1991. A contingency model of DSS development methodology. J. Management Infor. Sys., 8: 149-166.
20. Gemino, A. and Y. Wand, 2003. Evaluating modeling techniques based on models of learning. Communications of the ACM, 46: 79-84.
21. Hardgrave, B.C., R.L. Wilson and K. Eastman, 1999. Toward a contingency model for selecting an information system prototyping strategy. J. Manag. Inform. Sys., 16: 113-136.
22. Hughes, J., 1998. Selection and evaluation of information systems methodologies: The gap between theory and practice. IEE Proceedings on Software, 145: 100-104.
23. Brinkkemper, S., 1996. Method engineering: Engineering of information systems development methods and tools. Information and Software Technology, 38: 275-280.
24. Fitzgerald, B., N.L. Russo and T. O'Kane, 2003. Software development method tailoring at Motorola. Communications of the ACM, 46: 65-70.
25. Kinnunen, K. and M. Leppanen, 1996. O/A matrix and a technique for methodology engineering. J. Sys. Software, 33: 141-152.
26. Vlasblom, G., D. Rijsenbrij and M. Glastra, 1995. Flexibilization of the methodology of system development. Inform. Software Technol., 37: 595-607.
27. Booch, G., I. Jacobson and J. Rumbaugh, 1999. The unified modeling language: User guide. Boston: Addison-Wesley.
28. Martin, J. and C. Mcclure, 1988. Structured techniques: The basis for CASE. Upper Saddle River, NJ: Prentice-Hall.
29. Edwards, A.D.N., 1988. Visual Programming Languages: The Next Generation. ACM SIGPLAN Notices, 23: 43-50.
30. Borland, 2001. Delphi Developer's Guide. Scotts Valley, CA: Borland.
31. Jacobson, I., M. Christerson, P. Jonsson and G. Overgaard, 1995. Object-Oriented Software Engineering: A Use Case Driven Approach. Boston: Addison-Wesley.
32. Rumbaugh, J., M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, 1991. Object-Oriented Modeling and Design. Upper Saddle River, NJ: Prentice-Hall.
33. Jacobson, I., G. Booch and J. Rumbaugh, 1999. The Unified Software Development Process. Boston: Addison-Wesley.
34. Kruchten, P., 2000. The Rational Unified Process: An Introduction. Boston: Addison-Wesley.
35. Shaw, M., 1990. Prospects for an engineering discipline of software. IEEE Software, 7: 15-24.
36. Shaw, M., 2003. Writing good software engineering research papers: Minitutorial. Proceedings of the 25th International Conference on Software Engineering, Portland, OR, pp: 726-736.
37. Tichy, W.F., P. Lukowicz, L. Prechelt and E.A. Heinz, 1995. Experimental evaluation in computer sccience: A quantitative study. J. Sys. Software, 28: 9-18.
38. Zelkowitz, M.V. and D.R. Wallace, 1998. Experimental models for validating technology. IEEE Computer, 31: 23-31.
39. IBM OOTC, 1997. Developing object-oriented software: An experience-based approach. Upper Saddle River, NJ: Prentice-Hall.
40. Yin, R.K., 2003. Case Study Research: Design and Methods. Thousand Oaks, CA: Sage.
41. Miles, M.B. and A.M. Huberman, 1994. Qualitative data analysis: An expanded sourcebook. Thousand Oaks, CA: Sage.
42. Aldisert, H.R.J., 1997. Logic for lawyers: A guide to clear legal thinking. South Bend, IN: NITA.
43. Missimer, C.A., 2004. Good arguments: An introduction to critical thinking. Upper Saddle River, NJ: Prentice-Hall.
44. Toulmin, S.E., 1958. The uses of argument. Cambridge, U.K.: Cambridge University Press.