Malaysian Journal of Computer Science, Vol. 9 No. 2, December 1996, pp. 14-28

PERCEPTIONS OF SELECTED MALAYSIAN INFORMATION SYSTEMS PRACTITIONERS TOWARDS SOFTWARE PROTOTYPING: AN EXPLORATORY STUDY

Mohd. Hasan Selamat*
Md. Mahbubur Rahim**
Noor Maizura Mohammad Noor*

*Department of Computer Science
Universiti Pertanian Malaysia
43400 UPM, Serdang
Selangor Darul Ehsan
Malaysia
Fax: 6-03-9486646
email: hassan@cs.fsas.upm.edu.my

**Dept. of Computing and IS
Institut Teknologi Brunei
Simpang 125, Jalan Muara
Bandar Seri Begawan 3786
Negara Brunei Darussalam
Fax: 673-2-330776
Phone: 380249

ABSTRACT

Studies the perceptions of the Malaysian IS practitioners about the use, applicability, problems, and benefits of prototyping approach in the development of software systems. This was accomplished by undertaking an exploratory survey among selected information systems practitioners. The results indicate that the adoption of prototyping approach is relatively a new concept to the Malaysian practitioners. It was found that prototyping models are not generally thrown away, and prototyping in third generation languages is common. Prototyping approach was further applied to develop a wide variety of applications ranging from real time to traditional business systems. Some of these findings are contrary to the existing literature on prototyping. Hence, despite the survey's restriction to small sample, the findings are important to information systems practitioners and academics.

Keywords**: **Software prototyping, Information systems practitioners, System development life cycle

1.0 INTRODUCTION

Developed in the late 1960s and early 1970s, the traditional systems development life cycle (SDLC) is still the most commonly practiced approach to software development [1]. The widespread use of the SDLC however does not mean that it has always been seen as satisfactory [2]. In SDLC, there exists a limit in the extent to which user requirements can be captured and specified [3]. It is because users are not good in understanding computer related terms, while developers are not experts in the problem areas that require to be automated [4]. Moreover, requirements are generally expressed in terms of natural languages. Such natural languages are excellent mediums for novels and poems, where the quality of such works is partly judged by the degree of ambiguity in the text, but their use in requirements does not easily convey a realistic sense of how the software system will fit the user's needs [5]. Furthermore, natural languages are not particularly good at encouraging questions as to whether a user need is really needed or superfluous. Neither they are at all good at encouraging questions as to whether user requirements are complete [6]. As such, misunderstanding arises between users and developers, which leads to incomplete and often inconsistent requirements [7]. The traditional approach further assumes that all the requirements of the users can be identified in advance [8]. Unfortunately, users are not always able to highlight their needs early, and they do not consider system definition to be a front-end activity with a definite ending point [9]. As a result, software systems built with the traditional SDLC approach often fail to meet user requirements [10, 11], and require substantial maintenance [12]. Moreover, in the traditional approach, volumes of new documents need to be generated, and several tasks are repeated, if user requirements are revised. Such generation of paperwork and sign-off documents is very time consuming and costly, and often delays the installation of a system [13]. These problems have created a tremendous bottleneck in the SDLC approach [14]. As such, the appropriateness of the SDLC approach has been criticised [2, 15, 16, 17].

Software prototyping approach has emerged to address some of the difficulties inherent in the traditional software development approach [8, 18, 19, 20, 21]. Some authors like Berrisford and Wetherby [22], Jason and Smith [23], Lingraj and Kathawala [24] suggested that prototyping is a revolutionary approach for developing software systems. Even prototyping has been considered as an alternative model that would replace the traditional life cycle approach [25]. In prototyping, an initial and usually simplified prototype version of the system is designed, implemented, tested and brought into operation. Based on the experience gained in the operation of the first

prototype, a revised requirement is established, and a second prototype is designed and implemented. The cycle is repeated as often as is necessary to achieve a satisfactory operational system [26]. The prototyping approach is more explicitly iterative than the traditional life cycle method. Unlike the traditional SDLC, which must capture the correct version of a system the first time around, prototyping encourages experimentation and repeated design changes. Thus, prototyping calls for more intensive involvement of users and developers alike [27].

The goal of prototyping approach is to deliver a software system that is functionally correct, easy to use and learn [19]. The existing literature claims that prototyping results in better identification and validation of information requirements [28, 29, 30, 31], easy system implementation and acceptance [4, 32], and reduced over-all total life-cycle costs [18, 33]. Prototyping allows errors and omissions to be detected earlier than the SDLC. Without prototyping, errors may not be detected until later in the development process [34]. Prototyping is also effective for establishing superior user interface [18, 35]. Even, prototyping may be the only way of eliciting requirements from a user who is unsure of the very nature of the system he or she wants built [1]. Several empirical studies [19, 30] conducted in US and Europe support some of these claims. Over the past several years, the use of prototyping has increased dramatically [36]. A number of studies indicate a growing use and acceptance of prototyping approach among the western information systems practitioners. For instance, Langle et al. [37] indicated that 33% of the US organisations used prototyping to some extent and 61% was reported in 1990 [17]. Prototyping was even found to be more widely used than almost all the structured software development tools [38]. However, little empirical research has been conducted, to investigate how the Malaysian Information Systems (IS) practitioners use prototyping approach, and how successful they have been. This study was initiated to seek the perceptions of the Malaysian IS practitioners about the use, applicability, problems, and benefits of the prototyping approach in the development of software systems. This was accomplished by undertaking an exploratory survey among selected information systems practitioners.

This paper proceeds as follows: the next section defines prototyping, and describes its benefits, problems, taxonomy, and identifies the major concerns related to prototyping. After that research questions are highlighted, and the research approach is delineated. This is followed by a presentation of the survey results. Findings of the survey are discussed, and compared with some related studies. Lastly, some conclusions are made.

2.0 SOFTWARE PROTOTYPING REVISITED

Software prototyping is concerned with the construction of prototypes. The word *'prototype'* is derived from the Greek roots *'protos*' and *'topos*' meaning the 'first model' [39]. A prototype is an operational model of the application system [15]. It implements certain aspects of the future system, and provides a concrete basis for discussions between developers, users and management. Quite often, software prototyping is a subject of debate, due to the lack of a widely accepted definition of prototyping [5, 18, 40]. The following definition of prototyping as suggested by Rahim et al. [41, 42] has been adopted by this study:

> *Prototyping is an approach to develop computer-based software systems, in which a series of working models are constructed quickly in an iterative, and structured fashion, and is facilitated by active user involvement.*

The fundamental assumption underlying prototyping approach is that end-users seldom have clear and concise understanding about their information needs, and they do not normally know what to expect of future systems [21]. This problem is particularly acute for users who have never used a computer system before. As a result, it is difficult for the users to specify their requirements in advance [8]. However, once users begin to use a prototype, it soon becomes clear to them where the problems lie [18]. Users understand working prototypes better than logical graphical models, and are able to suggest refinements, while developers can understand where they need to make changes to meet user needs [28]. Demonstration of prototypes by developers help users to expose their unstated assumptions and trigger some of the inevitable requirements changes [43]. Thus, prototyping helps developers to build the right system [44, 45]. Furthermore, by allowing the end users to interact with prototypes, it is possible to validate their requirements much easily. Prototyping further enables management to check whether the software system development can be carried out in order to avoid unnecessary use of resources and to make sure that the wrong product is not built. Thus, prototyping helps to reduce risk in system development project [3, 43, 46, 47, 48].

In prototyping, users can further test their views by interacting with working prototypes in a live environment. Therefore, prototyping increases active user participation in the software system building process [49]. Such participation generates a sense of contribution and ownership among the users, lessens their learning curve [48]), and facilitates early acceptance of the system [4]. Maintenance is also expected to be less owing to the accurate identification of user requirements. As such,

software prototyping is often considered as an 'insurance policy' for success in systems development [50].

Despite these benefits, prototyping approach is not free from weaknesses. Potential problems of prototyping include: higher initial cost for the requirements phase of the development cycle, poor documentation, and unrealistic expectations from users, among others [42]. Prototyping approach has less control on project schedule and cost, due to unknown number of iterations involved in prototyping [21, 51]. Moreover, in the prototyping approach developers often rush to coding, thus, bypassing analysis and design stages [18, 52]. This leads to an ill-controlled project drifting through endless iterations [53].

Early efforts viewed prototyping as a single method that can be used to supplement the traditional system development life cycle approach [17]. More recent research studies, however, recognise the existence of a variety of prototyping methods. Unfortunately, there is a little agreement about the taxonomy. In general, four popular categories of prototyping are identified in the computer literature: user interface prototyping, rapid throw away prototyping, experimental prototyping, and evolutionary prototyping. The purpose of user interface prototyping is to establish how the interface between users and computer systems will look like and behave. The focus is on such matters as screen layout, dialogue style and ergonomics [6]. This techniques has, of course, been widely used for many years. The rapid throw away method produces throw away (disposable) models that simulate limited system functions, including interactions with a database. The central aim of rapid prototyping is to clarify requirements for the target system. It is useful at the early stages of software development, as it focuses on the communication problems between users and developers. Once the prototypes are developed, they are not delivered to the users, rather they form a basis for the rigorous design of the proposed system. In other words, in throw away prototyping, only the derived requirements are kept, but the code is thrown away [50]. A throwaway prototype implements only those requirements that are poorly understood. Thus, after the prototype is complete, the developer writes software specifications, incorporating what was learned, and constructs a full-scale system based on that specification [54]. Rapid prototyping can, however, be used in conjunction with the traditional SDLC approach. It augments user participation and understanding when applied at the requirements analysis stage. Thus, this approach does not replace the traditional SDLC method; but only improves it. One particular problem that may arise at some organisations is that undocumented prototypes that are intended to be thrown away are kept and become poorly planned bases for large complex systems that consequently becomes difficult to manage [50]. Moreover, when such prototypes are retained for use as an operational system, performance and input validation problems are most likely to occur [55].

In recent years, the availability of powerful design environments has given rise to evolutionary prototyping, in which a series of working models are constructed in an iterative and structured manner in order to deliver a complete software system [43]. Evolutionary prototyping builds successive models that evolve into a finished operational system. In contrast to a throw away prototype, an evolutionary prototype is built in a quality manner and implements only confirmed requirements. Evolutionary prototyping works well when most of the critical functions are well understood [54]. In general, the use of evolutionary prototyping has been deemed more appropriate for the development of small, specific decision support systems (DSS) than for large transaction processing systems [19]. Throw away and evolutionary prototyping have almost nothing in common except the word "prototyping". They are built differently, implement different functions, serve different purposes, and have different outcomes [54].

Experimental prototyping, on the other hand, involves building a prototype of a proposed solution to a particular problem [56]. The prototype is then evaluated by experimental use, in order to determine the adequacy of the proposed solution. In other words, experimental prototyping provides an opportunity to explore design alternatives after requirements have been defined. It is not a tool for helping to establish requirements in the first place [6]. This kind of prototyping enables users to further specify their ideas about the type of computer support required. Developers, for their part, are provided with a basis for appraising the feasibility and suitability of a particular application system [15].

Existing literature cautions that prototyping approach should not be adopted without suitable tools. However, the kind of tools needed is dependent upon the type of the prototyping approach used [39]. An example of user interface prototyping tool includes Smalltalk, a facility that is suitable for prototyping exotic user interface with windows pull-down menus with very little programming effort [55]. Rapid throw away prototypes can be created with simple tools, such as word processors or graphics packages. Often Prolog is used as a tool to develop rapid prototypes [57]. While evolutionary prototypes require more sophisticated tools such as database management systems, fourth generation languages or special purpose prototyping tools [58]. Even though these tools are extremely powerful in establishing a prototype, but the prototypes are extremely resource intensive and they run too slowly [55]. This inefficiency prevents using prototypes in a large scale high volume production environment [59]. In recent years, with the emergence of more powerful CASE tools, prototyping can be practiced

more effectively than before. CASE tools allow rapid prototypes to be built quickly. If the final system is constructed using a 3GL, CASE tools print out all the menu, forms, reports and commands in a logically organised document, providing chapter and section automatically that can be used as the functional specification document. This saves designers hours of labor [7]. If an I-CASE is used, it enables the results of prototype, to be used directly into the tool, for generation of program code in a 4GL. Currently, among these three types, tools for user interface prototyping dominate [3].

This study investigated a number of matters discussed above by undertaking an exploratory survey. A number of questions were asked to examine the extent the benefits, problems and the other interesting issues related to prototyping approach were actually experienced by the Malaysian information systems practitioners.

3.0 RESEARCH APPROACH

This study adopted a survey approach in order to find answers of the key questions as outlined in the earlier section. A questionnaire consisting of two parts was developed based on the available literature and experiences of the authors. Part A captured background information about the responding practitioners, while part B collected perceptions of the practitioners on the use of the prototyping approach. Some of the questions required the respondents to rate features of software prototyping on a five point Likert scale, some required ranking, while others required 'yes'/'no' answers. The questionnaire was reviewed by several academics. Later, the questionnaire was pilot tested among selected IS practitioners from industry. Based on their suggestions, one new question was included, which asked for the opinions of the practitioners regarding the degree of effectiveness of the software tools. Other minor changes involved rewording and rearranging of some of the questions. Revised questionnaires were sent to thirty randomly selected IS practitioners working in twenty different organisations during January 1995 to April 1995. These organisations were located in Kuala Lumpur and Selangor areas. These two areas were chosen because of the researcher's easy access to these areas. Besides, most of the major businesses including public offices are located in these two areas. Moreover, there is no reason why organisations located in these two areas would be different from those chosen from other areas of Malaysia. Similar assumptions were made by Carey and McCloed [38] while conducting the use of system development practices within Texas based organisations.

It must be emphasised that the surveyed sample constitutes thirty randomly selected IS personnel, not the number of organisations. Thus, it is hoped that the responses received from them would follow a normal distribution.

4.0 RESEARCH QUESTIONS

Most of the literature on prototyping are conceptual, and there exists a lack of empirical studies to provide a comprehensive evaluation of prototyping approach based on the field experience. In largely unexamined fields of study such as prototyping, it is advisable to start with qualitative field studies to gain an understanding. As such, this study has employed a survey approach which should be viewed in the light of the exploratory nature of empirical research. Furthermore, no prior survey on the use of prototyping approach was conducted among the Malaysian IS practitioners. Thus, a survey was undertaken in order to examine how prototyping was being viewed by the Malaysian IS practitioners. More specifically, this research attempted to seek the perceptions of the IS practitioners on the following key issues related to software prototyping:

- ☞ To what extent practitioners use prototyping ?
- ☞ What kinds of prototyping approaches are being adopted ?
- ☞ What types of applications are developed using prototyping ?
- ☞ Which categories of software tools are used in prototyping ?
- ☞ What kinds of documentation are performed in prototyping ?
- ☞ Is prototyping used within the phases of the traditional approach ?
- ☞ What are the reasons for adopting prototyping ?
- ☞ What problems are generally encountered in prototyping ?
- ☞ What benefits of prototyping are borne out in practice ?

5.0 RESULTS

Twenty organisations participated in the survey. The size of the organisations measured in terms of the number of employees, varied widely from less than 100 to over 1000. However, a dominant proportion of the organisations (70%) was large, as they had over 500 employees. Eight out of twenty organisations (40%) had large information systems department as they employed more than fifty IS personnel. Most of the surveyed organisations (70%) were in well established business for over a decade. The characteristics of the participating organisations are shown in Table 1.

Thirty information systems practitioners from twenty organisations participated in the survey. They represented a considerable spectrum of IS job categories. Essential characteristics of these practitioners are illustrated in Table 2. The practitioners were in responsible positions, and were familiar with the development methodologies used within their respective organisations. A majority of the practitioners (64%) were analysts, 17% were IS managers, while the remaining (19%) included EDP officers and executives of other category. Of the thirty respondents, 63% had a bachelor degree, 13% had a Master's degree, while the remaining 23% had a postgraduate degree in computing or information technology. Nearly two-thirds of the practitioners (60%) had over five years of experience in systems development, 27% had two to five years experience, while only limited 13% of the practitioners were relatively new in the field of information systems as they had less than two years of working experience.

Out of thirty, twenty two practitioners (73%) from thirteen organisations were found to use software prototyping approach. The experience of these practitioners in terms of applications developed through prototyping is reflected in Fig. 1. It can be observed that the adoption of prototyping is relatively new, because a considerable proportion of the practitioners applied prototyping approach in less than six projects. However, nine out of thirty participants (30%) developed over five applications using prototyping approach.

Malaysian information systems practitioners generally practiced more than one kind of prototyping. This is shown in Fig. 2. Throw away prototyping was found to be widely used, as half of the practitioners who adopted prototyping approach indicated its use. However, despite the popularity of rapid throw away prototyping, a considerable portion of the practitioners (41%) performed evolutionary prototyping. User interface and experimental prototyping were also found to be used, but to a lesser extent.

Table 1: Organisational characteristics

Total Employee in Organisation	**Number**	**Percent**
Less than 100	5	25
101- 500	1	5
501- 1000	1	5
More than 1000	13	65
Total	20	100
Number of IS Employee		
1 -10	5	25
11-50	7	35
51 - 100	4	20
Over 100	4	20
Business Experience of Organisations		
Between 1 to 5 years	4	20
Between 5 to 10 years	2	10
More than 10 years	14	70

Table 2: Practitioner's profile

Job Title	Number	Percent
IS Manager, Director, Head of Application	5	17
EDP Officer	1	3
Systems Analyst, Analyst Programmers	19	64
Research Officer	3	10
Asst. Manager	1	3
Computer Designer	1	3
Total	30	100
Academic Background		
Master	4	13
Post-graduate Diploma	7	23
Bachelor	19	63
Years of IS Experience		
Less than 2 years	4	13
Between 2 to 5 years	8	27
More than 5 years	18	60

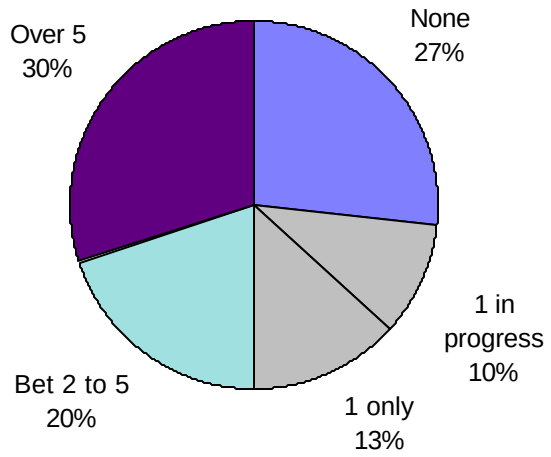

Fig. 1: Applications developed with prototyping

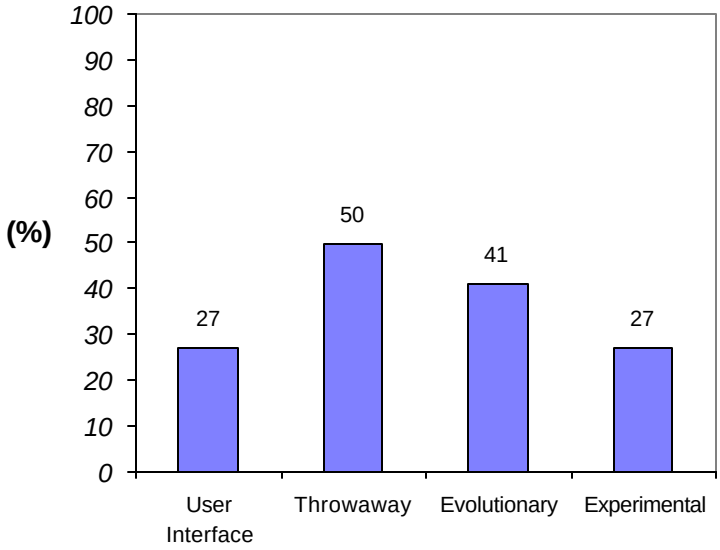

Fig. 2: Types of prototyping used by the practitioners

Prototyping approach was adopted within different phases of the traditional SDLC approach. This is illustrated in Table 3. Prototyping was used in requirements analysis phase to clarify requirements. It was also used within design phase to produce better interface. Some practitioners also developed prototypes to experiment between alternative designs. The use of prototyping was found to be the least in coding and testing phases. The cumulative percent in percent column in Table 3 is more than 100%. It is because often a practitioner used prototyping in more than different phases of SDLC.

Table 3: Use of prototyping in SDLC

SDLC Phases	Number	Percent
Analysis	11	50
Design	17	77
Coding	4	18
Testing	3	14

Prototyping approach was adopted to develop a wide variety of applications. Table 4 indicates that prototyping was primarily used in the areas of traditional business applications and executive information systems. Conversely, the use of prototyping was limited to decision support systems and expert systems. Interestingly, prototyping was used to develop real time applications. Business applications were primarily well structured in nature, thus, user interface prototypes were used to design user interface and experimental prototypes were used to chose between alternative designs. Executive information systems and DSS that are either semi or ill-structured, were developed primarily through evolutionary prototyping. Prototyping was further used to develop applications of varying size. However, most of the applications developed with prototyping was medium in size. Interestingly, some large applications were also found to be developed using prototyping approach.

This study found that practitioners used a wide range of software tools to construct prototypes. However, the adoption of third and fourth generation languages, and database packages were most frequently cited. Graphics software were used by limited practitioners. Surprisingly, sophisticated software packages like CASE tools were rarely used. None of the practitioners used applications generators. This is shown in Table 5.

Documentation is an essential aspect of systems development. It is recognized as an important written material for computer operations and maintenance. Existing literature reports that in prototyping, developers have the tendency to rush to coding. Thus, documentation is often ignored [52]. As such, undocumented prototypes are produced. A question was asked to inquire whether the practitioners documented prototypes as they would do if the systems were built with the traditional SDLC approach. The results are shown in Fig. 3, which indicates that a considerable proportion of those practitioners who adopted prototyping approach (41%) ignored documentation during prototyping. Only a limited number of the practitioners (4%) placed more efforts on prototype documentation as compared to the SDLC approach.

Table 4: Characteristics of applications built with prototyping

Application Types	**Number**	**Percentage**
Expert systems	1	4
Traditional business applications	14	64
Decision support systems	4	18
Compilers	3	14
Executive information systems	8	36
Real time applications	5	23
Type of Problem		
Well-structured	16	73
Semi-structured	14	64
Ill-structured	8	36
Size of Application		
Small	6	27
Medium	14	64
Large	10	45

Table 5: Use of software tools in prototyping

Prototyping Tools	**Number**	**Percentage (%)**
3GLs	14	64
4GLs	10	45
Database software	7	32
Graphics software	2	9
Applications generators	0	0
CASE	1	4
Others	1	4

The types of documents that were prepared while performing prototyping are listed in Table 6. The most prevalent forms of documentation were system flow charts and program flow charts. The use of data dictionary, data flow diagram and structure charts was found to be limited. This finding is not surprising given that most of the practitioners practiced rapid prototyping and user interface prototyping. It can be expected that the use of data dictionary, structured diagrams like data flow diagram, structure charts and action charts would increase with the gradual rise in the adoption of evolutionary prototyping.

The responding practitioners were provided with a list of reasons that motivated them to adopt prototyping approach, and were asked to identify the top three. The responses were aggregated using a simple weighting scheme, and are presented in the descending order as shown in Table 7. The top three reasons include: better identification of user requirements, reduction of development time, and active participation of end-users in the system development process.

Other considerations such as enhancing the image of the IS people and improving relationship with end-users, and the reduction of maintenance tasks were rated less important in adopting prototyping. The results further indicate that practitioners recognised the importance of the role played by the end-users in the development of information systems. As such, during prototyping, developers encouraged active involvement of the users in the development process. They considered that such user participation would assist them to identify and collect user requirements. Thus, uncertainty in determining user requirements could be reduced. These findings support that participating information systems practitioners fully understood the fundamental concept of prototyping. Survey results further reveal that Malaysian practitioners adopted prototyping for shortening application development time. The existing literature on prototyping, however, does not always consider this as an important reason.

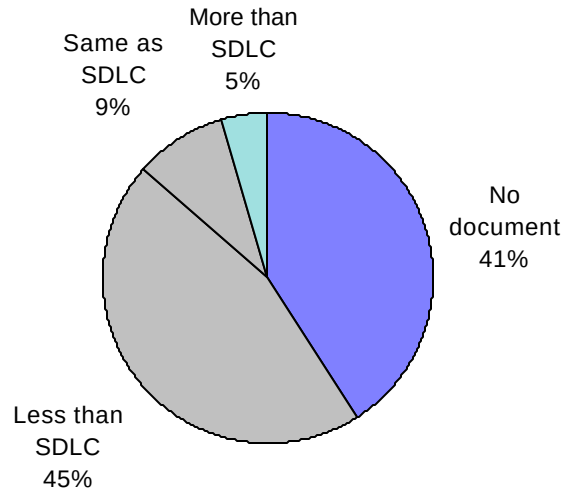

Fig. 3: Documentation in prototyping

Table 6: Types of documents used in prototyping

Types of Documents	Number	Percentage
System flow chart	10	45
Program flow chart	7	32
Data dictionary	4	18
Data flow diagram	4	18
Structure chart	4	18
Action chart	4	18
Narrative description	3	14
Others	0	0

Table 7: Reasons for adopting prototyping

Reasons for using Software prototyping	Rank			Weighted Average
	1	2	3	
To improve the capability of identifying user requirements	11	6	4	112
To reduce application development time	5	4	2	58
To encourage user participation in system development	2	5	5	57
To reduce application maintenance	3	2	7	56
To improve the image of IS people to end users	1	3	4	37
To improve relationship with end users	0	2	0	10

The responding practitioners were further provided with a list of problems that were encountered while developing software systems using prototyping approach. They were asked to rank the top three problems. The responses were aggregated using a simple weighting scheme, and are presented in the descending order in Table 8. The difficulty to control prototype iterations and estimate project schedule were identified as the most pressing problems. Another important problem was that software prototyping did not reduce project completion time, instead, in most cases, systems developed with prototyping were found to consume more time than anticipated. This frustrated the end-users and developers alike.

The existing literature on prototyping claims that several benefits can be gained using prototyping approach. A question, was thus, asked that required practitioners to rate a series of possible benefits of prototyping on a five point Likert scale, where 1 stands for very unsatisfactory, and five means very satisfactory. The responses from twenty two practitioners who practiced prototyping approach were compiled and are shown in Table 9. This table supports that adoption of software prototyping approach, indeed, helped developers to better capture requirements of the end-users.

Table 8: Problems in prototyping

Problems encountered by the practitioners	**Rank**			**Weighted Average**
	1	2	3	
Prototype iterations tend to continue infinitely	4	8	2	30
Difficult to estimate project schedule	6	2	4	26
Systems developed using prototyping consumes more time than anticipated	5	4	2	25
Prototypes created unrealistic user expectations	5	2	6	25
Users did not participate actively	1	4	3	14
Developers did not like to be criticised	1	2	3	10
Difficult to estimate budget	0	0	2	2

Table 9: Benefits of prototyping

Benefits	**Mean Rating**
Better identification of user requirements	4.1
Easy acceptance of the system by users	4.0
Higher job satisfaction of practitioners	3.8
Higher user satisfaction	3.6
Less maintenance is required	3.6

6.0 DISCUSSIONS

This study has found that a large proportion (73%) of the information systems practitioners from selected Malaysian organisations are using software prototyping approach. This indicates that prototyping approach has gained acceptance by the Malaysian IS community. However, the adoption of prototyping is relatively new, because around half of the practitioners (43%) have developed between two to five projects using prototyping approach.

An interesting observation is that all the four types of prototyping approaches as cited in the computer literature were found to be practiced by the Malaysian IS practitioners. Among them, rapid throw away prototyping was more widely used compared to other varieties. Practitioners applied rapid prototyping primarily to identify user requirements and to examine design alternatives while developing traditional business applications, which were well-structured in nature. Evolutionary prototyping was used to develop decision support and executive information systems. This is line with the observations of Kendall and Kendall [60] who suggested that less structured problems involving decision support systems are well suited to prototyping, and warned that traditional business applications are not good candidates for prototyping. However, the Malaysian IS practitioners also used prototyping for development of traditional commercial applications. This is not consistent with current opinions on prototyping [39].

On the other hand, user interface and experimental prototyping were used to a limited extent. The findings of this study are in agreement with those of Doke [17] and Guimaraes [50], both of them reported that US firms practiced illustrated (throw away) prototyping most frequently compared to other varieties of prototyping. However, a recent survey among forty US firms by Martin and Carey [19] found that evolutionary prototyping became more popular compared to rapid throw away prototyping. It is due to the emergence of suitable tools and growing awareness and favorable attitude towards evolutionary prototyping. With the rising awareness, continuous advancement of powerful tools, and user pressure, more and more Malaysian information systems practitioners are likely to adopt evolutionary prototyping in the near future. Thus, a time will arrive when a majority of the software applications will be built not by the traditional systems development life cycle approach, but by the prototyping approach.

Another striking observation is that the Malaysian IS practitioners applied prototyping approach even to develop real time applications. This is against the suggestions of Rakos [7] who cited that real time and scientific systems fare less well when prototyping approach is used. Malaysian practitioners further used prototyping to develop small, medium as well as large sizes applications. This is in agreement with Utz [61] who suggested that any software system more than several thousand lines of code should be prototyped, but against the advice of Alavi [28] and Laudon and Laudon [27] who advocated that prototyping is most effective for smaller applications, it cannot be applied easily to massive, mainframe based systems with complex processing instructions and calculations.

Guimaraes and Saraph [30] proposed that the benefits arising from the adoption of prototyping approach can be categorised into three different categories such as: managerial benefits (MB), end-user benefits (EUB) and technical benefits (TB). The survey results as shown in Table 9 suggest that both EUB and MB dominate. Technical benefits like: identifying optimum structure for metadata, validating database domains, testing or validating alternate systems designs to improve system performance were less perceived. Table 9 further supports that prototyping approach helped better identification of system specifications, and improved user-developers communication. As a result, software systems built with the prototyping approach met user requirements. This facilitated early acceptance of the system by the users, and raised satisfaction of the users with the system. Moreover, developers themselves were satisfied by developing a system that fulfilled user requirements. Thus, the Malaysian IS practitioners were able to develop the right system, albeit late. Moreover, practitioners adopted prototyping approach not for increasing efficiency. This is in line with the suggestions of Carey and Currey [62] who advised to use prototyping to define requirements more accurately rather than more quickly.

Another managerial benefit of prototyping is the impact of prototyping on systems maintenance. Table 9 indicates that software systems built with the prototyping approach needed less maintenance. This is not surprising because, in a typical system built with the traditional approach, up to fifty percent of maintenance that occurs in the first nine months or so immediately after delivery, result from mis-specification. The use of prototyping helped to produce better, more acceptable specifications, thus reduced maintenance.

Despite the reported benefits, the Malaysian information systems practitioners also encountered several problems while developing software systems through prototyping approach. The problems are shown in ranking order in Table 8. Difficulty in controlling the number of iterations in prototyping has been identified as the most pressing problem. It is not surprising, because several other authors like Mayhew et al [51], Rakos [7], and Rahim [63] argued that if developers allow any number of iterations to

the prototype, the requirements could go forever. The solution to this problem is that like the traditional approach, a specific time period could be set during which developers would seek feedback from users to evaluate how well the prototype is performing [60]. Alternatively, it is possible to set a limit to the number of prototypes. This has been supported by Rakos [7] and Connel and Shafer [64], who suggested that prototypes could be built in less than five versions, with three being the most common.

Software prototyping was used along with software development tools. Developers employed third and fourth generation languages, database packages and to a lesser degree, computer-aided software engineering (CASE) tools. Surprisingly, procedural third generation languages were found to be the most widely used tools, with more than half of the participants reported its use. Many of them stated that COBOL was used in evolutionary prototyping. The reason is that COBOL is still very popular among practitioners, and is regularly used to build applications. Thus, the use of COBOL helped the practitioners to avoid the conversion problems from prototypes (programmed in 4GLs) to operational systems that were running in COBOL. Additionally, fourth generation languages and database software were identified as the second and third most popular tools to develop prototypes. 4GLs were chosen because they have the power to handle complex screen logic, can perform special purpose calculations, and can invoke special routines (Rakos, 1990). Those who programmed prototypes in 4GLs used them primarily for throw-away purposes. The use of 4GLs in throw away makes development life cycle fast. But they are often inefficient in terms of machine utilisation. However, a few practitioners reported that they used the same 4GL to produce operational systems based on prototypes.

Mildly surprising was the comparatively low reported use of CASE tools: only one practitioner used a CASE tool to prototype a system. Perhaps, the benefits of CASE tools are not yet cost-effective. This finding did not support the suggestion of Rakos [7] who advocated that both disposable and evolutionary prototyping can be performed using CASE tools. Interestingly, functional languages like Prolog or Lisp or object oriented programming like C++ was not cited as a tool to construct prototypes.

7.0 CONCLUSIONS

This study reports the perceptions of thirty IS practitioners on prototyping. Out of thirty, twenty two practitioners (73%) from thirteen organisations were found to use software prototyping approach to some extent. The adoption of prototyping is relatively new, because a considerable proportion of the practitioners applied prototyping approach in less than six projects. Malaysian information systems practitioners generally practiced more than one kind of prototyping. However, throw away prototyping was found to be widely used, as half of the practitioners who adopted prototyping approach indicated its use. Despite the popularity of rapid throw away prototyping, a considerable portion of the practitioners (41%) also performed evolutionary prototyping. User interface and experimental prototyping were found to be used, but to a lesser extent. Prototyping approach was adopted within different phases of the traditional SDLC approach. Prototyping was used in requirements analysis phase to clarify requirements. It was also used within design phase to produce better interface. Some practitioners also developed prototypes to experiment between alternative designs. The use of prototyping was found to be the least in coding and testing phases. Prototyping was primarily used in the areas of traditional business applications and executive information systems. Conversely, the use of prototyping was limited to decision support systems and expert systems. Interestingly, prototyping was used to develop real time applications.

This research has supported many of the findings of the prior studies. However, some of the findings of this study are somewhat contrary to prior understanding of the use of prototyping in industry and hence warrant further investigation. The results of this survey are important to information systems managers who are considering or are already using prototyping in their systems development programs. They can use the results of this paper to compare against the systems developed in their own environment. The results are also important to researchers who need background information on the prototyping phenomenon or who are searching for research opportunities in this area.

Lastly, the small number of respondents selected from only two geographical areas may hamper generalisation of the results. Nevertheless, the results are valuable and warrant exploration. They can serve as a basis for further in-depth survey. Such in-depth survey should collect data from a larger number of respondents covering a wide range of industries located throughout the country. This would produce more reliable findings. Another limitation of this study is the omission of an important question regarding the percentage of applications that practitioners develop during a certain time interval (e.g. one year). Thus, research questionnaire should be revised to identify the actual percentage of applications that are produced through prototyping in relation to SDLC approach.

REFERENCES

[1] E. Yourdon, *Decline & Fall of the American Programmer*, Yourdon Press, PTR Prentice Hall, 1992, pp. 92-105.

[2] C. Avgerou and T. Cornford, *Developing Information Systems: Concepts, Issues and Practice*, Macmillan Press, 1993, pp. 119-137.

[3] E. Wallmuler, *Software Quality Assurance*, Prentice Hall, U.K., 1994.

[4] D. Appleton, "Data Driven Prototyping", *Datamation*, November 1983, pp. 259-268.

[5] W. Morrison, "Communicating with Users during Systems Development", *Information and Software Technology*, Vol. 30, No. 5, 1988, pp. 295-298.

[6] C. P. Allen, *Effective Structured Techniques: From Strategy to CASE*, Prentice Hall, 1991.

[7] J. J. Rakos, *Software Project Management for Small to Medium sized Projects*, Prentice Hall, Englewood Cliffs, New Jersey, USA., 1990.

[8] B. H. Boar, *Application Prototyping: A Requirements Strategy for the 80's'*, John Wiley and Sons, Inc., USA, 1984.

[9] L. Scharer, "Pinpointing Requirements", *Datamation*, April, 1981, pp. 149-151.

[10] B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, 1981.

[11] B. W. Boehm, T. E. Gray, and T. Seewaldt, "Prototyping Vs: Specifying: A Multi-Project Experiment", *IEEE Transactions of Software Engineering*, Vol. 10, No. 3, 1984, pp. 290-302.

[12] D. C. Ince and P. Mayhew, "Software Prototyping - Techniques and Management", In *Proceedings of the Second IEE/BCS Conference: Software Engineering 88*, London, UK, July 1988, pp. 1-2.

[13] K. C. Laudon, C. G. Traver and J. P. Laudon, *Information Technology Concepts and Issues*, Boyd & Fraser Company, USA, 1995.

[14] W. C. Andrews, "Prototyping Information Systems", *Journal of Systems Management*, September 1983, pp. 16-18.

[15] R. Budde, K. Kautz, K. Kuhlenkamp, and H. Zullighoven, *Prototyping - An Approach to Evolutionary Systems Development*, Springer-Verlag, New York, 1990.

[16] D. C. Ince, and S. Hekmatpour, "Software Prototyping: Progress and Pitfalls", *Information and Software Technology*, Vol. 29, No. 1, 1987, pp. 8-14.

[17] E. R. Doke, "An Industry Survey of Emerging Prototyping Methodologies", *Information and Management*, Vol. 18, 1990, pp. 169-176.

[18] J. M. Carey, "Prototyping: Alternative System Development Methodology", *Information and Software Technology*, Vol. 32, No. 2, March 1990, pp. 119-124.

[19] M. P. Martin and J. M. Carey, "Converting Prototypes to Operational Systems: Evidence from Preliminary Industrial Survey", *Information and Software Technology*, Vol. 33, No. 5, June 1991, pp. 351-356.

[20] K. Kautz, "Communication Support for Prototyping Projects", *Information and Software Technology*, Vol. 35, No. 11/12, December 1993, pp. 647-651.

[21] M. C. Er, "Prototyping, Participative Phenomenological Approaches to Information Systems Development", *Journal of Systems Management*, August 1987, pp. 12-15.

[22] T. Berrisford and J. C. Wetherby, "Heuristic Development: A Redesign of Systems Design", *MIS Quarterly*, Vol. 3, March 1979, pp. 11-19.

[23] M. A. Jason, and L. D. Smith, "Prototyping for System Development: A Critical Appraisal", *MIS Quarterly*, December 1985, pp. 305-316.

[24] B. P. Lingraj, and Y. Kathawala, "Prototyping: Models and Issues", In *Proceedings of Decision Sciences Institute*, Vol. 1, 1988, pp. 416-17.

[25] P. Loucopoulos, and V. Karakostas, *System Requirements Engineering*, McGraw-Hill Book Company, 1995.

[26] J. N. G. Brittan, "Design for a Changing Environment", *The Computer Journal*, Vol. 23, No. 1, January 1980, pp. 13-19.

[27] K. C. Laudon, and J. P. Laudon, *Business Information Systems*, The Dryden Press, USA, 1991.

[28] M. Alavi, "An Assessment of the Prototyping Approach to Information Systems Development", *Communications of the ACM*, Vol. 27, No. 6, 1984, pp. 556-563.

[29] M. Alavi, and J. C. Wetherbe, "Mixing Prototyping and Data Modelling for Information System Design", *IEEE Software*, May 1991, pp. 86-91.

[30] T. Guimaraes and J. V. Saraph, "The Role of Prototyping in Executive Decision Systems", *Information and Management*, Vol. 21, 1991, pp. 257-267

[31] G. Tate and J. Verner, "Case Study of Risk Management, Incremental Development, and Evolutionary Prototyping", *Information and Software Technology*, Vol. 32, No. 3, 1990, pp. 207-214.

[32] P.A. Dearnley and P. J. Mayhew, "In Favour of System Prototypes and Their Integration within the System Life Cycle", *The Computer Journal*, Vol. 26, No. 1, 1983, pp. 36-42.

[33] F. W. Nickols, "Prototyping: Systems Development in Record Time", *Journal of Systems Management*, September 1993, pp. 26-30.

[34] R. M. Stair, *Principles of Information Systems : A Managerial Approach*, Boyd & Fraser Publishing Company, USA, 1996.

[35] I. Graham, "Structured Prototyping for Requirements Specification in Expert Systems and Conventional IT Projects", *Computing & Control Engineering Journal*, March 1991, pp. 82-89.

[36] M. P. Martin, *Analysis and Design of Business Information Systems*, Prentice Hall, USA, 1995.

[37] G. B. Langle, R. L. Leitheiser and J. D. Naumann, "A Survey of Applications Systems: Prototyping in Industry", *Information and Management*, Vol. 7, 1984, pp. 273-284.

[38] J. M. Carey and R. McLeod, "Use of System Development Methodology and Tools", *Journal of Systems Management*, March, 1988, pp. 30-35.

[39] R. P. Cerveny and E. J. Garrity, "Why Software Prototyping Works", *Datamation*, August, 1987, pp. 97-103.

[40] B. C. Hardgrave and R. L. Wilson, "An Investigation of Guidelines for Selecting a Prototyping Strategy", *Journal of Systems Management*, April 1994, pp. 28-35.

[41] M. M. Rahim, M. H. Selamat and A. T. Othman, "Model of Structured Prototyping: A Case Study", In *Proceedings of the Third Conference on IT and its Applications*, Leicester, U.K., April 2-3, 1994, pp. 25-46.

[42] M. H. Selamat, M. M. Rahim and A. T. Othman, "Software Prototyping: Clarifying Common Misconceptions", Accepted *in Jurnal Teknologi Maklumat*, 1995.

[43] L. Luqi and W. Royce, "Status Report: Computer-Aided Prototyping", *IEEE Software*, November 1991, pp. 77-81.

[44] G. Tate, "Prototyping: Helping to Build the Right Software", *Information and Software Technology*, Vol. 32, No. 4, 1990, pp. 237-244.

[45] M. M. Rahim, M. H. Selamat, and A. T. Othman, "Surmounting Hurdles of Prototyping Adoption: An Action Strategy Framework", *In Proceedings of the South Asia Regional Computer Conference*, Karachi, Pakistan, November 1994, pp. 350-361.

[46] M. J. Earl, "Prototype Systems for Accounting, Information and Control", *Accounting, Organisations and Society*, Vol. 3, No. 2, 1978, pp. 161-170.

[47] R. Harrison, "Prototyping and the Systems Development Life Cycle", *Journal of Systems Management*, August 1985, pp. 22-25.

[48] S. Belardo and K. R. Karwan, "The Development of a Disaster Management Support System through Prototyping", *Information and Management*, Vol. 10, pp. 93-102.

[49] M. H. Selamat, M. M. Rahim and A. T. Othman, "End-User Involvement in Software Prototyping", *Jurnal Teknologi*, 1994, pp. 7-14.

[50] T. Guimaraes, "Prototyping Orchestrating for Success", *Datamation*, December 1987, pp. 101-106.

[51] Mayhew, P J, P. A. Dearnley, "Controlling Software Prototyping: A Change Classification Method", *Information and Software Technology*, Vol. 31, 1989, pp. 59-65.

[52] R. Vonk, *Prototyping The Effective Use of CASE Technology*, Prentice Hall International Ltd, U.K, 1990.

[53] Bart O'brien, *Demands & Decisions: Briefings on Issues in Information Technology Strategy*, Prentice Hall, U.K. 1992

[54] A. M. Davis, "Operational Prototyping: A New Development Approach", *IEEE Software*, September 1992, pp. 70-78.

[55] D. Bell, I. Morrey, and J. Pugh, *Software Engineering, A Programming Approach*, Prentice Hall, 1992.

[56] P. J. Mayhew, and P. A. Dearnley, "An Alternate Prototyping Classification", *The Computer Journal*, Vol. 30, No. 6, 1987, pp. 481-484.

[57] M. M. Rahim, *Introducing Control and Structure in Software Prototyping*, M.S. Thesis, Department of Computer Science, Universiti Pertanian Malaysia, Serdang, Selangor D.E., Malaysia, 1992.

[58] S. J. Andriole (1992) *Rapid Application Prototyping: The Storyboard Approach to User requirements Analysis*, QED Technical Publishing Group, USA, 1992.

[59] Mensching, J. R. and Adams, D.A. (1991) *Managing an Information System*, Prentice-Hall International, Inc., New Jersey, USA., pp. 38-381.

[60] E. K. Kendall and J. E. Kendall, *Systems Analysis and Design*, Prentice Hall, New Jersey, USA, 1992.

[61] W. J. Utz, *Software Technology Transitions*, Prentice Hall, U.K., 1992.

[62] J. M. Carey and J. D. Currey, "The Prototyping Conundrum", *Datamation*, June 1989, pp. 25-33.

[63] M. M. Rahim, M. H. Selamat and A. T. Othman, "Application of the User Satisfaction Approach to Control Software Prototyping: An Experience", Accepted in the *Singapore National Academy of Science Journal*, 1995.

[64] J. Connell, and L. Shafer, *Structured Rapid Prototyping: An Evolutionary Approach to Software Development*, Yourdon Press, U.S.A, 1989.

BIOGRAPHY

Mohd. Hasan Selamat, is a Senior Lecturer at the Department of Computer Science, Universiti Pertanian Malaysia. His research interest includes CASE, end-user computing and software prototyping. He has published several dozens of research papers in international and Malaysian journals and proceedings of international conferences. He is also the recipient of the best paper award in IT conference in UK.

Md. Mahbubur Rahim, Tel.: 673-2-330427 (Extn:145 or 152), Fax: 673-2-330776, received M.S. in Computer Science from Universiti Pertanian Malaysia in 1992. Currently, he is a Lecturer at Department of Computing and Information Systems, Institut Teknologi Brunei. His research interest includes CASE, and software prototyping. His research papers have appeared in several international journals including Information and Software Technology, International Journal of Information Management, Asia-Pacific Journal of Information Management. He is also the recipient of the best paper award in IT conference in UK. Currently, he is a member of the Australian Computer Society.

Noor Maizura Mohammad, is a tutor at the Department of Computer Science, Universiti Pertanian Malaysia. She graduated from the same university in 1994. Her research interest includes system development methods, CASE, end-user computing, and software prototyping.