http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 8 (4): 558-564, 2009
ISSN 1812-5638

Screen-Based Prototyping: A Conceptual Framework

[1]E. Kheirkhah, [1]A. Deraman and [2]Z.S. Tabatabaie
[1]Department of Computer Science, Faculty of Information Science and Technology,
National University of Malaysia, 43600, Bangi, Malaysia
[2]Faculty of Computer Science and Information Technology, University Putra Malaysia, Malaysia

Abstract: In any software development process, Requirements Engineering (RE) has been recognized as a critical factor in determining the quality of the software projects. In this study, an efficient technique, that is screen-based prototyping, is proposed to increase users' involvement in RE tasks and bridge the communication gap between end-users and software developers. This prototyping technique can be employed by most of requirements engineering methodologies. Screen-based prototyping employs the use-case driven approach in constructing prototypes and realize each use-case using a sequence of screens. Graph structure and related concepts are used to implement the prototypes and create various scenarios of use-cases.

Key words: Requirements engineering, end-user, prototyping, screen-based

INTRODUCTION

Despite significant progress that has been made in software development, software projects still suffer from some problems like over-budget, late delivery and poor quality. The success of a software system depends on how well it fits the needs of its users and its environment (Nuseibeh and Easterbrook, 2000; Parnas, 1999). Requirements engineering is accepted as one of the most crucial stages in software design and development. Requirements engineering first defines the problem scope and then links all subsequent development information to it. Several tasks and techniques have been proposed to accomplish the RE process. Using appropriate requirements engineering techniques during software development process has the potential of making a software system successful.

A substantial amount of research has been conducted to develop new methods, techniques and tools for requirements engineering. However, software projects still face to several problems. Communication gap between different stakeholders is one of the prevalent problems in this area. The system developers tend to speak with technical terms while the users prefer to use their natural language used in their daily business. As a result, the analysts may not have a clear idea about problem domain and users may not satisfy with the final product. End-users usually are not computer experts. They do not familiar with software engineering methods and techniques. However, end-user involvement in the software development process is undeniable necessity to ensure the quality of the software project at hand. Therefore methods which are easy to use and understand by end-users, support end-user involvement and facilitate communication, are required to bridge this gap.

Prototyping can be considered as a useful method in requirements engineering tasks. In this study, a conceptual framework for software prototyping, which is called screen-based prototyping, is proposed as a solution for above mentioned problems.

REQUIREMENTS ENGINEERING

The requirements engineering is an important part of software development process and plays a vital role in ensuring the quality of a software system. Requirements Engineering (RE), by definition is the discipline of determining, analyzing, pruning, documenting and validating the desires, needs and requirements of stakeholders for a system (Alan Davis *et al*., 2007).

As stated by Cheng and Atlee (2007), requirements reside primarily in the problem space whereas other software activities reside primarily in the solution space. That is, requirements descriptions, ideally, are written entirely in terms of the environment, describing how the environment is to be affected by the proposed system. In contrast, other software artifacts, focus on the behavior of the proposed system and are written in terms of internal software entities and properties.

According to Pohl (1994), requirement engineering is a systematic process of developing requirements through an iterative co-operative process of analyzing the

Corresponding Author: Esmaeil Kheirkhah, Department of Computer Science, Faculty of Information Science and Technology, National University of Malaysia, 43600, Bangi, Malaysia

problem, documenting the resulting observations in a variety of representation formats and checking the accuracy of the understanding gained. RE is inherently iterative and consists of a number of interrelated sub-processes. A correct, consistent and complete method, to collect, understand, specify and verify requirements is important. At the beginning of the RE process, unclear personal views of the system exist. Those views are usually recorded using informal languages, whereas at the end of the RE process a complete specification is expressed using formal languages on which an agreement should be reached.

SOFTWARE PROTOTYPING

Software prototyping is the process of creating an incomplete model of the future full-featured software program, which can be used to let the users have a first idea of the completed program or allow the clients to evaluate the program. Prototyping as a useful technique can be used in most of software development methodologies (Khalifa and Verner, 2000; Weidenhaupt *et al*., 1998). Prototyping method helps software developer to find and specify requirements, feasibility study of development strategies, design user interface, narrow the communication gap among stakeholders, increase level of end-user involvement in software development process and visualize future application for stakeholders.

There are various types of prototypes for different purposes in software engineering. Prototypes can be categorized, for example, as throwaway versus evolutionary, horizontal versus vertical, architectural versus requirements, textual versus visual and executable versus non-executable prototypes (Asur and Hufnagel, 1993; Leffingwell and Widrig, 2000; Kotonya and Sommerville, 1998; Wiegers, 1999).

Architectural prototypes are focused on the performance and the feasibility of the technology used, whereas requirements prototypes are suggested to be used in requirements acquisition and user interface design (Leffingwell and Widrig, 2000). Wiegers (1999) presented horizontal prototypes as behavioral prototypes that do not necessarily contain any real functionality, whereas a vertical prototype implements a specific part of system functionality in a quality manner. The goal of a horizontal prototype is refining unclear requirements while a vertical prototype is used, for example, in algorithm optimization (Mannio and Nikula, 2001). According to Kotonya and Sommerville (1998), throwaway prototypes are discarded when the final system has been developed, whereas evolutionary prototypes are made available to users early in the development process and then extended to produce the final system. Sommerville and Sawyer (1997) described executable prototypes as software constructed using a high level programming language and non-executable prototypes as paper prototypes and other mock-ups of the system.

Visual prototyping methods include such techniques as diagrams, state charts, storyboards and scenarios, whereas textual methods consist of formal prototyping languages (Asur and Hufnagel, 1993). Sutcliffe (1997) proposed that the use of prototypes, mock-ups, examples, scenes and narrative descriptions of contexts could all be called scenario-based approaches.

SCREEN-BASED PROTOTYPING

The proposed approach for prototyping, that is screen-based prototyping, prepares software developers with a very easy to use prototyping concepts and toolset which helps them to design a prototype of the software system at hand in a rapid and straightforward fashion. Screen-based prototyping employs the use-case driven approach in constructing prototypes. One prototype is designed for each use case selected for prototyping. Storylines will be generated and user classes are created in order to show the relationship among use-cases, user's actions and interaction between users and use-cases.

In order to have a clear perception of above mentioned concepts, Fig. 1 demonstrates hierarchical diagram of a software project from screen-based approach's point of view which firstly decomposed the project into several use-case and then each use-case

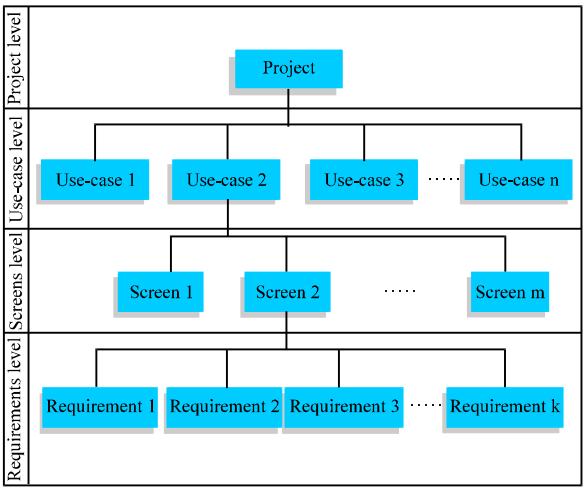

Fig. 1: Hierarchical demonstration of a project in screen-based approach

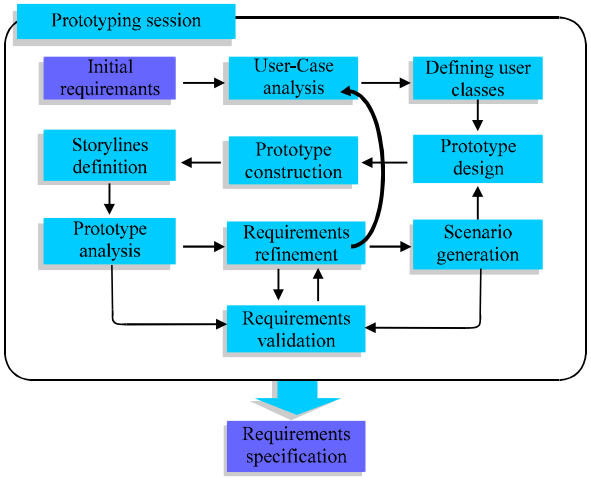

Fig. 2: Overall schema of prototyping session

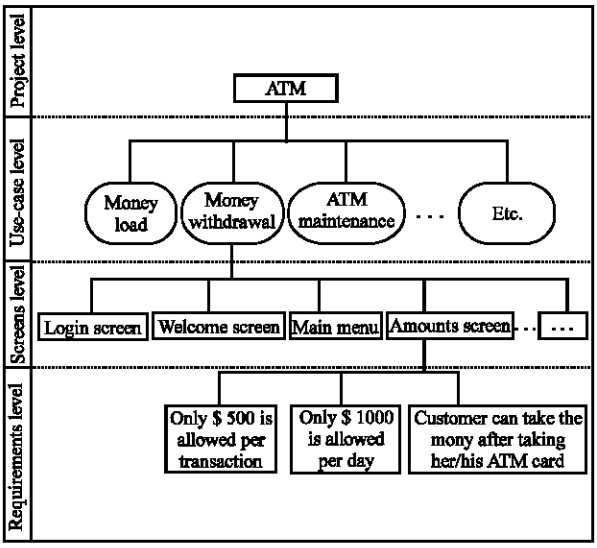

Fig. 3: Example of an ATM project

visualized using a couple of screens and per screen comprising some requirements in the next level.

Prototyping can be accomplished through a series of sessions with participation of various software system stakeholders which we call them prototyping sessions. Due to rapid development of the software prototype in the screen-based prototyping approach, software developers will be profited by early feedback of software users and stakeholders which will ensure quality of the software project. Figure 2 demonstrates overall schema of a prototyping session.

Prototyping session starts with an initial knowledge of problem domain and complementary knowledge will be acquired during this prototyping session and subsequent prototyping sessions. In order to achieve prototyping session's goals, a top-down manner will be employed, that is the moderator of session together with the other stakeholders try to identify important use-cases, define user classes and then design and construct the prototype for each use-case. In prototyping sessions, only uses-cases related to the participants in that session will be prototyped and other use-cases will be left for the subsequent prototyping sessions with participation of relevant stakeholders.

Prototyping session starts with an initial knowledge of problem domain and complementary knowledge will be acquired during this prototyping session and subsequent prototyping sessions. In order to achieve prototyping session's goals, a top-down manner will be employed, that is, the moderator of session together with the other stakeholders try to identify important use-cases, define user classes and then design and construct the prototype for each use-case. In prototyping sessions, only uses-cases related to the participants in that session will be prototyped and other use-cases will be left for the subsequent prototyping sessions with participation of relevant stakeholders.

As mentioned above, a project can be decomposed to several use-cases which were identified and analyzed in prototyping sessions. Visual methods in software engineering are more acceptable by end-users since they will have a clear imagination of what is happening by using these methods. Screens as a very efficient feature can help to visualize use-cases in a project. In this approach each use-case can be realized using one or more screens. A use-case prototype is a series of screens which simulate the use-case actions for elicitation, analysis, verification and validation of the requirements in a project.

In order to have a clear perception of above mentioned concepts, Fig. 3 demonstrates hierarchical diagram of an ATM software project from screen-based viewpoint. Execution order of screens (i.e., storyline) is not mentioned completely in the above hierarchical diagram. However, the storyline for each use-case is an important factor in realizing a use-case.

The requirements of the project lie in the last level of the above diagram. These requirements usually are atomic requirements. Some of the requirements may have been implemented in the prototype and some of them not, because the goal of prototyping is rapid development of few aspects of the software program being developed. These requirements will be acquired, recorded, prioritized and analyzed during screen-based RE process.

Mathematical model: The screen-based prototyping approach discussed above can be further converted into an equivalent mathematical model as given below.

Screen-based requirements engineering is started with initial knowledge of problem domain and the knowledge will be completed gradually within the process of Requirements Engineering (RE). As mentioned earlier, we applied use-case driven approach for screen-based requirements engineering. A use-case is a sequence of actions (here is a sequence of screens) that an actor performs within a system to achieve a certain goal. From this point of view, a software system can be decomposed into several use-cases. These use-cases may be changed during RE process. Therefore, each use-case can be defined at the state of time I as follows:

- $u_{j, I}$ is jth identified use-case at the state of time I. where, I = 1, 2,…, n and j = 1, 2,…, m

Then, u_i the set all identified use-case at the state of time I can be define as follows:

$$u_i = \sum_{j=1}^{m} u_{j,i} = \{u_{1,i} + u_{2,i} + ... + u_{m,i}\}$$

Description of a use-case is the most important feature of a use-case. Let S_P be the requirements specification of the project P. Description of use-case $u_{j,i}$ is defined as follows:

$D_{j,i}$ is the description of jth use-case at the state of time I.

If I⇒1 then the use-case description tends to end-user expression, but if I⇒n then the use-case and its description tends to be more complete and consistent. Figure 4 demonstrates use-cases evolution in a project.

The screens play important role in screen-based approach. In this approach each use-case can be realized using one or more screens. Each screen can be defined as an ordered pair (e, r), in which e is a subset of E (the set of screen entities like text-box, list-box, button, etc.) and r is a subset of R_i (the set of all requirements of project P at the state of time I) related to the screen. Let Scr_i be the set of all screens related to the all use-cases in project P at the state of time I. Therefore, Scr_i can be depicted as follows:

$$Scr_i \cong \{(e, r) \mid e \subseteq E, r \subseteq R_i\}$$

Screens, as mentioned above, are the most important component in screen-based prototyping framework. We defined Scr_i as a set of all screens related to the use-cases in project P. But it seems to be more than a simple set of objects. Although, each screen or a subset of screens is used to realize a use-case, it does not mean use-cases partition Scr_i to distinct subsets of screens, because some screens may be used in several use-cases. On the other hand the order of screens plays vital role in realizing use-cases. In fact execution order is a significant relationship among screens in each use-case. Figure 5 shows an example of such relationship.

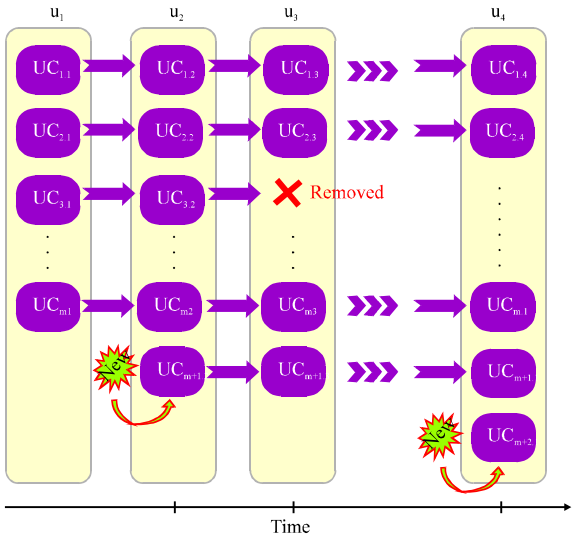

Fig. 4: Use-cases evolution timeline

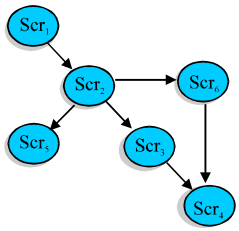

Fig. 5: An example of relationships among screens related to a use-case

Graphs can be considered as a good solution to implement these relationships among screens. Screens and their relationships (flows) can be represented as a directed graph. In screen-based approach we call the graph of a use-case, use-case storyline. Use-case storyline is the flow of its screens in order to realize the use-case. On the other hand, due to changeable nature of stakeholders' requirements, we may need to add, remove or change some screens in prototypes of use-cases during prototyping tasks.

Therefore, operations on graphs, such as addition or deletion of a vertex or an edge, merging and splitting of vertices, etc., will be useful to make changes in use-cases' prototypes. Here, we define a directed graph of screens for each use-case.

To this end, let $u_{j,i} \in U_i$ be a use-case and $Scr_{j,i} \subseteq Scr_i$ be the set of screens which realize $u_{j,i}$ at the state of time I. Then the graph $g_{j,i}$ is the set $\{Scr_{j,i}, O_{j,i}\}$ comprising the set $Scr_{j,i}$ of screens (vertices) together with the set $O_{j,i}$ of connections (edges), which are subsets of $Scr_i \times Scr_i$. That is:

$$g_{j,i} = \{Scr_{j,i}, O_{j,i}\}$$

where, $Scr_{j,i} \subseteq Scr_i$

$$O_{j,i} = \{ (x,y) \mid x \in Scr_i , y \in Scr_i \}$$

Let us call the set of all such graphs in project P as G_i at the state of time I. Then the prototype of use-case $u_{j,i}$ can be defined as follows:

$$P: U_i \rightarrow G_i; P(u_{j,i}) = g_{j,i} = \{Scr_{j,i} , O_{j,i}\}$$

Thus, the proposition each use-case can be realized by a set of screens can be expressed as follows:

$$\forall u_{j, I} \in U_i \; \exists \; g_{j,i} \in G_{p,i}; P(u_{j,i}) = g_{j,i}$$

It means each graph $g_{j,i}$ in G_i prototypes only one use-case $u_{j,i}$ in U_i. For instance, the graph in Fig. 6 can be represented as follows:

$g = \{Scr, O\}$
Where:
$Scr = \{scr_1, scr_2, scr_3, scr_4, scr_5, scr_6\}$
$O = \{(scr_1, scr_2), (scr_2, scr_3), (scr_3, scr_4), (scr_2, scr_6), (scr_6, scr_4), (scr_2, scr_5)\}$

The process of screen-based prototyping is iterative, that is, the prototype of the project will be built in an iterative fashion and can be repeated until satisfactory level of domain knowledge is acquired. As described in above constructing the prototypes is performed through prototyping sessions. During prototyping sessions, several changes may occur on different level. These changes can fall into the following categories:

- **Use-case changes:** Include, add a new use-case, remove an existing use-case and modify description of a use-case
- **Storyline changes:** Include, add a new screen to the graph, delete an existing screen and modify edges of a graph related to the use-case
- **Screens changes:** Modify the content of a screen related to a graph of a use-case
- **Requirements changes:** Add, remove, or modify the requirements of a screen

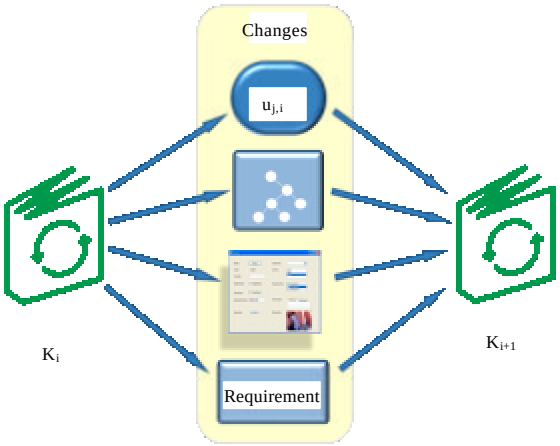

Fig. 6: Relationship between acquired knowledge of problem domain and entities of screen-based prototyping

Changing use-cases may cause changing storyline changes, screen changes and even requirements changes. Also changing storylines may lead to changing screens and requirements. The contents of screens may change during prototyping sessions which may cause changes in requirements related to the screen. Figure 6 displays the relationship between the acquired knowledge of the problem domain at the state of time I (i.e., K_i) and the changes of components in the screen-based prototyping.

After each prototyping session, the acquired knowledge of problem domain will be more complete. If we call the acquired knowledge of problem domain as K_i at the state of time I, then the function S maps K_i to K_{i+1}. That is:

$$S (K_i) = K_{i+1}$$

K_0 is initial knowledge about problem domain

In fact, the function S is the act of prototyping session which may make above mentioned changes on acquired knowledge of problem domain to move us one step ahead in order to have better understanding of the software project at hand. It means when I⇒1 we have only initial knowledge of the problem domain, when I increases then the knowledge about problem domain will be more rigorous and complete. Figure 7 demonstrates these steps.

Scenario generation: Scenarios are formal, sequential descriptions of the steps taken to carry out the use case, or the flow of actions that conduct during a use-case instance. These can include multiple scenarios, to cater for exceptional circumstances and alternative processing

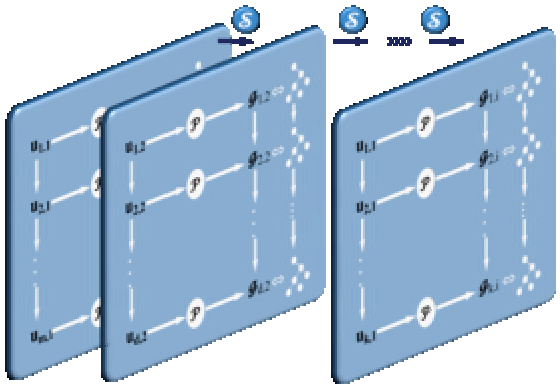

Fig. 7: Steps of knowledge evolution in screen-based approach

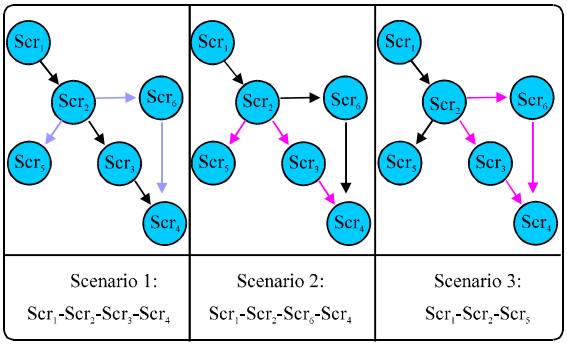

Fig. 8: Three scenarios based on various traversals of the graph

paths. Due to using graphs to represent storylines in use-cases, scenarios of a use-case can be generated easily. Various traversals of a graph of screens in screen-based approach indicate various paths of the graph from first node (first screen) to the leaf nodes (end screens). There are several graph traversal algorithms which can be employed to this end. The set $\tau_{j,i}$ of all scenarios related to the jth use-case (i.e., $u_{j,i}$) at the state of time I can be expressed as follows:

$$\tau_{j,i} \cong \{t \mid t \text{ is a traversal of graph } g_{j,i}\}$$

Figure 8 shows three scenarios related to the graph in Fig. 5 based on various traversal of the graph from Scr_1 to leaf screens (i.e., Scr_4 and Scr_5)

Let Scr_1 to Scr_6 be as follows:

- **Scr_1:** Customer selects or enters amount
- **Scr_2:** If balance <= amount then Scr_6
  If balance > amount then Scr_3
  If user cancelled transaction then Scr_5
- **Scr_3:** Pay cash
- **Scr_4:** Show main menu
- **Scr_5:** Stop transaction
- **Scr_6:** Show error message

Therefore, the three scenarios can be described as follows:

Scenario 1: Customer selects or enters amount. Balance is enough so cash will be paid and main menu will be shown again

Scenario 2: Customer selects or enters amount. Balance is not enough so error message is shown and the main menu is appeared

Scenario 3: Customer selects or enters amount. User cancelled transaction. Transaction is stopped

CONCLUSION AND FUTURE WORK

In earlier sections screen-based prototyping has been proposed and its components have been defined and discussed. The components of the proposed approach are fallen into the following categories

- **Functions:** Functions refer to activities to be accomplished during the course of screen-based requirements engineering. Following are recognized functions through the process of screen-based RE:
  - **D:** Description function which map each use-case to its description
  - **P:** Prototyping function which maps each use-case to a directed graph of screens
  - **S:** Prototyping session function
- **Dynamic entities:** Dynamic entities are defined as entities within screen-based RE that will act as argument for the function element. Following are our identified dynamic entities:
  - **K_i:** Acquired knowledge of problem domain at the state of time I
  - **U_i:** The set of all identified use-cases relevant to the project at the state of time I
  - **Scr_i:** The set of all screens related to all use-cases of the project at the state of time I
  - **$g_{j,i}$:** Prototype of a use-case $u_{j,i}$, implemented as a directed graph
  - **$\tau_{j,i}$:** The set of all Scenarios related to the use-case $u_{j,i}$ generated from graph $g_{j,i}$
  - **$R_{j(k),i}$:** Requirements relevant to each screen

- **Static entities:** Static entities are defined as entities within screen-based prototyping process that acts as the agents to execute related functions. These entities include all stakeholders of the project (i.e., requirements engineers, end-user, etc.)

In this study we briefly described software prototyping concepts and types. Screen-based prototyping was proposed in this study as a solution to communication problem among developers and other stakeholders. The framework of present approach was also demonstrated and its main components were discussed. Irrespective of the usefulness of the screen-based prototyping, this approach is capable of being employed by end-user developers in the area of End-User Development (EUD) and also by requirements engineers in a typical requirements engineering environment.

The completion of the screen-based prototyping approach, its toolset and its usability in a real industrial setting requires further research, which focuses on refinement of the framework, its design and implementation of its supporting toolset.

REFERENCES

Alan Davis, A.H., O. Dieste, N. Juristo and A. Moreno, 2007. A quantitative assessment of requirements engineering publications-1963-2006. Proceedings of the 13th International Working Conference on Requirements Engineering, Foundation for Software Quality, Jun. 11-12, Trondheim, Norway, pp: 129-143.

Asur, S. and S. Hufnagel, 1993. Taxonomy of rapid-prototyping methods and tools. Proceedings of the IEEE 4th International Workshop on Rapid System Prototyping, Jun. 28-30, IEEE Xplore, London, pp: 42-56.

Cheng, B.H.C. and J.M. Atlee, 2007. Research directions in requirements engineering. Proceedings of the International Conference on Software Engineering, May 23-25, IEEE Xplore, London, pp: 285-303.

Khalifa, M. and J.M. Verner, 2000. Drivers for software development method usage. IEEE Trans. Eng. Manage., 47: 360-369.

Kotonya, G. and I. Sommerville, 1998. RE, Processes and Techniques. 1st Edn., John Wiley and Sons Ltd., New York, ISBN: 0471972088.

Leffingwell, D. and D. Widrig, 2000. Managing Software Requirements: A Unified Approach. 1st Edn., Addison-Wesley, USA., ISBN: 0201615932.

Mannio, M. and U. Nikula, 2001. Requirements elicitation using a combination of prototypes and scenarios, technical report. Telecom Business Research Center, Lappeenranta University of Technology Finland.

Nuseibeh, B. and S. Easterbrook, 2000. Requirements engineering: A roadmap. Proceedings of the Conference on The Future of Software Engineering, Jun. 4-11, Limerick, Ireland, pp: 35-46.

Parnas, D.L., 1999. Software engineering programs are not computer science programs. IEEE Software, 16: 19-30.

Pohl, K., 1994. The three dimensions of requirements engineering: A framework and its applications. Proceeding of the Selected Papers from the 5th International Conference on Advanced Information Systems Engineering Table of Contents, 1994, Paris-Sorbonne, France, pp: 243-258.

Sommerville, I. and P. Sawyer, 1997. Requirements Engineering: A Good Practice Guide. 1st Edn., John Wiley and Sons Inc., Chichester, ISBN: 0471974447.

Sutcliffe, A., 1997. A technique combination approach to requirements engineering. Proceedings of the 3rd IEEE International Symposium on Requirements Engineering, Jan. 6-10, Annapolis, MD, USA., pp: 65-74.

Weidenhaupt, K., K. Pohl, M. Jarke, P. Haumer and T.H. Aachen, 1998. Scenarios in system development: Current practice. Software, IEEE, 15: 34-45.

Wiegers, K.E., 1999. Software Requirements Redmond. 1st Edn., Microsoft Press, Washington DC, ISBN: 0735606315.