Middle-East Journal of Scientific Research 8 (1): 274-281, 2011
ISSN 1990-9233

Improving Web Engineering and Agile ICONIX Process

[1]Morteza Poyan rad, [2]Homayon Motameni, [2]Hamid Alinejad Rokny and [1]Reza Pourshaikh

[1]Electrical and Computer Engineering Faculty,
Qazvin branch, Islamic Azad University, Qazvin, Iran
[2]Department of Computer Engineering, Sari branch, Islamic Azad University, Sari, Iran

Abstract: Web Engineering is the application of systematic, disciplined and quantifiable approaches for development, operation and maintenance of Web-based applications. It is both a pro-active approach and a growing collection of theoretical and empirical research in Web application development. Web development process, due to help to developers, has been divided into the manageable parts and delivering techniques, which can manage and complete the web projects. In this article an outlook of web engineering, differences between web engineering and software engineering, important approaches in web development, new advancements, a model for web engineering and implementing the application processes will be delivered.

Key words: Web engineering · Software engineering · ICONIX process · Agile

INTRODUCTION

Web developers, clients, government agencies, users, academics and researchers have increasingly become interested in Web engineering. In addition, this new field has attracted professionals from other related disciplines such as software engineering distributed systems, computer science and information retrieval [1]. Web-based systems change and grow rapidly in their requirements, contents and functionality during their life cycle - much more than what we would normally encounter in traditional software, information and engineering systems. Web-based system development is a continuous activity without specific releases as with conventional software. Thus, a Web-based system is like a garden - it continues to evolve and grow. With managing the diversity and complexity of web application development, although everything has progressed rapidly in the Internet and Web arena, nothing has changed significantly in the way that most people develop Web sites and applications. Web-based systems and applications now deliver a complex array of content and functionality to many heterogeneous end users and this trend will continue [1].

Unfortunately, however, the practices that developers follow for Web application development today are as poor as they were, when the Web was in its infancy. Many organizations and developers have successfully developed large, high-performance Web sites and applications, but others have failed or face the potential for major failures. The primary causes of these failures are a lack of vision, shortsighted goals, a flawed design and development process and poor management of development efforts-not technology [2]. Web Engineering is the application of systematic, disciplined and quantifiable approaches to development, operation and maintenance of Web-based applications. It is a response to the early, chaotic development of Web sites and applications as well as recognition of a divide between Web developers and conventional software developers. Viewed broadly, Web Engineering is both a conscious and pro-active approach and a growing collection of theoretical and empirical research [3].

The new Web engineering discipline deals with the process of developing Web-based systems and applications. The essence of Web engineering is to successfully manage the diversity and complexity of Web application development and hence, to avoid potential failures that can have serious implications. Web engineering helping practitioners develop and maintain high-quality Web systems and applications. It presents a Web design framework that facilitates application reuse, an object-oriented approach to legacy integration, a tool for constructing Web documents with visual simulations, Web metrics and a case study highlighting experiences in developing flexible Web services. This issue further

Corresponding Author: Dr. Homayon Motameni, Department of Computer Engineering,
Sari branch, Islamic Azad University, Sari, Iran.

explores Web-based systems development, practices and presents multidisciplinary perspectives that help shape this dynamic area of Internet and Web application development [2]. The rest of the paper is organized as follow: in the next section we present a comparison between software engineering, web engineering, in section three, the importance of web engineering has been stated and in section four, we describe the ICONIX process as a model of web development. Finally, section five concludes the paper.

Comparing Software Engineering and Web Engineering: Software engineering uses a regular, systematic and measurable approach for software maintenance and development and is complementing, developing, managing and specifying the software systems. It is a rule, which aims to produce software with high quality, on time, with a certain fund and user friendliness. Nowadays by spreading the web applications in common trade strategies, a need to construct the suitable system applications plays an important role. So we need a rule based method (Web Engineering) to create and develop the web applications. Web engineering is a process to create web applications with high quality. It is not exactly the same as software engineering. However, it includes many concepts, basic rules, lots of techniques and management methods that are used in software engineering. Against some of the developers and software engineer's ideas, web engineering is not a child for software engineering, although both of them have challenged in programming and developing [4]. Despite this, web engineering has accepted a lot of principles in software engineering but a lot of new approaches, methodologies, tools, techniques and strategies have been created based on the web [5].

Web based systems development is different from traditional software, remarkably. Web development is a combination of industry, press, software development, marketing, computing, intercommunications, art and technology. Constructing the complicated web application requires a team with various specific ability. So, web engineering is a multiple system and needs a lot of various proficiencies such as analysis and design, software engineering, hypertext and hyper media engineering, engineering requirements, human and computer interaction, user interface development, data engineering, information indexing and retrieval, testing, modeling, simulation, project management and graphic design. In opposition, web applications have been created for meeting the needs with high quality, while most of the traditional software could do this, with a relatively low quality [6-7]. Most of the web applications are integrating by keeping the information and they are like a garden, which is grown up and completed continuously.

With quick evolution of internet technology, to achieve the final production, web application development cycle should be a short periodic time. Otherwise, before achieving the final production, development process may fail due to technology evolution [8]. Because of the heterogeneous architectures, which have applications running on them, it has been suggested that software development begin with a high-level design. In a way that, before the requirement collecting, restrictions of the presented architecture should have been studied, by means of web media. These restrictions will help us to exclude the requirements [9]. Also in requirements collecting, fewer decisions have been made and more restricted options will exist. Therefore, requirements engineering in the web is easier than open models. The feasibility phase in web application will be executed in less time and lower cost rather than traditional applications [10-11].

Modeling in traditional applications has been accepted as an effective tool for analysis and authenticating of requirements [10]. But traditional modeling is based on hypotheses which are not suitable for web applications. In web applications, although a constructed model as a tool in development process will be used, but this production will be used in real world modeling and with real workers. Therefore, modeling the web applications toward traditional modeling should follow the standard principles and have a good degree of quality [8].

Nowadays, users see the web applications by means of internet browsers, but on the other hand, due to a competitive challenge between browser producers, they do not want to follow each other’s principles and standards. Therefore, some of the attributes, which are executable in browser A, are not executable in other browsers. When a browser does not support some attributes, it means, that a web site has lost a lot of visitors and customers. Because of this, web application developers have encountered many problems, which traditional application developers are not dealing with and, a web site dealing with all browsers is directly based on developer efforts [7].

One of the new aspects of web applications is that they are changing a lot. Traditional applications are dealing with marketing, sales, transportation and because of high cost of these processes, software companies are

Table 1: software engineering VS web engineering

Users domain	Users domain	Users domain
Number of simultaneous users	Low	High
Time deadline	We have time deadline but less than web engineering. We have more time to develop the applications	Web development for reaching the final production should be accomplished in short time. Otherwise, because of technology evolution, development process may fail.
User's requirements	Specified	Changing rapidly
Fund	Different ranges, depend on company	Relatively comparative, in a small domain
Requirements engineering	Due to importance of collecting the information, requirements engineering is essential and more complicated.	For requirements collecting, fewer decisions will be made and restricted options will exist. Therefore, requirements engineering will be easier in web engineering.
Feasibility testing phase	Needs more time and money	Needs less time and money
Hardware and software areas constraints	Certain and specified	Uncertain and changing
Security and logical issuance	Not very important because of certain and limited users. There is more conservation.	Very important. Because users are not certain and it includes all the web
Changing rate	Due to some factors like high cost, changing rate is not rapid and users, in exerting their interests, are not impatient.	Users have been allowed to change the data rapidly, so changes should be accomplished swiftly.
Modeling importance	An effective tool for requirements analysis and authentication	Modeling in web engineering should adhere to specific standards and conditions and it is not as important as in software engineering
Common goals	Programming and software development to achieve a qualified software which is on time and user friendly with a certain fund	Programming and software development to achieve qualified applications which are on time and user friendly, with a certain fund
Adhering to standards and protocols	Not so much important and just in necessary occasions	Very important
User idea about final production	Important	It goes to culture, achieving systems, etc.
User interface	Important	Very important. One of the most essential goal of web development
Network necessity	It depends on the conditions of used software	Very important

usually collecting the maintenance changes and after passing some time create the new version and send it to the customers [11]. Users are not important and are not trying to change these applications by themselves, but in web applications, which do allow customers to achieve the changes, essential changes should be exerted rapidly and all of the errors should be debugged. In this application instead of a monthly or a yearly maintenance cycle, we have a daily or hourly maintenance cycle. Therefore, if a site appeared with new ideas and good services in internet, many users will surf it and a number of those users, which are, trying to visit this site will become elevated. Consequently, if this site could not deliver good services to this high number of users, it will lose the customers. If this site wants to interest the users again, surely it should cost a lot in advertising. Therefore, web applications should be engineered in a way that they can increase their capabilities of giving the services to numerous customers. A comprehensive comparison between software engineering and web engineering has been shown in table 1.

The Essence of Web Engineering: Successful Web-based system development and deployment is a process, not just an event as currently perceived and practiced by many developers and academics. Web engineering is a holistic approach and it deals with all aspects of Web-based systems development, starting from conception and development to implementation, performance evaluation and continual maintenance. Building and deploying a Web-based system involves multiple, iterative steps. Most Web-based systems continuously evolve to keep the information current and to meet user needs.

Web engineering represents a proactive approach to creating Web applications. Web engineering methodologies have been successfully applied in a number of Web applications (for example, the ABC Internet College, 2000 Sydney Olympics, 1998 Nagano Winter Olympics, Vienna International Festival and many more). These and other success stories are encouraging Web developers to adopt Web engineering principles [1-2]. To successfully build large-scale, complex

Web-based systems and applications, Web developers need to adopt a disciplined development process and a sound methodology, use better development tools and follow a set of good guidelines.

The emerging discipline of Web engineering addresses these needs and focuses on successful development of Web-based systems and applications, while advocating a holistic, disciplined approach to Web development. Web Engineering uses scientific, engineering and management principles and systematic approaches to successfully develop, deploy and maintain high-quality Web systems and applications. It aims to bring Web-based system development under control, minimize risks and improve quality, maintainability and scalability of Web applications. The essence of Web engineering is to successfully manage the diversity and complexity of Web application development and hence, avoid potential failures that could have serious implications [3-4].

Ten Key Steps for Successful Development: Based on our experience in building many Web-based systems and on our research, we recommend the following 10 key steps to follow for successful Web development and deployment [3-5]:

- Understand the system's overall function and operational environment including the business objectives and requirements.
- Clearly identify the stakeholders - that is, the system's main users, the organization that needs the system and those who fund the system's development.
- Specify the technical and non-technical requirements of the stakeholders and the overall system.
- Develop an overall architecture of the Web based system that meets the technical and non-technical requirements.
- Identify subprojects or sub processes to implement the architecture. If the subprojects are too complex to manage, further divide them until they become a set of manageable tasks.
- Develop and implement the subprojects.
- Incorporate effective mechanisms to manage the Web system's evolution, change and maintenance. As the system evolves, repeat the overall process or some parts of it, as required.
- Address the non-technical issues such as revised business processes; organizational and management policies; human resources development; and legal, cultural and social aspects.
- Measure the system's performance.
- Refine and update the system.

Although the Web engineering discipline is young, it is gaining the attention of some researchers, developers, academics and other major players in Web-based systems implementation such as customers and clients. Web engineering needs to evolve and mature to effectively handle the unique challenges that Web-based systems and applications pose [3].

A number of members of the Web engineering community have commented on the lack of a suitable Web engineering process for the development of Web applications. At present, most of the industry focus is on tools, which aid and assist the implementation of Web-based systems. Yet, as this survey has shown, the greatest problems facing the field are at the Analysis, Requirements, Testing (verification), Evaluation (validation) and Maintenance stages of the Web engineering life cycle. If a Web engineering process is to be successful then it must address the following [5]:

Short Development Life-cycle Times: According to our results, the average development life-cycle time is less than three months. This figure is dramatically lower than that of traditional software development. If a Web engineering process is going to be successful, then it has to cope with exceptional time pressures.

Delivery of Bespoke Solutions: Unlike traditional software engineering, Web engineering must cope not only with the development of software components but also with the development of data and the inter-dependencies between them.

Multidisciplinary Development Teams: Many have commented on the diverse variety of disciplines involved in the development of Web-based systems (Boehm, 1988). A Web engineering process must take into account the different types of developer required to build a successful solution. Ensuring that all involved understand their roles and responsibilities, where overlap occurs and how to resolve conflict in the best interests of the project in question.

Small Development Teams Working in Parallel on Similar Tasks: Like traditional software development, large numbers of Web developers are split into smaller teams. Each team providing an interface to the deliverables that they produce, enabling other teams to view the deliverable as a black box. A Web engineering process should also allow different types of developer to

communicate amongst their peers, beyond particular projects and teams. This will help ensure consistency and will prevent duplication of effort amongst teams.

Analysis and Evaluation: There is a clear need for focus on Analysis and Evaluation stages in Web engineering processes. Now, there is little attention paid to either of these stages. A Web engineering process should encourage developers to address the following questions:

- Why are we going to develop a Web application?
- What problems or goals will the Web application address?
- How will we know if the solution addresses these problems or goals?
- How will we measure and validate the deliverables?

Requirements and Testing: In addition to understanding, what issues the Web-based solution should address and how we will measure the success of the deliverables in tackling these issues, a Web engineering process needs to focus more on Requirements and Testing. A Web engineering process should force its users to ask the following questions:

- Are we building the product in the correct way?
- How will we ensure that we have built the product correctly?

Maintenance: If the longevity and quality of Web-based solutions are to be improved, then more attention needs to be paid to the issue of Maintenance. While a well-documented system may not be essential during the development of a Web application, it is certainly necessary for ensuring the proper maintenance and update of the deliverables.

Not only must the process address the points 1-7 above, but it should also be independent of tools and technologies. This is not to say that supporting CASE tools are not important. But such is the diversity and rapid change of technologies used to build Web applications, that a Web engineering process should remain as distant as possible from mechanisms used to implement Web applications.

ICONIX Process: ICONIX Process originated several years before the Unified Modeling Language UML and the Unified Process (UP) as a synthesis and distillation of the best techniques from the original methodologies that formed the UML, Jim Rumbaugh's Object Modeling Technique (OMT), Ivar Jacobson's Objectory method and Grady Booch's Booch method [10-11].

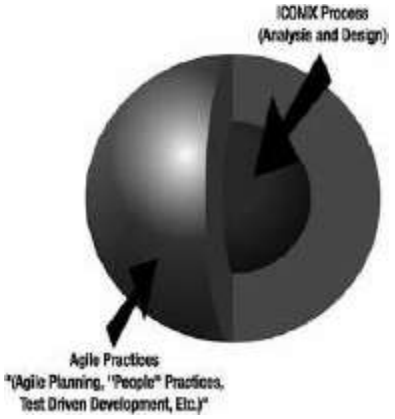

Fig. 1: Agile ICONIX in a nutshell

We attempted a synthesis of these three very different schools of object-oriented (OO) thought because the strengths and weaknesses of these methodologies seemed to complement each another. OMT was useful for problem domain (analysis-level) modeling, but it was not as strong for solution space (detailed design) class modeling, while the Booch method was strong at the detailed level but unintuitive at the analysis level. The Objectory method took Booch's concept of a dynamic model and extended it with a systematic approach that leads cleanly from a use case view to a detailed design (sequence diagram) view of the runtime behavior. Both OMT and the Booch method were stronger for static (class) modeling, while the Objectory method is mostly focused on dynamic runtime behavior. Booch also introduced component and deployment views of the system.

ICONIX Process is a use case-driven analysis and design methodology. Its focus is on how to get reliably from use cases to code, in as few steps as possible. In this book, we describe ICONIX Process and show how it was applied to a real-life project. We also describe in detail how to apply ICONIX Process to a broader agile project environment. This combination of process and practices is shown in Figure 1. Informally, we refer to this combined process as Agile ICONIX.

Although much of ICONIX Process can be considered agile, some parts of it stand in stark contrast to recent agile thinking. In particular, agile methods often tell us not to bother keeping design documentation and source code synchronized. The theory is that once code has been written, we do not need the diagrams anymore. ICONIX Process, on the other hand, suggests exactly the opposite: the more tightly synchronized the design documentation is with the code, the faster, more maintainable and more accurate (i.e., closer to the customer's requirements) your project will be over

Table 2: comparison between agile, ICONIX and heavy

	Agile methods	ICONIX methods	Heavy methods
Approach	Adaptive	Adaptive	Predictive
Success measurement	Business value	Software working	Conformation to plan
Project size	Small	Medium	Large
Management style	Decentralized	Decentralized	Autocratic
Documentation	Low	Low	Heavy
Emphasis	People-oriented	Plan-Driven and Feedback-Driven	Process-oriented
Cycles	Numerous	Regular (fixed size iteration)	Limited
Domain	Unpredictable/Exp-loratory	real-world objects and relationships	Predictable
Team size	Small/Creative	Small/Creative and productive	Large

successive releases. Luckily, ICONIX Process also aims to make this process easier (again in contrast to other agile processes). To achieve this tight synchronization between diagrams and code, we need to cut down on the number of diagrams that we have to draw (and therefore maintain). It is also important to know which diagrams are going to be important to ongoing iterations and which can safely be discarded. "Minimal but sufficient" analysis and design is right at the core of ICONIX Process [11].

If we think of a big, high-ceremony process as elephantine and an extremely minimal/agile one as mouse-like, it is amusing to note that some people try to equate the mouse to the elephant because both are gray mammals with tails. In reality, however, there are significant differences between a mouse and an elephant and none of those animals does the work of a horse very well. We have interpreted the goals of agility as the ability to followings:

- Respond to changing requirements in a robust and timely manner.
- Improve the design and architecture of a project without massively influencing its schedule.
- Give customers exactly what they want from a project for the dollars they have to invest.
- Do all this without burning out your staff to get the job done. These goals are sometimes overlooked in the rush to address the values. The agile values are important but, we feel, are there to fulfill these goals. The key differences are listed in followings:
- With ICONIX Process, an emphasis is placed on getting the requirements right up front. This is done through a variety of practices that root out ambiguity in the requirements and uncover gaps in the use cases and domain model.
- With ICONIX Process, design is not something that you "quickly do a bit of" before coding (where the real design work happens). Instead, design modeling and coding are tightly interleaved, each providing feedback to the other. Essentially, in ICONIX Process, we "model like we mean it" - that is, we intend to drive the coding of the system from the model.
- With ICONIX Process, the design and the code do not diverge over time. Instead, they converge, with the design model and the code becoming more in harmony. In table 2, we present a comparison among agile, ICONIX and heavy method of software development.

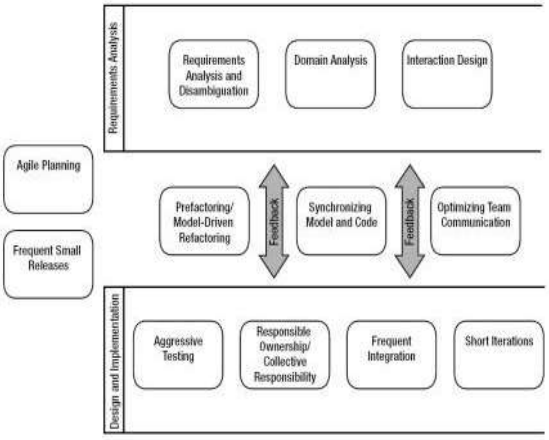

Fig. 2: Core agile practice

The Agile ICONIX Practices: This list gives a little more detail about the practices shown in the diagram of figure 2. These practices are also listed in a suitable adoption order (i.e. adopt Practice 1 and then practice 2 and so on). So, in a sense, this list doubles as an "adoption roadmap":

ICONIX Process Practices: These practices cover ICONIX Process analysis and design (plus interaction design) which is a useful way of controlling change by reducing requirements churns.

Requirements Analysis and Disambiguation: Reduce requirements churn using a variety of techniques (including robustness analysis and domain analysis - see the next practice).

Domain Analysis: Capture a common language that can be shared by everyone involved in the project, including the customer and users.

Interaction Design (Insert Your Favorite Interaction Design Technique Here): Interaction design, performed as early in the project life cycle as possible, can have a profound effect on the team's understanding of the problem domain. *Take the time to understand the user's needs* (e.g. by interviewing actual end-users) and to balance their needs in the context of the customer's business requirements.

This in turn reduces requirements churn and therefore reduces the expense incurred from responding to changing requirements.

Prefactoring/model-driven Re-factoring: Apply the right design practices to get the design as "on the nose" as possible, up front. We have coined the term "prefactoring" to mean using up-front design modeling, to re-factor and improve the design, before writing the code.

Agile Practices: The remaining practices can be thought of as "traditional" agile practices.

Aggressive Testing: As we describe in the Agile ICONIX book, Test-Driven Development (TDD) can be adapted to become a very effective form of design review ("micro-tests" are driven by the controllers and boundary objects in your UML diagrams). This process is particularly good for exploring all of the "rainy day scenarios" in your usage cases. Additionally, if further code re-factoring needs to be done, then it is important to have a set of unit tests in place.

Frequent Small Releases: It pays to release the product to the customer often, so that you can get feedback as early as possible on whether, or not, the product is heading in the right direction. Note that a "release" is not necessarily a "full customer ship"; it may be a prototype release, a proof-of-concept, a QA testing release, etc.

Frequent Integration (a.k.a. Very-nearly-continuous Integration): integrating all production code at frequent intervals. We also view it as essential to have a dedicated build PC which gets all the latest code, builds it and runs all the automated tests, ideally once per hour.

Synchronizing Model and Code after Each Release: With this practice we are enabling the next bout of work ("enabling the next effort" to describe it in Agile Modeling terms). After each release, pause briefly to bring the code and the design back in sync. However, only update what you need to - if a diagram is no longer needed, discard it (or to keep the managers happy, banish it to the bowels of your version control system).

Agile Planning: An agile project is one that is planning-driven rather than plan-driven. In the book, we give a chapter over to various agile planning techniques.

Responsible Ownership/collective Responsibility: This practice is deliberately left open to interpretation because you will need to tailor it according to the personalities, abilities and politics of the individuals on your team. The practice ranges from "extreme" collective ownership to the other extreme, individual ownership (a.k.a. specialization).

Optimizing Team Communication: This practice primarily means looking at the way your team is currently organized and moving team members around (both physically and in terms of their responsibilities) to improve the level and quality of communication.

Short Iterations: For planning purposes; dividing a release into fixed-size iterations. A project often finds a natural rhythm. For example, each week on Monday morning, you might have a team progress meeting to discuss issues from last week and sort out what each person will do this week. It pays to make the planning process follow the same rhythm, by dividing the project into fixed-size planning iterations of 1 week each. As with the other practices, this practice should be tailored to suit your own project's "natural rhythm" [10-14].

Note that several of these practices mention up-front design. For the purposes of defining these practices, our definition of "up-front design" is simply "design performed prior to coding", which can be broken down into fine-grained iterations. "Up-front design" is not the same as "Big Design Up-Front" (BDUF). BDUF could

mean 6+ months of up-front design without a single line of source code - in other words, a very bad thing! (Figure 2).

CONCLUSION

A need in Web engineering is a proactive approach in a real world, which is important and the duty of web developers is to apply a process, which can enforce and specify the applications and different work environments, suitably. ICONIX process is a flexible design with a high amount of knowledge and experimentation in web engineering and has some common characteristics from heavy and agile methods, which make it an adaptive process for a lot of different requirement of Web projects. ICONIX Process describes how to get from usage cases to code reliably, in as few steps as possible. It is a minimal object modeling process that is well suited to agile development and defines a practical core subset of the Unified Modeling Language (UML).The core subset of agile practices is Agile ICONIX. Agile ICONIX takes ICONIX Process (a minimal object-modeling process) and describes how to use it in an agile project. Agile ICONIX retains the core philosophy of ICONIX Process, which is that less is more. The less we have to do to safely achieve our goal, the quicker and more effectively, we will do it. The agile practices that Agile ICONIX uses are a combination of current agile thinking and some practices of its own, which create a core subset of agile practices the bare minimum that you would need to do in order to achieve the goals of agility.

REFERENCES

1. Ginige, A. and S. Murugesan, 2001. Web Engineering: An Introduction. University of Western Sydney, Australia, IEEE Press.
2. Murugesan, S. and A. Ginige, 2008. Web Engineering: Introduction and Perspectives. IGI Global, distributing in print or electronic forms without written permission of IGI Global is prohibited.
3. Deshpande, Y., S. Murugesan, A. Ginige, S. Hansen, D. Schwabe, M. Gaedke and B. White, 2002. Web Engineering. J. Web Eng., 1(1): 3-17.
4. Powell, T.A., 1998. Web site engineering: beyond web page design. Prentice-Hall, Upper Saddle River, EAN:978- 0136509202. Format: Paperback. Publisher: Prentice Hall PTR. Published.
5. McDonald, A. and R. Welland, 2005. Web Engineering in Practice. Department of Computing Science, The University, Glasgow G12 8QQ, Scotland.
6. Elbaum, S., S. Karre and G. Rothermel, 2003. Improving web application testing with user session data. Proceedings. 25th International Conference on Software Engineering, pp: 49-59.
7. Sommerville, I., 1996. Software engineering. 5th edition, Addison-Wesley, Harlow, UK.
8. Bleek, W., M. Jeenicke and R. Klischewski, 2002. Developing web-based applications through e-prototyping. Computer Software and Applications Conference, Proceedings of 26th Annual International.
9. Boehm, B., 1988. A spiral model of software development and enhancement", In R. Thayer, editor, Software Engineering Project Management IEEE Computer Society Press, pp: 128-142.
10. Zowghi, D. and V. Gervasi, 2001. Why is RE for web-based software development easier? In Proceedings of the 7th International Workshop on Requirements Engineering: Foundation for Software Quality.
11. Ricca, F. and P. Tonella, 2001. Analysis and testing web applications. In Proceedings of International Conference on Software Engineering, pp: 25-34.
12. Amirian, P. and A.A. Alesheikh, 2008. Implementation of a Geospatial Web service Using Web Services Technologies and Native XML Databases. Middle-East J. Sci. Res., 3(1): 36-48.
13. Farhoodi, M., M. Mahmoudi, A.M. Zare Bidoki, 2009. Query Expansion Using Persian Ontology Derived from Wikipedia. World Appl. Sci. J., 7(4): 410-417.
14. Motameni, H., A. Movaghar, I. Daneshfar, H. Nemat Zadeh and J. Bakhshi, 2008. Mapping to Convert Activity Diagram in Fuzzy UML to Fuzzy Petri Net. World Appl. Sci. J., 3 (3): 514-521.