International Journal for Research in Applied Science & Engineering Technology (IJRASET)

Requirements Analysis and Design in the Context of Various Software Development Approaches

Zille Subhan[1], Ali Tariq Bhatti[2]
[1]National University of Computer and Emerging Sciences (NUCES), Lahore Pakistan
[2]North Carolina A&T State University, Greensboro NC USA
[1]zsubhan@hotmail.com, [2]atbhatti@aggies.ncat.edu, ali_tariq302@hotmail.com

Abstract:-Software Engineering is a significant domain of Computer Science that deals with all the activities associated with the software development. Basically, a software is developed in a number of phases and each phase is closely linked to all other phases. The success and failure of a phase can heavily affect the other phases. Both requirements analysis and software design are significant phases of the software development process. In fact, the successful completion of a software development task heavily depends on the successful completion of these two phases. This paper is a comparative study of requirements engineering and design phases of different software development approaches. The major objective of this research is to present a detailed analysis of requirements and design phases of traditional and agile software development approaches.

I. INTRODUCTION

Software Engineering (SE) is a complete and isolated field of engineering that is concerned with the development and maintenance of software systems. Basically, the objective of the software engineering is to provide the software engineers with a wide variety of guidelines, processes, techniques, and principles through which they can develop dependable, affordable and efficient systems at the same time as satisfying all the requirements specified by the customers for those systems. In fact, the role of software engineering is becoming more and more important and critical with the emergence of huge and costly software systems and their implementations in safety-critical areas. It has a close relationship with a large number of other disciplines such as mathematics, physics and computer sciences [1].

Basically, software engineering provides a wide variety of principles, practices and tools for software engineers throughout the software development process. However, the software development process is followed through a software development life cycle. In this scenario, a software development life cycle is a conceptual framework or a map, which defines the stages of a software development process. Basically, a software development life cycle divides the software development process into a number of stages. Additionally, each stage of a software development life cycle is aimed at achieving a particular objective [20].

Up till now, a large number of software development process models have been developed taking into consideration the objectives of a particular software. For instance, waterfall process model, spiral model, iterative models, agile models and so on. Though, each software process model is based on a specific ideology. However, the basic objective of all the software process models is to support the software development process by dividing the development effort into a number of stages. In this scenario, each software development model divides the software development process into different stages such as: requirements analysis, system design, coding, testing, implementation and maintenance. In addition, the execution of these stages depends on the requirements of a particular software [11]. However, the execution of these stages flows in a sequence from upper to lower stage.

This paper presents a detailed analysis of two most important stages of a traditional software development life cycle, these stages are: requirements analysis and software design. This paper presents a detailed analysis of requirements analysis and software design. This paper also discusses how different software development life cycles process these two stages. This paper also compares these two stages of traditional software development life cycle with agile software methodologies. In addition, requirements analysis and software design processes of different software development methods will be also be discussed in this paper.

Traditional Software Development Life Cycle

Basically, software development lifecycle is a theoretical outline or process that demonstrates the phases or steps required for the development of a software application. In fact, this process outlines all the steps from preliminary feasibility study to installation and maintenance of a software application. In simple words, a software development life cycle is a map that guides the software

©IJRASET 2015: All Rights are Reserved

development team all the way through the system development process. It tells them what steps they should perform at a certain stage [21, 23, 27]. Figure1 demonstrates the stages of a traditional software development lifecycle:

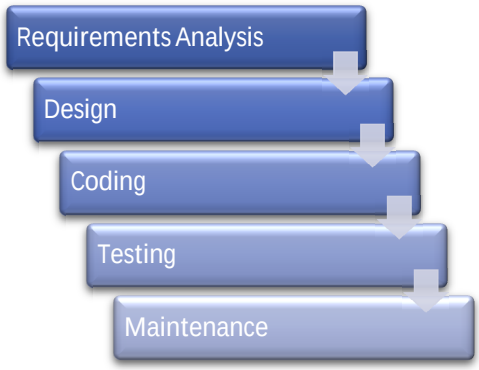

Figure 1 Some common stages of a software development life cycle, Image Source: [11]

Without a doubt, the software development life cycle is believed to be the most important aspect all the way through the software development process. Over the past decades, there have emerged a large number of methodologies that have been developed to tackle different challenges that happen all through software development. In this scenario, agile software development methodologies emerged as a response to a number of software development problems which were not addressed by traditional software development approaches. However, traditional software development methodology (waterfall model) is still being used for the development of software applications [9, 8].

As demonstrated in the figure1, a software development life cycle divides the entire software development effort into a number of stages. In this scenario, requirements analysis and system design are two most important processes of a software development life cycle. In fact, these two stages are believed to be the most important stages of almost every software development model.

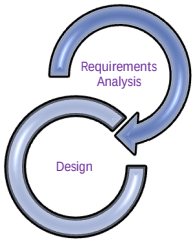

Figure 2 Requirements Analysis and Software Design: Two most important processes of a Software Development Life Cycle

II. REQUIREMENTS ANALYSIS

Before the software developers begin the development of a software application, they must identify and understand what actually they are going to develop and implement. Basically, requirements analysis is the process through which software developers gain necessary understanding of the intended system. Sometimes this requirements analysis causes several confusions among project stakeholders because of their different understanding level. However, the best way to reduce these confusions among stakeholders is to write down all the system requirements specified by different stakeholders in a document. This document is known as software requirements specification (SRS) document [3].

An SRS contains both functional and non-functional requirements. A functional requirement refers to the actual functionality of the system. For instance, in the context of a library management system, the system must allow the librarian to search for a book. On the other hand, a non-functional requirement does not refer to a functional requirement directly, but it has a close relationship with a functional requirement. For instance, in the context of a library management system, a librarian must be able to see all the search results within a second. The requirements analysis process is aimed at discovering all kinds of requirements. These requirements can include functional and non-functional requirements, performance requirements, and quality requirements and so on. Though, the basic objective of a requirement is to specify what the system should do and what the customer wants, however, it does not specify how these requirements will be accomplished or how a particular requirement will be programmed [3, 24].

©IJRASET 2015: All Rights are Reserved

A. The Importance Of Requirements Analysis

As shown in the figure 3, if a bug is not fixed in the initial stages of a software development life cycle it can cause a massive increase in the overall cost of the project. Hence, a poorly specified requirement or a poor requirement analysis can turn out to be very expensive for both software development firm and the customer. The research has shown that it takes only $1 to find and repair a defect at the stage of requirements analysis. However, if that initial defect is detected at the end of the software development life cycle, then the cost of its repair can raise to $100 [3].

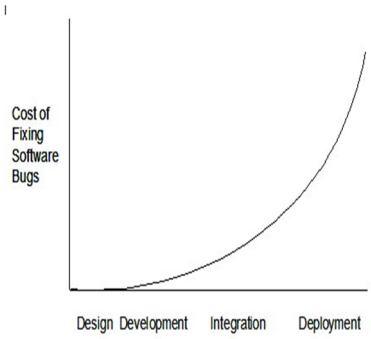

Figure 3 Cost of defects increases with each stage, Image Source: [13]

In addition to the increased cost of the software development poor requirements analysis sometimes causes customer's bad experience with the resulted software application. Though, software developers can attain a large number of advantages by detecting and repairing requirements at the requirements analysis stage. Despite this opportunity a large number of software development projects still manage to fail. One major reason of these failures is that in many cases, users of a system do not know what actually they want their system to do or they cannot express their needs in an appropriate manner causing serious issues among various project stakeholders. However, they start expressing their requirements when the software development progresses. As a result, they come with a new requirement each time they view the developed system [19, 3].

In addition, the failure of software development begins when a user states the system development requirements in a less effective way. In this way the system developed on the basis of such faulty system requirements become a disaster. In this scenario, the inability to state the user requirements can be due to lack of software working knowledge, poor understanding of software working or less effective business process information. However, mistakes in requirements recording can be done on both sides at client side or at the developer side [12, 10]. Agile software development methodologies are believed to be an excellent solution to these problems. These software development methodologies allow customers to change their requirements at any stage of the software development life cycle [19, 3] (a discussion on requirements analysis and design philosophies will be provided in the next sections).

At the end of requirements analysis, software engineers get a complete description of software's working properties, which demonstrate software's interface and their relationships with other components of the system, along with constraints and limitations under which these relationships should be established and maintained [16].

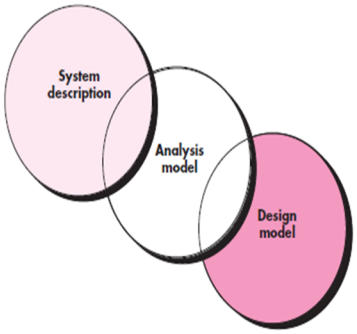

Figure 4 Requirement Model: A bridge between system description and design model, Image Source: [16]

B. Requirements Engineering

Requirements Engineering (RE) is a domain of software engineering which deals with all the aspects associated with requirements.

©IJRASET 2015: All Rights are Reserved

In fact, requirements engineering has emerged as a broad category of software engineering that initiates during the initial communication activity and remains until the design activity starts. However, software development firms can tailor requirements engineering activities according to the requirements of a project, the needs of the process, skills of software development team, and the product. The requirements engineering process initiates at the bases of software development project when software development firms initiates communication with project stakeholders in order to gather requirements. Requirements engineering is a complete discipline that involves a wide variety of process and offers suitable ways to understand what the project stakeholders want from the system, assess needs, discuss a sensible solution, analyze the feasibility, offer the solution explicitly, manage the requirements as they are converted into a functioning system and validate the specification [16].

C. Sources Of Requirements

In view of the fact that a software application is developed for customers, so they are believed to be a significant source of requirements. In fact, the entire requirements analysis process puts emphasis on collecting and analyzing the requirements from customers. Actually, the requirements analysis process itself involves a variety of stakeholders for instance, the actual users of the software application, the people paying for a software application, and the people nominated to specify the requirements. Instead of considering the requirements of only end-users, the software development firms should attempt to take into consideration the needs and wishes of almost all the associated stakeholders. During requirements analysis phase software development firms can face two major challenges. The first challenge has already been discussed above, and that is customers hardly know accurately at the beginning of the software development life cycle what they want the software development firm to develop. The second challenge is that, customers do not know all the requirements in the beginning of the software development life cycle. Hence, in many cases it is impossible for them to specify all the requirements in the beginning of software development [3].

Requirements can be gathered from a wide variety of sources. In this scenario, requirements are gathered not only from project stakeholders, but also from different books and documents. In fact, a large number of books have been written on requirements collection and gathering. These books describe processes and the extent to which requirements should be collected from project stakeholders, contrasted with other sources for instance published material. Basically, some software systems require working with mission-critical situations. In this scenario, it is essential for the software development firms to read books and literature to understand the working of different machines for which they are going to develop a software system. However, for general software applications, requirements can be gathered from project stakeholders [3, 24].

III. SOFTWARE DESIGN

A software design is the graphical representation of the requirements specification of the intended software application. Software design allows software developers to implement the specified requirements by graphically depicting them into various parts. In a software design, various components of a software application are illustrated using different notations. This design allows the software developers to understand the relationship between different components so that they could actually program the intended system. A software design is similar to a map of a building which is used by the contractor and engineers to build a building. Basically, a software design can contain two parts: first part can be a high-level design or application architecture, which is a critical component of the software development. On the other hand, other part is known as ‘‘detailed design.’’ Developing a detailed software design is believed to be a good practice for the reason that a detailed application design allows software engineers to study a detailed design with the purpose of detecting errors and defects before the actual implementation instead of inspecting simply the code. Though, the creation of a detailed software design can require some additional time and effort, but the benefit attained through a completely detailed design can compensate the loss of time and effort [3; 24].

Software design appears as initial iteration when the requirements engineering process reaches to its end. The software design allows software engineers to put into practice a wide variety of thoughts, philosophies, and practices that result in the development of a superior product or system. The software design allows software designers to build an outline of application for software engineers that they will use to implement all customer’s specifications appropriately. In this scenario, software design provides an excellent support throughout the software development process. In order to appropriately meet the requirements of project stakeholders, software designers must search through a wide variety of design changes. The software design process begins with the creation of a high-level design and moves ahead with the division of that high-level design into more detailed patterns [16].

A. The Goals Of Software Design

©IJRASET 2015: All Rights are Reserved

One of the basic objectives of software design is to allow software engineers to determine the accuracy of the system. It is the best way to identify whether the software requirements are being met or not. Also, a second major goal of software design is to achieve flexibility of software development because an efficient software design takes into consideration changes in the requirements. A change is inevitable and it keeps occurring throughout the software development life cycle and even after the system implementation. In this scenario, the software design must be flexible enough to accommodate any future changes. However, a software design can be made more understandable by increasing the level of its modularity. In fact, software modularity is believed to be a key to understandability. A software is believed to be modular if it can be divided into different modules all of which can be separately addressed and managed [3; 24].

In view of the fact that a modular application is isolated and easier to understand, hence a module can be replaced or modified without affecting other modules. Additionally, a modular application can be easily planned, developed, modified, documented, and tested. A modular software design allows software developers to divide and complete the software development by working on different modules. A software design works as a communicator between requirements specification and the implementation because a designer converts software requirements into an understandable format for the software developers in order that they could be actually implemented. Basically, a software development project involves a number of people all the way through its lifetime and a design is used for the communication among these people. In this scenario, if a software design is not clear, it can cause serious challenges [3, 24].

In addition, software engineers can also improve the quality of design by making it more simple and easy to understand. They can achieve this level of understandability and easiness by breaking high-level design into smaller manageable components. A good software designer always attempts to make a design document that depicts accurate description of the system (i.e. how different components of the system communicate and how the system works). Additionally, software design also involves a variety of other aspects such as coupling and cohesion. In this scenario, coupling is the extent to which different components or modules of a system interact with each other. Cohesion is a degree of how independent a component is. The basic objective is not to divide a software application into a number of modules, but also their associated elements and relationships should be in the same module. Coupling refers to the extent to which different modules of a system communicate with other modules. The high level of coupling makes the software design difficult to understand. Hence, in order to modularize a software design effectively, software designers need to minimize coupling and maximize cohesion. This ideology allows software designers to capture large and complicated operations into simpler ones [3, 24].

IV. TRADITIONAL SDLC VS. AGILE SDLC

The traditional software development life cycle is based on a rigid set of steps with a documented sign-off on the completion of each one. In this scenario, a comprehensive and in-depth requirement analysis is performed in an attempt to determine the system needs and requirements to form a Software Requirements Specification (SRS) document. In addition, clients are forced to "sign-off" on the requirement specification document before system development proceeds to the next step. Moreover, the other steps of the system development lifecycle include a comprehensive system design and actual implementation and testing [5, 22, 16].

Actually, the traditional software development lifecycle is composed of a rigid set of development stages that are aligned with each other in a less flexible way. As a result, the entire software development process faces several serious issues and concerns regarding software development. For example, what if the design stage of a software development process uncovers requirements that are strictly impracticable or very expensive to establish or develop? What if issues and errors found in the software requirements and design stages are encountered in implementation phase? In addition, the time duration between preliminary investigation and testing typically spread over several months. What if basic needs, requirements or priorities of a client are changed or system users understand they ignored critical needs all through the software development analysis phase? In fact, there are numerous issues and concerns which make the traditional software development lifecycle a more rigid and inflexible process. In addition, there can be numerous other issues which can create serious problems for entire software development process and can result in project failure or do not able to convene the user’s expectations when deployed [5, 22, 16].

Moreover, when software development firms look for a better approach to software development, they see Agile as one of the best approaches. In fact, agile software development techniques came out of the real-life development experiences of experienced and skilled software professionals who had practiced the main issues and limitations of customary waterfall software development on the project following project. In addition, the techniques promoted by agile development are a straight response to the matters and

©IJRASET 2015: All Rights are Reserved

issues connected with customary software development, both in terms of general viewpoints and precise processes [26, 2, 7].

Furthermore, the agile software development methodology is in its simplest appearance, presents a lightweight structure for facilitating system development teams and also offering frequently developing technical and functional setting; pays deep attention to release business value quickly. Hence, due to its focus on the development side as well as its associated advantages, businesses are able to considerably minimize the common risk linked with software development [26, 2, 7].

Especially, agile development speeds-up the release of early business value, and throughout a system development process of continuous feedback and planning, is capable to make sure that worth is long-lasting to be maximized all through the system development process. Due to this quick and iterative feedback and planning loop, system development teams are capable of bringing into line the delivered software with preferred business requirements, simply adjusting to transforming system development requirements all over the software development process. In addition, by calculating and determining status based on the irrefutable fact of testing software, working, a great deal of more precise visibility into the real development of projects is achievable [26, 2; 7].

V. FEATURES OF AGILE REQUIREMENTS ENGINEERING AND SYSTEM DESIGN

A. Documentation

Agile software development methodologies hardly have complete and dependable requirements documentation. Though, some agile software development methodologies comprise a kind of documentation or suggest the adoption of a requirements document (such as Scrum, DSDM, and Crystal) however the decision of the key extend and contents is left to the development team and not discussed thoroughly. Hence, due to lack of documentation agile software development process face long-term problems, however, lack of documentation increases the speed of actual software development. The documentations can answer his questions if he simply read through the entire documents. Additionally, asking questions to other team members will surely delay work, due to the time required to clarify a large project. However, these long term problems are reduced as agile methods often generate compact and clean code [15].

B. Non-functional Requirements

Agile software development methodologies do not effectively deal with non-functional requirements. In this scenario, systems users describing the system requirements what they want the system to do usually do not take into consideration associated non-functional requirements such as maintainability, resources, safety, performance and portability. In fact, in some cases, non-functional requirements can be related to functional requirements and are hard to explain. However, the majority of non-functional requirements should be identified in development for the reason that they can have an effect on the selection of programming language, database or operating system [15].

C. Requirements Management

Requirements management is the process of managing changes to the software requirements. Since agile software development methodologies do not focus on appropriate documentation hence it is difficult to keep track to changes in requirements. Requirements management should be a formal process, but agile software development methodologies do not follow it formally [15].

D. Object-Oriented System Analysis And Design

In the field of software development, Object-Oriented Analysis and Design (OOAD) is a software development approach based on the idea that a software application should be developed using a set of recyclable elements known as objects. In this scenario, in place of dividing data and processes applied on data in the organized way, objects comprise the characteristics of both. Basically, the agile software development is an excellent response to the thought that it is nearly impossible to describe, front side, all the functionality of a software application. In view of the fact that functional requirements always change, as well as the application environment can also change, which can require further changes to the programming concepts and coding ideas [25]

Without a doubt, in many software development projects the development of a database is also a part of overall software development which normally requires the implementation of refactoring of the database code (for instance mapping, schema and data) similar to other software code. The use of object oriented concepts in the database improves the efficiency of database development through re-programming its access layers, and moving the existing stored data to the new database schema regularly.

©IJRASET 2015: All Rights are Reserved

In addition, the overall worth of agile software development is sensibly well implicit. On the other hand, it is problematic to find an ultimate worth of the investment through object determination and agile development. In this scenario, the execution of somewhat shortened techniques offers some standpoint on the possible worth of combining the development of a database into the agile development methodology [25].

There are many differences between object oriented system analysis and design and traditional approach to system design. It is believed that the traditional approach to system design is very effective when it is used for the development of traditional projects, for instance projects that use procedural programming such as C language. On the other hand, the object-oriented design is used to develop software applications that use object-oriented programming languages such as Java and C++. In this scenario, this system design approach allows software development team to emphasize on decomposition of huge and complex techniques and algorithms into smaller ones. Usually, traditional software development approach is used to develop large scale software projects and requires large duration. On the other hand, object oriented design requires more time than traditional approach and as a result the development cost of the system also increases [14].

In addition, traditional approach to system development is based on routine practices such as requirements, analysis, design, development, and testing and deployment. On the other hand, object oriented design is based on unified modeling languages and involves the development of various diagrams such as class diagram, use case, development diagram, communication diagram and sequence diagram. Normally, the decision regarding the selection of traditional software development approach is made on the basis of the type and size of projects. On the other hand, object oriented design is used when the software development team is experienced to deal with the complexity of the objects [14].

If object oriented design is used to develop real-time systems it will ensure the development of, more reliable, secure, and easy to maintain code because the object oriented design approach focuses on determining objects and their activities in place of using functional decomposition of the system [17].

VI. REQUIREMENTS ANALYSIS AND SYSTEM DESIGN IN THE CONTEXT OF OTHER SOFTWARE DEVELOPMENT METHODOLOGIES

A. Scrum Methodology

Scrum is very well-known agile software development methodology. Scrum software development starts with the customer's story. This story can be anything which a customer uses to describe their requirements. For instance, end-user needs to add a new contact in address book, in order that he can interact with the person in the future by email or postal mail. These stories are further explored by the software development team. In this scenario, the customer does not provide the full, thorough clarification of things and requirements that need to be completed in a project, however the burden shifts on the shoulders of software development team for the reason that the software development team better understands how to solve the problem they are provided with. Hence, in the Scrum software development a sprint planning meeting involves the discussion on the required results. In the Scrum methodology, the software development process is measured through a series of sprints. Normally, these sprints are delivered within a duration of two weeks. In this scenario, a meeting is conducted in the beginning of each sprint, in which project team determines what requirements they can complete, and then develop a sprint backlog on the basis of this analysis as well as a list of the actions needs to be carried out during the sprint [4, 18].

B. DSDM (Dynamic System Development Methodology)

Dynamic Software Development Methodology (DSDM) looks a great deal similar to the eXtreme Programming technique, however, in this methodology the concentration is paid to the design of a software product. This methodology suggests that software development teams should complete just adequate design up front at the initiation stage of software development with the purpose of determining and clarifying the structure of the complete solution as well as to develop an agile plan for delivery of the project. For instance, a client wants the software development to develop an ATM software, it is a good practice to develop some part of the design in the early stage of the software development. In this scenario, software development team can better understand the requirements and feasibility of software. This design can include only some basic functionality of the system. Hence, this practice starts up the basis for effective software development and delivery. In addition, the design phase of the DSDM differs with the analysis and design phase of the traditional waterfall software development because in the traditional software development a detailed design is completed while in DSDM only some details are covered (Craddock, Roberts, Richards, Tudor, & Godwin, 2012).

C. Lean Software Development Methodology

©IJRASET 2015: All Rights are Reserved

Basically, the lean methodology is widely used in production industries. However, with the passage of time its usage has been increased for the software development as well. This methodology is aimed at reducing the waste in the software development. This is done by sensibly dividing projects into small chunks with the purpose of identifying what components or requirements of a software system are inessential and can be eliminated. For instance, in case of an ATM software, an initial analysis can be carried out to determine what components can be excluded such as additional buttons or graphics but basic functionality should not be compromised. After that in light of the requirements the size of a team is determined and desired resources are provided to the team. After that the project is broken into smaller components to make the management simple. For instance, in case of the ATM project first component can be "view credit" and second can be cash withdraw and so on [6].

VII. CONCLUSION

Both requirements analysis and software design are the most important stages of a software development life cycle. If a defect or mistake is not detected at the requirements analysis stage it can cause serious challenges for the entire software development. In view of the fact that all the stages of a software development life cycle flows in a sequence, hence the success of a particular stage heavily depends on the successful completion of a preceding stage. In this scenario, if the requirements analysis stage is not completed effectively it will have a significant effect on the design stage. One major drawback of traditional requirements engineering process is that it does not provide any support for changes. For instance, if a change occurs during implementation stage the software development team has to complete the entire software development life cycle in order to accommodate change. In order to deal with the issues of traditional life cycle, a new software development methodology emerged that is known as Agile software development methodology. It provides an excellent support for changes occurring throughout a software development life cycle. This paper has presented a detailed analysis of requirements analysis and software design with respect to traditional software development life cycle as well as agile software development. This paper has also discussed the concepts of requirements analysis and software design with respect to other software development methodologies. However, both requirements analysis and software design are significant attribute of all the software development approaches.

REFERENCES

[1] ACM, Inc. (2006). Software Engineering. Retrieved from ACM.org:http://computingcareers.acm.org/?page_id=12
[2] Bender RBT Inc. (2003). Systems Development Life Cycle: Objectives and Requirements. Retrieved July 22, 2014, from http://www.benderrbt.com/Bender-SDLC.pdf
[3] Braude, E. J., & Bernstein, M. E. (2010). Software Engineering: Modern Approaches, 2nd Edition. New York: Wiley & Sons, Inc.
[4] Cohn, M. (2012). Scrum. Retrieved from Mountain Goat Software: http://www.mountaingoatsoftware.com/agile/scrum
[5] DocStoc. (2009, June 25). Software Development Life Cycle Models. Retrieved July 19, 2014, from http://www.docstoc.com/docs/7726041/Software-Development-Life-Cycle-Models
[6] Duggan, P. (2013, February 07). Going with the Flow – The Lean Approach to Successful Project Management. Retrieved from Backbase Blog: http://blog.backbase.com/2935/going-with-the-flow-the-lean-approach-to-successful-project-management/
[7] Erdil, K., Finn, E., Keating, K., Meattle, J., Park, S., & Yoon, D. (2003, December 16). Software Maintenance As Part of the Software Life Cycle. Retrieved July 23, 2014, from http://hepguru.com/maintenance/Final_121603_v6.pdf
[8] Hughes, B., & Cotterell, M. (2002). Software Project Management. Glasgow: McGRAW-HILL Publishing.
[9] Jiang, L., & Eberlein, A. (2008). Towards A Framework for Understanding the Relationships between Classical Software Engineering and Agile Methodologies. APOS '08 Proceedings of the 2008 international workshop on Scrutinizing agile practices or shoot-out at the agile corral (pp. 9-14). New York: ACM.
[10] Kaur, R., & Sengupta, J. (2011). Software Process Models and Analysis on Failure of Software Development Projects. International Journal of Scientific & Engineering Research, Volume 2 Issue 2, pp.1-4.
[11] Kumar, N., Zadgaonkar, A. S., & Shukla, A. (2013). Evolving a New Software Development Life Cycle Model SDLC-2013 with Client Satisfaction. International Journal of Soft Computing and Engineering (IJSCE), Volume 3 Issue 1, pp.216-221.
[12] May, L. J. (1997). Major Causes of Software Project Failures. Retrieved from http://info.psu.edu.sa/psu/cis/biq/se501/a/a1/majorcausesofsoftwareprojectfailures.pdf
[13] Michael, C., Wyk, K. v., & Radosevich, W. (2005, September 23). Risk-Based and Functional Security Testing. Retrieved from US-CERT.Gov: https://buildsecurityin.us-cert.gov/articles/best-practices/security-testing/risk-based-and-functional-security-testing
[14] Munassar, N. M., & Govardhan, A. (2011). Comparison between Traditional Approach and Object-Oriented Approach in Software Engineering Development. (IJACSA) International Journal of Advanced Computer Science and Applications, Volume 2 Issue 6, pp. 70-76.
[15] Paetsch, F., Eberlein, A., & Maurer, F. (2003). Requirements Engineering and Agile Software Development. Proceedings of the Twelfth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03) (pp. 1-6). IEEE.
[16] Pressman, R. S. (2010). Software Engineering : A Practitioner's Approach, 7th Edition. New York: The McGraw-Hill Companies, Inc.
[17] Quillin, M. J. (2001, November 25). Object Oriented Analysis and Design: What is it? How Does it Work? Why is it used? Retrieved from

©IJRASET 2015: All Rights are Reserved

http://www.umsl.edu/~sauterv/analysis/488_f01_papers/quillin.htm#III.%20Object-Oriented%20Analysis%20vs.%20Traditional%20Methods
[18] Rees, D. (2013). Agile Project Management with Scrum. Retrieved from ITWales.com:http://www.itwales.com/998612.htm
[19] Rehman, I. U., ullah, S., Rauf, A., & Shahid, A. A. (2010). Scope management in agile versus traditional software development methods. NSEC '10 Proceedings of the 2010 National Software Engineering Conference (p. 10). ACM.
[20] Rouse, M. (2009, May). Systems Development Life Cycle (SDLC). Retrieved from TechTarget.com: http://searchsoftwarequality.techtarget.com/definition/systems-development-life-cycle
[21] Ruparelia, N. B. (2010). Software development lifecycle models. ACM SIGSOFT Software Engineering Notes, Volume 35 Issue 3, pp. 8-13.
[22] Shelly, G. B., & Rosenblatt, H. J. (2009). System Analysis and Design, 8th Edition. Cengage Learning.
[23] Smuts, H., Merwe, A. v., Kotze, P., & Loock, M. (2010). Critical success factors for information systems outsourcing management: a software development lifecycle view. SAICSIT '10 Proceedings of the 2010 Annual Research Conference of the South African Institute of Computer Scientists and Information Technologists (pp. 304-313). New York: ACM.
[24] Sommerville, I. (2011). Software Engineering, 9th Edition. Boston: Addison-Wesley.
[25] Versant Corporation. (2009). OBJECT PERSISTENCE AND AGILE SOFTWARE DEVELOPMENT. Redwood City, CA: Versant Corporation.
[26] VersionOne, Inc. (2012). Benefits of Agile Development. Retrieved July 26, 2014,from http://www.versionone.com/Agile101/Agile_Benefits.asp
[27] Yeo, A. W. (2010). Global-software development lifecycle: an exploratory study. CHI '01 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (pp. 104-111). New York: ACM.

BIOGRAPHY

Ali Tariq Bhatti received his Associate degree in Information System Security (Highest Honors) from Rockingham Community College, NC USA, B.Sc. in Software engineering (Honors) from UET Taxila, Pakistan, M.Sc in Electrical engineering (Honors) from North Carolina A&T State University, NC USA, and currently pursuing PhD in Electrical engineering from North Carolina A&T State University. Working as a researcher in campus and working off-campus too. His area of interests and current research includes Coding Algorithm, Networking Security, Mobile Telecommunication, Biosensors, Genetic Algorithm, Swarm Algorithm, Health, Bioinformatics, Systems Biology, Control system, Power, Software development, Software Quality Assurance, Communication, and Signal Processing. For more information, contact **Ali Tariq Bhatti** at ali_tariq302@hotmail.com.

©IJRASET 2015: All Rights are Reserved