Yet another Set of Requirement Metrics for Software Projects

Shahid Iqbal and M. Naeem Ahmed Khan

Shaheed Zulfiqar Ali Bhutto Institute of Science and Technology (SZABIST)
Islamabad, Pakistan
shahid.Iqbal.skt@gmail.com, mnak2010@gmail.com

Abstract

Software project management has emerged as a new discipline with wide-ranging ideas and across-the-board insights for effectively managing key areas of software projects. The remarkable work of the software project managers, professionals and researchers across the globe have resulted in substantial improvements in this field. Likewise, software project failure rate has decreased considerably due to the use of effective software project management tools and techniques by the software houses. Now more efficient, robust and quantitative measures are being practiced in the areas of software requirement gathering, analysis, design, architecture, development, quality assurance, integration, deployment and support. A number of metrics are used by the requirement engineers, system analysts, software engineers, team leads, software project managers and other professionals to successfully manage, execute and complete the software projects. As the software industry moves towards a more mature state, the need for employing more effective tools, techniques and benchmarks for managing software projects has become indispensable to minimize the negative risk factors and improved adherence to quality assurance. Particularly, requirement metrics are useful in identifying risks of a project by locating errors in the requirements document. These metrics validate the gathered requirements against the actual requirements by evaluating whether the requirements are complete or not. A range of metrics are used for measuring the requirements e.g., volatility metrics check changes in the requirements, traceability evaluates links among the requirements within a document and requirements completeness metrics verifies whether the specified requirements are complete or not. Multiple metrics are recommended to be used to assess the health of a software project to ensure overall quality as a single metric cannot suffice. In this paper, we explore how different metrics relating to different areas of software project, especially in requirement engineering, can be useful to manage the software projects for knowledgeably. The focus of this research study is to evaluate and highlight the importance of various performance metrics and propose additional metrics for requirement gathering and management.

Keywords*: Software Project Management, Project Performance Metrics, Requirement Management, Metrics for Quality Management, Requirement Engineering*

1. Introduction

Software projects have encountered major difficulties for a long time due to use of poor metrics by the project managers to gauge performance of software projects. Most of the causes of project failures, cost and schedule overrun are often traced back to requirement engineering issues such as requirement creep, poorly documented requirements, requirements that are impossible to comply with (inverse requirements) and requirements that remained futile to meet the user needs. Requirement management best practices help improve customer satisfaction, lower the system development costs and increase project success rate. Software project metrics aim at measuring and evaluating performance characteristics of a software system and can be employed to identify product defects and assess software quality [13]. Likewise, requirement metrics, when incorporated in requirement gathering and elicitation process, assist in analyzing the quality of requirements as well as identifying the reasons for software reengineering or malfunction. Requirement metrics also help define the output measures of the software processes.

In this paper, we highlight some of the core metrics, skills and processes which can help project managers and key stakeholders to effectively manage and monitor IT projects. Particularly, our focus is on the metrics which are helpful for software requirement management. We also propose an additional set of metrics for requirement management which can be helpful for better evaluation of software projects.

1.1. Defining Metrics

Very true to famous quote of a management consultant Peter Drucker: “If you can’t measure it, you can’t manage it”, if a project manager and team members are not able to precisely measure what they are going to do then it would not be possible for them to effectively manage and improve the performance of a project. The success of a software project has always been the primary goal of the software industry, however the metrics that help to measure the success or failure of a project are very diverse and these metric hardly have a good deal in commonalities [4]. Using appropriate performance metrics is vital to correctly manage a software project, as otherwise it would be difficult for a project manager to ensure that project is progressing according to the plan. Metrics are also helpful to determine current status of a project and evaluate its health. Early identification of risks associated with different project tasks provides an opportunity to focus efforts on the most critical project errands. Metrics to evaluate project status are broadly categorized into requirements, risks, source code, tests, defects and documents. However, no single metric is representative of project’s status, rather a combination of metrics portray a comprehensible picture about of the health of a project [1].

1.2. Metrics to Manage Efforts

An effort is defined as: *“an exertion of strength or power, whether physical or mental, in performing an act or aiming at an object; more or less strenuous endeavour; struggle directed to the accomplishment of an object”* [16]. In general, an effort is considered as the total amount of time spent on accomplishing a task that results in developing a product or service [4]. The planned amount of time required for accomplishing a work is called *‘Planned Effort’* and the actual amount of time spent is called the *‘Actual Effort’*. Effort can be measured in hours, days or weeks depending on the specifications and essentials of the project.

1.3. Productivity Metrics

Productivity is defined as: *“the efficiency with which output is produced by a given set of inputs”* [15]. Productivity is commonly measured by the ratio of output to input. An increase in the ratio indicates an increase in productivity and conversely, decrease in the ratio indicates decline in productivity. Alternatively, productivity is characterized as the number of *‘simple tasks’* delivered per day [4]. Since the definition of a ‘*simple task’* raises a lot of ambiguity, therefore, a better explanation of productivity can be described as the amount of time required by a resource to deliver an output within five hours [4]. Hence, by taking into account the eight working hours per day, the productivity can be calculated by using the following ratio:

$$Productivity = ((Planned\ Effort / 5) / Actual\ Efforts) \times 8$$

1.4. Quality Metrics

According to Gilmore [4], *"Quality is the degree to which a specific product conforms to a design or specification"*. Quality is accounted for throughout the project lifecycle as the defects found at a later stage cause severe impact on the project. That's why, quality is usually described as the number of severe, medium or low defects delivered throughout the lifetime of the project [4].

1.5. Effective Measurement

Asthana and Olivieri [8] describe the following four key components of an effective measurement process.

a. *Defining the software development issues clearly and the software measures (data elements) that support insight to issues.*

b. *Processing the software data into graphs and tabular reports (indicator) that support the analysis of issues.*

c. *Analysing the indicators to provide an insight into the issues.*

d. *Using the results to implement improvements and identify new issues and questions.*

1.6. Organization of this Paper

This paper is organized into six sections; and this section, being the introductory section, throws light on the significance of project metrics. The second section provides background information related to the expediency of the performance metrics - with particular reference to the requirement metrics - and is primarily based on the literature survey conducted during the entire course of this research. A descriptive instance of software requirement engineering process model is summarized in the third section. The anthology of our proposed requirement metrics is outlined in the next section. An anecdote about the effects of metrics on requirement engineering process in provided in the fifth section and finally we conclude in the last section.

2. Background and Literature Survey

In the recent past, the effective uses of software project management tools and techniques have improved the rate of success of software projects to a significant extent. In this perspective, Software Project Management (SPM) has emerged as a new discipline that encompasses wide-ranging ideas and across-the-board methodologies to efficiently manage all the fundamental knowledge areas of software projects. A number of performance metrics and techniques are being used to ensure the quality of software requirement elicitation, design, development and deployment. The performance metrics are now used throughout the software development lifecycle - from specification to maintenance and support. These metrics can also be used to evaluate the status of a software development project and measure different attributes of a software product that give valuable insight about the stability of software code and fitness for beta testing and deployment [1].

Requirement management has evolved as an important aspect of the software development process and is an area of active research. In a broader perspective, requirements management involves information storage, organization, traceability, analysis, visualization, change/configuration management and documentation. One aspect of good requirements management practices is to measure and collect requirement metrics, rather than relying on just gut feeling. Though requirements metrics can help in understanding and improving the requirement management process, however, implementing a metrics framework is quite challenging as it may face obstacles from stakeholders.

Software metrics range from internal product attributes such as size complexity and modularity to external process attribute such as effort, productivity and reliability [2]. Many professionals in the software industry are still practicing old approaches for managing software project as they are not acquiescent to the use performance metrics, or they might find these techniques overwhelming. Still, loads of practitioners find it ominous and daunting to apply software management best practices in their own setups [2].

Managing a software project is a complex task and is further complicated due to continued increase in the size and complexity of the software-intensive systems [6]. Project managers are required to manage each and every step of the project execution process so that tasks may be completed within the agreed parameters. Employing progress and performance metrics is a way to help a project team to timely achieve its targets and milestones [4]. Project managers should also be aware of the performance concerns and issues of a software project as software performance is an important non-functional attribute of software systems for developing quality software [5].

Managing the development of information system projects in an effective manner has become a practical problem in academic research [7]. SPM entails the management of all the issues involved in the development of a software project like scope and objectives identification, evaluation, planning, project development approaches, software development efforts and cost estimation, activity planning, monitoring and control, risk management, procurement, resource allocation and quality control, managing contracts and project teams [6]. A project manager should dispense more emphasis on managing triple constraints (i.e., project scope, time and cost) as sponsors and top management are mostly interested in these areas. These triple constraints also refer to the fact that the failure of software project is mostly attributed to project not delivered on *time* or that it does not meet the key requirements *(scope)* and therefore has *cost* overrun implications [6]. Consequently, the size, complexity and strategic importance of information systems undergoing development process require stringent measures to ensure success of these projects.

Although several improvements techniques for software project management have been proposed but still the ratio of software project failure is higher as compared to the other commercial, construction and industrial projects. Software projects possess weaknesses and strengths of different nature; for instance, some of the strengths associated with software projects include flexibility, ease of creating backups, scalability, replication and reusability of components; while a few weaknesses include invisibility, complexity, difficulty to assign additional manpower to delayed projects and the need for regular upgrades [3].

In this study, we have endeavoured to explorer multiple techniques, metrics and artefacts to effectively manage software projects. We attempt to suggest some performance metrics to effectively manage the requirement gathering phase - the most important and initial phase - of a software project.

3. Software Requirement Engineering and Process Model

Requirements are the driving force behind the development of a software project. Each phase in software development like analysis, design and testing etc., directly or indirectly, depends on the requirements. Software requirement engineering is a process to measure the degree to which a software system meets the purpose for which it was planned by identifying needs of all the stakeholders of the system; and then documenting these requirements in a form that is amenable to analysis, communication and subsequent implementation.

Requirements constitute the earliest phase of the software development life cycle. They are statements of intended needs which convey understanding about a desired result that can be independent of its actual realization. The main objective of the requirements engineering process is to provide an overview of what is required and defining a clear, consistent, precise and unambiguous statement of the problem. Familiarity with the system's environment is imperative to get the unambiguous requirements. If the system's environment is not well understood then extracting the actual requirements of the system will become more difficult. Therefore, more familiarity with the environment will definitely decrease the complexity of the process. A requirement error found at the early stages e.g., during requirement gathering or elicitation phase, comparatively costs much lower if found at later stages of software development or deployment.

There could be many reasons behind the success of requirement engineering processes; however, a high level of consideration for the requirements elicitation, analysis, specification, validation and management can make the requirement gathering process more accurate and successful. Each of these techniques is further divided into subtasks to extract and manage the requirements in more efficient way e.g., the requirements elicitation process can be accomplished through the subtasks:- Question and Answer Method; Customer Interviews; Brainstorming and Idea Reduction; Storyboards; Prototyping; Questionnaires ;Use Cases; and, Requirements Management.

Since a requirement describes a capability that the system must provide; therefore, it is either derived directly from user needs or is stated in a contract, standards, specifications or other formally used documents [11]. Examples of requirements include: inputs to the system; outputs from the system; functions of the system; attributes of the system; and, attributes of the system environment.

In the software development process model, developing and handing over the software within the agreed schedule is an important activity [9]. Software project manager should monitor the project status during the whole project lifecycle; and in this regard, selection of a suitable process model can increase the success rate of the project. Software process model can provide the necessary key information for risk assessment and embraces more powerful expression capability than traditional risk analysis methods [14]. The requirement analysis, design, coding, testing and maintenance phases of a process model entail measuring and monitoring the flow of the activities taken by the team [10]. A comparison of industry processes and the software development processes as proposed by Lin *et al.* [12] is provided in Table I.

Table 1. Industry and Software Development Processes [12]

Industry Process	Software Development Process
Design	Requirement analysis
	Architecture/Designing
	Coding
	Testing
	Release
Sale	Sale
Maintenance	Maintenance

4. Proposed Requirement Metrics Anthology

Requirements metrics are useful in identifying risks of a project by identifying errors in requirements document. These metrics validate the written requirements against actual requirements and evaluate whether the requirements are complete or not. There are many metrics used for measuring the requirements e.g., volatility metrics checks the changes of the requirements, traceability evaluates links between requirements to requirements within a document and requirements completeness metrics checks whether the specified requirements are complete or not. Single metric cannot ensure overall quality therefore multiple metrics should be used for measurement.

Based on our experience of gathering requirements for a number of software projects, we propose a set of additional metrics to evaluate the effectiveness of the requirements. Our proposed requirement metrics are mentioned in Table II. These metrics can be used during the entire course of action of requirement engineering process. These metrics offer a blend of quantitative measures to judge the performance and quality of the requirement gathering and elicitation phase. The criterion (standard) mentioned against each of these metrics is a recommended level to confirm the validity of a requirement; however, it can be slightly changed depending on the nature of the project and needs of the organization.

In addition to the aforesaid assortment of requirement metrics, we propose the following steps to be followed by the software engineers or requirement engineers in order to gather the software requirements in a more accurate and planned manner.

a. Formulate a team of experts in which majority of members have the relevant domain knowledge.

b. List down all the stakeholders and their role in the project.

c. Clearly express the data collection procedures.

d. Make a prototype of the measurement process.

e. Prepare list of potential quality requirements for the system.

f. Setup a discussion forum for the requirement engineering team.

g. Reconcile the quality requirements list for future validation of the same.

h. Perform a cost-benefit analysis of all the requirements.

i. Organize a meeting with the stakeholders for further verification and refinement of the process.

j. Keep track of changes in the requirements.

k. In case a new requirement pop ups then perform step “e” onward.

l. Interpret the performance metric results.

m. Determine the expected software quality by making software quality predictions.

n. Ensure compliance of the software artefacts with the requirements.

o. Document the performance metric results.

Table II. Proposed Metrics to Ensure Quality in Requirement Engineering Process

#	Name of Metric	Description	Formula	Criterion
1	Uniqueness	The *uniqueness* metric can be used to obtain percentage of the requirements that have been uniquely explained by all the reviewers.	$\frac{R_i}{R_t} \times 100$ where, R_i = Requirements with distinctive explanation R_t = Total requirements	Based on our experience of executing software projects, we recommend that if the result of this metric is $< 95\%$ then it would indicate that requirements are ambiguous and hence need rework.
2	Correctness	*Correctness* metric calculates the percentage of all the requirements that have been correctly validated.	$\frac{R_c}{R_t} \times 100$ R_c = Requirements having the same interpretation R_t = Total requirements	If the output of this metric is $< 80\%$ then it is assumed that the correctness cannot be verified; whereas, accuracy of results being $\geq 80\%$ is acceptable.
3	Changed Requirements	This metric counts percentage of the requirements changed during each phase of system development.	$\frac{R_{ch}}{R_c} \times 100$ R_{ch} = Total requirements to be changed R_c = Total correct requirements	The result of this metric describes the percentage of the total requirements to be changed.
4	Misinterpreted Requirements	This metric is used to check percentages of the requirements that have been misinterpreted during the elicitation phase.	$\frac{R_c}{R_t} \times 100$ R_c = Total requirements to be changed R_t = Total requirements	If result is $\leq 5\%$, then the new changes should be accommodated into the requirement document and if percentage is $> 5\%$ then elicitation process needs to be redone for confirmation of the same.
5	Understandable Requirements	This metric is used to count the number of requirements that are understandable to all the users and reviewers.	$\frac{R_u}{R_t} \times 100$ R_u = Requirements understood by the users R_t = Total requirements	The acceptable criterion for this metric is that all the requirements should be understood by the users i.e., 100% result of this metric is deemed to be acceptable.
6	Modifiable	This metric counts the number of requirements that are required to be changed or modified.	$\frac{R_m}{R_t} \times 100$ R_m = Requirements modified R_t = Total requirements	The outcome of this metric is percentage of requirements modified throughout the system development lifecycle.
7	Traced	Measuring level of tracing the requirements and any changes is difficult because this activity spans throughout the system development lifecycle.	$\frac{R_{tr}}{R_t} \times 100$ R_{tr} = Requirements traced R_t = Total requirements	The result of this metric is percentage of the requirements traced throughout system development lifecycle.
8	Requirement Testing	This metric tracks testing of the requirements.	$\frac{R_{ts}}{R_t} \times 100$ R_{ts} = Requirements tested R_t = Total requirements	This metric exhibits percentage of the requirements that have been successfully tested.
9	SRS Quality	This metric is used to check the correctness of SRS.	$\frac{R_{ts} - (R_{ef} + R_{ed} + R_{em})}{R_t} \times 100$ R_{ef} = Errors found in SRS R_{ed} = Errors deleted from SRS R_{em} = Errors modified in SRS R_t = Total requirements	This metric illustrates percentage of the correct requirements found in the SRS document.

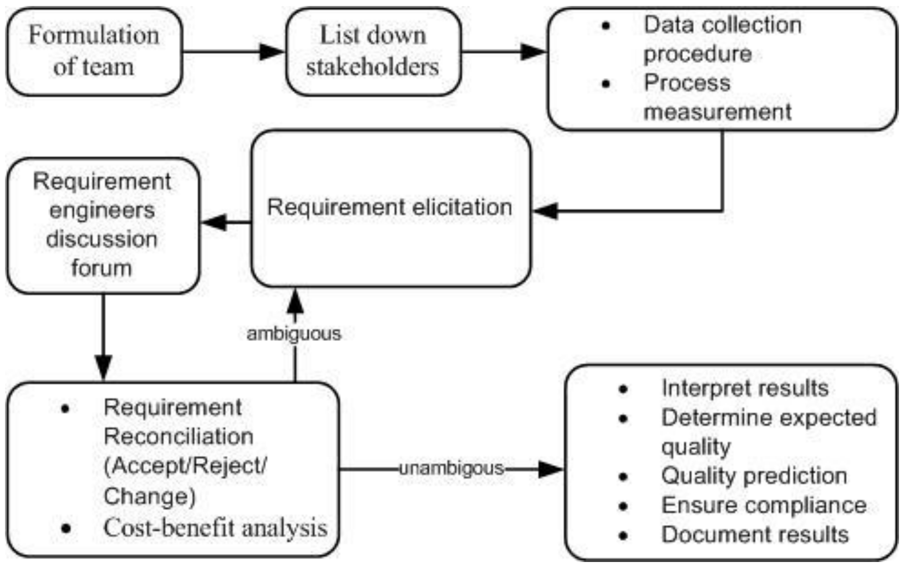

Figure 1. Proposed Process Model for Requirement Engineering

The aforementioned recommended steps for our proposed requirement engineering process model are illustrated in Figure 1.

5. Effect of Metrics on Requirement Engineering Process

The project monitoring techniques using a set of metrics help the requirement engineers to quantitatively measure the effectiveness and fitness of any requirement collected during the elicitation process of requirement engineering. Figure 2 illustrates how metrics can improve the quality of the gathered requirements.

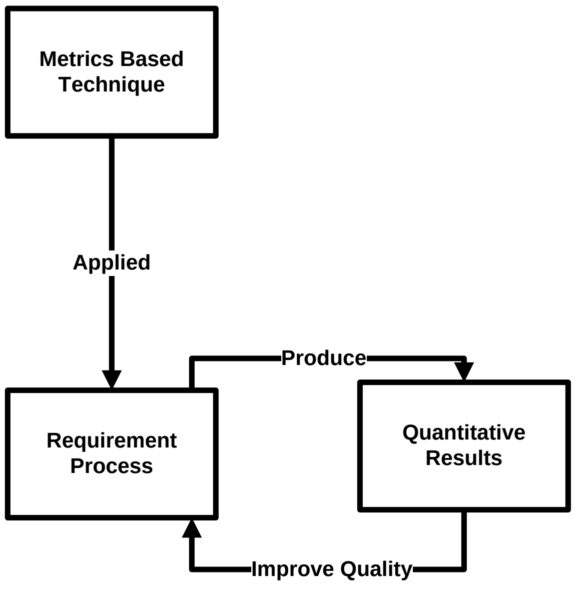

Figure 2. Effect of Metrics on Requirement Engineering Process

By employing our proposed requirement metrics, software development companies and teams can significantly improve the quality of their requirement gathering process which in return will not only help in delivering the software projects in conformity with the schedule and budget, but will also serve as an effective tool for the project managers to better administer the software projects. From the authors own experience, the software developed using the abovementioned techniques have successfully been deployed.

6. Conclusion

Use of metrics play critical role to measure performance of software projects. The results obtained through these metrics act as performance indicators for different artifacts and activities of the software processes and this information is grouped together to measure health of the projects. In this perspective, the software process model, productivity metrics, efforts management metrics and requirement metrics provide vital support for effectively measuring and controlling software projects. In this paper, we have presented a set of additional requirement metrics that can be used in combination with the existing metrics and procedures to administer the software projects in a more befitting manner. The criteria for measuring the results of these metrics have also been articulated in the paper. The specific criteria delineated against each metric is in fact the recommended level for gauging utility and effectiveness of that metric which can slightly be changed to cater for any native exigency pertinent to a particular project. Our practical experience of using the aforementioned metrics in different software projects show that these metrics are very effective to keep abreast with the current state of the affairs of the different software processes; and the software developed by employing these metrics have successfully been deployed. We presume that by adopting these requirement metrics, the software development companies can benefit from a deeper insight about their projects which in return will not only help to better administer the software, but also contribute towards achievement of the business goals in an effective manner. To further enhance and extend scope of our proposed metric, we intend to look into the metrics for design and development phases of the software projects as a prospective future work to this research.

References

[1] G. Atkinson, J. Hagemeister, P. Oman, and A. Baburaj, "Directing Software Development Projects with Product Metrics", In Proceedings of the Fifth International Software Metrics Symposium, Bethesda, MD , USA **(1998)** pp. 193 - 204.

[2] P.M. Johnson, H. Kou, M.G. Paulding, Q. Zhang, A. Kagawa, and T. Yamashita, "Improving Software Development Management through Software Project Telemetry", IEEE Software **(2005)** August.

[3] A. Sukhoo, A. Barnard, M.M. Eloff, J.A. Van der Poll, and M. Motah, "Accommodating Soft Skills in Software Project Management", Issues in Informing Science and Information Technology: 2 **(2005)** pp. 691-703.

[4] C. Sirias, "Project Metrics for Software Development", **(2009)** Available online at: http://www.infoq.com/articles/project-metrics.

[5] D.E. Geetha, T.V.S. Kumar, and K.R., "Predicting Performance of Software Systems during Feasibility Study of Software Project Management", In Proceedings of the 6th International Conference on Information, Communications & Signal Processing **(2007)** pp. 1-5.

[6] R.C. Nienaber, and A. Barnard, "A generic Agent Framework to Support the Various Software Project Management Processes", Interdisciplinary Journal of Information, Knowledge and Management: **(2007)** February.

[7] Q.Z. Wang, and J. Liu, "Project Uncertainty, Management Practice and Project Performance: An Empirical Analysis on Customized Information System Development Projects", 2006 IEEE International Engineering Management Conference, Bahia **(2006)** September 17-20.

[8] A. Asthana, and J. Olivieri, "Quantifying Software Reliability and Readiness", The MITRE Corporation **(2009)**

[9] R. Kamalraj, B.G. Geetha, and G. Singaravel, "Reducing Efforts on Sofware Project Management using Software Package Reusability", IEEE International Advance Computing Conference, Patiala, India **(2009)** pp. 1624-1627.

[10] G. Liu, "Tracking Software Development Progress with Earned Value and Use Case Point", In Proceedings of the 2009 International Workshop on Information Security and Application (IWISA 2009), Qingdao, China **(2009)** November 21-22.

[11] Rational Software, IBM Information Centre, "Managing and Composing Requirements", Available at: http://publib.boulder.ibm.com/infocenter/reqpro/v7r1m0/index.jsp?topic=/com.ibm.reqpro.help/integ/c_req_metrics.html.

[12] L. Yang, S. Wei, Z. Chi-long, and T.H. Lei, "On practice of Big Software Designing", Journal of Software: 5(1), **(2009)** pp. 81-88.

[13] G. Lajios, "Software Metrics Suite for Project Landscapes", In Proceedings of the 13th European Conference on Software Maintenance and Reengineering, Kaiserslautern, Germany **(2009)** March 24-27, pp. 317-318.

[14] G. Jiang, and Y. Chen, "Coordinate Metrics and Process Model to Manage Software Project Risk", In Proceedings of 2004 IEEE International Conference on Engineering Management **(2004)** October 18-21, pp. 865-869.

[15] YourDictionary, ""Productivity Business Definition", Available at: www.yourdictionary.com/business/productivity, Last Visited: **(2010)** June 25.

[16] BrainyQuote, "Definition of Effort", Available at: www.brainyquote.com/words/ef/effort158526.html, Last Visited: **(2011)** June 25.

Authors

Shahid Iqbal completed his Master of Science in Software Engineering from Shaheed Zulfikar Ali Bhutto Institute of Science and Technology (SZABIST), Islamabad, Pakistan. He has been serving in the field of Information Technology. His research interests include, software engineering, project management, analyzing and developing new methods and tools for effective management.

M. Naeem Ahmed Khan obtained D.Phil. degree in Computer System Engineering from the University of Suusex, Brighton, England, UK. Presently, he is affiliated with Shaheed Zulfikar Ali Bhutto Institute of Science and Technology (SZABIST), Islamabad. His research interests are in the fields of software engineering, cyber administration, digital forensic analysis and machine learning techniques.