REVIEW OF RESEARCH
IMPACT FACTOR : 5.7631(UIF) UGC APPROVED JOURNAL NO. 48514 ISSN: 2249-894X
VOLUME - 8 | ISSUE - 5 | FEBRUARY - 2019

A SYSTEMATIC STUDY ON VARIOUS SOFTWARE DEFECTS AND THEIR IMPACT ON SOFTWARE PROJECT DEVELOPMENT

Meer Tauseef Ali[1] and Dr .Syed Abdul Sattar[2]
[1]Research Scholar, Department of Computer Science, Rayalaseema University, Kurnool, A.P.
[2]Professor, Director (R&D) & Principal, Nawab Shah Alam Khan College of Engg. & Tech., Hyderabad.

ABSTRACT :

As new Software project success measures are being discussed, the projects failing to be complete within the traditional constraints continue to hamper major corporations, global business and Government agencies. Even though majority of projects fail to be completed on time and on budget every year, trends are not showing any big improvements in these areas. As large projects continue to face performance issues, project managers can relate project size to increases in severity and frequency of many common project challenges. During review of many of these surveys the correlation between project size and project failure is evident. Global business specifically to IT projects different methodologies do not provide a large enough benefit to combat the challenges of increased project size. The objective of this study is to define and implement Defect Prevention mechanism, for the various stages of software project development, to reduce the quantity of defects and then to determine a strategy for decreasing the testing effort needed for development projects in an organization.

KEYWORDS : *Defect data , Severity, Probability, Phase injected, Agile methodology.*

INTRODUCTION:

As more and more projects are utilizing different methodologies to deliver fast business value identifying project success is consistently being discussed. The amount of studies and surveys conducted every year on project management failures is plentiful. According to an IBM study, only 40% of projects meet schedule, budget and quality goals. Further, they found that the biggest barriers to success are people factors[1]. Geneca, a software development company, noted from its studies that fuzzy business objectives, out-of-sync stakeholders and excessive rework mean that 75% of project participants lack confidence that their projects will succeed. [2] 50 % of businesses had an IT project fail during 2012 year, according to a survey by cloud portfolio management provider Innotas. [3] The primary reason, according to 74 percent of respondents, was a lack of resources to meet project demands. The Portland Business Journal found similarly depressing statistics: Most analyses conclude that between 65 and 80% of IT projects fail to meet their objectives, and also run significantly late or cost far more than planned. “Many of these statistics show the high number of projects that ultimately are delivered late and over budget while providing some of the drivers behind these failures. [4]

Only 26 percent of all projects succeed. Just 42 percent of organizations report having high alignment of projects to organizational strategy. This lack of alignment of projects most likely contributes to

the surprising result that nearly one half of all strategic initiated (44 percent) are reported as unsuccessful. Project success rates are rising. Organizations today are wasting an average of US$97 million for every US$1 billion invested. [5]

As study goes through this research area one can quickly correlate many of these issues back to a single variable in the project and that is project size. When speaking about large and small projects factors discussed are costs, complexity and durations. Software development organizations are frequently being challenged with overcoming the after effects of failed project delivery. Report to illustrate the correlation between project size and failure. The common drivers of project failure will be identified and discussed in terms of project size. Any software which must be delivered to the consumer should be defect free. Therefore finding out defects and remedies for the same plays an important role in the software development process. The ability to predict defects is also of primary importance in this regard. All these factors must be taken into consideration during the software testing phase. Software testing is employed in the software development process to find out any occurrence of defects in the software. Numerous defect analysis and prediction techniques are used, which facilitate in defect prevention. In software defect prediction we predict the occurrence of defects by observing the existing trends and occurrence of defects.

2. RELATED LITERATURE

According to Perry, W. E. et al[6], The Software Development process always has deficiencies with regard to test activities. As per Spillner A. et al, [7] real testing activity starts after the coding stage at which point the implementation appears to be complete. Jalote, P. [8] stated that each development stage injects defects into the application. The percentage of defects injected may vary from one stage to another.

Humphrey, W. S.et al [9] stated that correcting defects at the later stages of the development is more complicated, expensive and time-consuming than correcting them at earlier stages. It is therefore the best way to optimize the development process costs and to shorten the development time.

According to Weber, D.G.et al[10] defect prevention activity is a mechanism for spreading lessons learned across the projects, Mays, R.G., et al. [11] objective of defect prevention are to define and implement techniques to reduce the total number of defects in the process of development and to prevent certain classes of defects from reoccurring.

As per Anderson, T., Barrett, P.A., Halliwell, D.N., Moudling, M.L et al [12][13] the defects may have been identified on other projects as well as in earlier stages or tasks of the current project. If we have defect prevention activity at every stage then there is a tendency to reduce the number of defects. Defect Prevention involves analyzing defects that were encountered in the past and taking specific actions to prevent the occurrence of those types of defects in the future.

According Abraham, P., Leeba, S., Abraham, D., Paul, R.et al[14] defects occurrence reduces by implementing defect prevention process in any organization. It has been observed that average 40-50% of defects are covered in every stage of the development by Defect prevention activities. If the detection of these 40% of defects is covered by Defect prevention activities then it reduces the cost of developing the software drastically. The rest (60%) of defects are mostly identified in the testing stage. [15]

If business software is developed without sufficient understanding of the requirement specifications, then this deficiency of the requirements costs more to operate and maintain the system in production. From the beginning, the quality team or development team should trace the requirement through horizontal or vertical traceability matrix.

2.1 Classification of Defects

Software Defects are normally classified on the basis of Severity, Probability, Priority, Related Dimension of quality, Related Module or Component, Phase Detected and Phase Injected.

2.1.1 Related Module /Component

Related Module / Component indicate the module or component of the software where the defect was detected. This provides information on which module or component is buggy or risky. For example whether it is found in Module A, B, C and so on.

2.1.2 Phase Detected

Phase Detected indicates the phase in the software development lifecycle where the defect was identified in, Unit testing, Integration testing, System testing or in Acceptance testing phase.

2.1.3 Phase Injected

Phase Injected indicates the phase in the software development lifecycle where the bug was introduced. Phase Injected is always earlier in the software development lifecycle than the Phase Detected. Phase Injected can be known only after a proper root-cause analysis of the bug. Whether the bug found in requirements phase, high level Design, detailed design, coding or in deployment phase. The categorizations above are just guidelines and it is up to the project/organization to decide on what kind of categorization to use. In most cases, the categorization depends on the defect tracking tool that is being used. It is essential that project members agree before hand on the categorization (and the meaning of each categorization) so as to avoid arguments, conflicts, and unhealthy bickering later.

2.2 Further Defect Classification

The further defect classification method involves manually evaluating defects to add contextual meta information. The Team Software Process suggests some of the following pieces of information as a starting point for creating defect taxonomy. Firstly, when a defect is reported thereby the feature related to which it is reported will be marked. Secondly the code in which the defect was found will be marked naming it as Module discovered.

A standardized description of the defect Is needed, such as "assignment," "data," or "syntax." so as to broadly distinguish Defect type. Sometimes the bug makes it into the software in the first place. This should also be standardized, for example "education," "oversight," or "mistake." In which of the activities the defect was injected. for example, "requirements," "design," or "implementation." What were the activities being performed when the defect was discovered? These would be the same as the list for phase injected.

This simple method has some disadvantages, the biggest one being that we usually will not be able to understand the reason a defect was injected until after we understand the root cause of the defect. Also, unless we were the one who wrote the code, then only we can guess at the reason (was this an oversight, mistake, or education?) the defect was injected.

As such, some knowledge of the system is required to perform the analysis. There are other methods for defect classification that overcome this problem but take a little more training and planning to apply well. Since the defects in a software system are a direct reflection of the process, knowledge, and skills of the team that injected them, once the defects are classified you have some serious power for enacting positive change.

3. SOFTWARE PROJECT DEFECT ANALYSIS

3.1 Benchmark for project success

As the agile methods is extremely prevalent in the software development industry, business value and speed to market have been tagged as the measure of success while traditional constraints are thought to have some flexibility. Project success can and will be defined by the client or company embarking on a project to enhance their business. However, large-scale projects fail due to the delays in schedule and enormous cost overruns. [16]

3.2 Time Constraints

Many project managers are tasked with providing accurate cost estimates and end dates for multi-year projects before gaining funding or project approval. Although we refine as we build out our plans, many of us can never live within our initial funding limits or schedule constraints [18]. However, when regarding initial cost and schedule estimates the longer the project are the less certainty around much of this original information [19]. This may seem counterintuitive as we learn more as the project goes on and are able to get a better idea of what the finished product will eventually be as the months pass.

3.3 Requirement Complexities

As the long duration of projects are kicking off many people can associate the grand scale of the project with unlimited space to add large complex items into it. In smaller projects, the idea of defining specific scope driving value is easier to achieve as the thought of a tight condensed budget and schedule becomes more practical.

3.4 Communication Barriers

For many large software projects effort is increased and therefore the number of resources usually follows. There can also be situations where silos are built in the form of various contracts awarded on the same project. This creates additional challenges with handoffs, risks and dependencies.

3.5 Usage of Methodologies

However, Agile software projects face similar trends with regards to project size and failure rates. In a study it is stated that 34% of agile failures can be related to poor planning. Although planning itself can be done with success on small and large projects it ties back to complexity. The planning of large projects with many dependencies regardless of methodology increases the complexity.

4. RESULTS AND INTERPRETATIONS:

Details of Defect Data

The following graphs summarize some of the more interesting trends we discovered by looking at the data. There were two very interesting findings. First, we had a major problem in requirements that resulted in significant rework. Second, we need to rethink the way testing is done and work to improve.

Table-1: shows the percentage count of defects injected

S.No	**Defects types**	**Percentages**
1	Communication	13
2	Communication(change)	25
3	Communication (requirement)	12
4	Data Issues	9
5	Default behavior	7
6	Deployment	1
7	Education	17
8	Misconfiguration	12
9	Overwrite	3
10	Process	2

Table-1 show the reason why various defects were injected. Defects that originate in tests were excluded from this particular analysis so that we could focus on other areas that might also be interesting. Communication in our vernacular refers to cases where the defect resulted from misinformation originating

in documentation or personal interactions (e.g. a meeting).We noticed that a disproportionately large number of defects resulted from miscommunication and so broke down communication into true communication problems, changes that were not effectively communicated, and new requirements that were only flushed out as reported bugs from the customer.

The information in Table-1 shows that 55% of reported non-test defects resulted from requirements or communication issues. While 18% of reported non-test defects were injected directly by the development team. This 18% includes oversights (things you just forgot), education (things you should have known how to do), and mistakes (things you understood and meant to do but botched somehow). We also included process issues in which the process we were following created an opportunity for a defect that was actually injected into the software and escaped to the customer for review. Defect injection reason percentage, shows that communication and requirements issues account for 55% of all non-test related defects. Further, 18 % of defects were injected directly by the team and might be avoided through better training, testing, or peer reviews.[20]

While understanding why we injected defects is interesting, understanding where the defects are injected is important to creating a plan for doing something about it. Table 2 shows the defect percentage count for each of the main areas of the software products. The top three trouble spots, "bug farms" were the display configuration (user interface), test cases, and data conversion and extraction code. That poor test cases were one of our greatest sources of defects is extremely disturbing.

Table-2: shows percentage count of defects injected into different software modules.

S.No	**Defects types**	**Percentages**
1	Display Configuration	25
2	Test Case	22
3	Crawl configuration	7
4	Conversion/ Extraction	14
5	Environment Configuration	3
6	End user confirmation	2
7	Taxonomy	4
8	Not a bug	3
9	Source Configuration	4
10	Unable to reproduce	7
11	others	9

Table-2 includes defects related to tests. Putting it all together (figure 1) really shows our trouble spots and gives a pretty fair view of where the team needs to improve, When we look at this data, we see that communication in general is a pretty big problem. There were acute problems when trying to communicate requirements in the User Interface, especially late feature requests and changes. Finally, the team just couldn't get it together for testing and made mistakes all over the place.

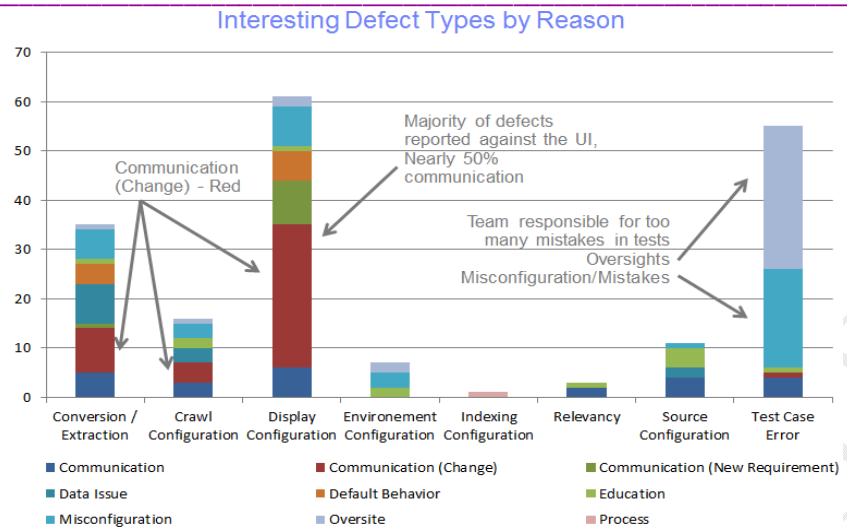

Figure- 1: Count of defects by module by reason injected.

Now getting into clear and more precise information on how the software projects development industry is coping in handling the Defects at large. As the study is about to explore the reason behind what exactly the scenario in software projects developments during 2011 to 2017. [21] [22]

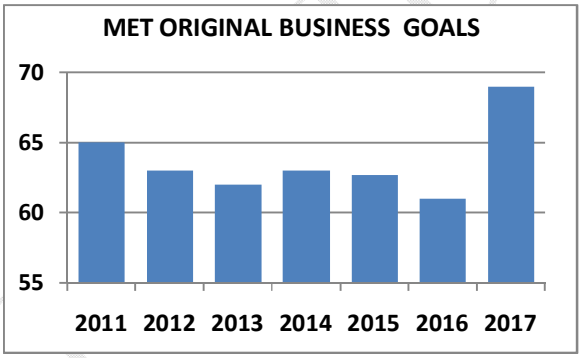

Figure- 2: shows the percentage of software projects achieving the original business goals.

The Figure 2 shows that since 2011 projects on average have been completed on time and on budget less than 60% of the time. Clearly depicts the Scenario of software development as there is drastic improvement in the dealing with business objectives.

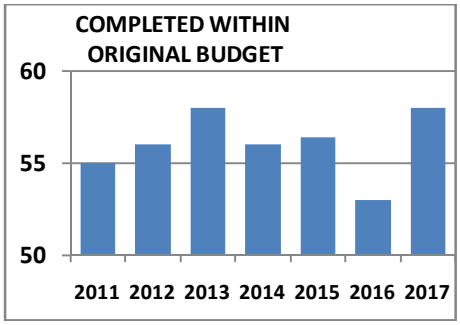

Figure -3: shows the percentage of software projects getting completed within original budget.

Figure 3 Shows the improvement in terms of completing the project within the allocated budget, there was too low in 2016 at 53% whereas in 2017 it reached 58% comparatively. Since 2013 the business market was coping with the drastic change, thereby the demand in business software was high which resulted in frequent changes in frequency causing the overall budget to be overrun.

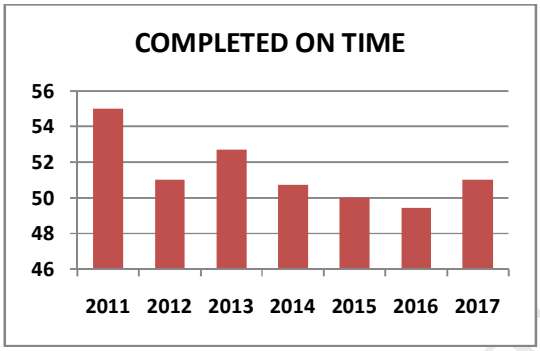

Figure- 4: shows the percentage of software projects completed on time.

Figure 4: shows details of software projects completing on time. In 2017 there is a remarkable growth in completing the project on time when compared to 2014, 2015 and 2016 respectively. By properly implementing the changes in software requirement much of the rework of testing is reduce, thereby allowing project to be completed in time.

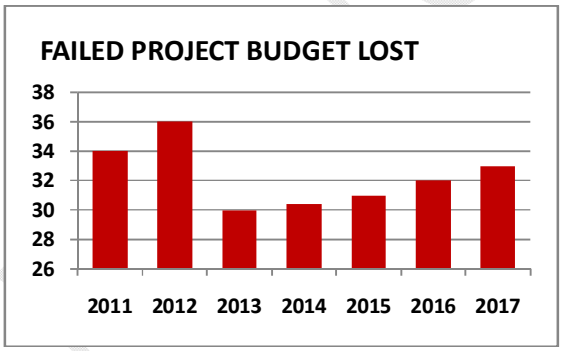

Figure -5: shows the percentage of software projects failed and the budget is lost.

Most often, software engineers or developers do not have a good design before coding or writing programs or software (Poor or no designs at all). This is the major cause of software failure today. Developers most times boot their computers and navigate to the programming language location and they will start coding without having a design. Such software is bound to fail. Once software does not have a very good design, the software is bound to fail as it is put into use.

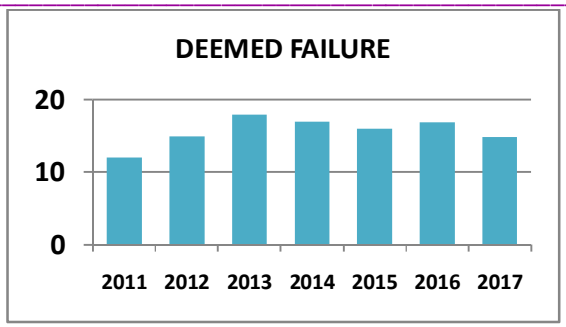

Figure- 6: shows the percentage of software projects Deemed failure .

This Figure 6: shows that, in 2017 there is a decrease in partial failure as compare to previous years, In the year 2013 and till 2016 duration, it was the tragic phase for software business due to which majority of the software wasn't able to compily the drastic changes in the business market.

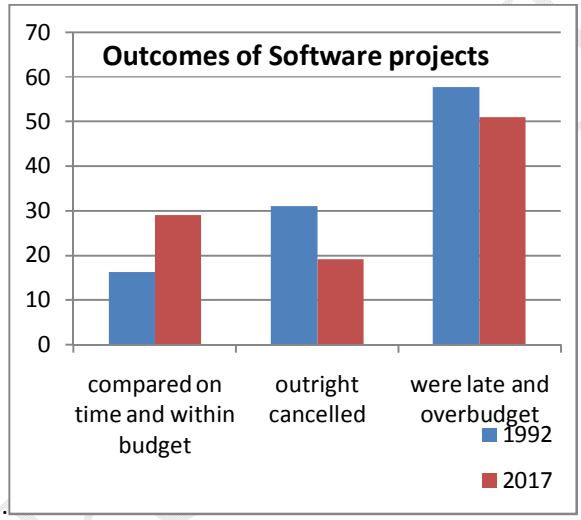

Figure- 7: shows the percentage outcomes of software projects

Again in Figure-7 most developers of software often have one major or comprehensive design called high-level design. For software to be properly developed and have minimal errors, the software must have both high-level and low-level (or detail) design. Before coding, both the high-level and detail designs must be properly done and correction made so as to minimize the errors that might arise from the software

Going through the software development 25 years history just 16.2% of all MIS projects were completed on time and within budget 52.7% were late and over budget and 31.1% were outright cancelled. The top two reasons then were lack of user input/involvement and incomplete requirements. Recently there is an improvement from 16.2% to 29% success rate. Certain projects were as high as 62%.

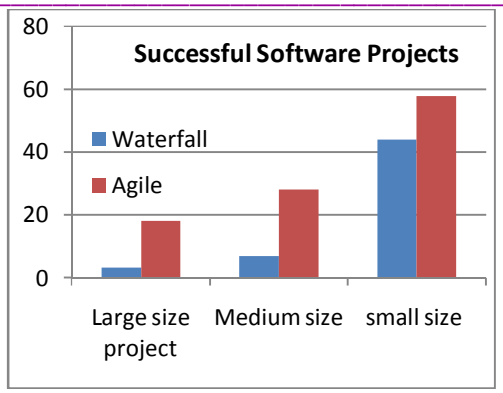

Figure- 8: shows the percentage comparisons between waterfall and agile methods success.

Figure-8 shows a comparison of the adaptation of methodologies in dealing with software projects success. Here is the comparison made on 3 different Classification sizes namely large size, medium size and small size. Looking at the successful project waterfall methodologies holds 3% for large size, 7% for medium size, and 44% respectively, whereas the Agile methodology holds 18% for large size, 27% for medium size and 58% for small size respectively. [23]

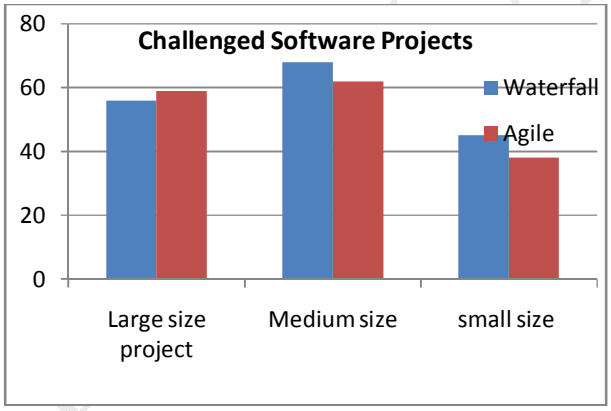

Figure- 9: shows the percentage comparisons between waterfall and agile methods for challenged software development projects.

In Figure- 9. shows comparison in terms of challenged software project the result for Waterfall methodology shows 57% for large size ,63% for medium size and 42% for small size respectively , But whereas the Agile methodology shows 59% for large size , 61% for medium size and 38% for small size respectively.

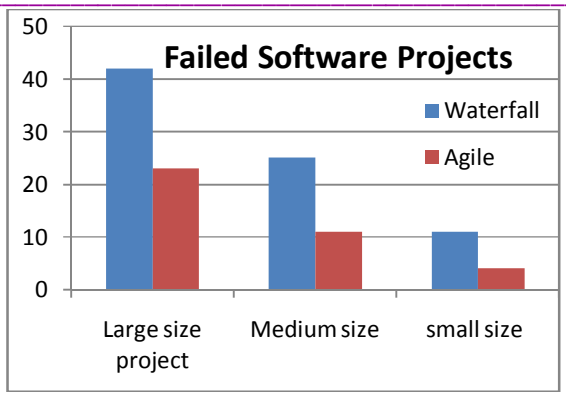

Figure- 10: shows the percentage comparisons between waterfall and agile methods for failed software development projects

Finally, Figure 10 shows the comparison in terms of software project failures between Waterfall and Agile methodology the result depicts 42% for large size, 25% for medium size and 11% for small size respectively whereas Agile shows 23% for large size, 11% for medium size and 4% for small size respectively. So, therefore the changing scenario in business software's suits for agile methodology as compared to Waterfall. [24]

By looking at the usage of a software methodologies we found that those did had more success, 38% vs. 31%. Those that Used a methodology were still 72% behind schedule, 29% didn't meet scope, 32% didn't meet quality requirements, 40% didn't meet expected benefits and the numbers are even worse for projects that did not use a methodology. A software project in its early stage is like an iceberg where you see the tip above the water and the underwater shape/size can't be predicted easily." [25]

5. CONCLUSION

With a focus on IT projects, management methodologies like Agile can be used with some success to mitigate some of the common issues and help project managers deliver value early. There are many reasons that can drive a project off the path and inevitably cause it to finish over budget or be heavily delayed. The success of projects continues to be discussed and studied as many organizations invest millions in hopes to get some value in return. However, when looking at success to failure comparisons, one of the biggest factors is the size of the project. With large projects these challenges are magnified and exaggerated compared to similar challenges on their smaller counterpart projects. There is a utmost need in applying proper methodology for given software projects and through information gathering approach and better communication among different phases of software development can only yield better results, the answers really drive to a improvised and optimize flaw down project from the start. The conclusion is relatively easy to arrive at if you believe all projects face similar challenges

REFERENCES:

[1]http://www-935.ibm.com/services/us/gbs/bus/pdf/gbe03100-usen-03-making-change-work.pdf
[2]https://www.geneca.com/why-up-to-75-of-software-projects-will-fail
[3]https://hbr.org/2011/09/why-your-it-project-may-be-riskier-than-you-think
[4]https://www.mckinsey.com/business-functions/digital-mckinsey/our-insights/delivering-large-scale-it-projects-on-time-on-budget-and-on-value.Portland Business Journal
[5]https://www.pmi.org/learning/thought-leadership/pulse
[6]Perry,W.E., Effective Methods for Software Testing, John Wiley & Sons, 2nd Edition, 1999 .
[7]Spillner,A.,From V-Model to W-Model - Establishing the Whole Test Process, Proceedings Conquest - 4th Conf. on Quality Engineering in Software Technology, (2000) .

[8]Jalote,P.,An Integrated Approach to Software Engineering, Narosa Publishing House, 3 rd Edition, 2007.
[9]Humphrey, W.S."Managing the Software Process", Chapter 17 - Defect Prevention, ISBN-0-201-18095-2 .
[10]Weber,D.G.,"Formal specification of fault-tolerance and its relation to computer security", Proc. 5th Int. Workshop on Software Spec. and Design, pp 273-277, Pittsburgh, PA, May 1989 .
[11]Mays,R.G.,Jones,C.L.,Holloway,G.J.,Studinski,D.P.,"Experiences with Defect Prevention" IBM Systems Journal, Volume 29, No. 1, 1990.
[12]Anderson,T.,Barrett,P.A.,Halliwell, D.N., Moudling, M.L., "An evaluation of software fault tolerance in a practical system", Proc. Fault Tolerant Computing Symposium 1985, pp. 140-145.
[13]Boehm,B.,Basili,V.,Software Defect Reduction Top 10 List, Computer, 34(1), 2001,135,137 www.sei.cmu.edu/publications/documents/92.reports/92.tr.022.html
[14]Abraham,P.,Leeba,S.,Abraham,D.,Paul,R.,"Defect Prevention Techniques for High Quality and Reduced Cycle Time" An ESSI Process Improvement Experiment (PIE), EuroSPI 98.
[15]Williams,L.Instilling a Defect Prevention Philosophy. Frontiers in Education Conference. 1998. Pages 1308-1312 .
[16]Edwards,J.(2017).7 simple ways to fail at agile. CIO. Retrieved from https://www.cio.com/article/3234366/project-management/7-simple-ways-to-fail-at-agile.html
[17]Flyvbjerg,B.,& Budzier,A.(2011).Why Your IT Project May Be Riskier Than You Think. Harvard Business Review. Retrieved fromhttps://hbr.org/2011/09/why-your-it-project-may-be-riskier-than-you-think.
[18]Mieritz, L(2012).Gartner Survey Shows Why Projects Fail 501. 2012: G00231952. Retrieved from https://thisiswhatgoodlookslike.com/2012/06/10/gartner-survey-shows-why-projects-fail.
[19]Nicholas,J.,&Hiding.(2010). Management principles associated with IT project success.International Journal of Management & Information Systems (Online Retrieved from https://ez.sjcny.edu/login?url=https://ez.sjcny.edu:2099/docview/1778076247?accountid=28722.
[20]Project Management Institute.(2015).Capturing the value of project management. Project Management Institute. Retrieved from https://www.pmi.org//media/pmi/documents/public/pdf/learning/thoughtleadership/pulse/pulse-of-the-profession-2015.pdf.
[21]The Standish Group. Retrieved from https://www.projectsmart.co.uk/white-papers/chaos-report.pdf.
[22]Standish Group Report CHAOS Report 2015.
[23]Wrike Complete Collection of Project Management Statistics 2015.
[24]An Agile Agenda. 6point6 Limited Registered in England and Wales Company (2017).
[25]Project Management Institute Pulse of the Profession 2017.

Meer Tauseef Ali
Research Scholar, Department of Computer Science, Rayalaseema University, Kurnool, A.P.

Dr. Syed Abdul Sattar
Professor, Director (R&D) & Principal, Nawab Shah Alam Khan College of Engg. & Tech., Hyderabad.