A Study to Investigate the Impact of Requirements Instability on Software Defects

Talha Javed[1], Manzil-e-Maqsood[2] and Qaiser S. Durrani[3]
[1, 2, 3] National University of Computer and Emerging Sciences,
852-B, Faisal Town, Lahore, Pakistan
{mspm008,mspm024,qaiser.durrani}@nu.edu.pk

Abstract

Software development is a dynamic process and is characterized by change. Software projects often begin with unclear, ambiguous, and incomplete requirements which give rise to intrinsic volatility. Constant change in requirements is one of the main causes of software defects and a major issue faced by the software industry. This paper describes the findings of our research-based study that investigates the impact of both the pre-release and post-release requirements changes on overall defects by defining measures, collecting data against those measures and analyzing the collected data through statistical techniques. Our findings, based on industry data from 4 software projects consisting of 30 releases, all in e-commerce domain, indicate that there is a significant relationship between pre/post release change requests initiated by the client and software defects. In addition, our data analysis indicates that changes in the design of the system at the later stages of software development i.e., during coding, testing and after release have a significant impact on the high severity defects that affect the major functionality of the system. Also, we found that insufficient time spent on the design phase and inadequate communication with the client could be some of the causes of requirements changes and consequently software defects.

Keywords: Requirements change, pre/post release changes, change request (CR's)[1], high/medium/low change requests, defects, severity-1/severity-2 defects

1. Introduction

Requirements are the foundation of the software development process. They provide the basis for estimating costs and schedules as well as developing design and testing specifications. So the success of a software project, both functional and financial, is directly related to the quality of its requirements. Although an initial set of requirements may be well documented, requirements will change throughout the software development lifecycle. Thus, constant change (addition, deletion and modification) in requirements during the development life cycle impacts the cost, schedule, and quality of the resulting product [4].

However the basic problem is not with changing requirements; the problem is with inadequate approaches for dealing with them**.** Requirements Evolution is due to both social and technical aspects. The *social viewpoint* is related to the stakeholders involved in the system, they range from end-users to software engineers, project managers and other business actors (e.g., standards regulators, market competitors, etc.). All stakeholders change their understanding of the ongoing system during its life cycle, hence requirements evolve. On the *technical viewpoint*, requirements may evolve due to production constraints, usage experience and feedback from other phases of the system life cycle (e.g., testing).

Ideally, the requirements once approved by the client should stabilize with no or very few major changes. According to *Capers Jones*, requirements change (RC) should come down to 3% in the design phase, 1% in the coding phase and ideally 0% during testing. However requirements change is always there but it can have very negative affect during the later stages of software development. For example: requirements change during the coding and testing stage can maximize the defect density as compared to other phase. Studies conducted by Jones have shown that the defect rates associated with the new features added during mid-development are about 50% greater than those of the artifacts associated with original requirements. [7].

Secondly it is important to realize that RC can be distinguished as: *(1) Pre-FS (Functional Specification) Changes* which refer to changes in the requirements during the early phases (i.e. elicitation, elaboration, analysis, modeling and negotiation) of software development before FS has been completed and signed off, *(2) Post-FS Changes* occur during the later phases of software development (i.e. design, coding, testing and development) after the FS has been formally signed off, *(3) Post-release changes* occur once the system has been deployed at the client side, after release [20]. *(First and second type of changes fall under the category of Pre-release changes).*

In the above context, it is worth mentioning that the first type of change is constructive if correctly done, because these would help in more complete requirements. However the second and the third types of requirements can be destructive as they may affect the productivity in terms of cost overruns, schedule overruns and quality *(adding defects while incorporating a change).*

Malaiya [17] has examined the relationship between changing requirements and defect density at the code phase and found the requirements volatility has an impact on defect density. According to *Capers Jones* [7], the maximum defects should never exceed 3.5 defects per function point (Sum of the defects found in requirements, design, code, user documents and bad fixes)[2].

Our research work investigates the impact of both the pre and post-release requirements changes on *overall defects* by collecting

[1] Changes in requirements (addition, deletion, modification) initiated by the client through the Change Request Forms (CRFs).

[2] The data presented here is derived from top 5% of the projects in the top 30% organizations *Software Productivity Research* has analyzed out of a total of 600.

and analyzing the data through statistical techniques. For this study we have collected data from *4 projects consisting* of *30 releases*. All these projects are from the *e-commerce domain*. Further, we have categorized both the pre and post-release CR's initiated by the client in three different categories[3]: high; medium; low. Similarly we have categorized the defects in two categories[4]: severity-1; severity-2.

The findings of this study provide some preliminary results in understanding the impact of RC on software defects and the possible causes of those changes. We believe that these results provide significant implications for software practitioners to understand the impact of RC and the associated risks. However, due to the intricacy of the RC phenomenon and the scarcity of empirical evidence available, it is important to conduct further investigations to better understand the causes and effects of RC, to identify effective processes and tools and techniques to control and manage RC.

This study is organized as follows: In section 2, we look at the work related to the impact of RC. Section 3 presents our hypotheses and the procedures for data collection along with brief details of the data gathered against the selected projects. In section 4, we have discussed the results based on our findings and the final section concludes our work with directions for future research.

2. Prior Literature

Recent studies have shown that both large and complex software projects experience many changes throughout the system development life cycle [4]. Studies conducted by *Barry* [3] have shown that the sources of RC are manifold (changing work environment, organizational complexity, government regulations, and conflicts among stakeholders in deciding on a core set of requirements).

Lamsweerde [1] conducted a survey of over 8000 projects from 350 US companies and revealed that one third of the projects were never completed and one half succeeded only partially, that is, with partial functionalities, major cost overruns, and significant delays. When asked about the causes of such failures, executive managers identified *poor requirements* as the major source of problems (about half of the responses) - more specifically, the *lack of user involvement* (13%), *requirements incompleteness* (12%), *changing requirements* (11%), *unrealistic expectations* (6%), and *unclear objectives* (5%).

On the European side, a recent survey of over 3800 organizations in 17 countries similarly concluded that most of the perceived software problems are in the area of requirements specification (greater than 50%) and requirements management (50%) [1].

Prior studies have investigated the impact of RC *on software productivity* [15]*, software releases* [16] *and its impact on isolated software development phases* [17]. *Lane* [15] investigated the impact of RC on effectiveness and efficiency of software development productivity and found that there was no direct impact of requirements change on these two concepts. *Lane's* findings further suggested that factors such as product size and organization size strongly influence the impact of RC on software development productivity. Another study conducted by *Zowghi* [19, 20] provided no strong evidence to support that RC has a direct impact on software development productivity (such as code quality, quality of project management and development capability). *Hyatt et al* [2] reported that RC must be considered as a part of project risk assessment. *Malaiya* [17] has examined the relationship between RC and defect density at the code phase and found the RC has an impact on defect density. However, to our knowledge, little prior research has been done to specifically examine the impact of RC on software defects throughout the SDLC and the root causes of those defects.

Our study focuses on what the previous studies fall short of coverage. First, investigating the impact of both the pre-release and post-release requirements changes (categorized as: high; medium; low) on overall defects (categorized as: severity-1; severity-2) throughout the SDLC. Second, identifying the possible causes of requirements changes.

3. Hypotheses and Research Site

3.1 Hypothesis-1

The purpose of this hypothesis is to test the relationship between the *pre-release* CR's initiated by the client and the defects introduced due to those changes. Here the variable 'pre-release CR's' has three categories: high=1; medium=2; low=3, and the variable 'defects' has two categories: severity-1; severity-2. To prove the hypothesis, we have used Cross-tabulation method and applied the Pearson's Chi-Square test.

3.1.1 Null Hypothesis (H_0)

There is no relationship between defects and the *pre-release* CR's (high, medium, low severity) initiated by the client and the two variables are independent.

3.1.2 Alternate Hypothesis (H_1)

There is a relationship between defects and the *pre-release* CR's (high, medium, low severity) initiated by the client and the two variables are dependent on each other

3.2 Hypothesis-2

The purpose of this hypothesis is to test the relationship between the *post-release* CR's initiated by the client and the defects introduced due to those changes. Here the variable 'post-release CR's' has three categories: high=1; medium=2; low=3, and the variable 'defects' has two categories: severity-1; severity-2. To prove the hypothesis, we have used Cross-tabulation method and applied the Pearson's Chi-Square test.

[3] **High:** If a CR affects the Design, major functionality or databases design of the system.
Medium: If a CR affects minor functionality or minor database changes.
Low: If a CR requires minimal GUI consistency changes.

[4] **Severity-1**: major defects, affecting the significant functionality of the system.
Severity-2: minor defects, mostly GUI related.

3.2.1 Null Hypothesis (H_0)

There is no relationship between defects and the *post-release* CR's (high, medium, low severity) initiated by the client and the two variables are independent.

3.2.2 Alternate Hypothesis (H_1)

There is a relationship between defects and the *post-release* CR's (high, medium, low severity) initiated by the client and the two variables are dependent on each other.

3.3 Research Site and Data Collection

Our research site is a leading software organization that develops diverse commercial applications. Brief details of the organization and the projects under study are given in Table-1.

Organization Details	
Organization size	140 employees (approximately)
Organization's maturity level	Tick-IT Certified; ISO Certified
Project Details	
Number of projects under study	Four Project A = 16 releases Project B = 10 releases Project C = 2 releases Project D = 2 releases
Domain of the projects under study	e-Commerce
Average duration of each release in a project	Project A = 56 days Project B = 67 days Project C = 38 days Project D = 38 days
Average number of resources utilized in each release of a project	Project A: Developers = 5 Database = 2 Quality Assurance = 3 SCM5 = 1 System Support = 2
	Project B: Developers = 17 Database = 3 Quality Assurance = 3 SCM = 1 System Support = 2
	Project C: Developers = 3 Database = 2 Quality Assurance = 2 SCM = 1 System Support = 2
	Project D: Developers = 3 Database = 2 Quality Assurance = 2 SCM = 1 System Support = 2
Technology used in the selected Projects	Project A IBM Net Commerce
	Project B Java/ J2EE
	Project C IBM Net Commerce
	Project D IBM Net Commerce
SDLC followed	Waterfall methodology
Communication Methodology with the onshore client	Conference calls, e-mails, meetings, telephone calls

Table-1: Data collected from organization under study

Further, for this study we have collected data against 30 releases of the four selected projects in e-Commerce domain by considering the following areas:

3.3.1 Requirements change

(Data collected from Functional Specification documents, Change Request Forms, Project Schedules)

- Pre-release and post-release CR's of high/ medium/ low severity against all releases of a project
- Pre-release and post-release CR's of high/medium/low severity initiated in different phases (specifications, design, coding, testing, shipment) of all releases of a project
- Requirement specifications (initial/pre-release/post-release) in all releases of a project

3.3.2 Defects

(Data collected from in-house Defect Repository System - Bug Base)

- Defects of high/medium/low severity against all releases
- Defects of high/medium/low severity due to pre-release and post-release CR's against all releases of a project
- Software Discrepancy Reports (SDR's)6 of high/low severity against all releases of a project

3.3.3 Project Duration

(Data collected from Quality Reports7, Project Schedules)

- Releases shipped on time/ with delay
- Duration (days) for each release of a project
- Time (days) allocated to different phases (Specifications, design, coding, testing) of a release
- For our investigation, it was not possible to collect data against the other factors affected by RC such as: project cost, size and effort, since the availability of data against them was one of the constraints. Also due to intellectual property protection issues, the fully functional system was not available to us and we could not store the data in persistent media. However, we had viewing access to the project documentation through the Configuration Management System for the duration of the study, which allowed us to record the data manually.

4. Discussion of Results

In this section we will discuss our findings based on the statistical analysis of the hypotheses

4.1 Hypothesis-1: Relationship between Pre-Release CR's and Defects

For hypothesis-1 we have combined all the releases of the four projects to determine if there is a relationship between the pre-release CR's and the defects. Our results in Table-2 indicate that there is a significant relationship between the number of pre-release CR's and defects since the significant value of the Chi-square test is less than 0.05. This proves that the two variables are not independent and our null hypothesis is rejected.

[5] **SCM:** Software Configuration Management

[6] **SDR:** It is the defect(s) in the software system that is reported by the client through formal SDR forms once the project/release has been shipped to the client.

[7] **Quality Report** is developed monthly by Quality Excellence Department that contains: project shipment details, planning and tracking details, project quality and productivity details and process management details.

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)
Pearson Chi-Square	11.526[a]	2	.003
Likelihood Ratio	8.391	2	.015
Linear-by-Linear Association	4.322	1	.038
N of Valid Cases	289		

a. 1 cells (16.7%) have expected count less than 5. The minimum expected count is 1.41.

Table-2: Chi-square results at significant level of 0.05

Figure-1 and Table-3 present the total number of defects found (categories: severity-1; severity-2) against the pre-release CR's (categories: low; medium; high). It is worth mentioning that these figures explain the combined results of all the 30 releases against 4 projects. However a detailed picture of percentage pre-release defect and CR's for each project is given in the Table-5.

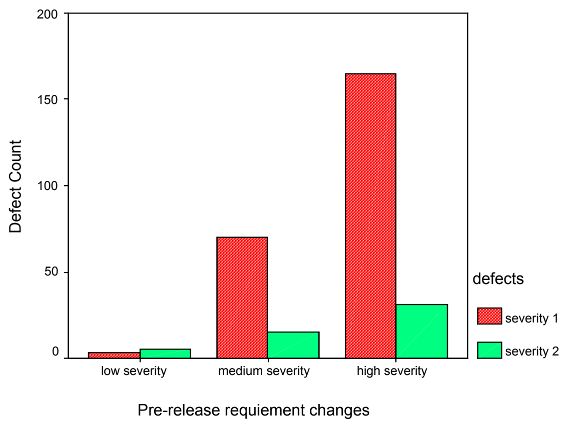

Figure-1: Defects versus no. of pre-release CR's

requiement changes * defects Crosstabulation

			defects		
			severity 1	severity 2	Total
requiement changes	Pre-rel sev. low	Count	2	1	3
		Expected Count	2.5	.5	3.0
	Pre-rel sev. medium	Count	72	15	87
		Expected Count	72.7	14.3	87.0
	Pre-rel sev. high	Count	165	31	196
		Expected Count	163.8	32.2	196.0
Total		Count	239	47	286
		Expected Count	239.0	47.0	286.0

Table-3: A cross-tabulation table displaying the number of defects in each category

Results in Table-4 & Table-5 show that maximum number of severity-1 defects (69%) is found due to a less number (37%) of high severity pre-release CR's. It is important to note that although majority of pre-release CR's are of medium severity (56%) but they only caused 30% of the severity-1 defects. Similarly, maximum number of severity-2 defects (66%) is found due to 37% of high severity pre-release CR's, although a major number of pre-release CR's (56%) are of medium severity but they only caused 32% of the severity-2 defects.

Overall Percentage of Defects of Severity-1 and Severity-2 due to Pre-Release CR's

Pre-Release CR's	%Severity-1 Defects	%Severity-2 Defects
Low Severity	1%	2%
Medium Severity	30%	32%
High Severity	69%	66%
Total	**100%**	**100%**

Table-4: Overall percentage distribution of defects due to pre-release CR's

Project-wise Percentage Distribution of Pre-Release CR's

Projects	High Severity CR's	Medium Severity CR's	Low Severity CR's	Total CR's	%age High Severity CR's	%age Medium Severity CR's	%age Low Severity CR's
Project A	9	34	3	46	20%	74%	6%
Project B	16	1	0	17	94%	6%	0%
Project C	1	5	2	8	12.5%	62.5%	25%
Project D	1	0	0	1	100%	0%	0%
Total	**27**	**40**	**5**	**72**			
Total %age	**37%**	**56%**	**7%**	**100%**			

Table-5: Project wise percentage distribution of pre-release CR's

These findings indicate that the high severity pre-release CR's have a significant impact on the occurrence of a majority of both the severity-1 and severity-2 defects. However medium and low severity CR's also contribute towards defects but their contribution is less as compared to high severity CR's. Some possible reasons to this conclusion are given below:

1. *High severity CR's are the ones that require changes in the design of the system*, as defined for this study. Such changes require major rework and may affect all the subsequent development phases. Due to these reasons, even very minor design changes can introduce high percentage of defects if ripple effects/bad fixes are not taken under consideration and sufficient time is not spent on quality assurance.

2. Second reason to the occurrence of severity-1 defects could be due to the *initiation of CR's in the later phases of the software development lifecycle.* Our data analysis presented in Table-6 illustrates that majority of the high severity pre-release CR's are initiated late during the development of all the four projects i.e., during coding and testing phases. However very few CR's are initiated during the development of RS, FS and design.

Pre-Release CR's initiated in different SDLC phases

Severity of CR's	Pre-Release CR's					
	Total No. of Pre-Rel. CR's	% RS	% FS	% Design	% Coding	% Testing
High	27	7%	4%	4%	33%	52%
Medium	40	0%	0%	2.5%	25%	72.5%
Low	5	0%	0%	0%	20%	80%

Table-6: Pre-Release CR's initiated in different phases of the SDLC

Ratio between pre-release CR's initiated in Coding phase and severity-1 bugs	
Pre-Release CR's initiated in Coding phase of all projects	20
Severity-1 defects caused due to pre-release CR's	38
Ratio	20:38 = 1:2
Percentage of severity-1 defects due to pre-release CR's during coding[8]	13%
Ratio between pre-release CR's initiated in Testing phase and severity-1 bugs	
Pre-Release CR's initiated in Testing phase of all projects	46
Severity-1 defects caused due to pre-release CR's	196
Ratio	46:196 = 1:4
Percentage of severity-1 defects due to pre-release CR's during testing[9]	68%

Table-7: Percentage of CR's initiated during Coding and Testing phases

Table-7 shows that the average percentage of severity-1 defects introduced during coding and testing due to pre-release CR's is 13% and 68% respectively. Further the defects introduced late in the software development are difficult to eradicate and may cause a significant reduction in the overall *defect removal efficiency*[10] of the work product. The overall defect removal efficiency of Project A is 73% and that of Project B is 84%. Project C and Project D, however, have defect removal efficiencies of 100%. Therefore, the average defect removal efficiency of all the projects is 89%. Studies done by Capers Jones have revealed that top ranked companies such as AT&T, IBM, Motorola, Raytheon and HP achieved defect removal efficiency levels of 99%.

3. Another reason could be the average percentage of time spent in different SDLC phases. Our data analysis in Table-8 indicates that on the average only 15% of the time is spent on design of all the four projects. This may lead to an indication that the design phase is not allocated sufficient time due to which most of the high CR's, that affect the design of the system, are initiated in the later phases i.e., during coding and testing. As a result of these high severity CR's a majority of both the severity-1 and severity-2 defects are introduced.

Percent Average Time Spent (in days) in different SDLC Phases (Excluding the time spent in incorporating the CR's)					
Project	**Requirement Specifications**	**Design**	**Coding**	**Testing**	**Others (Post Shipment Reviews, Shipments, Installations etc.)**
Project A	28%	12%	23%	26%	11%
Project B	32%	18%	25%	17%	8%
Project C	17%	15%	28%	20%	17%
Project D	16%	16%	33%	33%	2.5
Overall Average Duration	**23%**	**15%**	**27%**	**24%**	**10%**

Table-8: Percent Average Time Spent (in days) on different SDLC Phases (Excluding the time spent in incorporating the CR's)

4.2 Hypothesis-2: Relationship between Post-Release CR's and Defects

For hypothesis-II, Table-10 indicates that there is a significant relationship between the number of post-release CR's and the defects since the significant value of the Chi-square test is less than 0.05. This proves that the two variables are not independent and our null hypothesis is rejected.

Chi-Square Tests

	Value	df	Asymp. Sig. (2-sided)
Pearson Chi-Square	23.774[a]	2	.000
Likelihood Ratio	21.282	2	.000
Linear-by-Linear Association	1.009	1	.315
N of Valid Cases	1400		

a. 0 cells (.0%) have expected count less than 5. The minimum expected count is 20.00.

Table-9: Chi-square results at significant level of 0.05

Figure-2 and Table-10 present the total number of defects found (categories: severity-1; severity-2) against the pre-release CR's (categories: low; medium; high). *It is worth mentioning that these figures explain the combined results of all the 30 releases against 4 projects. However a detailed picture of percentage pre-release defects and CR's for each project is given in Table-12.*

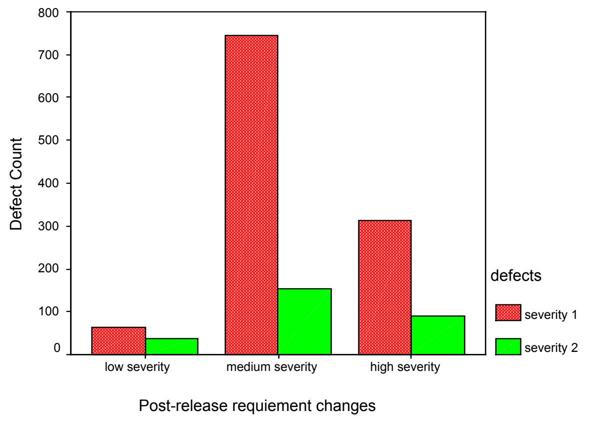

Figure-2: Defects versus no. of post-release CR's

requiement changes * defects Crosstabulation

			defects		
			severity 1	severity 2	Total
requiement changes	Post-rel sev. low	Count	63	37	100
		Expected Count	80.0	20.0	100.0
	Post-rel sev. medium	Count	744	154	898
		Expected Count	718.4	179.6	898.0
	Post-rel sev. high	Count	313	89	402
		Expected Count	321.6	80.4	402.0
Total		Count	1120	280	1400
		Expected Count	1120.0	280.0	1400.0

Table-10: A cross-tabulation table displaying the number of cases in each category

Results in Table-11 & 12 show that maximum number of severity-1 defects (66%) is found due to 71% of medium severity post-release CR's. However it is worth mentioning that only 15% of the high severity post release CR's have caused 28% of the overall severity-1 defects, which is comparatively a high defect rate per

[8] (Severity 1 defects due to pre-release CR's in coding phase/Total defects due to pre-release CR's) * 100 = (38/286)* 100 = 13%

[9] (Severity 1 defects due to pre-release CR's in testing phase/Total defects due to pre-release CR's) * 100 = (196/286)*100 = 68%

[10] **Defect Removal Efficiency** = Total defects found during development/ (Total defects found during development + Defects reported by Customer in 1 year after deployment) * 100

CR as compared to the previous one (a high severity CR can cause two severity-1 defects, whereas a medium severity CR can cause one severity-1 defect). Similarly, maximum number of severity-2 defects (55%) is found due to 71% of medium severity post-release CR's but is comparatively less defect rate per CR as compared to 15% of high severity post-release CR's causing 32% of the severity-2 defects (a high severity CR can cause about two severity-2 defects, whereas a medium severity CR can cause one severity-2 defect).

Overall Percentage of Defects of Severity-1 and Severity-2 due to Post-Release CR's		
Pre-Release CR's	**%Severity-1 Defects**	**%Severity-2 Defects**
Low Severity	6%	13%
Medium Severity	66%	55%
High Severity	28%	32%
Total	**100%**	**100%**

Table-11: Overall percentage distribution of defects due to post-release CR's

Percentage of Post-Release CR's							
Projects	**High Sev CR's**	**Medium Sev. CR's**	**Low Sev. CR's**	**Total CR's**	**%age High Sev. CR's**	**%age Medium Sev. CR's**	**%age Low Sev. CR's**
Project A	21	185	33	239	9%	77%	14%
Project B	18	17	0	35	51%	49%	0%
Project C	6	12	4	32	19%	37.5%	12.5%
Project D	1	4	0	5	20%	80%	0%
Total	**46**	**218**	**37**	**311**			
Total %age	**15%**	**71%**	**12%**	**100%**			

Table-12: Project wise percentage distribution of post-release CR's

These results indicate that high severity post-release CR's have a more profound impact on severity-1 and severity-2 defects than medium severity post-release CR's. However medium and low severity CR's also contribute towards defects but their contribution is less as compared to high severity CR's. There could be many reasons to this conclusion but here we have stated only those that we have found through data analysis.

1. Our findings indicate that 81%[11] of the CR's, for all the 30 release of four projects, are initiated once the project is shipped to the client (post-release) and only 19%[12] of the changes are initiated before release. Furthermore, 81% of post-release CR's have caused 67%[13] of severity-1 defects and 16%[14] of the severity-2 defects. On the other hand, 19% of pre-release CR's have only caused 14%[15] of severity-1 defects and 3%[16] of the severity-2 defects. These figures indicate that majority of the CR's are initiated by the client once the system is released and are the major source of severity-1 defects.
2. One reason to this high percentage of post release changes is the lack of communication between the client and the development side. Either the client is not getting proper feedback at major system milestones or the client is not taking much interest till the final product is ready for deployment. Both these issues add towards the post release changes and subsequently the defects. This information was gathered through an interview with one of the project managers of the organization under study.

5. Conclusion

In this paper we have presented some preliminary results based on a research work analyzing the impact of both the pre-release and post-release requirements change on *overall defects* by defining measures, collecting data against those measures and analyzing the collected data through statistical techniques. Also, we have reported some of the possible causes of requirements changes and in turn software defects.

Prior studies **[17]** have examined the relationship between changing requirements and defect density at the coding phase and found that the requirements volatility has an impact on defect density. However, to our knowledge, little prior research has been done to specifically examine the impact of changing requirements (categorized as: high; medium; low) on defects (categorized as: severity-1; severity-2) throughout the SDLC and the root causes of requirements changes.

Our study is based on industry data collected from 4 projects, all in e-commerce domain, consisting of 30 releases. Results indicate that there is a *significant relationship between pre/post release change requests and overall defects*. In addition, our data analysis indicates that changes in the design of the system at the later stages of software development i.e., during coding, testing and after release have a significant impact on the high severity defects that affect the major functionality of the system. Also, we found that insufficient time spent on the design phase and inadequate communication with the client could be some of the reasons for requirements change and consequently software defects.

Like most other researches in the context of requirements changes, this study also has several limitations. Due to the intricacy of the requirements change phenomenon and the scarcity of empirical evidence available, there is a need to validate our findings by considering projects from different domains and explicitly controlling people-related factors, such as development expertise in a particular domain and the communication methodology with the stakeholders.

[11] (No. of post release CR's / Total no. of CR's) * 100
(311/ 383) * 100 = 81% (From Table-5 and Table-12)

[12] (No. of pre-release CR's / Total no. of CR's) * 100
(72 / 383) * 100 = 19% (From Table-5 and Table-12)

[13] (No. of post release severity-1 defects / Total no. of defects) * 100
(1120 / 1686) * 100 = 67% (From Table-3 and Table-10)

[14] (No. of post release severity-2 defects / Total no. of defects) * 100
(280 / 1686) * 100 = 16% (From Table-3 and Table-10)

[15] (No. of pre-release severity-1 defects / Total no. of defects) * 100
(239 / 1686) * 100 = 14% (From Table-3 and Table-10)

[16] (No. of pre-release severity-1 defects / Total no. of defects) * 100
(47 / 1686) * 100 = 3% (From Table-3 and Table-10)

Acknowledgements

We would like to thank Professor Muhammad Hanif Mian and Usman Khan from National College of Business Administration and Economics (NCBA&E) for providing help in the statistical analysis of the hypotheses; Waqas Zyad and Aamir Majeed of CresSoft (Pvt.) Ltd., for solving queries regarding the projects under study and giving maximum resources to help accomplish our work.

References

[1] Lamsweerde, A. (2000): Requirements engineering in the year 00: A research perspective. *In proceedings of the 22nd International Conference on Software Engineering (ICSE'2000), Limerick, Ireland,* 5-19, ACM Press

[2] Hyatt, L. and Rosenberg, L. (1996): Software Metrics for Risk Assessment. *International Academy of Astronautics (IAA) 29th Safety and Rescue Symposium, Risk Management and Assessment Session, Beijing, China.*

[3] Barry, E. (2002): Software Project Duration and Effort: An Empirical Study. *Information Technology and Management* 3: pp. 113-136.

[4] Zowghi, D. (2002): A Study on the Impact of Requirements Volatility on Software Project Performance. *In proceedings of Ninth Asia-Pacific SE Conference (APSEC' 2002), IEEE Computer Science.*

[5] Grady, R. (ed) (1987): *Software Metrics Establishing a Company Wide Program*, Prentice Hall, New Jersey, USA.

[6] Kan, S. (ed) (2002): *Metrics and Models in Software Quality Engineering,* Addison-Wesley Publishers.

[7] Jones, C. (ed) (1997): Software Quality: Analysis and Guidelines for Success, International Thomson Computer Press.

[8] Davis, A. (1993): Identifying and Measuring Quality in Software Requirements Specifications. *In Proceedings of First International Software Metrics Symposium*, Baltimore, pp. 141-152.

[9] Pfahl, D. and Lebsanft, K. (2000): *"Using Simulation to analyze the impact of Software Requirements Volatility on Project Performance"*, Information and Software Technology, 42, pp. 1001-1008.

[10] Davis, A. (2003): The Art of Requirements Triage. *IEEE Computer Science* 36(3): pp. 42-49.

[11] Collier, B., DeMarco, T. and Fearery, P. (1996): A defined process for Project Post Mortem Review, *IEEE Software* 13(4): pp. 65-72, IEEE Computer Society Press.

[12] Weber, M. (2002): Requirements Engineering in Automative Development: Experiences and Challenges. In proceedings of *IEEE Joint International Conference on Requirements Engineering (RE'02)*, 331.

[13] Pressman, R. (ed) (2001): *Software Engineering: Practitioners Approach*, pp. 243-297, McGraw-Hill Companies, New York, USA.

[14] Lam, W., Shankararaman, V. and Saward, G. (1999): Requirements Change: A Dissection of Management Issues, *EUROMICRO'99 Workshop on Software Process and Product Improvement, Milan, Italy,* pp. 2244-2251.

[15] Lane, M. and Cavaye, A. (1998): Management of Requirements Volatility Enhances Software Development Productivity. *In proceedings of the 3rd Australian Conference on Requirements Engineering (ACRE 98), Geelong, Australia.*

[16] Start, G., Skillicorn, A. and Ameele, R. (1998): An Examination of the Effects of requirements changes on software Releases. *In CROSSTALK, The Journal of Defence Software Engineering.*

[17] Malaiya, Y. and Denton, J. (1998): Requirements Volatility and Defect Density. *In proceedings of the 10th International, Symposium on Software Reliability Engineering, Fort Collins,* pp. 285.

[18] Harker, S., Eason, K. and Dobson, J. (1993): The change and evolution of requirements as a challenge to the practice of software engineering. *In proceedings of the IEEE International Symposium on Requirements Engineering,* pp. 266-272, *San Diego, California, USA, IEEE Computer Society Press.*

[19] Zowghi, D., Offen, R. and Nurmuliani, N. (2000): The Impact of Requirements Volatility on Software Development Lifecycle. *In proceedings of the International Conference on Software, Theory and Practice (ICS2000), Beijing, China.*

[20] Zowghi, D. and Nurmuliani, N. (1998): Investigating Requirements Volatility During Software Development: Research in Progress. *In proceedings of the 3rd Australian Conference on Requirements Engineering (ACRE98), Geelong, Australia.*