Transfer of Knowledge Pod before Code Construction for Effective Defect Management

Bhagavant Deshpande, Dr Suma V

Abstract- The software industry is facing multiple challenges to reduce the cost of production of software and to increase the quality of production. This can be achieved by working smartly through better models of process than applying conventional practices. Since, high quality software has one of the dimensions being defect-free, it is required for all software developing organizations to ensure development of software with minimal or negligible defects. To address defects, there are several strategies which all organizations are following in their developmental process. However, there are still defects which get injected during the process and make the software to be not up to the satisfaction of the customers. Therefore, an empirical investigation is carried out in various software industries in order to study the impact of pre-production defects. Investigation results have further led towards introduction of knowledge pod as an integral part of software development process. This paper however has brought out the need for integration of knowledge pod before code construction phase of software development process. Implementation and stringent follow up of knowledge pod ensures reduced defect injection rate and hence leads towards developing high quality software resulting in total customer satisfaction.

Keywords— Software Engineering, Software Life Cycle, Software Quality, Defect Detection and Prevention, Software Testing, Software Quality Assurance and Control

Introduction

Since, high quality software has one of the dimensions being defect-free, it is required for all software developing organizations to ensure development of software with minimal or negligible defects. The basic reason being this as objective of software organizations is to ensure total customer satisfaction. Any organizations who fail to attain customer satisfaction will not be able to continue their business in the market. Hence, main motto of all organizations in the business market is to develop customer satisfied software products [1][2].

- *Bhagavant Deshpande is a research scholar at JJTU Rajasthan in computer science and engineering in India. E-mail: deshcapricorn@gmail.com*
- *Dr Suma V is currently, Head of Dayanandsagar Research and Industry incubation Center India. E-mail: sumavdsce@gmail.com*

To address defects, there are several strategies which all organizations are following in their developmental process [3][4]. However, there are still defects which get injected during the process and make the software to be not up to the satisfaction of the customers. There are various reasons as to why the defects get injected at every phase of software development. Poor performance of business analysts makes requirements engineering process not to be perfect. Political reasons, economical conditions, standards, and technology to name a few are some of the reasons why requirements engineering are not successful. Further, requirements misunderstood, ambiguity, misconception, mis-interpretation, amalgamation of requirements, poor understanding of the concepts, unaware of the concepts etc also adds to make requirements engineering process a not successful phase. Due to these mistakes in requirements collected, understood, analyzed and specified, defects gets injected at the phase of requirements engineering process in the software development.

Defects gets injected in design process due to architectural decision decisions, lack of domain knowledge of the designers, lack of expertise and knowledge of metrics, measurements and relationships

between the components in the system, makes design to be flaw full. Portability issues, interoperability issues, platform issues, dependency issues and other such lack of quality attributes in the design makes the design to have defects.

Since, requirements defects gets propagated if unnoticed, it gets into design and ultimately into the code construction phase. Though, steps have to be formulated to reduce this propagation, this research aims only at defect reduction at code construction phase such that ultimately pre-production defect count is reduced. Further, this knowledge in code construction phase enables one to at least take measures to remove all defects introduced at previous phases of software development.

Therefore, this research focused upon understanding the defect and its various facets during software development process thereby enabling to provide an effective strategy to reduce defect injection rate during pre-production activities.

Literature Survey

In order to comprehend defect facets, it is essential to have knowledge of defect pattern at various phases of software development. Since, defect has the tendency to propagate and magnify, early defect detection prevents defect migration from requirements phase to design and from design phase into implementation phase [5]. It enhances quality by adding value to the most important attributes of software like reliability, maintainability, efficiency and portability [6].

[7]Spiewak and McRitchie suggest that the best practice of identification and fixing of process defects enable one to achieve the product quality. However, all defects are not of same nature and thus do not have same impact on the quality of the product. Defect Prevention is one of the most significant activities in the software development process. An analysis of defects at the early stage reduces the time, cost, and the resources required for rework [8][9]

The consciousness of defect patterns enables to identify majority of defects close to defect inception point. In fact Li Meng, Xiaoyuan He, and Sontakke Ashok [10] emphasize on defect prevention. Ching-Pao Chang, Chih-Ping Chu, and Yu-Fang Yeh [11] explain defect prevention as the prevention of defect occurrences in advance and that it is not early defect detection.

Research Methodology

This research has conducted an empirical investigation on various software industries. Since, population of software is huge and industries developing software is also huge, this research narrowed down to matured industries where defect management process is well in implementation, Thus, all software industries investigated were certified as CMMI Level 4 and 5 industries. Also, in order to reduce the type of projects that got developed in these industries, this research further narrowed down to analyze non-critical applications. The reason behind this is critical applications will not have defects as they are threat to life and value of the country such as medical software projects, defense projects involving software components, space crafts, and missiles etc where software is involved. Therefore, this research aimed at investigating only non-critical applications.

Yet again, this population of non-critical applications is also so huge that this research again narrowed down towards investigating applications from the domain of healthcare, retail and telecom projects. The research started with collecting data for these projects under the assumptions of projects being developed using common technology, platform, programming language, tools and process models. The data was collected from various sources as defect prevention centers, logs, developers, project managers, quality assurance team and so on. Data was collected using telephone, mails, personal contacts, interviews and face to face communications. Having obtained the data using the templates made for data collection, the data was analyzed from the perspective of customer satisfaction, project success and so on.

Research Work

The aim of this research is to comprehend the impact of pre production defects during software development with the main area of focus to be on impact of existence of high severity defects as defect leak. Defect leak are defect escape from pre production cycle to post production cycle.

Table 1 depicts randomly sampled projects from the telecom domain. The table provides information about total project development time, cost for developing the entire project, complexity of the project, number of defects captured during the production cycle, number of defect classification, number of defect escapes, number of customer reported defects and customer satisfaction index of every project. Projects in the table are arranged in ascending order of total project development time since complexity is same for several projects.

Table 1 Pre Production defect profile for Telecom domain projects

Domain	**Parameters**	**Project-1**	**Project-2**	**Project-3**	**Project-4**	**Project-5**	**project-6**	**Project-7**	**Proejct-8**	**Project-9**	**Project-10**
Telecom	Project hours of development (*)	1260	1390	1460	1475	1890	2140	2850	3250	3440	4100
	Cost (**)	1400	1400	1800	1400	2100	16270	2900	3700	3200	4200
	Complexity (***)	3	3	3	3	3	3	4	4	4	4
	# of defects captured	67	72	79	62	91	84	92	107	99	114
	# of defects classification	6P1	9P1	13P1	8P1	14P1	14P1	17P1	19P1	19P1	21P1
	# of escapes	3	2	4	2	4	3	5	5	5	5
	# customer reported defects	1	1	1	1	2	1	2	3	3	3
	# customer satisfaction index(CSI)	9.3	9.2	9.1	9.3	9.2	9.2	9.2	9.1	8.9	9

(*)- Measured in Man hours; (**) – Measured in US Dollars; (***) – Measured on a scale of 1 to 5

Inferences from Table 1

It was observed from the projects that success of any project depends upon customer satisfaction index. Customer satisfaction index is all the time evaluated in these industries with a rate of 1 to 10 where 1 indicates extremely poor customer satisfaction level and 10 indicate total customer satisfaction. A level of 9 and above is always expected in all highly established software industries who have sustained in the dynamic market. An index level of 8 to 9 indicates acceptable band while below 8 is rework case and projects are termed as challenged projects. However, a customer

satisfaction index rated anywhere up to 7 is deemed to be not tolerable in software industries. Hence,

$$Project\ Success \propto Customer\ Satisfaction\ Index$$

Eq(1)

However, from investigation of these data, it was found out that customer satisfaction index is further depending only on defect count. This is because, whenever time increased, cost increased also customer satisfaction index was high. High indicates 9 and above. Thus, it was inferred from the data collected randomly over retail, telecom and healthcare projects that customer satisfaction in these industries especially which are at level CMMI 4 and 5 have no troubles with time or cost or even resources but it only matters with defect counts. Since, these industries are well established and hence human resource and technology support is adequate and therefore time, cost is not an issue and hence customer satisfaction level due to time, cost is not a criteria to be looked into. Thus, this research inferred that

$$Customer\ Satisfaction\ Index \propto \frac{1}{Defect\ Count}$$

Eq(2)

Further, this research progressed to analyze if complexity of the project also has an impact on defect count. Complexity is measured using either function points or use cases in all the industries. It was found out that as complexity increases defect count also increases but not exponentially. Thus, this study now directed towards analyzing the type of programmers who write code and the defect injection possibility from them. From the study, it was observed that whenever project complexity is either small or medium, the proportion in which developers are allocated in the project team has a pattern.

Further, this research moved to investigate the experience level of programmers. Table 2 depicts the randomly sampled telecom projects and their personnel information. Projects P1 to P3 depicts randomly sampled projects of small complexity and Projects P4 to P5 represents medium complexity projects. These projects are enhancement type of projects which are of type change request. Table further provides information about the number of programmers used, their experience in terms of largely experienced numbers and average experienced number of programmer5s who are put into the project.

Table 2. Sampled Projects with project type and programmers profile at implementation phase of software development

Projects	P1	P2	P3	P4	P5
ProjDesc	Telecom	Telecom	Telecom	Telecom	Telecom
Technology	Window7	Window7	Window7	Window7	Window7
Prog Lang	Java	Java	Java	Java	Java
Total Projdevp time(*)	800	1200	4000	6200	8400
Project Complexity	Small	Small	Small	medium	medium
Project Type	Change Request	Change Request	Change Request	Change Request	Change Request
Total No. Progm	5	8	16	19	22
No. of Lr.Exp. Progm	1	2	3	3	5
Percentage	20%	25%	18.75%	15.78%	22.72%
No.of Avg.Exp. Progm	2	2	5	7	8

Percentage	40%	25%	31.25%	36.84%	36.36%
No of Fresher	2	4	8	9	9
Percentage	40%	50%	50%	47.36%	40.90%

ProjDesc- Project Description; Prog Lang- Programming Language; Total Projdevp time- Total Project Development Time; (*)- measured in person hours; Total No. Progm – Total Number of Programmers; No. of Lr.Exp.Progm – Number of largely experienced programmers; No. of Avg.Exp.Progm- Number of average experienced programmers

It was observed that nearly 25 percent of developers in projects having medium and small complexity are developers having large experience. They are having an experience level of 8 and above. The team of developers also comprised of developers having experience between 2 to 8, who are average experienced developers. This proportion was found to be up to 40 percent in those projects. The project team comprised of fresher and less experience developers such as less than 2 years experience in a proportion of up to 50 percent in these projects.

Having obtained this pattern of developers allocation in projects by project managers, this investigation further directed towards knowing the type of defects they introduced during their code construction time. This is because all defects will not be of same nature. It was found that nearly 10 percent of defects injected by these programmers are blocker type in nature and blocks the entire application. Nearly 20 percent of defects introduced by the team of programmers having above-said proportion in their experiences were critical type of defects. In these industries, both blocker and critical type of defects were deemed to be hiving highest severity since they have ultimate impact on the project functioning. Hence, they termed these defects under the severity of P1.

Further, it was found from the data collection that nearly 40 percent of defects are major type and also 40 percent of defects are minor type. These defects are introduced by these set of programmers during their code developing span. Since, they do not block the functioning of applications, industries consider them under the severity level medium and call them as P2 type of defects. Programmers of this combination has also injected defects which are trivial in nature and has a cosmetic effect which means they do not have any high or medium impact on functioning of the applications. They are up to 40 percent in proportion and hence such defects are called as P3 type of defects.

From the investigation of the data collected across projects, it was found that customer satisfaction index was depending on defect count. However, when this investigation further got into its depth, it was found out that customer satisfaction index do not just depend on defect count but it ultimately depends on the type of defect. Thus, it was apparent that

$$Customer\ Satisfaction\ Index \propto \frac{1}{P1\ types\ of\ defect\ count}$$

Eq(3)

Now, it was still under investigation to find out as to how this P1 type of defects address customer satisfaction index. Thus, digging more into the research, it was found that as defect escapes increases P1 type of defects decreases. This is because defect escape is last opportunity where quality team and user acceptance team can unearth defects under the criteria of pre-production defects. Subsequent to this testing is the product being installed and deployed at the customer's location. Thus, defect escapes enable one to weed out the pre-production defect after which the defects when identified in the customer's site are termed as customer reported defects. This customer reported defect count and type of customer reported defect influences customer satisfaction index. Thus, it was found that

$$Defect\ Escape\ Count \propto \frac{1}{P1type\ of\ Cusomter\ Reported\ Defect}$$

Eq(4)

Thus, as more number of defects are captured during pre-production phase, lesser is the number of customer reported defects. However, it is the type of customer reported defects which influence customer satisfaction and hence, it is important to ensure that defect escapes capture most of the P1 and also P2 type of defects in order to ascertain total customer satisfaction since P3 defects reported by customers are hardly observed in these projects. There were few projects which had variations to the above made inferences since variances are within the acceptable framework of the application. All parameters cannot be objectively assed and some should be subjectively accessed such as documents. Documents should be evaluated for non ambiguity.

Integration of Knowledge Pod before Code Construction for Reduction of Pre-Production Defects

Further, moving ahead with the research investigation on the sampled collected projects from the sampled industries, it was necessary to analyze these inferences against the root cause analysis.

This research thus directed towards RCA (Root Cause Analysis) for the observed defect pattern with the combination of the programmers in the team. Table 3 thus provides RCA for the pattern observed. These listing are very generic and the list goes elaborated in perspective of applications.

Table 3. Root Cause Analysis for defects injected during code construction phase

Sl No	Root Cause Analysis
1	Improper understanding of requirements
2	Missed requirements
3	Coding standards are not followed
4	Syntax errors
5	Errors such as browser compatibility
6	Operating system compatibility
7	System errors
8	Errors due software version
9	Plug in- third party software
10	Logical errors
11	Missed functionality
12	Regression error
13	Insufficient unit testing
14	Unit test results not recorded
15	Unit test defects not fixed
16	Insufficient unit test cases
17	Test data

18	UI defects

Table 3 infers that defects injected by the combination of experienced, average experienced and less experienced programmers in the sampled projects are due to various reasons. Some of the common reasons are listed in the table which acts as an awareness module for the programmers not to inject such defects. Further, this table throws light on investigating the severity of these type of defects. Thus, the investigation led towards analyzing the defects causes which leads to defect types. Table 4 provides an insight of severity of defects when defects introduced are due to the above listed reasons.

Table 4. Severity of defect

Sl No	Root Cause Analysis	Severity of the defect
1	Improper understanding of requirements	P1
2	Missed requirements	P1
3	Coding standards are not followed	P2
4	Syntax errors	P1/P2/P3
5	Errors such as browser compatibility	P2
6	Operating system compatibility	P2
7	System errors	P2
8	Errors due software version	P2
9	Plug in- third party software	P2
10	Logical errors	P2
11	Missed functionality	P1
12	Regression error	P2/P3
13	Insufficient unit testing	P2
14	Unit test results not recorded	P3
15	Unit test defects not fixed	P2
16	Insufficient unit test cases	P2
17	Test data	P3
18	UI defects	P3

Table 4 infers that whenever defects are due to improper understanding of requirements, missed requirements, syntax errors, missed functionality and such type of reasons, the defect injected if not detected in pre-production cycle will lead towards customer reported defects having the severity P1. This certainly brings down the customer satisfaction index. Table further infers that whenever defects are due to reasons such as coding standards not followed, syntax errors, errors due to browser compatibility, operating system

compatibility, system errors, errors due to software version, plug in errors due to third party software, logical errors, regression errors, insufficient testing leading to defect residual, unit test defects which were left fixed and defects due to the reasons of insufficient unit test cases attribute towards P2 type of defects if undetected during pre-production phase and if they are identified and reported by the customers when the product is in the operational state in their locations.

From the table 6 it is further understood that defects which are introduced by the combination of programmers in the team which may result due to syntax errors, unit test results failed to be recorded, insufficient unit test cases, type of test inputs chosen for testing and user interface defects are cosmetic in nature. Hence, these defects needs to be detected in pre-production phase such that though they are P3, it should not result as customer reported defects.

This knowledge further enabled this research to investigate on the probability of which type of programmers injects what type of defects. This team of programmers as analyzed in all the projects comprises of largely experienced developers, average experienced developers and less experienced or fresher combination of programmers. Hence, it was required to find out which type of programmer may inject what severity of defects. The deep investigations carried out in these industries across the sample of projects led towards emerging of probability of type of defects being introduced by the type of programmers. Table 5 provides the probability of type of defects injected by the type of programmers.

Table 5. Probable type of defects injected by programmers

Sl No	Root Cause Analysis	Severity of the defect	Type of programmers
1	Improper understanding of requirements	P1	Largely experienced
2	Missed requirements	P1	Largely experienced
3	Coding standards are not followed	P2	Average experienced / Less experienced
4	Syntax errors	P1/P2/P3	Average experienced / Less experienced
5	Errors such as browser compatibility	P2	Average experienced / Less experienced
6	Operating system compatibility	P2	Average experienced / Less experienced
7	System errors	P2	Largely experienced / Average experienced
8	Errors due software version	P2	Average experienced / Less experienced
9	Plug in- third party software	P2	Largely experienced / Average experienced
10	Logical errors	P2	Less experienced
11	Missed functionality	P1	Largely experienced / Average experienced

12	Regression error	P2/P3	Less experienced
13	Insufficient unit testing	P2	Average experienced / Less experienced
14	Unit test results not recorded	P3	Less experienced
15	Unit test defects not fixed	P2	Less experienced
16	Insufficient unit test cases	P2	Largely experienced / Average experienced
17	Test data	P3	Average experienced / Less experienced
18	UI defects	P3	Less experienced

Table 5 infers that programmers who have their experiences in various levels may inject probable type of defects. The basic reasons for this mode of defect injection are the type of modules allocated to them for development. Experienced developers are provided with responsibilities' which are at a higher end and hence their lack of awareness leads towards severe type of defects.

This research therefore provides a solution where every developer should be provided with the awareness checklist which indicates the probable type of defects, their impact levels and probability of the programmers injecting them. Figure 1 illustrates the integration of knowledge pod before code construction for reduction of pre-production defects during software development process.

Benefits of knowledge pod when integrated in software development process

- According to this new approach where the knowledge pod needs to be integrated into the software development process especially after design specifications are provided and before code implementation actually happens.
    - This knowledge pod is a checklist mode of document which has to be provided to all programmers irrespective of their experience levels since it is found out that even an experienced programmer also injects defects of high severity.
    - Further, developers of all levels of experience are prone to introduce defects which can be prevented only through awareness of the type of defects,
    - This knowledge pod therefore acts as a catalyst where it directs all programmers about the type of defect which are probable to be introduced by which type of developers and hence precautionary actions to be taken not to execute the same.
    - Knowledge pod further acts as an indicator for developers to know if they had introduced such type of defects in earlier projects and if so to enable them to overcome them in further stages of development
    - Knowledge pod acts as a measurement scheme for developers individually and in team to check their performance efficiency
    - Knowledge pod also enlightens them and acts as a travel light to proceed with further code implementation activities using this as a best practice to be strictly followed
    - Knowledge pod also helps management team to access the efficiency of individual developers and team in any project and thereby process visibility can be obtained
    - Thus, knowledge pod is both a qualitative and quantitative framework which enables the software organizations to deliver software products having minimal or negligible pre-production defects and further enables to produce customer satisfied products.

- Integration of knowledge pod in the software development process reflects the process maturity of the company
- Implementation and stringent follow of knowledge pod ensures continual process development and assured customer satisfaction.

Thus, this research finally concluded with coming out with a novel approach where knowledge pod comprising of possible type of defect knowledge in association with the probability of its severity on customer satisfaction index and also the most likelihood of knowledge on the type of programmers committing the type of defect is brought out. This act as a solution to the problem caused due to post-production defects and its impact on customer satisfaction index. This solution is also a strategy to be followed to reduce pre-production defect during software development process.

Acknowledgement

The authors would like to thank all the industry personnel who have helped in coming out with this research under the framework of Non Disclosure Agreement.

Conclusion

Software has gained its highest peak of significance in all domains of human life style. Therefore, software companies generating software products should ensure developing of high quality software systems. Since quality can be visualized in several ways, it is always essential to look at that angle of quality which has highest impact on customer satisfaction level. Defect-free software is one such methods through which customer satisfaction can be achieved at maximum level. However, developing defect-free software especially in non-critical applications is always a dream. This is because due to the impact of devastation and loss that gets in critical applications, industries ensure defect-free software components in such applications.

This research therefore aimed at analyzing the impact of defects in non-critical applications for which a deep investigation was carried out in several software industries. It was proved that defect count alone is not a factor to influence satisfaction of customers, but it is the type of defect which is a concern. Proceeding in this direction, it was essential to know how these defects are distributed in the code construction phase of software development since our areas of focus was more on people than on process.

Hence, further research indicated that the distribution pattern of programmers in code construction phase comprised of up to 25 percent largely experienced developers while up to 40 percent of team were made of average experienced programmers. The project developer team also consisted of up to 50 percent being either less experienced developers or fresher in code construction activities.

This understanding of team distribution in code construction phase made this research to progress to explore the various root causes for the type of defects being introduced. The knowledge thus gained through this research is put forth in the form of knowledge pod and is suggested to be integrated in the software development process. Integration of knowledge pod after design specifications but before the start of actual code construction by the programmers enables these programmers to gain awareness and transfer knowledge to them to comprehend the type of defects and the probability of them getting injected in their hands. Upon the gain of knowledge, it is a measurement for these developers to ensure that they do not introduce such defects and thereby provide a qualitative code than just a quantitative code. This research has its limitation where knowledge pod is applicable only to non-critical applications such as retail, telecom and healthcare projects. The work is applicable on platforms and developing environments as studied in this entire research on the sampled projects. The work is further limited to legacy projects where they can be further undertaken for maintenance or enhancement purposes. The knowledge pod is provided only to programmers and not across all the project personnel.

References

[1] Suma V. and Gopalakrishnan Nair T.R.: Effective Defect Prevention Approach in Software Process for Achieving Better Quality Levels, International Conference on Software Engineering, WASET, Singapore, August 29-31, September 01, 2008.

[2] Suma V.andGopalakrishnan Nair T.R.: Enhanced Approaches in Defect Detection and Prevention Strategies in Small and Medium Scale Industries, IEEE 3rd International Conference on Advances in Software Engineering, Sliema, Malta, October 26-31, 2008

[3] Suma V. and Gopalakrishnan Nair T.R.: Better Defect Detection and Prevention Through Improved Inspection and Testing Approach in Small and Medium Scale Software Industry, International Journal of Productivity and Quality Management (IJPQM), Vol. 6, No. 1, 2010, pp.71-90.

[4] T. R. Gopalakrishnan Nair, Suma. V, Pranesh Kumar Tiwari, "Analysis of Test Efficiency during Software Development Process", Submitted, 2nd Annual International Conference on Software Engineering and Applications (SEA 2011), Singapore, 12th -13th December 2011

[5] Bary Boehm and Victor R. Basili: Software Defect Reduction Top 10 List, IEEE Computer, Vol. 34, No.1, January 2001, pp.135-137
http://www.cebase.org/defectreduction/top10

[6] Brain Randell, Alexander Romanovsky, Cecilia M F Rubira, Robert J Stroud, Zhizue Wu and JieXu: Implementation of Blocking Coordinated Atomic Actions Based on Forward Error Recovery, Journal of Systems Architecture, Vol. 43, No.10, 1997, pp. 687-699.

[7 Spiewak R. and McRitchie K.: Using Software Quality Methods to Reduce Cost and Prevent Defects, CROSSTALK, The journal of Defense Software Engineering, Vol. 21, No. 12, 2008

[8] Bhagawant Despande, Jawahar J Rao and Suma V, "Comprehension of Defect Pattern at Code Construction Phase during Software Development Process", 3rd International Conference on Frontiers in Intelligent Computing Theory & Applications (FICTA), 14th-15th November, Bhubaneswar, India.**Index: Springer**, ISI Proceedings, DBLP, Ulrich's, EI-Compendex, SCOPUS, Zentralblatt Math, MetaPress

[9] Pattern Analysis of Post Production Defects in Software IndustryDivakar Harekal, , V. Suma Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014 Advances in Intelligent Systems and Computing Volume 328, 2015, pp 667-671

[10] Li Meng, Xiaoyuan He and Sontakke Ashok: Defect Prevention-A General Framework and Its Application, Sixth International Conference on Quality Software (QSIC'06), Beijing, China, October 27-28, 2006, pp. 281-286

[11]Ching-Pao Chang, Chih-Ping Chu and Yu-Fang Yeh: Integrating In-Process Software Defect Prediction with Association Mining to Discover Defect Pattern, Information and Software Technology Journal, Vol. 51, No.2, 2009, pp. 375-384