Abstract—Enterprise Resource Planning (ERP) software belongs to a class of extremely complex software. There are a million lines of codes written for it, as a result of which they are prone to errors. Most large software products have elaborate quality control processes, involving many tasks performed by different groups using a variety of techniques. The defects found are generally recorded in a database, which is used for tracking and prioritizing them. However, this data also provides a wealth of information, which can be analyzed for improving the process. Software defects are expensive and time consuming. Due to its coding errors, defects continue to plague the industry with disastrous impact on the enterprise application software category. The cost of finding and correcting these defects represents one of the most expensive software development activities. This paper highlights the importance of finding and analyzing defects.

Keywords—Enterprise Systems, Defect Analysis, Defect Prevention, Quality Control, Software Process Improvement

I. INTRODUCTION

An Enterprise Resource Planning (ERP) system is a business management system that comprises of integrated sets of comprehensive software, which can be used, when successfully implemented, to manage and integrate all the business functions within an organization.

Enterprise Resource Planning software (ERP) belongs to a class of large-scale software known as enterprise application software. These complex, expensive, powerful, proprietary systems are off-the-shelf solutions, requiring consultants to tailor and implement them based on the company’s requirements. They are expensive to build, debug and maintain.

Some researchers theorized that in the 70s, although there was a vision to create an enterprise application, software to integrate individual information systems within organizations, computing capacity and programming language capabilities were too limited to cause such applications to be built. Instead, organizations constructed “islands of automation”, meaning that a new application was built for each individual with information processing needs within the same organization.

However by the 80s, the vision of enterprise applications began to be realized, although somewhat slowly, as organizations were able to interface the individual information systems into a homogeneous one.
Mid-1990s was a turning point for the development and adoption of enterprise application software, since more and more organizations embraced the client-server architecture over the traditional and comparatively expensive mainframe architecture. They also stated that by the year 1998, approximately 40% of all organizations with annual revenues of more than one billion dollars had implemented some form of enterprise application software.

As stated before, enterprise application software is a complex software application and is prone to a large number of defects. The enterprise application software is written in a generic manner to suit the general needs of various organizations (each of which had their own individual requirements). When any of these organizations actually purchased any particular enterprise application software, the software had to be heavily customized to meet the organization’s specific requirements and those customizations often led to defects.

In 2006, the enterprise application software market was valued at US$100 billion, but software failures were estimated to cost the industry between US$20 billion and US$50 billion. Although various aspects of the software were defective, the coding defects were a significant concern.

On one hand, efforts have been made to find practical ways to verify that the ERP and other types of software are free of defects. On the other hand, complete software verification for software with large code bases, such as ERP software have proved to be unpredictable and theoretically intractable. To gain a deeper understanding of the effectiveness of the software process, it is essential to examine the details of defects detected in the previous projects and to study how the same can be eliminated due to process improvements and newer methodologies.

This paper will focus on the area of software development through development models and provide a comprehensive view on defect prevention techniques and practices that can be followed in software development.

ERP applications often use a single, centralized data repository for all modules. Therefore, when these applications have performance issues, they can potentially impact all areas of business that use the common repository. This connectivity creates the need for robust testing and monitoring programs to improve the health of applications, critical to a company’s mission. When testing ERP applications, one should always take a holistic approach.

One more aspect which shows why organizations lay emphasis on testing and defect analysis is the market competition. ERP solution providers worldwide face fierce competition in a relatively limited market for their products, especially in a country like India. These type of software systems are expensive and their deployment requires specific know-how and extensive due diligence. Normally, integration with existing software and Customer Relationship Management systems (CRM) can be challenging, especially when trying to integrate with outdated software platforms. This can be handled by using well-established integration tools and experienced vendors.

India’s software vendors are usually more flexible than their Western peers, which allows them to react faster and offer tailor made solutions, when different ERP software modules are required to feature specific functionality, or some particular modules need to be added or excluded from the software. Ramco Systems Limited is a classic example of successful Indian Software vendor capable of providing quality software products.

With growing competition and economic crisis, it becomes imperative for ERP vendors to deliver their products on time with high quality standards. However, building excellent enterprise application software requires tremendous effort, costs significantly more, and takes much longer to launch in the market. As a result, many enterprise application software developers, in an effort to reduce costs and development time, build software that are “just good enough”, instead of being excellent.

It has been observed that in the competitive software market, the challenge facing software development companies was either to release poor quality software to the market early, or to release high quality software late. The challenge in deciding when the software was “good enough” to be released into the market place was sometimes based on “gut feeling”, but several mathematical defect prediction techniques existed that could aid in the process. These included tracking defects, defect pooling, defect seeding, and defect modeling.
Business process transactions have an impact on many departments and divisions within a company, and on various modules within the ERP application itself. When validating process functionality, it is important to capture test samples of these business processes within automated testing solutions for rapid repeatability. The inherent complexity of crossing the multiple business stacks makes it critical to functionally test every ERP application.

During software development, defects are likely to emerge. With the advent of SDLC processes, many companies have formulated their own defect prevention mechanisms and various studies have been conducted to predict and prevent defects.

II. IMPORTANCE OF TESTING ERP SOFTWARE

As described earlier, Enterprise Resource Planning software (ERP) belongs to a class of large-scale software called enterprise application software and is highly complex. According to an estimate, SAP R/3 is one of the most complex software as it has 250 million lines of codes written for it. Along with this, the customized version of the software available for different customers makes it more vulnerable towards performance issues, as there is a trade-off between satisfying customer demands and increasing codes. In order to avoid potential errors that may affect key application functionalities and disrupt critical business processes, organizations are required to perform comprehensive tests before going live.

Traditionally, organizations have been dealing with this challenge by using a manual trial-and-error approach to test their customizations. However, the shortcomings of this method are exacerbated as the ERP system becomes more complicated. This includes, for example, an inability to detect all problems, mounting costs due to increased utilization of development and testing resources, extended implementation time, and inability to accurately calculate project timeline and costs.

To tackle the shortcomings of traditional approaches, there is a growing need for solutions to automate testing processes. These solutions should provide comprehensive capabilities such as impact analysis of customizations and changes, as well as automatic detection of potential problems across the entire application lifecycle – from development to configuration changes, maintenance, and upgrades. However, the balance between the uses of manual and automatic testing also depends on the organization structure.

Changes must be tested in a controlled environment that imitates production operations to ensure that the critical business operations are not interrupted when a change is being implemented. It should also ensure that product-related changes do not negatively affect how the application supports the business. The following are the challenges faced while testing ERP software:

- **Complex Technology:** Requiring complex implementations with specific programming and configurations, and industry specific business processes.
- **Broader Scale:** Multi-geography projects requiring deeper and wider levels of enterprise-scale software test management experience.
- **Integration Challenges:** Requiring integration with newer as well as legacy and third-party applications.
- **Product Upgrades and Interim Releases:** Application vendors going in for interim market releases and product upgrades that require on-going testing, modeling of relevant business scenarios as an integrated suite of automated test cases.
- **Performance Issues:** Lack of a comprehensive performance test plan, leading to performance issues at various levels (application, database, load characterization, user roles, critical paths, ramp rates, tools, protocols, injection points, load origination).

Organizations have been following both, a manual testing approach and an automatic testing approach. A manual testing approach is a trial-and-error approach that requires lengthy and cumbersome manual testing processes.
Various researchers and authors have highlighted the inefficiency of this testing method as one of the main reasons that the upgraded projects often result in mounting costs, delays, and system outages due to undetected errors. However, the use of automatic testing also depends on the organization structure and can be in sync with manual testing.

Ramco System’s Aviation ERP software follows the same approach. Once the manual testing is over, automatic testing is carried out to identify bugs before the system goes live. This approach has helped in stabilizing Ramco Aviation software by finding bugs at different stages of ERP implementation, after the initial phase of testing is over.

III. DEFECT ANALYSIS AND PREVENTION

Software quality problems can rapidly increase the costs of software maintenance and development. Moreover, the cost of finding and correcting defects represents one of the most expensive software development activities. While these defects may be inevitable, we can minimize their number and impact on our projects. Often, there are different ways of looking at software defects.

For instance, if a user makes an error in using the software, or if the software is used incorrectly, then it will not behave as expected. However, the people who design and build the software can also make mistakes during the design and build. These errors are called defects or bugs. Software defects can be defined as “Imperfections in software development process that would cause the software to fail in meeting the desired expectations”.

The process of managing problems and defects includes a large number of difficulties and challenges but has been given little consideration in software engineering research.

From a review of literature of various authors some research variables have been:

- Programming language: This factor involves choice of programming language and programming language paradigm, as well as the reason for using the selected programming language to develop ERP software.
- Customization: This factor represents the level of changes made within a specific period.
- Complexity: This factor gives an indication of the size of ERP software in terms of its McCabe cyclomatic complexity.
- Epistemological commitment: This factor involves decisions made by the developers of ERP software with respect to satisfying requirements and development time.
- Code auditing: This factor involves the application of manual code auditing, automated code auditing, or a combination of both, to ERP software.
- Legal requirements: This factor involves the perception of the developers of an ERP system that they should be legally required to restrict the defect density of their ERP software to no higher than the industry average.
- Testing and tracking: This factor refers to the use of regression testing and tracking of defect density before and after the testing process.

Thus, it is very important to understand the nature of defects and carry out defect prevention techniques. Two key factors in detecting defects are the quality of teams performing the quality control tasks and the process/techniques being used. Understanding their effectiveness will clearly help in identifying improvement opportunities. The other dimension of defect analysis is the three dimensional when–who–how analysis.
The ‘when analysis’ takes into the fact that analyzing defects over a timeline is not a reliable method as development is continuous and distributed, and phases like design and test are not demarcated clearly in time. Authors have suggested instead a milestone-based approach as these milestones are clearly defined in time for a product, and typically the entire code is synchronized at these milestones, making it easier to use them for dividing the timeline. A possible set of milestones could be as follows:

- Preview – when the product is “released” for preview.
- Beta1 – the first Beta release of the product.
- Beta2 – the second Beta release.
- Beta3 – the third Beta release.
- RC – final release check.
- RTM – release to manufacturing.

The milestone dates and the timestamp of the defects can be used to classify the phase of a defect. It is expected that the number of defects in each phase will come down.

The ‘who analysis’ involves a different team performing some QC tasks as the constituting category. The teams may be divided into ‘Internal’ team and ‘Other’ team which may include people from different areas of software development, and in some cases may involve customers. Researchers have found surprising patterns for this approach. Other teams find more defects in a particular component than the internal team. This clearly highlights internal testing of the components as an area of improvement.

The third important factor in analyzing the QC processes to analyze the effectiveness of the different techniques being used. How a defect is identified, is generally captured in the database. To do a meaningful analysis, we should classify them into a few categories which are mutually exclusive but collectively exhaustive. One possible categorization could be:

- Reviews – Any kind of review or inspection.
- Code Analysis – Any static analysis tool.
- Self-Host – Special testing done when a group self-hosts the system.
- Test Pass – Regular testing.
- Stress – Stress and performance testing.
- Customer – External and internal customers.

The how and who analysis can easily be combined with the ‘when dimension’ to understand how the distribution of defects change with phases. Once the three dimension analysis is done for each component, it can be aggregated for the product. This again allows analysis of component level defects, which will help in the Quality Control process.

One such method of analysis is the ‘Early and Late defect relationship’, where the whole purpose is to explore the relationship between early defect measures and late defects found in the components. The idea is to establish a relationship that can determine early in the product cycle, the components that are most likely to have late defects.

This early warning can then be used by product managers to take suitable actions that will minimize the volume of late defects. Predicting defective components has been an active area of research for quite some time now. As said earlier, it has been found in research that the maximum numbers of defects are found by ‘Other’ test teams rather than the testing team. This gives the opportunity to test out all the components with the help of other test teams and analyze the data there on. Although this method is not feasible, it has a large impact on specification quality.

Defect Prevention is an important activity in any software project. The purpose of Defect Prevention is to identify the cause of defects and prevent them from recurring. 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.
Defect Prevention can be applied to one or more phases of the software lifecycle to improve the software process quality. Implementation of defect preventive action not only helps in developing a quality project, but is also a valuable investment. The benefits of adopting a defect prevention strategy would be enormous. To list a few, defect prevention would reduce development time and cost, increase customer satisfaction, reduce rework effort, decrease costs and improve product quality.

To study the prevalence of defect in software development process, one project was identified in a software company. The project was basically an enhancement of existing functionality of the product module. When projects on enhancement of existing products take place, it becomes all the more important, since existing functionality and integrity of components may get affected. Different organizations have different approaches towards the release of the product enhancement and all related activities. Similarly, in this project, product enhancement was released in two versions of the product.

The strategy was to release the enhancement in one of the versions and proceed with unit testing. Once the defect fixes are reviewed and deployed in the tested version the enhancement is moved to another version, for another round of unit testing. Multiple levels of testing across releases help to bring down the count of defects. Although this approach seems to be time consuming the quality of the product delivered is of prime importance. Development involved both, Agile and Waterfall methodologies. These type of customizable approaches are being followed by organizations because of the tremendous time constraint and pressure in delivery of quality product.

Software quality metrics are a subset of software metrics that focus on the quality aspects of the product, process, and project. In general, software quality metrics are more closely associated with process and product metrics than with project metrics. Nonetheless, the project parameters such as the number of developers and their skill levels, the schedule, the size, and the organization structure certainly affect the quality of the product.

<table>
<thead>
<tr>
<th>Lifecycle Phase</th>
<th>Activity</th>
<th>Defect Description</th>
<th>No. of Defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirements (REQ)</td>
<td>Review</td>
<td>Evaluation of business requirements from user point of view; and also for inadequate requirements definition.</td>
<td>10</td>
</tr>
<tr>
<td>Design (DSN)</td>
<td>Review</td>
<td>Evaluation of Technical requirements and system architecture with regards to user requirements. Error in developing design, or inadequate design, or technical inadequacy in design.</td>
<td>6</td>
</tr>
<tr>
<td>Code (CDE)</td>
<td>Review</td>
<td>Evaluating and fixing errors in the initial stages of code development by authors of code or design owners.</td>
<td>10</td>
</tr>
<tr>
<td>Unit Testing (UT)</td>
<td>Test</td>
<td>Unit/Component/Module Testing for verifying functioning of software and is done in isolation from the rest of the system. It can include both, functional and non-functional testing.</td>
<td>13/4</td>
</tr>
<tr>
<td>System Testing (ST)</td>
<td>Test</td>
<td>Integration/Regression/Acceptance/System testing which is concerned with the behavior of whole system/product defined by the scope of development project.</td>
<td>9/0</td>
</tr>
</tbody>
</table>

Table 1: Defect Description and Count of Defects.
Table 1 shows the type of defects considered throughout the product enhancement development project. Defects are usually categorized as Fatal, Major or Minor depending on the priority attached to fixing the bug. However, to gain a deeper understanding of the effectiveness of the software process in developing any product/application, it is essential to examine the details of defects more minutely.

Orthogonal Defect Classification (ODC) is a concept, which enables developers, Quality Managers and Project Managers to evaluate the effectiveness and completeness of the verification processes. For small and medium projects, in order to save time and effort, the defects can be classified up to first level of ODC, while critical projects and typically large projects need the defects to be classified deeply in order to analyze and understand defects.

The project considered for this paper comes under the category of small and medium size project; the analysis of defect is done by using first level of ODC defect classification. First level of ODC includes classifying the defects under various defect types like Requirements, Design, Logical (Logical defects are found by testing the code using functional/unit testing), and Documentation. Defects are classified under these types and then analysis of defects is carried out.

Figure 1 illustrates the defect type pattern across the project. It was found that nearly 55% of the defects were found during testing, 24% of defects were found during design and code review and the rest were requirements review defects.

After defects are logged and documented, the next step is to review and analyze them using root cause analysis techniques. Before rRoot Cause Analysis (RCA), a Pareto Chart was prepared.

Pareto analysis seeks to discover from an analysis of defect reports which “vital few” causes are responsible for most of the reported problems. The old adage states that 80% of reported problems can usually be traced to 20% of the various underlying causes. By concentrating your efforts on rectifying the vital 20%, you can have the greatest immediate impact on product quality.

Figure 2 illustrates the Defect Pareto Chart. From the Pareto Chart it is clear that 80% of the defects are falling under the category of unit and system testing. These defect types should be given higher priority and must be attended first. From Pareto principle, it is clear that putting more effort on Design and Code Review can lower the number of defects found during testing.

SRoot Cause Analysis is a class of problem solving methods aimed at identifying the root causes of problems or events. The practice of RCA is predicated on the belief that problems are best solved by attempting to correct or eliminate root causes, as opposed to merely addressing the immediately obvious symptoms. By directing corrective measures at root causes, it is hoped that the likelihood of problem recurrence will be minimized.

There are several methods of carrying out Root Cause Analysis, often involving brainstorming ideas and discussing them. Same concept was followed while carrying out Root Cause Analysis for the project. A standard brainstorming procedure was followed to do Root Cause Analysis. Based on the Pareto analysis which states that more emphasis should be given to Requirements, Design and Code Reviews, all the possible causes for these defects were identified and debated among the team and all suggestions were
Most software product development efforts ensure that defect data is logged regularly as this data is essential for product release decisions. For large products, the defect database may have tens or even hundreds of thousands of records in it. Though a considerable amount of information about each defect is recorded, this rich data is often not used for evaluating and improving the elaborate quality control processes that the products usually utilize.

The traditional methods for analyzing defect data are generally not suitable as they assume a waterfall-type model, while the product development is almost always iterative. Implementation of defect preventive action not only helps in giving a quality project, but is also a valuable investment.

Training needs to be conducted for two main categories of people, existing developers and potential developers. For existing developers, who are involved in developing, testing or auditing ERP software, new specialized training courses will be needed to rapidly build capacity in the developer organization, so that these professionals can work diligently to reduce the defect density of their software, under the oversight of the industry collaborative group.

New and existing training programs (e.g. software engineering, programming, and software quality assurance courses at tertiary and other institutions) will need to be designed or redesigned to introduce to and build competency among potential ERP developers, testers and auditors. Defect prevention practices enhance the ability of software developers to learn from those errors and, more importantly, learn from the mistakes of others.

**IV. CONCLUSION**

Most software product development efforts ensure that defect data is logged regularly as this data is essential for product release decisions. For large products, the defect database may have tens or even hundreds of thousands of records in it. Though a considerable amount of information about each defect is recorded, this rich data is often not used for evaluating and improving the elaborate quality control processes that the products usually utilize.

The traditional methods for analyzing defect data are generally not suitable as they assume a waterfall-type model, while the product development is almost always iterative. Implementation of defect preventive action not only helps in giving a quality project, but is also a valuable investment.

Training needs to be conducted for two main categories of people, existing developers and potential developers. For existing developers, who are involved in developing, testing or auditing ERP software, new specialized training courses will be needed to rapidly build capacity in the developer organization, so that these professionals can work diligently to reduce the defect density of their software, under the oversight of the industry collaborative group.

New and existing training programs (e.g. software engineering, programming, and software quality assurance courses at tertiary and other institutions) will need to be designed or redesigned to introduce to and build competency among potential ERP developers, testers and auditors. Defect prevention practices enhance the ability of software developers to learn from those errors and, more importantly, learn from the mistakes of others.