Experiences in implementing Defect Prevention activities in Software Product Development Life Cycle

Tathagat Varma[1]*, Quality Manager,*
Philips Software Centre, Bangalore, India

Abstract

The maturity of a software development process or a software product creation process to help prevent defects in the final product at an early stage of the development has significant importance in modern times. An effective defect prevention process not only helps in reducing the "surprises" (read frustration or dissatisfaction) in the end-product, it also helps in keeping the overall cost of non-quality (i.e. wastage or rework) under constant supervision and control.

Most projects and project managers have problems in implementing defect prevention activities in an ongoing project. We have implemented defect prevention program in the maintenance phase of a product. The product in question had all possible 'excuses' against embarking on such a program –

- *absence of user-level or technical documentation of any kind,*
- *absence of formal reviews or well structured testing on the originally developed code,*
- *absence of original designers and developers,*
- *a rather unstable product, and*
- *very tight delivery schedules.*

We attacked the problem in multiple phases:

In the first phase, we first went about reverse engineering and 'documenting' the product as per our software processes at the same time when our organisation was briskly moving from SEI-CMM Level 2 to 3 (May '1997 to Dec '1997). The aim was not Level 3, the aim was to gain thorough understanding of the product. Level 3 was like the checkpoint on overkill on documentation, etc.

In the second phase, we consolidated our understanding by being able to repetitively deliver the software. The defect densities were very high, and we did not have any idea on how we were going to improve our performance in subsequent releases. We made feeble attempts to prevent defects. The success was very limited.

Somewhere near the end of second phase, we committed to define and deploy a Defect Prevention Program for our next major release. Unfortunately, the work on this next release got stalled, but we had a good process definition in place. We started using the principles of defect prevention in the then ongoing developments that helped us to reduce the defect densities, and ultimately, in Jun '1998, we delivered a software with no known defects.

This paper highlights the experiences gained by the author and his project team while implementing well-planned and structured defect prevention activities in his project during the second phase.

The Defect Prevention Program was carefully modelled as per the expectations of SEI-CMM Level 5 though the organisational maturity was at Level 3. This paper also addresses the additional elements that were introduced in the project's software process.

1.Background

In Q2 '97, we took over the product development ownership of a client-server architecture-based software product for workflow automation in radiology departments. It was developed in Sweden, and was developed by a group of competent designers and engineers. However, formal software processes were not deployed anytime during its developments, and when we took over the product, we were confronted with the following major risks:

1. There was no documentation on the Requirements, Design, or the Test cases (that the software passed or failed). The same also applies to inline documentation in the source code. Whatever little documentation was available, was in Swedish.
2. The source code was almost never peer reviewed.
3. The testing was never formally done. No one knew if the testing was 'complete' or not.
4. It was possible that some modules are never formally reviewed and / or tested before being delivered to the customer / end-customer.
5. The number of errors reported increased linearly with every hour of testing. We had more than 1000 defects after just one calendar-month of testing by six test engineers.

2.Initial effort at Defect Prevention

We planned the following activities to correct the situation fast:

1. Reverse Engineering – We documented major issues in top-level and detailed design from looking at the code. This was also reviewed by the original designers and developers and found to be alright. In parallel, we also took help of the original designers and developers in translating inline documentation for major modules to English.
2. Defect Analysis – We started working on establishing root cause behind major defects. After a few iterations, enough data on defects was available.

The information from reverse engineering was used to train the team members on the architecture and other related issues of the software.

The data emanating from the causal analysis was circulated to all the team members. All team members were expected to make sure that they do not make same mistakes in future. This strategy was not successful because:

1. It was, at best, a weak and unstructured effort to prevent defects.
2. We were still learning how to do a good defect analysis. We would often find new mistakes happening in the next round of a review or testing.
3. We were looking at pennies and not at pounds. The result was that while we were able to contain smaller defects of limited and acceptable damage, we did not know how to deal with the defects that had more impact.

However, the experiment was encouraging because:

1. It gave a confidence to us that defect prevention is possible, even in the absence of complete information on the product.
2. Our approach was fine, in general. We now only had to focus it further for 80/20 defects, i.e. those 20% of defects that took almost 80% of the rework effort.

In the next phase, we decided to formalise the defect prevention as per the expectations of SEI-CMM Level 5 KPA on Defect Prevention.

3.Defect Prevention Process

When we were planning to adopt CMM's key practices on Defect Prevention, we had following odds against us:

1. A Defect Prevention process was not available within PSC's software processes.
2. There was no information available within PSC on how to define and deploy such a process.
3. No such information was available from within other Philips organisations.
4. There was practically no information or industry data available, except for [1], [2], [3] and [4] and SEI-CMM version 1.1.

In the absence of any start-up information, we planned our approach in the following manner:

1. We studied the Level 5 KPA on Defect Prevention.
2. We defined the Defect Prevention plan for our project[2].
3. We performed a gap analysis to ensure that the definition of the plan was compliant with the key practices expected in the KPA.
4. We consolidated all defect analysis reports and prepared a checklist for each software process, like requirements, design, implementation, configuration management, etc.

5. This checklist was used during using the relevant process to ensure that same mistakes were not being made again.
6. The concerned team members used to have a meeting to kick-off a new activity. This meeting would focus on educating the team members on most common mistakes made by people in that phase and how to prevent them.
7. Before releasing the work product for appropriate quality control activity (i.e. review or testing) the co-ordinator would ensure that the checklist has indeed been satisfied.
8. During the review / testing, if a new type of defect was found, appropriate preventive actions were identified. The appropriate checklist used to be updated.

In a nutshell, this process gives the closed-loop mechanism in which we defined and deployed the Defect Prevention process. The following figure explains it:

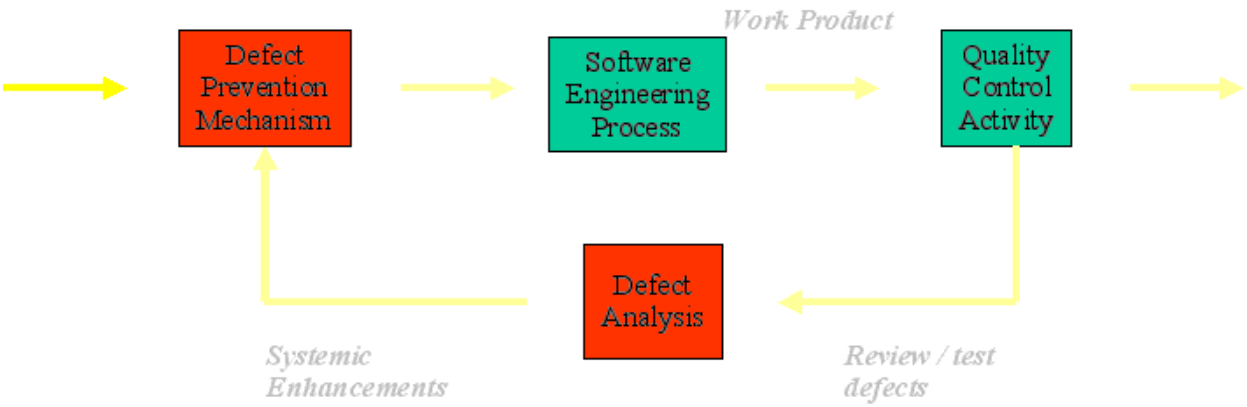

Figure 1: Defect Prevention Model

In the normal deployment of software processes, the process blocks “Defect Prevention Mechanism” and “Defect Analysis” are either missing or not properly planned. In such cases, the software engineering

processes never get improved because there is no feedback on the processes. Further, the review and test data is never used to find opportunities for process improvement.

In contrast, when these process blocks are well-planned, they help understand the effectiveness of processes, and create opportunities for continuous process improvements.

4.Defect Analysis

The objective behind defect analysis was to establish 80/20 kind of return-on-investment. We were interested to find systemic enhancements for those 20% of the defects that were responsible for 80% of the cost of non-quality.

Each defect was analysed for its following attributes:

1. Bug from – There are fair chances that the defect found was always existing in the original code or it was introduced by PSC.
2. Root Cause – The root cause of a defect could be (a) Systemic, or (b) Misexecution. In case of it being Systemic, it could further be because of (a) Incomplete / Incorrect process (b) Transcription error (c) Lack of Communication, or (d) Training. The cases involving misexecution were not considered cases for software process improvement.
3. Phase where it could have been trapped – Could be among (a) Requirements (b) Design (d) Implementation (e) Testing.
4. Preventive Action – What can be done to avoid such defects from happening in future. It is felt that every systemic root cause is an opportunity for process improvement. How can the process itself be improved to plug the gaps to make sure such a systemic defect can be prevented in future.

Consolidated results were recorded in a template as shown in Table 1. Corresponding systemic enhancements were captured in the following template:

Phase	**Systemic root causes**	**Preventive / follow-up action**
<Requirements> / <Design> / …	…	…

The following pie-chart shows the defect distribution based on root-causes:

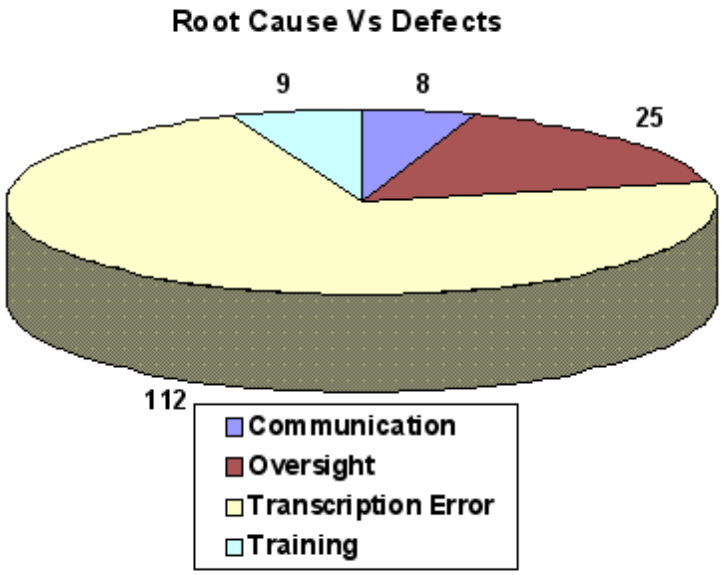

Figure 2: Distribution of defects as per systemic root-causes

5.Problems faced

The project was being executed under a tight schedule and often varying project priorities and delivery schedules. We were working on three parallel versions of the software – one was undergoing field runs at a hospital (and crashing every day because of some software bugs), another was undergoing maintenance at PSC, and the third one was partially under new development. Under normal circumstances, it would have meant sacrificing anything 'extra' to just take care of the immediate needs. However, we made sure that the process is never short-circuited. Some of the problems we faced and the steps we took to rectify them are:

1. External factors were daunting. We had to change plans very often, which almost meant scraping the results of a defect analysis, etc. (because we would not use the results again). However, we continued sticking to the plan and not dropping it midway.
2. PSC's software processes, though already at CMM Level 3 maturity, required adding certain extra process elements in a few processes. However, PSC's software processes allowed tailoring its processes for project-specific requirements, and so this was not a major issue.

Other than these, there were no real problems faced by the team members in carrying out this activity.

6.Lessons learnt

The major lessons learnt while planning and executing defect prevention activities were:

1. Defect prevention activities can be applied to any activity within the realm of software development.
2. If the willingness exists to accept the fact that we make mistakes that are controllable, defect prevention is not only possible, it will be very effective as well.
3. There is no “one-size-fits-all” solution for all projects. Every project must define its own mechanisms for defect prevention. There could be an organisation-specific defect prevention process and guidelines, but the finer details will vary for different projects.
4. There is an extra initial investment in performing activities related to defect prevention (like defect analysis, initiating systemic improvements, organising and attending various kick-off meetings, using defect prevention checklists, etc.). However, the returns are often not immediate. General recommendation is that one must give six months to start getting returns – if it is taking more than that, it is time to take stock of things.
5. In the retrospective, I feel that complete ownership of this activity by me (as the Project Leader) and the Software Quality Engineer was a critical success factor.

7.Results

In this section, we look at some representative results:

7.1Post-Release Defects

When the software was taken over, the curve between defects vs. testing time was linear. In one month of testing, over 1000 defects were logged.

The following graph indicates the trend in post-release defects or problem reports (PRs) :

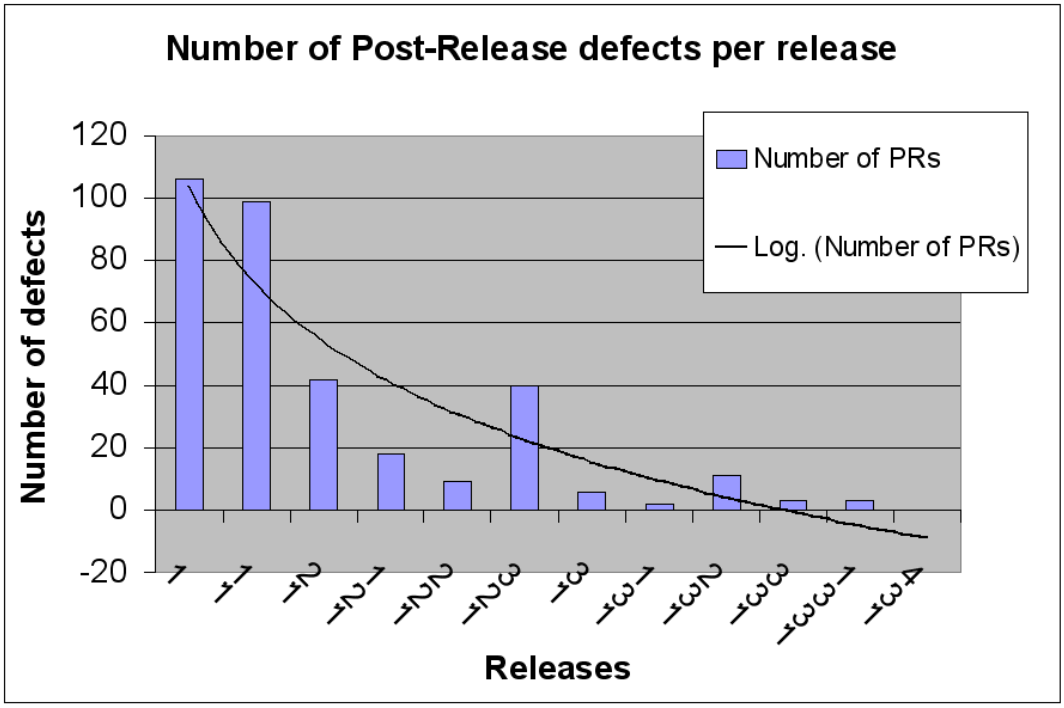

Figure 3: Post-release defects vs. releases

Initial versions (1.0, 1.1 and 1.2) were developed without using defect prevention techniques. We performed defect analysis, but we did not enforce preventive actions in successive developments. After release 1.2, we started using checklists for Functional Requirements Specifications, Detailed Design, Code reviews, Configuration Management, Integration, etc.

In the graph, however, we see the following behaviour:

1. There is a general trend in reduction of defects in each release. In 1.3.4, no defect has been reported. (The trendline has been drawn using logarithmic trendlining function of MS-Excel, and its value of less then zero has no meaning.)
2. There are sudden spurts in number of defects (1.2.3 and 1.3.2). These versions were tested by the end-user, and many situations under which the software was tested had never been simulated earlier. However, we do see that the defects ultimately come under control.

7.2Defect Distribution

The following table gives phase-wise and cause-wise defect distribution before initiating any of the defect prevention mechanisms:

Phase	Root Cause - Corresponding Errors	
Requirements	Lack of Communication	3
Oversight	1	
Transcription Error	5	
Training	0	
Design	Lack of Communication	2
Oversight	3	
Transcription Error	0	
Training	0	
Unit Testing	Lack of Communication	0
Oversight	0	
Transcription Error	19	
Training	5	
System Testing	Lack of Communication	3
Oversight	2	
Transcription Error	17	
Training	5	

Table 1: Defect Analysis data before deploying Defect Prevention mechanisms

The next table gives similar data for a version for which Defect Prevention mechanisms were deployed:

Phase	Root Cause - Corresponding Errors	
	Root Cause	**Total No. Of Errors**

Requirements	Lack of Communication	0
Oversight	0	
Transcription Error	0	
Training	0	
Design	Lack of Communication	0
Oversight	0	
Transcription Error	0	
Training	0	
Unit Testing	Lack of Communication	0
Oversight	2	
Transcription Error	15	
Training	0	
System Testing	Lack of Communication	0
Oversight	2	
Transcription Error	15	
Training	0	

Table 2: Defect Analysis data after deploying Defect Prevention mechanisms

7.3Configuration Management

Configuration Management was performed using a simple tool, CVS. Shell scripts were written to customise the tool for project-specific requirements like no multiple checkouts, etc. At any time, there were around 15 software engineers working on three parallel versions of the software, all with different build and release contents and delivery schedules. At times, a bug fix had to be implemented in all versions of the software. When we performed the initial releases, we faced a lot of problems in

controlling all these activities. We applied the Defect Prevention Model and came out with two major guidelines / checklists:

1. Checklist for Configuration Management activities
2. Checklist for Integration

Together, these checklists helped us ensure that all errors in integration were corrected and prevented to happen in future.

The following table gives the checklist used for Configuration Management activities:

S.No.	Activity	Yes/No
1.	Is the lead-time for preparing a test environment for System testing during a release is one man-day checked? (This includes preparing both an upgrade as well as a fresh installation environment.)	
2.	Has the SQA Engineer sent an e-mail stating that the code is ready for integration?(Integration can start only after that).	
3.	Is the repository locked during the system testing?	
4.	Is the bug fixes done and checked in only after one round of system testing is completed?	
5..	Are all the forms compiled after renaming?(psc_stock_yes.pll, psc_modality_yes.pll, psc_siap_yes.pll and psc_medrep_yes.pll to psc_stock.pll, psc_modality.pll, psc_siap.pll and psc_medrep.pll respectively.)	
6.	Is the c:\Project X\form directory checked for the files: Project X.ini.eng, Project X.ini.fr, Project X.ini.sw8?. Note that Project X.ini should not be present.	
7.	Does the c:\Project X\form directory have files: Project X.hlp, Project Xeng.hlp, Project Xfr.hlp?	
8.	Does the c:\Project X\menu directory has files: dc_menu.mmx.eng, dc_menu.mmx.fr and dc_menu.mmx.sw8?. Note that the file dc_menu.mmx should not be present	
9.	Does the c:\Project X\lib directory have the files: psc_stock_yes.pll, psc_modality_yes.pll, psc_siap_yes.pll, psc_medrep_yes.pll, psc_stock_no.pll, psc_modality_no.pll, psc_siap_no.pll and psc_medrep_no.pll?	

7.4Software Installation

In the area of software installation[3], the results of defect prevention were very evident. When we inherited the software, there were issues in installation that were not acceptable to the customer. Almost no installation would happen error-free. We realised that the root causes were uncontrolled changes, lack of coding guidelines and no formal reviews of the installation script. As a corrective action, we enforced coding guidelines, made installation scripts as a configuration item and made sure every change in it was controlled and reviewed before the installation script would form part of the baseline. As a preventive action, we also created a defect prevention checklist [5]. In mid-Q2 '98, we were able to release software that had zero defects in software installation

7.5Informal reviews

Another defect prevention mechanism we experimented with, was performing informal reviews on documents under preparation. This ensured that most of the major issues got rectified even before the document was released for review. Though purists of software processes might disagree on this approach, we felt this was very effective in exchanging early information on the work product under preparation, and help us contain the cost of eventual rework following a review.

8.Areas of further improvements

The project activities were transferred from Bangalore to another customer site in June '98. As a result, we could not implement one complete round of the development life cycle. The following tasks could not be completed:

1. We were still catching defects late in the life cycle. While we practically did not record any defects in the code reviews during the later stages of the project, unit testing (for newly developed modules) was a weak area. We were working on improvements in this area when we had to close the project.
2. We were able to contain the defect densities, but we were still not able to detect more defects at PSC before releasing to the user. In fact, the customer (i.e. our Philips Medical Systems customer) found 10 errors for every 1 error found by us. We attributed it to lack of training in the domain. Unfortunately, we could not improve this situation.

However, after successfully deploying defect prevention activities in other areas, we were confident that we could improve the overall process by applying similar principles.

9.Conclusions

The work done by the project team was instrumental in ensuring that the product is maintainable to the fullest possible. We were not only able to close all existing defects, we were also able to prevent many new and regression defects.

The project team was also able to demonstrate that it is possible to innovate in a runaway project. If takes time for such implementations to show improvements and the initial investment could be viewed as extra, unnecessary and unavailable, but it pays off in the long run.

The work done by our team was used as a basis by a Quality Improvement Team at PSC to define such a process at the organisational level. Currently, this process is being piloted by a project.

10.References

[1] "Defect Prevention" by Richard E Jones, Leeds Metropolitan University, 1996, available at www.yacc.co.uk/~rej/defects.html

[2] "Defect Prevention" by Sunita Menon, University of Houston, Clearlake, Society for Software quality Spring 1996 Grant Winner; available at www.ssq.org/grant/S96essay.html

[3] "Defect Prevention" by Paula Ashley available at www.processtrat.com/pp_5.htm

[4] "Three questions about each bug you find" by Tom Van Vleck, available at www.lilli.com/threeq.html

[5] "Defect Prevention Program for Project X" dated 06-Apr-98. Circulation Philips internal only.

[6] "Post-project evaluation report for Project X" dated 28-Jul-98. Circulation Philips internal only.

[7] "Causal Analysis Report of Post-Release Defects of Project X" dated 12-Jun-98. Circulation Philips internal only.

1 The author can be contacted at tathagat@blr.pin.philips.com

2 A sample Defect Prevention plan is available upon request within Philips. Please send e-mail to the author.
3 This software involves multiple migration paths from previous version of the software and involves creating / updating over 250 tables in Oracle. In addition, if existing data is available in an installation of a previous version, all such data needs to be migrated to the new version. In a real-life situation, all this needs to be done in a hospital without interrupting the services. Though

software installation might appear a trivial issue in normal cases, this was an important requirement for our project.