Driving Quality Improvement Through Root Cause Analysis

Amol Patil
apatil@mimecast.com

Abstract

This paper discusses the approaches used at a software company building solutions for healthcare payers . Achieving engineering process improvements and gaining confidence in quality led to the adoption of RCA as a primary driver for engineering best practices. This paper covers what worked effectively to introduce a structured RCA program and also discusses the challenges encountered. Key success factors and an overall methodology are highlighted that include;

- Scaling improvements for large, distributed engineering teams
- Customer satisfaction and its relationship with defect removals that can be used to trigger process improvements
- Effectiveness of improvements evaluated using metrics

Biography

Amol Patil is Director of Quality Engineering & Services at Mimecast. He is responsible for ensuring world-wide quality standards for Mimecast products that use A.I. and Machine Learning models for threat detection and analysis of emails and attachments. Prior to Mimecast, he held Quality Engineering leadership positions at Healthedge and PTC. He is a Software Engineering leader with experience in leading and scaling distributed engineering teams to deliver high quality customer focused solutions. At PTC, he was responsible for building geographically distributed Engineering teams with strong ownership and morale. At Healthedge, he has achieved high automation levels and shorter lead times using CI/CD, improved productivity and ultimately improved customer confidence and building corporate value. Amol has experience in many markets; Cyber Security, Healthcare, IoT, PLM, Mobile and cloud platforms and has been delivering new products and platform integration's using any cloud-based DevOps infrastructure leveraging Docker, Terraform and Kubernetes. Amol has a M.S. degree in Industrial Engineering from University of Cincinnati.

1.1 Opportunity for increasing the confidence in Quality

Software development teams should have a way of understanding what is most important to the customers and how they are doing with delivering great customer experiences. Teams unfortunately get that feedback through post-release defects. Teams then consume more than half their software development effort by applying it towards defect repair and testing. The CoPQ (Cost of Poor Quality) has already kicked in and a vicious cycle of engineering rework and rising customer care costs puts the relationship between customers, account managers and the Engineering organization into stress over the perception of quality.

Engineering adhered to all the practices that needed to be followed, like;

1. Understood the business needs
2. Brainstormed the solution
3. Implemented the solution
4. Examined the results through different levels of testing

All of this was achieved through an efficiently running software development engine that included

- Continuous integration
- Automated build and build verification
- TDD: Test Driven Development
- Automated regression testing

Engineering teams are now thinking about where it went awry.

This presents an opportunity for taking a step back and retrospect on why this happens and how the confidence in quality can be improved going forward.

1.2 Understanding the leading factors contributing to quality perceptions

In a typical enterprise software deliverable, there are major releases and minor releases.
The content and the date for receiving the major release are determined in advance and made available in the product roadmap. New features and business needs are addressed in the major release. Then, the minor releases are used to make improvements to the system that has been in production on the supported tech platform. In the minor releases, no new functions are added to the major release, and only improvements are made through bug fixing and software maintenance through software refactoring.

Now, when the real platform that houses the software in production has a large number of issues, the problems are visible for everyone to see it. Things start to get unsustainable when stakeholders cannot put a finger on why the system isn't functioning as expected. Doing a root cause analysis will provide the evidence and point in the direction of weak points. Engineering leaders can then drive outcomes based on that analysis.

I have always felt that, if you can solve a problem by asking the right questions the improvement path kind of makes itself visible. Let's start applying this questioning strategy beginning with engineering perceptions

1. Understood the business needs
   - What is the business routine that helps the customer generate revenue and how is it changing with this major release
   - Which systems are integrated upstream, midstream, and downstream that facilitate the customers business
2. Brainstormed the solution
   - Were all the components that constitute the system definition engineered to work together
   - Does the system scale
3. Implemented the solution
   - Was the software construction done consistently with best practices that consider functional and non-functional requirements
   - Was the system built with safeguards and default behavior
4. Examined the results through different levels of testing
   - Do the tests provide sufficient code coverage and scenario coverage
   - How effective are the tests to catch system level problems

A retrospective analysis of known defects can be applied to generate answers to all the above questions. Those answers should lead to gaps being identified and measures to improve defect avoidance. This type of root cause analysis and actionable improvements are being presented in this paper.

2.1 RCA Methodology

Following the principle that 'A defect in the software is a defect in the process', the leadership team decided to adopt the Orthogonal Defect Classification approach to understand and solve the underlying software problems and quality perceptions. The ODC technique was adopted as a methodology to characterize software defects and translate into process defects.

2.2 Orthogonal Defect Classification (ODC) technique

When successfully initiated, the ODC technique can be used for categorizing defects and reducing the cost of analysis based on predefined attributes. The main purpose of ODC is to extract semantic information from software defects to take actions against their re-occurrence to improve the process. In this sense, ODC can be used as a technique to realize the specific goals

Statistical KPI's like defect flow, defect density, defect remediation rates and test coverage are measured against test beds containing artificially created data and against a hardware footprint that is imperfectly sized. These types of KPI's do not set a relation to the system where defects originate and tend to fall short in identifying the root causes. A simple defect classification scheme like the one highlighted below provided distributions of defects against semantic data in each category.

Defect Origin → Defect Trigger → Defect → Defect Type → Fix Category

In this study of 200 defects reported and fixed in a span of 6 months in the 2020/2021 timeframe, the focus was on manually capturing and analyzing of defect data. The defect categories in this scheme are explained in the tables below

2.2.1 DEFECT TRIGGER:

No.	Defect Trigger Name	Description	Example
1	3rd Party Integration	Defect symptoms manifest when interaction happens with any external Integration	*Repricers, groupers, Claims editing systems*
2	Component Interoperability	Defect symptoms manifest when interaction happens with other components of the HE product portfolio	Platform, Care product, Custom code
3	Specific Data Condition	Data sources input had something unique or unexpected. System works as designed for majority of the input, but a certain small % of data fails functional coverage due to the distinctive nature of the data	*-- Payment cycle not correct* *-- Member links not updated after changes* *-- Validation policies not triggered* *-- Showing incorrect information in the UI interface* *-- Logic retrieving incorrect claim processing details*
4	Data Consistency	The business transaction failed to change affected data only in allowed ways	*-- Data processed differently by Payer Engine and the webservices* *-- Data replication / streaming. Mismatch / Missing records between OLTP and DW* *-- Mismatch in Source EDI and the Payer engine*
5	Selective Transaction	A few handful transactions out of many failed to complete or completed but not as expected. High material impact	*-- Items in the work basket. 1 claim, 3 members, 5 accounts etc.* *-- Items that put the HIC out of compliance* *-- penalties, interests*
6	Batch Transaction	Event driven / scheduled operational items that are part of regular business routines failed to complete. High % of the volume routed from auto-processing to manual processing.	*-- Adjudication sending large # of items to workbasket, payment batches not completing, member enrollment, Selective bootstrapping.* *-- Transient data clean up scripts, custom server scripts* *-- Manual re-adjudication/repair became necessary*
7	Software Upgrade	Codebase upgraded and end user tried to repeat a transaction that was working as expected in the previous version	*-- Job performance is not on par with pre-upgrade performance and is currently impacting regular business*

			-- Claims Search is taking over 2 minutes after upgrade -- Mapping for a field has changed -- Data Integrity between OLTP and DW
8	Configuration	Changes in configuration happened before the symptoms manifested	*-- Updating configurable parameters of the product to support new business functions*
9	Design	Logic Design and Data Design	*-- Index added to improve the performance, but caused contention during heavy inserts/updates/deletes of the attachments*
10	Volume	The business routines work as designed with expected results up to a certain threshold. Beyond the threshold, transactions and computing fails	*Search throws OOM error.*

2.2.2 DEFECT ORIGIN:

No.	**Defect Origin Name**	**Description**	**Example**
1	Customer Prod	Production environment	*-- PROD*
2	Customer Non-Prod	Non-Production environments but in control of the customer	*-- QA, Dev, SIT, Stress, Test, Pre-Prod*
3	Internal Non-Prod	Environments in control of the Software Engineering group	*Example: -- Engineering or Agile Services environments -- Perf CI environments*

2.2.3 DEFECT TYPE:

No.	Defect Type Name	Description	Example
1	System Integration	System integration with various 'external' components not native to the product under implementation	*-- Repricers, Fraud Detection* *-- Upstream loaders, Downstream replicated datastore* *-- Custom code*
2	Regression	Business routines under use stop working as expected after a change in functional configuration/customization/logic	*-- Payment cycle not correct* *-- Member links not updated after changes* *-- Validation policies not triggered* *-- Showing incorrect information in the client* *-- Logic retrieving incorrect claim processing details*
3	Performance Improvement	Functionality is not broken, but the code was refactored to show performance gains	*-- 'Out of Memory' problems* *-- Services responding slowly* *-- UI results responding slowly* *-- Concurrent calls to a single record (member/account/claim)* *-- Database locks and transaction locks* *-- Calling many more rows than required*
4	Negative Use Case	End users transaction was not conforming to the expected incoming data sources leading to un-expected outcomes	*User passed 1010-10-10 in the date range that expects date range to be within 1800-01-01 to 3000-01-01*
5	Func. Requirements not documented, but expected to work	Major functionality works, but specific scenario(s) do not work as expected	*These are scenarios or conditions that end up causing major business impact. Expected case not completely understood or missed*
6	Data Validation	Needs adjustment to the software logic that properly validates the data and values before used in computational logic or database storage	
7	Instructions Not Clear	End user followed the documentation, but for areas where there was ambiguity or lack of clarity, the end user expected the functionality to be supported as per the business use case	

2.2.4 FIX CATEGORY:

No.	Fix Category Name	Description	Example
1	User Interface	Fix addressed through UI changes	*Grid View changes*
2	Build/Package/ Merge	Solution existed but was not included due to procedural issues	*Rebuilt the package to include newer or missing libraries or missing LOC of known solutions not merged forward*
3	Functional Coverage	New functional logic introduced to handle missing or wrong functionality	*LOC that introduced logic to process and compute newer types that are known at time of requirements gathering*
4	Logic Coverage	Addressed an inadequate (efficiency) or wrong (correctness) algorithmic realization	*LOC that introduced new* *-- Service methods* *-- getAction()* *-- setAction()*
5	Defensive Coverage	Address poorly defined code boundaries and data validation for unexpected data resources	*LOC that introduced handling of* *-- Amounts set to 0* *-- null or !null values* *-- handling of terminated / missing / canceled / inactive / invalid states of transactions* *-- Service date mismatches* *-- Exception handling: NPE, NumberFormatException*
6	Checking	Affects program logic that would properly validate data and values before they are stored or used in computation.	*-- Matching approved conditions for Authorizations, Agreement details, Service provisions* *-- Initialization of control blocks or data structures*
7	Runtime Resource Handling	Addressed the code to handle proper management of shared and real-time resources	*Free resources at runtime* *-- Network connections* *-- database connections* *-- file streams* *-- occupied memory* *-- Timeouts, socket timeouts* *-- Serialization/multi-threading*
8	Database Query	Queries adjusted/introduced/enhanced to handle the case highlighted in the defect	
9	Database Design	Schema adjustments made	
10	Data Migration	Migration scripts modified/introduced to upgrade to a newer software version	
11	Documentation	Addressed the technical documentation for missing instructions/information	

3.1 Data Analysis

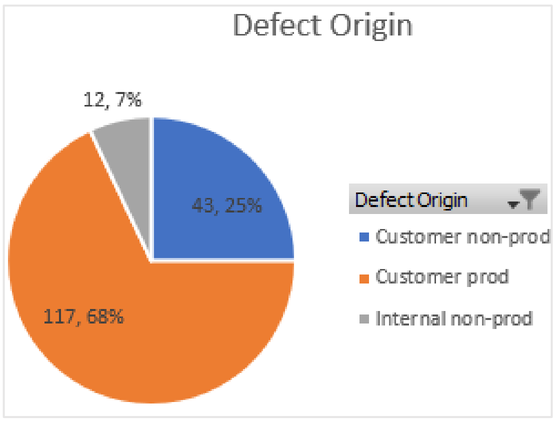
The defect set was chosen for a product that is subject to improvement.

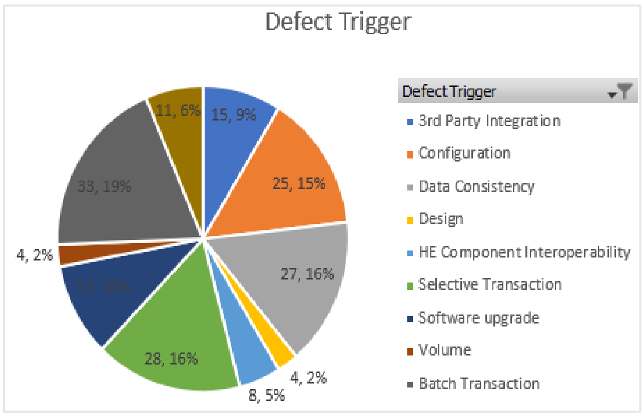
The data was prepared by analysis completed by a group of engineers that undertook training in the RCA program pilot implementation. The definitions were approved through a consensus method. The defect classifications for each category were recorded and discussions were held to clarify whether the process is proceeding according to the modeled process.

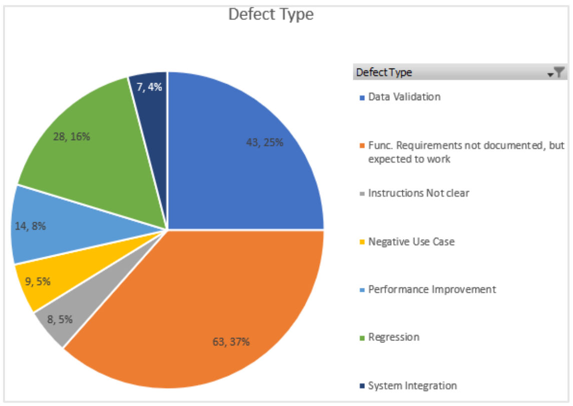
- The Defect Origin and Defect Trigger was obtained from the incident report management tool (*Salesforce*), activity information, and the person who reported the defect
- The Defect Type and Fix category was identified by analyzing the traces and explanations given for the correction of defects in the defect management tool (*Jira*) and the source code management tool (Bitbucke*t*)

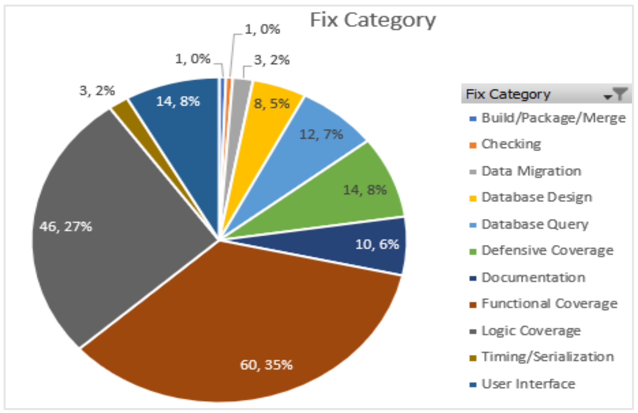
By profiling the defects using the classification scheme and analyzing the incidents, problematic areas and underlying process gaps bubbled up to the top as the likely causes of software defects.

The figures below show the distribution of each category









3.1.1: The figures below show the examples of distributions of Defect Types with respect to Defect Triggers

→ Defect Type ↓ Defect Trigger	Data Validation	Func. Requirements not documented, but expected to work	Instructions Not clear	No default behavior for negative use case	Performance Improvement	Regression	System Integration	Grand Total
3rd Party Integration	4	4	1	1		3	6	**19**
Batch Transaction	13	12			4	9		**38**
Configuration	3	14	4		3	4		**28**
Data Consistency	18	3	1	2		6	1	**31**
Design		2			1	1		**4**
Component Interoperability	1	3	1	1	2	3		**11**
Selective Transaction	7	15	1	3	4	6		**36**
Software upgrade	4	7		1	1	6	1	**20**
Specific Data Condition		11				1		**12**
Volume		1		1	2			**4**
Grand Total	**50**	**72**	**8**	**9**	**17**	**39**	**8**	**203**

3.1.2: The figures below show the examples of distributions of Defect Types with respect to Fix category

→ **Defect Type** ↓ **Fix Category**	**Data Validation**	**Func. Requirements not documented, but expected to work**	**Instructions Not clear**	**No default behavior for negative use case**	**Performance Improvement**	**Regression**	**System Integration**	**Grand Total**
Build/Package/Merge						2		**2**
Checking		1				1		**2**
Data Migration	2	1				1		**4**
Database Design	2	1			3	2		**8**
Database Query	1	5			2	4		**12**
Defensive Coverage	18			1				**19**
Documentation	1	1	8			2		**12**
Functional Coverage	9	37		1	7	8	5	**67**
Logic Coverage	16	20		4	2	12	2	**56**
Timing/Serialization	1				1	1		**3**
User Interface		6		3		6	1	**16**
Runtime Resource Handling					2			**2**
Grand Total	**50**	**72**	**8**	**9**	**17**	**39**	**8**	**203**

3.1.3: The figures below show the examples of distributions of Defect Triggers with respect to Fix category

→ **Defect Trigger** ↓ **Fix Category**	**3rd Party Integration**	**Batch Transaction**	**Configuration**	**Data Consistency**	**Design**	**HE Component Interoperability**	**Selective Transaction**	**Software upgrade**	**Specific Data Condition**	**Volume**	**Grand Total**
Build/Package/ Merge			1					1			**2**
Checking		1				1					**2**
Data Migration		1					1	2			**4**
Database Design		1			2		2	3			**8**
Database Query		6		1			2	3			**12**
Defensive Coverage		7		12							**19**
Documentation	2	1	4	1		1	1	1	1		**12**
Functional Coverage	7	12	10	6		3	12	6	8	3	**67**
Logic Coverage	9	6	8	9	1	5	13	3	2		**56**
Timing/Serialization			2	1							**3**
User Interface	1	3	3	1	1	1	3	1	1	1	**16**
Runtime Resource Handling							2				**2**
Grand Total	**19**	**38**	**28**	**31**	**4**	**11**	**36**	**20**	**12**	**4**	**203**

3.2 Root Cause Disposition

The educated evaluation from the data analysis will help reach conclusions on the root cause or the best fit for the root cause. Some examples;

- ↓ Defect Type of ‘Requirements not documented, but expected to work’ could not be discovered in the requirements intake process and therefore reached the codebase as conditions that could not be satisfactorily handled. This is a gap in the requirements gathering and requirements breakdown stage. Source – 3.1.2
- ↓ The Fix Category of ‘Logic coverage’ not handling ‘Data validation’ type of defects indicates that Design, and Code reviews were found inefficient. This engineering gap needs to be addressed. Source – 3.1.2
- ↓ Regression type of defects were mostly triggered when running batch transactions. This could be an area of improvement for inspection and testing as code reviews and unit tests were not sufficient. Source – 3.1.1
- ↓ Defects triggered by ‘Specific Data Conditions’ needed to be adjusted for coverage – functional and logic. Story Grooming practices were not considering all possible data conditions. Source – 3.1.3

During the discussion of the defect classification findings with engineers, the following evaluations and comments were showing up in the heatmap as frequent causes with respect to the ODC attributes:

- Requirement reviews can be more effective
- Design and Code reviews can be more effective
- Grooming can be more regular and effective
- The effectiveness of system level tests can be increased
- Defects in database queries can be reduced
- The number of changes in the design due to misunderstandings in requirement stage can be reduced
- The effectiveness of unit tests can be increased to discover code-related defects in logic and validation

Since the process outcomes were the measure of success for implementing process improvements, a root cause disposition for each defect was created. Accountability and ownership were assigned to each function for new process enactments and measuring their success and effectiveness. The table below shows the final root cause dispositions that were put in use by the Engineering department.

3.2.1 ROOT CAUSE DISPOSITION:

No.	Root Cause Disposition	Behavior Change Responsibility	Example
1	Engineering Practice Gaps	Tech Leads will bring about an actionable change & measure effect of implemented actions	*-- extend training offers and attendance on architecture and improve systems design skills* *-- enhance or tighten the DoD* *-- introduce or update the checklist for application domain to be used in backlog refinement*
2	Lack of Business knowledge	PO's will bring about an actionable change and use it to measure outcomes PM's take the use cases back to customer and plug the gaps for future PFRs	*-- extend training offers and attendance on application domain* *-- Continuous and iterative improvement in requirements in-take* *-- Update story template to provide NFR*
3	Performance not considered	Performance refactoring team	*-- Non-Functional requirements section to be added to the requirements*
4	Instructions not clear	Tech Writers to bring about an actionable change to remove ambiguity and introduce clarity in technical write ups	*-- Use Defect Trigger categories to better document the functional usage of the features*
5	Architecture Gap	Architecture review Board should take up actionable changes to bring tech stack changes that will last the next 10 years	*-- Defects resulting from obsoleted and deprecated libraries* *-- data governance and data model governance policies for engineers*

From all the root cause dispositions available for selection, each defect was tagged with the root cause

3.2.1: The figures below show the examples of distributions of RCA with respect to Defect Type

Row Labels	Instruct-ions Not clear	Negative Use Case	Regress-ion	Perfor-mance Improve-ment	Func. Requirements not documented, but expected to work	Data Validation	System Integration	Grand Total
Engineering Practice Gaps	1	8	36	7	28	39	7	**126**
Lack of Business knowledge	7		1	3	44	10	1	**66**
Performance not considered		1		7		1		**9**
Instructions not clear			2					**2**
Grand Total	**8**	**9**	**39**	**17**	**72**	**50**	**8**	**203**

3.2.2: The figures below show the distributions of RCA with respect to Fix category

Row Labels	Build / Pack-age / Merge	Data Migra-tion	DB Design	DB Query	Docu-menta-tion	Timing / Serial-ization	Logic Cover-age	Func. Cover-age	UI	Def. Cover-age	Check-ing	Runtime Resource Handling	Total
Engineering Practice Gaps	2	3	7	9	2	2	39	30	12	19	1		**126**
Lack of Business knowledge		1		2	8		16	34	4		1		**66**
Performance not considered			1	1		1	1	3				2	**9**
Instructions not clear					2								**2**
Grand Total	**2**	**4**	**8**	**12**	**12**	**3**	**56**	**67**	**16**	**19**	**2**	**2**	**203**

This RCA was assigned using a narrative flow that can lead to simplified way of reaching the RC conclusion

- Defect Trigger in Defect Origin led to Defect Type that was addressed in this Fix category leading to the RC Disposition

In an agile environment, it is difficult to find dedicated time and resources to perform detailed fish bone diagram or 5 Why's method. So, in practice an RCA can be done for each defect in a spreadsheet using the defect characteristics. It does not take much time when institutionalized within the process.

Once RCA is assigned, the next step is to determine the concrete improvements that will lead to a change in process.

4. Measures for Defect Avoidance

The results of the analysis through this simple defect classification scheme will be used to build new engineering improvements that can help answer and measure the questions asked previously in Section 1.2. Agile teams need to constantly decide what practices to keep and which practices to discontinue. The RCA distributions can help to come up with a disciplined and structured approach to improvement schemes.

Engineering Practice gaps and Lack of Business knowledge are the top 2 categories covering 75% of the defect population. (Source - 3.2.1 and 3.2.2). These 2 RCA areas can be used to come up with defect avoidance practices and reduce defect driven rework

Each grouping will provide a correlation that can be used. For example: Defects are requiring mostly major functional coverage changes to address the issue (Source - 3.2.2) and large portion of the defects are stemming 'lack of business knowledge' (Source - 3.2.2). The defects of Fix category "Functional Coverage" and "Logic Coverage" dominate by far all the other fix categories. 68% or more than 2/3rd of the defects were found in Customers production environment (Source - 3.1.1), leading to believe that the defects are insidious in nature until the customers business routines are run by customers business users. This analysis shows that important areas to look for improvements are reviews, grooming, domain and system knowledge and test strategies.

To address the causes, that make defects detected only by the customers end users, a quality improvement plan was implemented and tracked. Here are some high-level examples of goals set for each department:

- Product Management Team: Requirements gaps
    - Use checklist for systemic understanding of data entities in use and business needs
    - Completeness of requirements should be ensured
    - Confirmation of the final requirements and acceptance criteria by the customer should be improved
    - The experience of software engineers should be extended through training
- Software Engineering: Engineering Gaps
    - Backlog refinement process to be more systemic and detailed using checklists
    - Coding practices. Examples:
        - Adding null checks for any method which returns a string
        - Null check for optional attributes
        - Transient record creation
        - Duplicate values
        - Checking for stale objects
    - Code coverage and static analysis reports to be incorporated into the Definition of Done
        - Green pipeline criteria introduced before approving pull requests
    - Default Behavior should be introduced when coding new functional coverages
        - Rather than throwing assert exceptions, the code should execute default behavior
    - Strengthen Definition of Done to include missing technical aspects like data validation
    - Adherence to review and best practices to be enforced and measured
        - Fun and friendly audits to check adherence

5. Conclusion

Implementing ODC way of Root Cause Analysis is very cost-effective and applied using a simple principle of "*Take what you already know and apply it to what you think you know to produce quality software".* The focus is on the data already collected (software defects). Defect profiling will also help the engineers to build their standards for design, architecture, policies to be focused on prevention.

The defect classification scheme can be implemented in stages by starting with a simple scheme and then moving on to in-process analysis. Fields can be tailored to your own organization. It can be tooled quickly to do in-process defect profiling. It can become a part of the Definition of Done to make sure the analysis is always complete. Fields added to data management tools like Jira and Salesforce that are completed in real time, will make the data collection virtually free.

Many things can impact the confidence in quality. To motivate the teams to make the needed investments in quality driven practices and defect avoidance mechanisms, the situations need to be evaluated first. The RCA method using defect profiling will help the organization understand the motivations for continuously building and adopting effective practice changes that will lead to greater confidence in quality.

References

http://www.chillarege.com/odc

Challenges of software process and product quality improvement: catalyzing defect root-cause investigation by process enactment data analysis by Mehmet So¨ylemez and Ayca Tarhan, 2016

A Case Study in Root Cause Defect Analysis by Marek Leszak, Dewayne E. Perry, and Dieter Stoll