An Industrial Case Study on Improving Quality in Integrated Software Product using defect dependency

Sai Anirudh Karre
Software Engineering Research Center
IIIT Hyderabad, India
sai.anirudh@research.iiit.ac.in

Y. Raghu Reddy
Software Engineering Research Center
IIIT Hyderabad, India
raghu.reddy@iiit.ac.in

***Abstract* – Product based organizations have diverse product offerings that meet various business needs. The products are in turn integrated to create integrated product suites. Rigorous product engineering is a must for creation of high quality integrated software products. Adequate measures must be taken to improve quality of the integrated product before every release of its module or sub-product. It is hard to imagine upgrading an integrated software product with unidentified defects prior to its release. In this paper, we share our observations on implementing a defect dependency metric to identify the dependency of a defect over a real-time industry defect dataset of an integrated software product. This defect dependency metric was captured and analyzed during release cycle(s) to avoid surprise issues post product launch.**

Keywords—integrated software products; software quality; defect; defect dependency; software metric; product development; rough-set theory; defect widespread

I. INTRODUCTION

Academic research in areas such as software architecture, automation frameworks and implementation methods has seen a tremendous growth in recent years and it has been observed that software industries apply them in real-time business to achieve better results [1][2]. Many software practitioners are currently trying to use methods and technologies proposed by academia to create products to the best of their abilities. There were many lessons learnt from industrial case studies over the past decade [3].

All new products are created with the intent of delivering better functional and quality objectives that meet or exceed end user expectations. Most software firms are now deliberately framing their mission statements with a *'grow fast or die fast'* strategy before they hit the market with a high quality product. As per Gartner's 2015 Magic Quadrant for Enterprise Integration Platform as a Service survey [4] most of the software industries that work on developing integrated software products still follow traditional approaches to develop and maintain quality standards of their existing products. As per their study, most of the new start-ups are concentrating on new trends in research for a better product(s) of similar class.

In most cases, it is easier for start-ups or new development projects to implement new trends in research on to software production. However it is a challenge for well-established and equipped products to adhere to these changes as it requires massive planning and human effort. Especially in integrated software, individual sub-products which are commonly referred as *product pillars* are bound together loosely for various functional and business reasons. Integrated software products become vulnerable if its sub-products are bounded with too many integration defects. For example, let's consider an integrated software product consisting of the following two sub-products: Supply-Chain product and Revenue Reporter product. Supply-chain sub-product generally tracks product billing while revenue reporter reports revenue. A common defect in the integrated product is *rounding-off of the product price*. As an end result, from an integrated product perspective, the revenue reports incorrect data. If the results are taken separately, rounding-off defect can be insignificant for chain-supply but critical for product billing. In such scenarios, the defect may be logged in different ways based on the product development team. The same defect may be considered as a severe defect for revenue reporter where as it may not even be logged in supply-chain [5]. Hence measuring the impact of such dependencies can be critical to the defect fix cycle and the release cycle.

Various methods have been proposed on detection of current defects and occurrence of defects, spanning the development life cycle. However, most of the methods revolve around defects in product rather than dependency of a defect over an entire product suite. Such a dependency measure can help quality teams to stabilize the product and avoid surprise defects post deployment. In this paper, we present a quantitative evaluation of the defect dependency metric introduced in our previous work. We realize the metric over a real-time industrial defect dataset of a large-scale integrated software product [5]. We discuss the consequences of the results that lead to creation of new practices and processes to improve development and testing methodologies of the integrated software product within the organization.

The primary author of this paper has been working in this domain for many years and has contributed to the integration of the integrated product suite in various roles. The primary author is also pursuing graduate studies on a part-time basis. Hence the authors could gain access to all the artifacts and the original data. Due to non-disclosure clauses, the name of the integrated product suite, its product pillars and the organization is being withheld. The product information shown in Table 1 makes use of alternate names to the existing (real) names. However the defect dataset presented in table II shows exactly the same numbers as present in the defect database for the various products and versions of the integrated software product.

The rest of the paper is organized as follows: Section II provides details of industrial examples of software quality

related to our work, section III explains the background of defect dependency with an example along with study design of our work, and section IV details the implementation setup of defect dependency metric on an industry defect dataset. Section V talks about results of our implementation and observations identified during every new release of our integrated software product. Finally in section VI we discuss the threats to validity and present some insights about future work.

II. Related Work

Software Quality Assurance (SQA) in integrated software products is a major activity during software production cycle. Advanced SQA practices were proposed by various researchers over past decade that became standard approaches in today's software production release cycle. Functional integration approaches, strategies and methodologies to integrate software by its features were initially proposed [7]. Cost based effort estimation method [8] for integrated software architecture model-COTS was proposed and deduced quality measures to choose right resource for right task. Fedrik et al. proposed quality based methods to improve software integration [9]. In [10], new methods were proposed on software product integration by analyzing build statistics with real time products as applied examples. In contrast to the existing work, a quality based dependency model [13] capable of supporting software architecture as an evolution to software production was proposed. Improvements to integration methods in requirement analysis phase using a model based object oriented approach was proposed in [11].

Researchers have presented interesting methods on implementation of integration in global software projects and veracious trends in integration [12][15][20]. Zeng et al. discuss about an interesting integration framework that includes product design concepts as a collaborative feature during development in their work [14]. Software quality based integration challenges during design and implementation phases, and its consequences were listed out through an industrial case study of enterprise software product by Rognerud et al. [16]. Quality related observations on heterogeneous architectural model for efficient integration among software modules were proposed in [17]. Optimization methods in software integration with testing efforts and test complexity were analyzed [18]. Most significant work on integration bugs specific to dependency on requirements [19] are defined during project inception were recorded. Latest work on successful integration process [21] for large scale software was proposed along with quality improvements and between development and quality teams. In parallel there was significant amount of work on software defect prediction by Chengnian et al. [22] that can help industry understand future defects with prediction methods. Overall, there is a lot work on software quality, but specific research pertinent to defect widespread and dependency of a defect over a product is limited. There aren't many practical implementations that provide examples of applying the defect dependency methods to case studies in industry. In this paper, we are trying to address this specific gap by producing our implementation results on an industry dataset.

III. Study Design

In this section we provide an overview of the defect dependency metric and the real time industry dataset.

A. Defect Dependency Metric

Large-scale software products are complex and as such are prone to defects. Software quality teams have to perform rigorous checks before releasing a fix to a defect. This includes ensuring that the fix will not cascade new defect(s) into the product. The setup can be simple in case of small products but not for complex software products or an integrated product suite. Quality teams mostly face integration issues with incorrect control flow and data flow between the sub-products or sub-modules with in entire integrated product. It is also tough to detect and track the source of a defect in a complex integrated system as this involves various other quality teams from different sub-products. Firms that integrate products due mergers and acquisitions have different set of challenges as these products may have evolved independently but not in an integrated fashion. In such a scenario, it is essential for product owners to understand the impact the defect so as to mitigate possible surprise defects from other modules of the integrated product. We introduced defect dependency metric to address this specific concern in our previous paper [5]. We proposed a Defect dependency metric (D*) to calculate defect dependency by demonstrating the application of Generalized Dependency degree (Γ) using rough set theory [6].

Defect dependency can be defined as a metric to study the widespread of a defect with unknown impact and unknown risk over a module(s) or component(s) or sub-product(s) of a software product(s). Defect dependency can be calculated for any software of any size, however heuristically it is more applicable for complex systems as it is difficult to comment on widespread of a defect without any evidence. Generalized Dependency degree (Γ) is a mathematical approach to calculate the dependency between the equivalent classes generated by equivalence relation using disjoint sets. Initial study using this approach was proposed in Rough Set theory and was later studied by Halxuan et al [23].

- Consider a rough set over an information system, it can be defined as an approximation space as a pair as S= (U, A) where U is a non-empty finite set called universal set and A is a equivalence relation defined on a U which is a nonempty finite set of attributes i.e., a: $U \rightarrow V_a$ for a ϵ A, where V_a is called the domain of a.
- Here X be a subset of U, then the lower approximation of X by A in S is defined as $\underline{R}X$= {e ϵ U | [e] $\subseteq$ A}, similarly the upper approximation of X by A in S is defined as RX= {e ϵ U | [e] $\cap$ $A \neq \emptyset$} where [e] denotes the equivalence class containing 'e'.

If we redefine above definition in terms of a defect dependency approach, consider a defect dataset (D) of a large scale complex software product (L). Then:

- If $P_1, P_2, P_3, P_4 \ldots\ldots P_N$ are sub products of L, then consider $D_{P1}, D_{P2}, D_{P3}, D_{P4} \ldots D_{PN}$ are defect subsets of respective sub-products of a universal defect dataset *D*.
- S = (*D*, D_e) is an approximation space, where *D* is a non-empty finite defect set and D_e is a equivalence relation defined over all defect subsets D_{Pi} where $\{i \in 1,2,3 \ldots . n\}$

To calculate the dependency of a defect subset attributes over another subset, we will evaluate the value for Γ (Generalized dependency degree) which is defined as

$$D^* = \Gamma(O, H) = \frac{1}{|D|} \sum \frac{|O(x) \cap H(x)|}{|H(x)|} \qquad (1)$$

Here O & H are two equivalent classes generated over an equivalence relation framed from some disjoint sets of universal set D. We have utilized this method to find dependency of a defect on our industrial defect dataset. It is a simple mathematical approach to understand the dependency of a one set over another. Each data point in the dataset contains collection of attributes that are pre-processed such that it can be applied over dependency metric. If we map this method to our real time dataset, D is the total defect dataset of our enterprise software product, O and H are two equivalent classes of equivalent sets which constitutes defects of two different sub-products O and H. In case there are more than two sub-products, we need to generate equivalent sets of all the defect product sub-sets, constructs equivalence class and apply this formula. There is no definite scale to the defect dependency metric, however the value varies between 0 and 10.

B. About Industry Dataset

Our industry defect dataset contains defects of an Integrated Human Resource Integrated System (IHRIS) product with 5 primary product pillars (as shown in Table I) that are integrated as a single product suite. Each product pillar has sub-products that are implemented in an integrated mode. As stated earlier, due to non-disclosure clause, we are use the common derived names of product and their sub-products instead of the original product names.

This integrated product is deployed as Software-as-Service, Stand-alone Hosted and On-premise subscription for most of the fortune 500 companies. New service pack is released and deployed (includes feature changes or major fixes to the defects) once every 2 months in a calendar year to all the customer instances. Also a maintenance pack is released twice a month in a calendar year that includes minor fixes for the defects reported between the release timeline. All the above products once cross-sold and deployed as individual products are now deployed as an integrated suite, i.e. all users accessing the integrated suite will be able to access respective product(s) or sub-product(s) as per their role permissions defined by the global administrator of the product suite.

The defect dataset constitutes defects from all the products and sub-products of the integrated suite that are extracted from the defect database of a defect tracking tool called JIRA™. Dataset contains defects raised by QA teams every sprint cycle along with defects reported by customers post product deployment. The authors worked with quality assurance teams and customers to extract the defects from the sprint cycles and evaluated the data using product managers' inputs.

TABLE I. PRODUCT INFORMATION

S. No	Product	Sub-product
1	Learning Management System (*LMS*)	Admin Mgmt.
		Learner mode
		Manager mode
2	Human Resource System (*HRS*)	Hire Mgmt.
		Compensation Mgmt.
		Succession Mgmt.
		Performance Mgmt.
3	Business Intelligence System (*BIS*)	BI Dashboards
		Data Downloader
		Data Uploader
4	Work force Manager (*WFM*)	Attendance Mgmt.
		Payroll Mgmt.
		Reimbursement Mgmt.
5	Web Services Manager (*WSM*)	Export Mgmt.
		Integration Mgmt.
		Web Service Admin mode

C. Real Time example for Defect Dependency

To understand the need of studying defect dependency, we provide a real time industry scenario consisting of three defects reported in three different sub-products of IHRIS software:

✓ *Scenario:* A manager uses the performance management sub-product to perform an employee's year-end performance assessment. The Manager rates employee's performance (between 0-5) along with comments. As per the manager rating, a pre-defined compensation hike shall be added to the employer salary in compensation sub-product along with relevant tax calculations as per policy in payroll sub-product.

✓ *Defects*: The sensitiveness of appraisal data necessitates encryption while storage. So, decryption was necessary to view the data in other modules. Defect #191 is raised, as the decryption method is not honored by the numeric data in manager comments. Later defect #278 and #286 were recorded due to defect #191 but were practically difficult to trace within a complex product without performing a defect dependency study.

✓ *Observations*: These three defects appear to be linked, however software quality teams normally would not have proactively identified defect #278 and #286 unless customers reported them. Defect#191 caused malfunction to compensation and payroll calculation. In cases like these, defect dependency study helps in detecting such defect spread and help product managers to prioritize defects accordingly.

Defect#191	Incorrect Decryption of Manager and Employee comments in Employee Performance Cycle
Module (Product)	Performance Mgmt. (HRS)
Cause	Decryption algorithm incorrectly converts NUMERIC data causing incorrect Manager ratings and comment
Fix	Decryption logic updated to honour NUMERIC data in Manager rating and comments during Performance Cycle.

Defect#278	Invalid hike % was imported to multiple users and corrupted existing user hike information
Module (Product)	Compensation Mgmt. (HRS)
Cause	Decryption logic in Performance Mgmt. caused issue.
Fix	Exception handling is improved to handle Invalid data in Compensation process cycle.

Defect#286	Unable to deduct monthly tax for Employees due to mismatch in YTD employee payment in Payroll
Module (Product)	Payroll Mgmt. (WFM)
Cause	Lack Exception handling in Performance Mgmt. caused corruption in tax calculation.
Fix	Created exception to deduct default monthly tax in case of data corruption for Employee monthly payroll payments

D. Study Workflow

Below are the details of study workflow and teams involved.

- The study was conducted over three service packs along with five maintenance packs of the above provided integrated software suite. The study was done over a period of 9 months between September 2014 and July 2015.
- The entire defect dataset of integrated product has been chosen and equivalence classes have been generated for all the sub-products and products.
- Defect dependency metric is applied over the equivalence classes and the metric value is calculated for all the defects identified by quality assurance (QA) team during every weekly sprint cycle.
- These defects include defects recorded during sprint cycle and defects raised by customers together. The metric results are combination of two sources (QA team and customers).
- QA team will evaluate the results of the metric over post release defects and compare them with the current defects recorded during sprint cycle for regression. Primary aim of this exercise is to avoid the possible spread of defects in upcoming release version.
- The value of defect dependency metric is the indicator for improvement study. QA teams progressively compare the metric values every release and sprint cycle.
- It has to be noted that there is no specific scale for this metric as it always depends on size of the defects and attributes (products chosen to evaluate) from dataset.
- QA Team shall present the results to product management team so that defects can be prioritized and an executive decision can be taken on implementing a plan for a new feature for a stable product(s) or sub-product(s) in upcoming service packs.

E. Study Design

This section describes the steps involved on calculating the metric using the industry dataset with specific.

- Each defect in this dataset is a data point. All sub-products are considered as subset i.e., there are 16 sub-products spread across 5 product pillars (shown in Table I). For example, if Web Services Manager is a pillar product, Export Mgmt., Integration Mgmt., and Web Service Admin mode are its subsets.
- Each set contains defects of its sub-product and they are entitled to be calculated together. Let D superset which contains defects of all sub-products i.e.

$$D = \{p_1 \cup p_2 \cup p_3 \cup \ldots\ldots\ldots\ldots \cup p_{16}\}$$

  p_i represents 16 sub-products from the enterprise product suite under union of D the superset.
- Equivalence relation is constructed using all the p_i sets considering all the entities of the individual sets
- Equivalence classes are created for each p_i set generating the classes of values that are common to all the p_i sets.
- All equivalent classes of p_i sets are now passed to calculate $\Gamma(p_1, p_2, \ldots, p_{16})$ to generate overall defect dependency metric D^*
- D^* is now the metric standard for all the input p_i set of defect for a specific release. This activity needs to be continued for every release to understand the dependency of a defect over p_i sets used to calculate D^*
- Post every release (including service pack and maintenance pack), D^* values are compared and reviewed to identify the improvement.

All the above steps are programmatically implemented using .NET 4.0 and SQL. Additional details in this regard are provided in the next section.

IV. Implementation Setup

In addition to the standard testing process, QA team and product managers executed the below implementation and evaluation plan for of the defect dependency metric. Fig. 1 shows the implementation flow of the study setup. JIRA™ is hosted against Microsoft SQL Server 2008 R2 at database level. Below 'D' is the JIRA defect database which stores defects raised by customers post product release and QA team during sprint cycle.

Using a data extract package (designed using Microsoft SQL Server Integration Services 2008 R2), we extract desired defects from available sub-products from the entire product suite. The data extract package contains SQL query logic to extract the defect dump for all the sub-products. This package pushes the defect dump to a testing database (T). We use this testing database to implement defect dependency metric. We construct another package called metric package (M) that contains the SQL query logic to construct equivalence relation and equivalence classes of sub-products chosen for metric calculation. Using .NET Code and SQL, D* is calculated and stored in testing database.

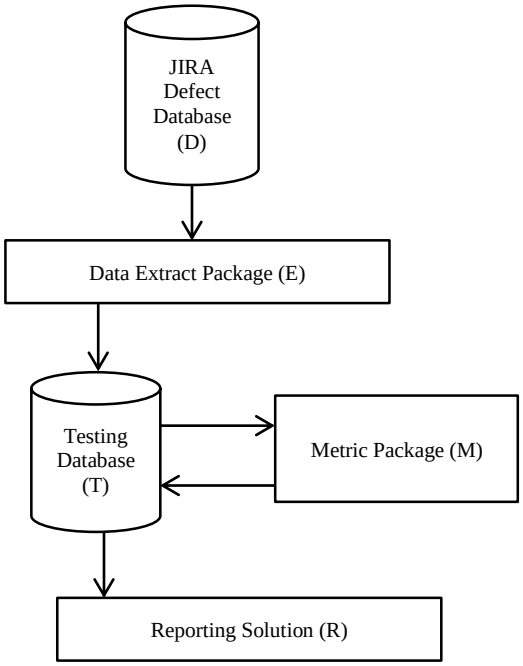

Fig. 1. Implementation flow

The implementation cycle is repeated during every release and every sprint cycle so that our QA teams can analyze and compare the metric results for taking fair decisions on improving product quality and defect prioritization. Product Managers and QA teams depend on Reporting tool (R) to visualize the trend of the metric periodically to understand and decide whether the results are conflicting or making real sense in practice.

V. Results and Observations

A. Implementation Results

We found interesting results across different version releases of our integrated software product. Table II contains the detailed trend data of metric values captured per product across entire produce suite specific to the released versions. Here {*V1, V2, V3*} being the service pack releases and {*V1.1, V1.2, V1.3, V2.1, V2.2, V3.1, V3.2, V3.3*} are the maintenance pack releases. *V1* is the considered as major service pack release and *V1.1, V1.2* and *V1.3* are its subsequent maintenance pack releases. Apart from these values, our QA team captured the metric values for every sprint release separately and for customer defects on weekly basis.

If we carefully observe, we can find the defect dependency values to be high in initial version *V1*. This was the base version of the implementation. We first calculated the metric value for *V1* version to analyze the health of the current integrated software suite and found that it had high defect dependency value of *6.78*. Human Resource System. (HRS) product was found to have high defect dependency value across overall product suite whereas Web Service Manager (WSM) was found to have low values. We started implementing the approach across different releases and found a significant changes in the quality of product and also a downtrend in the values of overall metric result for every product within a given specific version i.e. if we consider an example, in case of Learning Management System (LMS) the metric dropped down from *1.84* to *0.99* from service pack version *V1* and by end of release of maintenance pack *V1.3* which signifies improvement and stability in the product. Similar trend was identified across other product pillars in the enterprise suite. Our QA team has found significant improvement in terms of quality of product as the widespread of defects are diminishing by end of stable release as observed by the decrease in metric value for the products in below table.

Fig. 2 is the graphical representation of values from Table II highlighted in bold and italic, provides the trend analysis of the metric values across all products across version. We find a significant downtrend during the end of every version i.e. from *V1* to *V1.3*, *V2* to *V2.2* and *V3* to *V3.3*. We were able to minimize the various dependent issues across the integrated suite raising the quality levels of the entire product. This methodology helped QA teams and Product Managers to prioritize and de-prioritize defects with developers. For example, the Sustenance Engineering team responsible for providing fixes by end of upcoming release of a service pack or maintenance pack was able to select a particular defect that needed fix in a particular release cycle.

As per Fig. 2, from version *V1* to version *V3* we find a rise in dependency issues on every standard service pack release i.e. *V2* and *V3*. We studied causes of this increase and found that rise in metric is due to dependency among the new features introduced in the respective pillar products. However, as the maintenance pack(s) were released with subsequent fixes, we found downtrend in metric results within a version, i.e., *V2.1* and *V2.2*. At the end of every version, we were able to determine the impact of most of the defects. This led to prioritization of addressing high defect modules thereby easing the dependency of the defect to specific part of the product and decreasing it's widespread.

B. Observations

We present our observations partially based on the retrospective session conducted between Product Managers and QA teams for trend analysis.

- ✓ It became tough to gain confidence from Product managers in initial sprint cycles, as the defect dependency was too high which brought down initial confidence levels. Also as the approach was mathematical (based on rough set theory), the QA team didn't seem to comprehend the methodology in the beginning. As a result we had to spend some time negotiating for adoption of the approach within the Quality assurance team.

- ✓ However, as we progressed further, there has been significant improvement on stability of the product. We found exponential decrease in environment and performance related defects across releases. From the table, we can see that the "overall" numbers have decreased for every sub-product in the integrated product suite for *V1* to *V3.3*.
- ✓ By end of *V3.3* version release, as per the QA team, upon evaluation it was found that there was about *71%* decrease in overall defects reported by customers post product release. There was a *52%* decrease in internal defects raised by QA teams during sprint cycles.
- ✓ Most of the functional defects were proactively identified and resolved in timely fashion. We believe this decreased the risk of software failure during product deployments. The defect dependency metric was able to identify the spread of defects and helped to track critical surprise defects before produce release. These proactive defects constitute 12% among overall defects recorded across versions before deployment.
- ✓ In case of control flow issues among sub-products, we still have to rely on our standard approaches which are practiced by QA teams. Most of such control flow issues were free from defect dependency and were found them to be fragmented and un-connected with other modules in specific product or a sub-product.
- ✓ Business Intelligence System Reporting product and Web Services Manager product were found to be most stable products during evaluation of this metric.

C. Lessons learnt

- ✓ During this implementation, we found few architectural flaws in two of the sub-product(s) that required total makeover in terms of integration. This wouldn't have been possible if the metric was never implemented.
- ✓ It was also identified that it is expensive to re-design the sub-modules when the product is actively used by most of the customers. Hence, the faulty sub-products were removed from the integrated product suite and were to be merged as components in one of the existing product for improved quality.

TABLE II. DEFECT DEPENDENCY RESULTS BY PRODUCT AND VERSION

S. No	***Product***	***Sub-product***	***V1***	***V1.1***	***V1.2***	***V1.3***	***V2***	***V2.1***	***V2.2***	***V3***	***V3.1***	***V3.2***	***V3.3***
1	Learning Management System (LMS)	Overall	***1.84***	*1.49*	*1.24*	*0.99*	***1.26***	*1.15*	*0.53*	***0.8***	*0.41*	*0.28*	***0.14***
		Learner mode	0.19	0.18	0.14	0.17	0.14	0.11	0.07	0.09	0.03	0.03	0.03
		Manager mode	0.37	0.33	0.26	0.21	0.21	0.16	0.05	0.14	0.11	0.07	0.02
		Admin Mgmt.	1.28	0.98	0.84	0.61	0.91	0.88	0.41	0.57	0.27	0.18	0.09
2	Human Resource System (HRS)	Overall	***2.47***	*2.1*	*1.88*	*1.71*	***1.97***	*1.78*	*1.13*	***2.54***	*1.65*	*1.28*	***0.56***
		Hire Mgmt.	0.45	0.39	0.33	0.29	0.51	0.45	0.31	0.44	0.31	0.17	0.08
		Compensation Mgmt.	0.39	0.31	0.32	0.29	0.39	0.32	0.29	0.28	0.19	0.12	0.07
		Succession Mgmt.	0.22	0.21	0.16	0.15	0.18	0.13	0.12	0.14	0.11	0.05	0.02
		Performance Mgmt.	1.41	1.19	1.07	0.98	0.89	0.88	0.41	1.68	1.04	0.94	0.39
3	Business Intelligence System (BIS)	Overall	***1.02***	*0.93*	*0.81*	*0.62*	***1.08***	*0.96*	*0.72*	***0.98***	*0.72*	*0.42*	***0.19***
		BI Dashboards	0.27	0.21	0.18	0.13	0.31	0.28	0.22	0.34	0.21	0.11	0.07
		Data Downloader	0.32	0.31	0.27	0.17	0.45	0.41	0.29	0.52	0.44	0.29	0.12
		Data Uploader	0.43	0.41	0.36	0.32	0.32	0.27	0.21	0.12	0.07	0.02	0
4	Work force Manager (WFM)	Overall	***1.15***	*0.96*	*0.89*	*0.73*	***1.05***	*0.85*	*0.71*	***1.2***	*0.73*	*0.38*	***0.19***
		Attendance Mgmt.	0.31	0.25	0.19	0.12	0.44	0.37	0.31	0.58	0.31	0.21	0.09
		Payroll Mgmt.	0.24	0.21	0.2	0.11	0.21	0.17	0.14	0.24	0.15	0.06	0.02
		Reimbursement Mgmt.	0.6	0.5	0.5	0.5	0.4	0.31	0.26	0.38	0.27	0.11	0.08
5	Web Services Manager (WSM)	Overall	***0.3***	*0.26*	*0.22*	*0.22*	***0.24***	*0.1*	*0.07*	***0.17***	*0.06*	*0.04*	***0***
		Export Mgmt.	0.16	0.15	0.13	0.13	0.12	0.07	0.04	0.09	0.04	0.04	0
		Integration Mgmt.	0.05	0.05	0.05	0.05	0.09	0.03	0.03	0.07	0.02	0	0
		Web Service Admin mode	0.09	0.06	0.04	0.04	0.03	0	0	0.01	0	0	0
6	Overall Metric		***6.78***	***5.74***	***5.04***	***4.27***	***5.6***	***4.84***	***3.16***	***5.69***	***3.57***	***2.4***	***1.08***

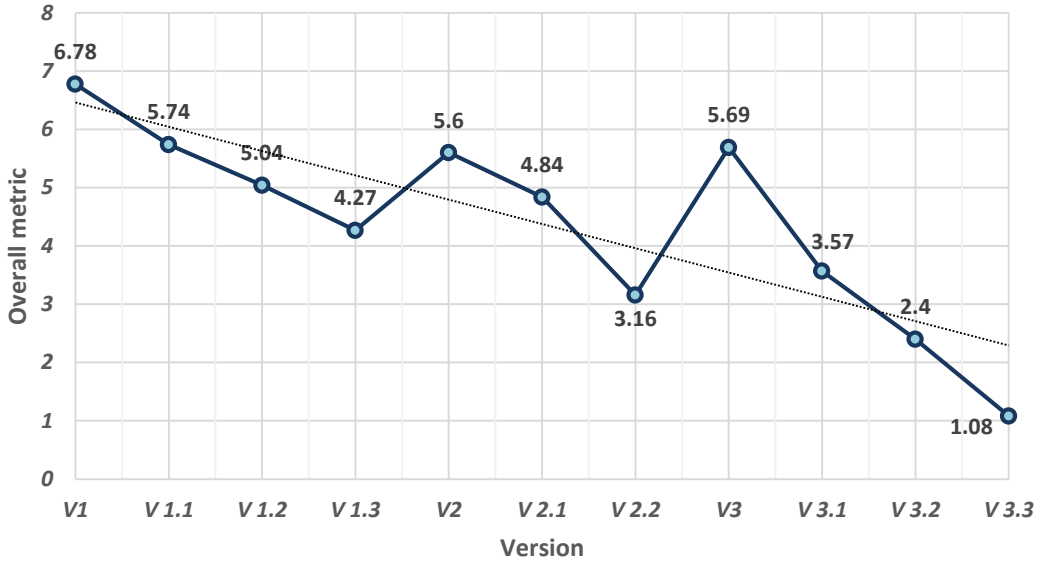

Fig. 2. Implementation flow – Trend Analysis of overall metric results across version

- ✓ QA teams have come up with improved test cases as part of future integrating testing, as traditional test cases are no longer contributing towards product quality.

In summary, defect dependency metric was one of the key contributors along with our standard processes for stabilizing our integrated software product to a greater extent. The QA team gave informal feedback that the metric was of great value and product managers stated that it has helped improve customer success across customer subscriptions.

VI. Threats to Validity

Our approach to calculate degree of Defect dependency metric is based on rough set theory. We implemented it against a real time defect dataset to improve and evaluate the quality of our large scale integrated software product during every release cycle since September 2014 to July 2015. We were successful in improving the integrated product suite. The main concern with our case study just like other case study papers is the possible extension and applicability of the work to other defect datasets. Given that we have applied it only to a single product suite, we can't convincingly state that it's applicable to other product suites too. However, it needs to be noted that our case study was based on an integrated software product that is used by most of the fortune 500 companies. It would be interesting to see if this methodology is adopted in tools used to build integrated software from mid-size software industries to large scale industries to understand its significance in reality. We believe that apart from defect dependency metric, heuristic approaches can also be used to solve our day-to-day quality issues. However we suggest fellow software practitioners to adopt our approach to improve software quality of their products. The scope of defect dependency metric is only to identify dependency of defect i.e. it's widespread; however an integrated software product can still be un-stable with no defect dependency. This can be because of poor functional and architectural design or due to control/data flow issues.

On the other hand, organizational constraints and its corresponding influence on the accuracy of metric can be questioned. However a series of evaluation by quality teams and meetings with product manager and key stake-holders of the project(s) helped us evaluate the efficiency of the metric during every release. Influence of teams with lack of process knowledge, skill set or technology used can be argued and the results may be interpreted differently at times. To limit this issue, the evaluation of this metric has to be attributed to only key decision makers within the organization.

VII. Conclusion and Future Work

In current study, we have implemented this metric only on product & sub-product defects. As an extension to this study, we will be working on alternate methods to identify dependencies and widespread of defect on various other artifacts at different levels of software production like requirement analysis, resource planning, integration strategy, maintenance and design. This will help an integrated software company to address quality issues at all levels. Lessons learnt by conducting such studies can address some of the open challenges and help take efficient decisions to produce better complex products. As a future work, we will be assessing the metric more comprehensively by getting feedback from developers and quality teams on how significant this method helps them to prioritize the defect as part of regular work. We will have to work on testing strategies while adopting this approach in real time so as to improve test cases and address proactive defects especially during maintenance phase.

Acknowledgments

We thank all the members of product management, quality assurance and deployment teams at SumTotal Inc. for providing the valuable assistance, suggestions and feedback on implementing our research.

REFERENCES

[1] Leupers, Rainer; RWTH Aachen; When, Norbert; Leupers, Rainer; Roodzant, Marco; Stahl, Johannes; Fanucci, Luca; Cohen, Albert; Janson, Bernd, "Technology transfer towards Horizon 2020", In proceedings of Design, Automation and Test in Europe Conference and Exhibition (DATE), March 2014.

[2] Laird, L; Ye Yang, "Transferring Software Engineering Research into Industry: The Stevens Way", In proceedings of IEEE/ACM 2nd International Workshop on Software Engineering Research and Industrial Practice (SER&IP), May 2015, pp.46-49

[3] Wohlin, C, "Empirical software engineering research with industry: Top 10 challenges", In proceedings of 1st International Workshop on Conducting Empirical Studies in Industry (CESI), 2013, pp.43-46.

[4] Massimo Pazzini, Yefin V. Natis, Paolo Malinverno, Kimihiko Iijima, Jess Thompson, Eric Thoo and Keith Guttridge, "Magic Quadrant for Enterprise Integration Platform as a Service, Worldwide", Gartner, March 2015, Report: G00270939.

[5] Sai Anirudh Karre, Y. Raghu Reddy, "A Defect Dependency approach to Improve Software Quality in Integrated Software products", International Conference on Evaluation of Novel Approaches to Software Engineering, Barcelona, April 2015, pp:110-117

[6] Pawlak Z, "Rough classification", In International Journal of Human-Computer Studies, 1999, pp. 369–383

[7] Jim-Min Lin, "Cross-platform software reuse by functional integration approach", In proceedings of 21st International conference on Computer Software and Application Conference, Washington DC, USA, Aug 1997, pp:402-408

[8] Daniil Yakimovich, James M. Bieman, and Victor R. Basili, "Software architecture classification for estimating the cost of COTS integration", International Conference on Software Engineering, Los Angeles, USA, May 1999, pp:296-302

[9] Fedrik Ekdahl and Ivica Crnkovic, "How to Improve Software Integration", Information & Software Technology Journal, Elsevier, 2005.

[10] Stig Larsson and Ivica Crnkovic,"Product Integration Improvement Based on Analysis of Build Statistics", European Software Engineering Conference, Dubrovnik, Croatia, Sept 2007

[11] Chih-Hung Chang, Chih-Wei Lu , and Chu W.C, "Improving Software Integration from Requirement Process with a Model-Based Object-Oriented Approach", International Conference on Secure System Integration and Reliability Improvement, Yokohama, Japan, July 2008, pp:175-176

[12] Gotel O, Kulkarni V, Scharff C, and Neak L, "Integration Starts on Day One in Global Software Development Projects", IEEE International Conference on Global Software Engineering, Bangalore, India, Aug 2008, pp:244-248

[13] Hongyu Pei and Ivica Crnkovic,"Using dependency model to support software architecture evolution", 23rd IEEE/ACM International Conference Automated Software Engineering-Workshops, L'Aquila, Italy, Sept 2008, pp:82-91

[14] Pengfei Zeng and Yongping Hao, "Towards a Software Integration Framework in Product Collaborative Design Environment", International Conference on Computer Science and Software Engineering, Wuhan, Hubei, Dec 2008, pp: 527-530

[15] Campbell, M., "The Future of Test-Product Integration and its Impact on Test", 24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, Chicago, USA, Oct 2009.

[16] Rognerud H.J, Hannay J.E,"Challenges in enterprise software integration: An industrial study using repertory grids", International Symposium on Empirical Software Engineering and Measurement, Lake Buena Vista, USA, Oct 2009, pp:11-22

[17] Chong-chong Zhao and Li-yong Zhao, "The research about software integration oriented heterogeneous architecture style", International Conference on Software Engineering and Data Mining, Chengdu, China June 2010, pp:311-315

[18] Steindl M and Mottok J, "Optimizing software integration by considering integration test complexity and test effort", In proceedings of 10th Workshop on Intelligent Solutions in Embedded Systems, Klagenfurt, Austria, July 2012, pp:63-68

[19] Junjie Wang, Juan Li, Qing Wang "Can requirements dependency network be used as early indicator of software integration bugs?", Rio De Janeiro, Brazil, July 2013, pp:185-194

[20] Jun He and Chandler, "Package reliability and performance trends in an era of product integration", 2014 IEEE International Reliability Physics Symposium, Waikoloa, Hawaii, June 2014, pp:2F.1.1-2F.1.5

[21] Yujuan Jiang, "Improving the integration process of large software systems", IEEE 22nd International Conference on Software Analysis, Evolution and Re-engineering, Montreal, Canada, March 2015, pp:598

[22] Yuan Tian, David Lo, Chengnian Sun: "DRONE: Predicting Priority of Reported Bugs by Multi-factor Analysis" In proceedings of International Conference on Software Maintaince (ICSM), Netherlands, Sept 2013, pp. 200-209

[23] Halxuan, Irwin, Michael, "Generalized Dependency Degree Between attributes", In proceedings of Journal of the American Society for Information Science and Technology, Sept 2007, pp:2280-2294