



Misapplications of Well-known Test Metrics

¹Mandeep Walia, ²Anu Gupta, ³Ravinder Kumar Singla

¹PhD Research Scholar, ²Associate Professor, ³Professor

Dept. of Comp. Sc. & Applications

Panjab University, Chandigarh, India

Abstract: Software testing plays a crucial role in software development life cycle. Constantly increasing product complexities accompanied by time-to-market pressures of delivering high quality software products achieving optimum cost points bring even more focus on software testing. Usage of measurement and metrics to drive software testing endeavors effectively has been a common practice. More and more organizations have deployed comprehensive metrics programs to enable their test managers make informed decisions based on objective evidences.

There are a gamut of metrics invented by the research community and a good subset of them in adoption by the software practitioner fraternity, however project managers still face challenges in applying these metrics and take optimum benefits out of this adoption. Given that every project and program has its own environmental characteristics, metrics adoption needs to be viewed in a project-specific environmental context. Moreover, the interpretation of metrics cannot be done in isolation and needs to be taken in conjunction with other associated metrics. Decisions taken based on individual metrics in isolation are likely to be misleading and sub-optimum. Though the literature has been full of varied kinds of metrics, practical guidelines towards adoption of these metrics are not available in a systematic form as a consolidated one-stop-shop solution.

There is a need to devise a guidance framework or a recommendation model for metrics adoption that project managers can depend up on as a guiding aid while selecting appropriate metrics for their project-specific needs. Such a model should guide project managers on which are the most appropriate metric sets for their project-specific needs, which are the associated metrics that should be considered in conjunction and how adoption of these metrics needs to be tailored to align with the project-specific environmental context.

I. Introduction

We start terming a discipline as mature, as it starts orienting itself more and more towards usage of objective evidences for any kinds of decision making. This saying has been applicable and true for the field of software engineering as well, which came in to inception around 50 years back. Though Software Engineering started as more of an art with “Programming Heroes” writing down complex and cumbersome software programs leveraging low-level programming languages, the field slowly started seeing usage of numbers to manage various attributes of underlying process and the end-product. Over the years, we saw inception and evolution of various measures and metrics associated with the field of software engineering especially the area of Software Quality Engineering. Though Software Quality Engineering went on to be an area of elevated focus as the software systems turned more and more complex, however quickly the software community was able to realize the significance of prevention over correction and hence the focus transformed in to “How can we deliver a high quality software with minimal costs getting incurred on activities related to software testing and other quality assurance activities”. Hence, the field of Software Testing, being seen as an Art for several years got the flavor of a scientific discipline with introduction of various measurement dimensions getting associated with different aspects of testing.

No doubt, software metrics serve valuable information to enable quantitative managerial decision making over the entire software lifecycle. Reliable support for decision-making implies effective risk assessment and reduction. Over the years, we saw a diverse set of metrics being invented by the research and practitioner communities. Starting late-90’s, more and more software organizations started adopting metrics as an enabler to manage and enhance their decision making abilities. With enhanced outsourcing at the same time, metrics even got an elevated attention both from vendor and customer teams in an outsourced setting. Most of the large-sized organizations institutionalized company-wide metrics programs as an endeavor to bring in improvements in a revolutionized manner.

However, while organizations institutionalized metrics programs and documented sets of metrics to be adopted depending up on the software engineering activity at hand, there also came in the human judgement side along

with project-to-project environmental differences, which in some or other way hindered a uniform implementation of these metrics programs. While project managers adopted, implemented and interpreted the recommended sets of metrics in a uniform way in most of the project situations, the results observed haven't been homogenous across projects. Due to this non-homogeneity, inter-project comparisons and comparisons with industry benchmarks became difficult and not leading to the right conclusions.

The primary reason for above sighted anomaly has been that each project differed from the other in terms of environmental intricacies. This required metrics implementation and interpretation to be tailored for project-specific needs, which the project manager community skipped in most of the cases. Due to this, though most of the projects adopted metrics satisfying organizational and CMMI (Capability Maturity Model Integrated) requirements, the benefits achieved were not identical. Some of the projects seemed to be misguided in spite of metrics in place to track most of the project aspects.

This paper discusses some of the most common metrics in the domain of software testing and presents a case study for sets of projects in industrial arena, where though the projects were adopting similar sets of metrics, but there were issues in implementation and interpretation of those metrics. The paper then provides recommendations on possible future research areas, which if pursued can provide solutions to such anomalies.

II. Need for Metrics in Software Testing Life Cycle

It is a well-known fact that measurement enables one to comprehend the scheme of things in a better way and target improvements. Most of us will understand and agree that there is a need to establish a software testing process that is cost effective and efficient to meet the market pressures of delivering low cost and quality software. Measurement is a key element of an effective and efficient software testing process as it helps evaluating the quality and effectiveness of the process. Moreover, it helps assess the productivity of the personnel involved in testing activities and enables improve the software testing procedures, methods, tools and activities.

Gathering of software testing related measurement data and proper analysis provides an opportunity for organizations to learn from their past history and enhancing their software testing process maturity. In the field of software testing, test managers are often challenged by the need to do more with less. A key goal for test managers is to ensure that the software product that gets released finally is a high quality product and this assurance needs to be provided with an optimum usage of resources and costs incurred towards software testing activities. During the course of software testing life cycle, test managers are tasked with making certain crucial decisions viz. devise smarter test strategies to select the most appropriate subset of test cases, which holds highest potential to catch defects, optimally sequence execution of test cases such that the test set executed during initial stages holds highest probability of catching defects [1], [2]. Additionally, test managers are also required to make predictions about final software quality based on the defect trends observed till now during the software development life cycle [3] [4].

Hence, in a nutshell, test managers are challenged to solve below mentioned problems as a part of their key responsibilities:

- 1) Continuously ensure efficiency and effectiveness of software testing process and software product quality [2]
- 2) Making decisions on the right test set selection for execution [2]
- 3) Making decisions on an optimal sequencing of test execution
- 4) Predicting end-software quality based on the defect trends observed till now [1]

Software metrics serve as a key enabler for test managers, while they embark on the journey to make above mentioned key decisions. They surely need a guidance model or framework, which is objective in nature (based on metrics) and can quickly act as an advisory aid for them for quicker and accurate actions related to these key needs of test managers [2].

In spite of a number of software testing metrics recommended in literature, the test manager community still struggles to find answers to following most crucial questions:

- 1) Which are the right metrics for me for adoption in my project?
- 2) Should I adopt these metrics "as is" based on their underlying definition or some amount of tailoring is required to be carried out to the metric prior to adoption given the peculiar characteristics of my project?
- 3) How to interpret observed values of metrics to make informed decisions? Do I need to take in to account any associated metrics to make those right decisions [5]?

- 4) Based on the defect trend observed till now, can I be confident about the software product quality, once released to market?

There are no clear-cut answers available to above sighted questions in literature as well as previous research outcomes. Moreover, the gamut of metrics invented seem to be scattered just here and there with no one-stop shop solution available.

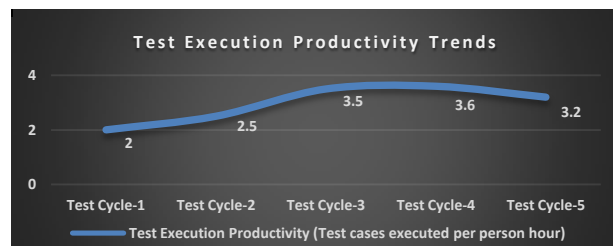
A. Common Test Metrics & their Purpose

Largely, software metrics can be viewed at from 3 dimensions and these dimensions in fact provide a basis for their classification for various purposes: product metrics, process metrics and project metrics. Software product metrics depict characteristics of the product such as its size, complexity, design aspects, performance and quality. Software process metrics can be used to measure and enhance processes of software development, maintenance and testing. Examples include defect removal effectiveness during development process, testing defect arrival pattern and defect-fix response time for maintenance process. Project metrics depict various project characteristics and its execution. Examples include number of software engineers, staffing pattern over project life cycle, cost, time and productivity [6].

For the purpose of this study, we focus on 4 of the most commonly used metrics to manage software testing process and product - Productivity, Test Coverage, Defects per Hour of Testing and Defect Arrival Rate. Below is a brief overview of these metrics along with their purpose, interpretation and usage.

B. Test Execution Productivity Trends

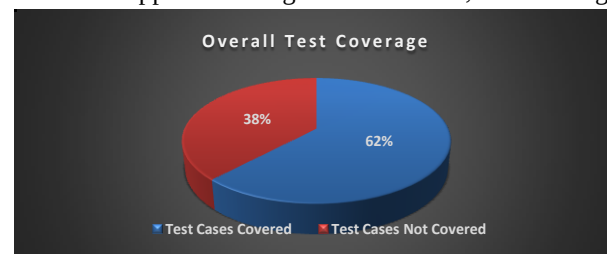
A helpful measure of test team's efficiency is the productivity at which the team carries out test execution. It is defined as the average no. of test cases executed by the team per unit of time, where the unit may be hour, day or week, but is generally taken as an hour or a day [6].



C. Test Coverage

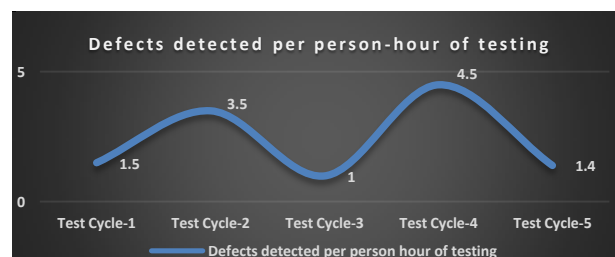
With ever increasing test banks, huge configuration support matrices and too frequently changing requirements, it becomes intuitively important for the test manager to smartly strategize in order to manage the resources at hand so as to ensure an optimum coverage of the test banks and supported configurations. Hence, test coverage serves as an important measure of the effectiveness of the testing process. The metric measures the amount (percentage or number) of components/test cases/ configurations actually tested successfully vis-à-vis the total no. of components/test cases/configurations for the product.

This metric provides the management team with a confidence on the product quality and acts as an important indicator for the effectiveness of the testing process. Adjoining figure illustrates the metric at a glance [6].



D. Defects per Hour of Testing

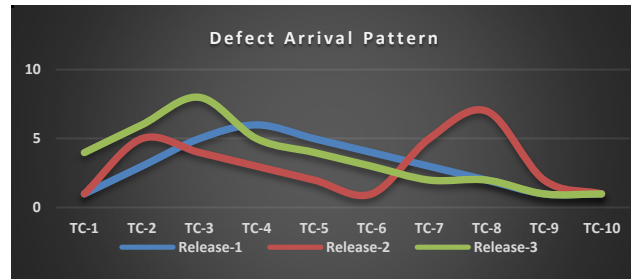
The effectiveness of the effort going in to test activities can be measured based on the no. of defects detected within this effort. A useful metric to gauge this effectiveness is the no. of defects detected per unit of time. As we move from initial stages of the software testing life cycle to the final stages, there should be a decreasing trend in the value of this metric. Adjoining figure depicts an illustration of this metric [6].



E. Defect Arrival Rate

Defect tracking and management is the key to driving software product quality through the software testing life cycle. One of the metrics used in this direction is the pattern of defect arrivals over time. Even with similar

overall defect counts during the test program, distinct patterns in defect arrival rates may mean different states of end-product quality. Adjoining figure depicts an illustration of this metric.



A defect arrival pattern with higher arrivals during earlier stages, an earlier peak and a decline during the final stages (compared to the baseline) is regarded as positive. Also, another way to term a defect arrival pattern positive may be a pattern that is constantly lower than the baseline throughout the testing life cycle and hence signifies that testing effectiveness is at least as good as the previous one. The concluding end of the defect pattern curve is especially important as it acts as a prime indicator of the product quality in the field. An elevated defect activity just before the product release is generally a sign of quality issues.

Based on the defect arrival patterns observed, an in-depth root cause analysis needs to be carried out to identify improvement actions. The teams should try to control defect arrival rates only in a natural way, which is a function of the test effectiveness, test progress and inherent quality of the product. Another related advice is to open defects as soon as they are found so they do not unnecessarily get delayed [6].

III. Misapplications of Common Test Metrics – A Case Study

To gauge the current state of metrics adoption by the software practitioner community, we conducted a pilot exercise by selecting multiple subsets of 5 software testing projects each in a software organization. We then studied the metrics adoption of 4 most common metrics for software testing life cycle by these projects: Productivity, Test Coverage, Defects per Hour of Testing, Defect Arrival Rate.

While making the selection of these project subsets, following project characteristics were taken in to account in order to ensure a homogeneity in the environment amongst all the projects and to ensure that there is no biasing involved:

Project Characteristic	Project State with respect to the characteristic
Project Type	Software testing projects being executed for market leading software product organizations in an outsourced environment
Team Structuring	Software Testing ownership lies with vendor, whereas Software Development with customer, multi-cultural teams
Project Duration	Typically 6 months, with test cycle durations of 2 weeks each
Financial Budget	(300 – 350) KUSD for 6 months period
Organizational Environment & Support	The project teams have the right environment for getting the job done. This covers everything from office space to desks and chairs to software development/testing tools
Type of Software Products under Consideration	3-tiered web-based products based on Java/J2EE technologies with comprehensive functionalities
Complexity of work	Medium to High
Communication needs	High with geographically distributed Test and Development teams
Development Methodology	Semi-agile, iterative model
Process Maturity	All projects adhering to the practices of and operating at CMMI Level-5
Team Skills	Fair
Team Size	8-12 team members including project manager, test leads and test analysts
Test Bank Size	5000 – 8000 Test Cases for each project
3rd Party Dependencies	No 3 rd party dependencies as such

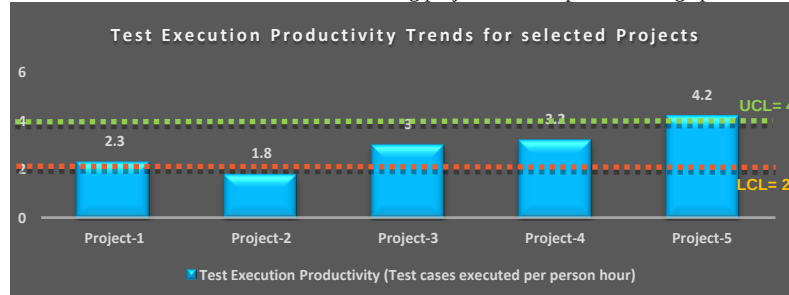
Table I: Characteristics of short-listed Projects

Below are the details of the Pilot exercise carried out along with the results observed for each of these metrics:

Metric: Productivity (Test Execution)
Purpose of the Metric:
- Indicates test team's efficiency - Defined as the average no. of test cases executed by a team per unit of time viz. an hour

Study Performed:

We studied selected subset of 5 software testing projects with respect to average productivity and below are the trends observed:



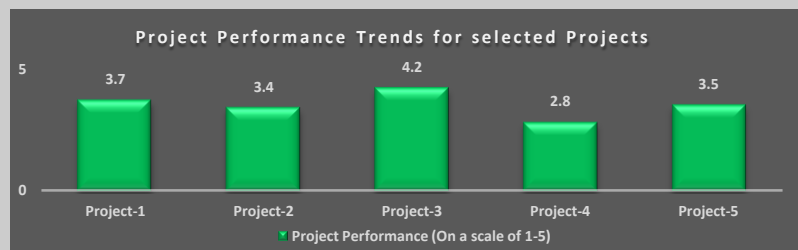
UCL: Upper Control Limit, LCL: Lower Control Limit

Looking at the productivity trends for the 5 projects, one would conclude the following:

- While project-5 is operating at the best level, project-2 is the worst
- Projects 3 & 4 are operating better than project-1

Other Perspective:

Below is a depiction of the performance evaluation exercise carried out for these 5 projects by an Internal Audit Board (IAB) within the organization primarily based on their speed of operations and customer feedback on their productivity. While carrying out the performance evaluation exercise, IAB takes care of normalizing customer feedback scores w.r.t. subjectivity in customer expectations and behavioral aspects:

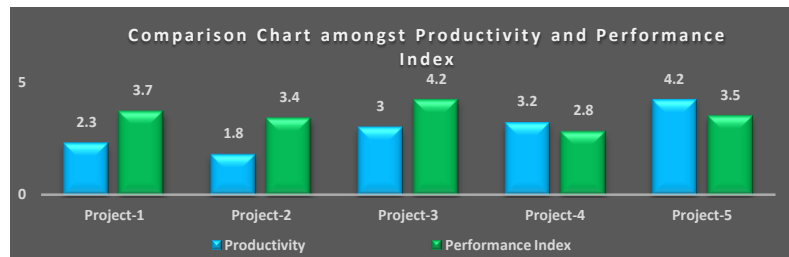


Looking at the performance scores for the 5 projects, one would conclude the following:

- Project-2 is not the worst performing project, while its productivity is the lowest
- Project-5 is not the best performing project, while its productivity is the highest

Comparative Analysis:

Below is a quick comparison chart amongst the Productivity and Performance Index for the 5 projects:



Seems awkward as the project with highest productivity is not the highest performing project and vice-versa

Further Investigation:

On further investigation, below anomalies were noted in the way these 5 projects have been adopting the metric of "Productivity" and using the same for measuring their performance:

- Though all the projects were using natural definition of test case execution productivity as (No. of Test Cases executed per person hour), the size of test cases in all the projects was different. The size of test case here is the no. of steps in a test case. This means that the projects with larger test cases are likely to show higher productivity trends and the ones with lower-sized test cases will generally seem to be performing better. This implies that the projects were not using a homogeneous base for measuring their productivity
- The test environment setup required in Project-2 was highly time consuming whereas Project-5 required a setup that was fairly quick to accomplish, hence influencing productivity

Conclusion:

Above evaluation exercise leads us to conclude:

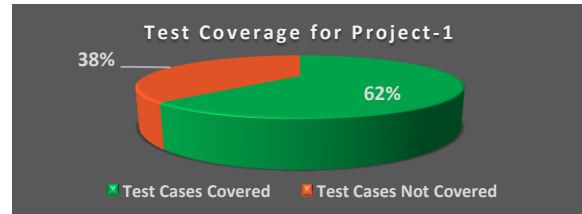
- All the projects have adopted Productivity as a metric "As Is" to measure their efficiency, however there is no baseline set for the size of a test case or normalizing the effort going in to test environment setup, resulting in the metric providing "misleading" views of project efficiency

Table II: Pilot Study – Test Execution Productivity**Metric: Test Coverage****Purpose of the Metric:**

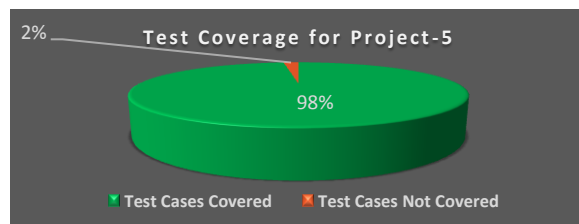
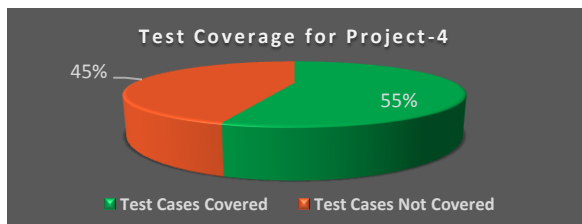
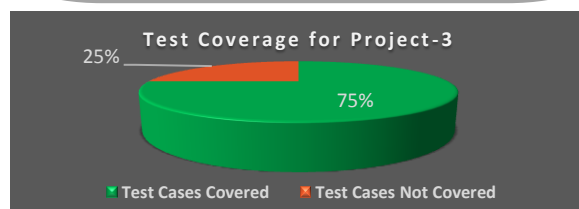
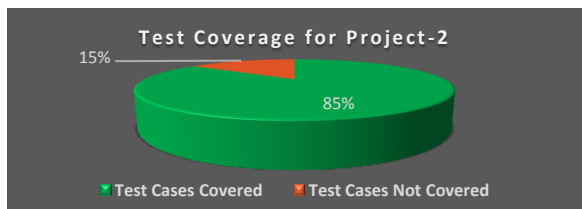
- Provides test management team and customer with a confidence on the product quality
- Defined as percentage of total test cases actually tested successfully

Study Performed:

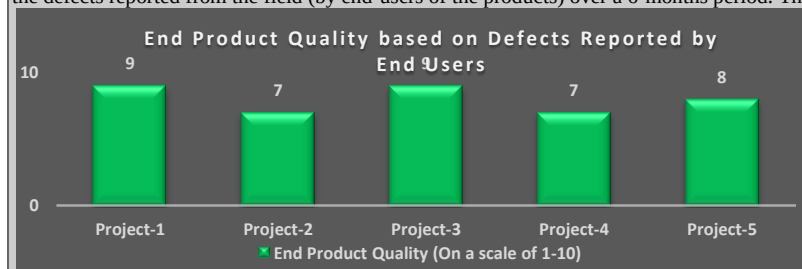
As a next step, we studied the above short-listed set of 5 software testing projects with respect to test coverage and below are the trends observed:

**Below are the observations from these trends:**

- Test coverage in all the projects varies from each other
- Project-5 has maximum coverage, whereas Project-4 has least
- Projects 1, 2 & 3 have a moderate coverage

**Other Perspective:**

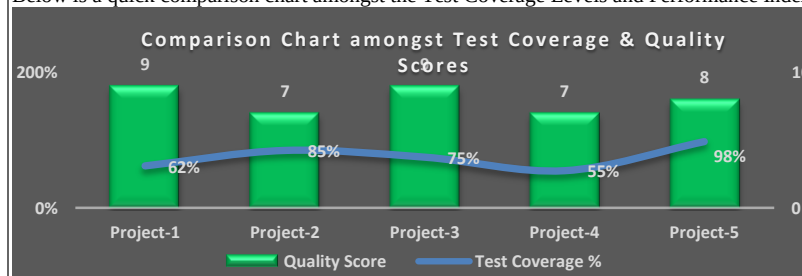
Now, below is a depiction of the end-quality of the software products delivered by these 5 projects. The end-product quality is derived based on the defects reported from the field (by end-users of the products) over a 6-months period. The quality has been depicted on a scale of 1 to 10:

**Looking at the end-product quality scores for the 5 projects:**

- For Projects-2 & 4, though there is a significant variation amongst coverage levels, quality scores are identical. Same phenomenon holds for Projects-1 & 3

Comparative Analysis:

Below is a quick comparison chart amongst the Test Coverage Levels and Performance Index for the projects:



There does not seem to be a perfect correlation between test coverage and end-product quality

Further Investigation:

On further investigation, below anomalies were noted in the way these 5 projects have been adopting the metric of “Test Coverage” and using the same for measuring the quality of work they deliver:

- Though all the projects were using high test coverage as a natural strategy to ensure high product quality, there were issues in the approach adopted for selecting the set of test cases to be executed – there were many test cases, which were not being executed, but were critical and failure-prone scenarios
- Projects, which were able to ensure a higher end-product quality in spite of a relatively lower coverage were covering sensitive failure-prone tests, leaving the safe and lower failure-prone areas leveraging smart test strategizing
- For Project-5, though only 2% test cases were left uncovered, the test bank itself was incomplete and missed on inclusion of some of key functional scenarios, which then slipped as defects to end-users of the product

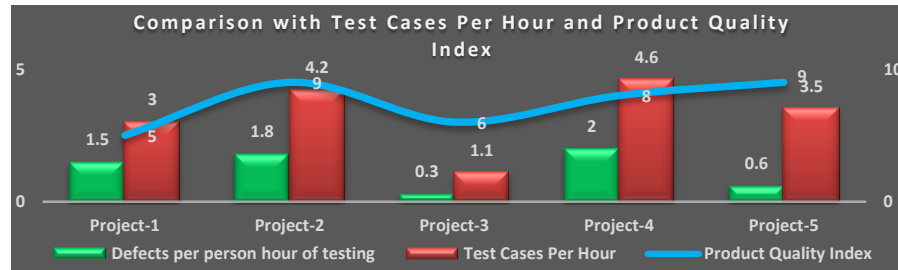
Conclusion:

Above evaluation exercise leads us to conclude:

- In spite of the fact that all the 5 projects are using “Test Coverage” as a metric to ensure product quality, there are issues in applying the metric. The application of metric needs to be tailored based on nature of test bank in a particular project, but there are no systematic guidelines available, which test managers can use while applying the metric

Table III: Pilot Study – Test Coverage

Metric: Defects per Hour of Testing													
Purpose of the Metric: • Indicates test team’s work effectiveness • Defined as the average no. of defects detected per person-hour of testing													
Study Performed: We then selected another set of 5 testing projects based on the qualifying criteria mentioned earlier and compared them with respect to average no. of defects detected per person-hour of testing, which gets executed:													
Defects detected per person-hour of testing <table border="1"> <thead> <tr> <th>Project</th> <th>Defects detected per person hour of testing</th> </tr> </thead> <tbody> <tr> <td>Project-1</td> <td>1.5</td> </tr> <tr> <td>Project-2</td> <td>1.8</td> </tr> <tr> <td>Project-3</td> <td>0.3</td> </tr> <tr> <td>Project-4</td> <td>2</td> </tr> <tr> <td>Project-5</td> <td>0.6</td> </tr> </tbody> </table>		Project	Defects detected per person hour of testing	Project-1	1.5	Project-2	1.8	Project-3	0.3	Project-4	2	Project-5	0.6
Project	Defects detected per person hour of testing												
Project-1	1.5												
Project-2	1.8												
Project-3	0.3												
Project-4	2												
Project-5	0.6												
Looking at defect trends for the 5 projects, one would conclude the following: • Testing happening in Projects-1, 2 & 4 is more effective while Projects-3 & 5 have less effective testing • Products under consideration in Projects-1, 2 & 4 are more defect prone as compared to products in Projects-3 & 5													
Other Perspective: As a next step, we measured the quality of the end-product delivered by these projects. Below is a depiction of the metric of “Product Quality Index” for these 5 projects. Here, the Product Quality Index is derived based on the defects reported from the field (by end-users of the products) over a 6-months period. The quality has been depicted on a scale of 1 to 10.													
Product Quality Index based on Defects Reported by End Users <table border="1"> <thead> <tr> <th>Project</th> <th>Product Quality Index</th> </tr> </thead> <tbody> <tr> <td>Project-5</td> <td>9</td> </tr> <tr> <td>Project-4</td> <td>8</td> </tr> <tr> <td>Project-3</td> <td>6</td> </tr> <tr> <td>Project-2</td> <td>9</td> </tr> <tr> <td>Project-1</td> <td>5</td> </tr> </tbody> </table>		Project	Product Quality Index	Project-5	9	Project-4	8	Project-3	6	Project-2	9	Project-1	5
Project	Product Quality Index												
Project-5	9												
Project-4	8												
Project-3	6												
Project-2	9												
Project-1	5												
Looking at the Product Quality Indices for the 5 projects: • There does not seem to be a perfect correlation between Defects Per Hour and end-product quality • For Projects-2 & 4, though Defects Per Hour is very high, end-product quality as well is on higher side • Similarly for Project-3, though Defects per Hour has been the lowest, product quality is on the lower side													
Comparative Analysis: We then tried to gauge the trend of No. of Test Cases executed per Hour of Testing in all these projects. Below is a depiction of the comparative analysis amongst Defects per Hour of Testing with “Test cases executed per hour” and “Product Quality Index” for these 5 projects:													
Looking at the test case execution and product quality scores for the 5 projects: • No doubt, the product quality index for Project-1 aligns with No. of Defects detected per hour of testing. Hence, Defects per Hour serves as a reliable predictor of product quality here • However, if you see Projects-2 & 4, though the Defects per Hour has been higher, Product Quality Index too is on the higher side. We observe that the main contributor for a higher Defects per Hour metric has been higher no. of test cases getting executed in these projects within an hour • While Defects per Hour reported for Project-5 is aligning with the Product Quality Index, Project-3 Product Quality Index is lower in spite of a low value of Defects per Hour. One main reason for a low value of Defects per Hour in Project-3 has been a low no. of test cases getting executed in this project within an hour													

**Further Investigation:**

On further investigation, below anomalies were noted in the way these 5 projects have been adopting the metric of “Defects per Hour of Testing” and using the same for predicting the quality of end-product:

- Though all the projects were using high no. of defects per hour of testing as an indication of a possibly defect-prone product, the factor of “Test cases executed per hour (Test Execution Productivity)” was completely ignored
- Some of the Projects, which were operating at higher levels of Test Execution Productivity were able to catch more defects in per hour of testing
- Other projects operating at lower levels of Test Execution Productivity were able to catch low no. of Defects per Hour of testing and hence projecting them as low-defect prone ones was not completely accurate

Conclusion:

Above evaluation exercise leads us to conclude:

- Though Defects detected per hour can provide an indication of the extent of defect-proneness of a software product, the calculation of the metric needs to be normalized on the basis of other associated metric namely “Test Execution Productivity”

Table IV: Pilot Study – Defects per Hour of Testing

Metric: Defect Arrival Rate**Purpose of the Metric:**

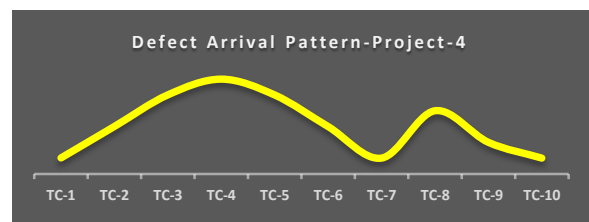
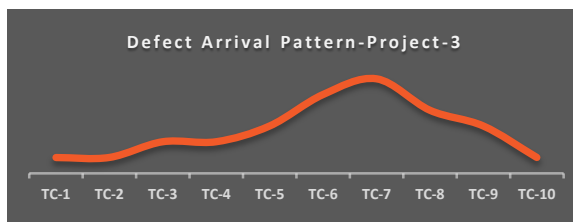
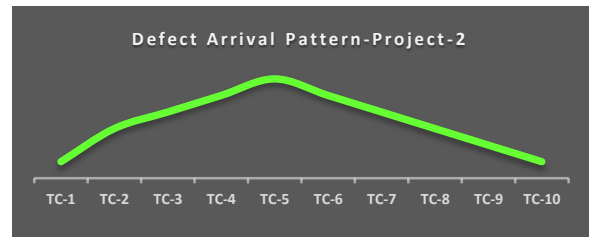
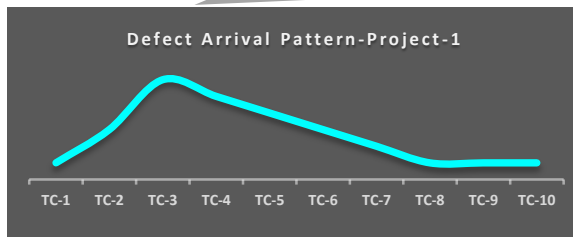
- Indicates pattern of defect arrivals over time, helps predict end-product quality
- Defect pattern with higher arrivals during earlier stages, an earlier peak and a decline during the final stages is regarded as positive

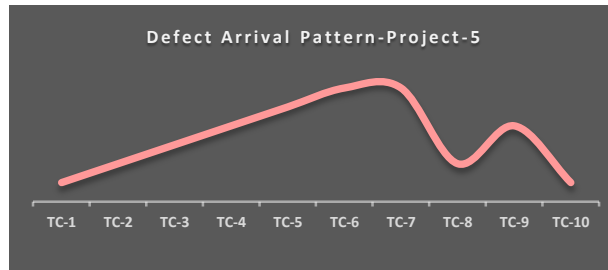
Study Performed:

We studied the short-listed 5 testing projects with respect to their defect arrival pattern and below patterns were observed (TC stands for Test Cycle here):

Looking at defect arrival pattern for the 5 projects, one would conclude the following:

- Arrival of defects follows an ideal pattern in Project-1 signifying the possibility of low defect leakage in to field
- Detection of defects for Project-2 peaks somewhere in the middle indicating medium risk of defect leakage
- For Project-3, since most of the defects are getting reported in later part of the test program, there is a high risk that there will be defects getting leaked to the field
- Project-4 starts with an ideal pattern and slows down, however there is again a flow of defects during test cycle-8 causing the project to fall in to high risk zone
- Project-5 looks to be the most risky out of the 5 projects as firstly the initial spike in defect reporting is seen delayed and then in test cycle-9, there is another stream of defects causing the project fall in to highest risk zone





Other Perspective:

Now, let's look at other side of the picture. Below is a depiction of the end-product quality of software products delivered by these 5 projects. The end-product quality is derived based on the defects reported from the field (by end-users of the products) over a 6-months period. The quality has been depicted on a scale of 1 to 10:

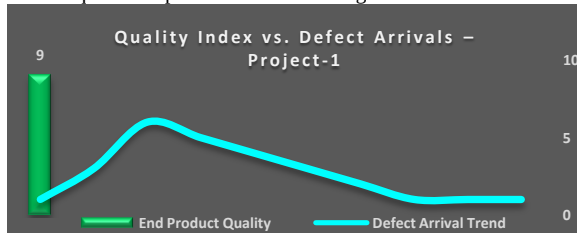


Looking at the end-product quality scores for the 5 projects:

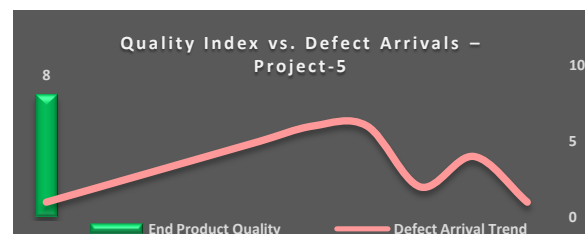
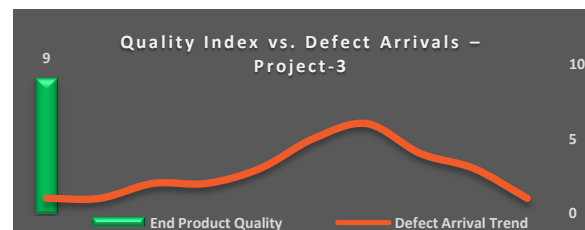
- Product Quality for Project-1 has been the highest as expected
- Product Quality for Project-2 has fallen in spite of the defect pattern indicating a medium level risk on product quality
- Product Quality for Project-3 has been exemplary in spite of defect pattern indicating high risk of defect leakage
- Product Quality for Project-4 has gone down as expected
- While Project-5 has been designated as the highest risk project for quality issues, the actual results have been contrary to the prediction

Comparative Analysis:

Below is a quick comparison chart indicating Defect Arrival Patterns and Quality Indices for the projects:



There does not seem to be a perfect association between defect arrival pattern and end-product quality especially for projects 2, 3 and 5



Further Investigation:

On further investigation, below issues were found in the way these projects have been adopting the metric of “Defect Arrival Rate” and using the same for predicting defect leakage to the field:

- Though all the projects were using defect arrival pattern to forecast quality, none of the projects were taking in to account the defect severity in to consideration
- For instance, for Project-2, most of the defects reported though late in cycle, were medium severity defects and development teams were able to quickly fix them up
- For Project-3, though defect reporting spiked during final test cycles, since most of these defects were cosmetic kind of defects, they were quickly fixed and end-product delivered to the field was a high quality one
- For Project-5, there were mostly low severity defects reported during 2nd spike seen in the defect arrival pattern, enabling the project achieve a decent quality product

Conclusion:

Above evaluation exercise leads us to conclude:

- While “Defect Arrival Rate” serves as a good predictor of the end-product quality, one must take in to account defect severity/defect type to gauge the amount of risk on the quality of final product

Table V: Pilot Study – Defect Arrival Rate

IV. Key Observations and Issues

We observe from the above study that though all the projects have adopted “the most popular” metrics as is with an intent to keep a check on their performance and target continuous improvements in the efficiency and effectiveness, the expected results are in jeopardy. The inter-project comparisons are not providing an accurate picture and it is not possible to derive any benchmark values for the metrics, which can be taken up as a guidance for similar projects. The projects while adopting metrics need to take the big picture in to account and the essence of metrics needs to be tailored. For example, the number of steps in a test cases need to be taken in to account while evaluating a project’s test execution productivity trends. The application of the metric of test execution productivity needs to be tailored based on test case size to have the correct picture. Similarly, while comparing the value of this metric amongst multiple projects, the test case size needs to be taken in to account. Otherwise, the metrics values and trends are bound to give a misleading direction. Additionally, the interpretation of a metric cannot be considered in isolation. It is vital to consider other associated metrics while making decisions. For example, while assessing the effectiveness of a testing effort on the basis of Number of Defects detected per hour of testing, it is vital to take in to account the amount of testing that gets carried out in an hour. For predicting the end-product quality on the basis of the metric of test coverage, it is important to consider the severity of test cases, which have not been covered. While gauging the health of a software product by observing the defect arrival pattern, it is important to consider severity and type of defects detected.

V. Way-forward & Recommended Research

Literature and Research related to software metrics recommend numerous software metrics, which can be utilized to enhance project performance and make informed decisions. They also emphasize importance of considering metrics associations while making the right decisions. However, the “many” metrics present in literature seem to be scattered across books, white papers, conference submissions and journals and there is no consolidated whole available. There is need of a recommendation model, which the test manager community can bank up on as a guidance aid to get an insight in to what are the “right” metrics amenable for project situations and which associated metrics to take in to account while making decisions based on a metrics set. Such a metrics model should enable usage of metrics in a consistent way across projects.

VI. Conclusion

It’s a well-known fact that adoption of metrics in software projects helps their managers make informed decisions, enabling them run these projects more smoothly and on the path of continuous improvement. This fact is substantiated by the gamut of literature and research outcomes related to metrics, which have been generated over past 40+ years. In spite of the enormous efforts made to bring about advancements in the field, industry’s adoption of various metrics is still at a basic level and has not changed much over past 20+ years. Moreover, the project manager community still struggles to identify the right set of metrics pertaining to their specific needs and looking for guidelines on how to make the right usage of selected metrics sets. There is a need to devise a guidance framework or a recommendation model for metrics adoption that project managers can depend up on as a guiding aid while selecting appropriate metrics for their project-specific needs. Such a model should guide project managers on which are the most appropriate metric sets for their project-specific needs, which are the associated metrics that should be considered in conjunction and how adoption of these metrics needs to be tailored to align with the project-specific environmental context.

References

- [1] Fenton, N. E., & Ohlsson, N. (2000). Quantitative analysis of faults and failures in a complex software system. *Software Engineering, IEEE Transactions on*, 26(8), 797-814
- [2] Karg, L., & Voelz, D. (2007). Scientific software test management-a research agenda. In *Software Engineering (Workshops)* (pp. 65-70)
- [3] Dekkers, C. A. (1999). The secrets of highly successful measurement programs. *Cutter IT Journal*, 12, 29-35. 25
- [4] Fenton, N. E., & Neil, M. (1999). A critique of software defect prediction models. *Software Engineering, IEEE Transactions on*, 25(5), 675-689
- [5] Goodman, P. (1993). *The Practical Implementation of Software Metrics*. McGraw-Hill, Inc
- [6] Realizing Efficiency & Effectiveness in Software Testing through a Comprehensive Metrics Model - <http://www.infosys.com/engineering-services/white-papers/Documents/comprehensive-metrics-model.pdf>
- [7] *Metrics and Models in Software Quality Engineering – Second Edition* by Stephen H. Kan
- [8] *Software Engineering: A Practitioner's Approach*, 6/e by Roger S Pressman, R.S. Pressman and Associates, ISBN: 0072853182, Copyright year: 2005