Journal Homepage: - www.journalijar.com

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH (IJAR)

Article DOI: 10.21474/IJAR01/10383
DOI URL: http://dx.doi.org/10.21474/IJAR01/10383

ISSN NO. 2320-5407

RESEARCH ARTICLE

A CONCEPTUAL SHORT COMMUNICATION PAPER ON PERFORMANCE APPRAISAL TECHNIQUE FOR SOFTWARE DEVELOPERS AND SOFTWARE QAS BASED ON SOFTWARE QUALITY METRICS

Fariha Al. Ferdous
Dhaka, Bangladesh

Manuscript Info

Manuscript History
Received: 30 November 2019
Final Accepted: 31 December 2019
Published: January 2020

Keywords: -
Project Management, Performance Appraisal, Employee Evaluation, Software Quality Metrics

Abstract

Performance appraisal techniques have been in research for a while and software quality metrics have been introduced for the employee evaluation process already. Currently, there are several organizations, where proper employee evaluation process gets failed due to inefficient usage of software quality metrics and deficiency of knowledge about them. Thus, the topic of this conceptual paper arose. In this regard, such software quality metrics have been studied and discussed and a conceptual research has been conducted in this paper for the purpose of easy performance appraisal technique research and deployment further in future for software developers and software QAs. Especially, the enhanced description and analysis of different software quality metrics and how proper utilization of software metrics can be effective in terms of employee evaluation, has been done here in an authentic way.

Introduction: -

Many organizations rely on Performance Appraisal techniques for their improvement. Eventually, various techniques have been formed to evaluate employees of different sectors of an organization. Numerous software quality metrics have been established for this purpose but none of them were investigated enough to make them into practice. Following these, number of software quality metrics have been discussed in this conceptual paper to support the performance appraisal techniques for Software Developers and Software QAs in organizations.

However, how should software developers and software QAs be appraised, has been off from the research as far investigated for this paper. Not much revealed about the performance appraisal techniques for such employees, although several software quality metrics have been provided in several articles and theories. Researchers have still been going on but could not result in practice yet. Organizations cannot seem to find ways to properly evaluate their employees, especially at IT firms for the software developers and QA engineers. Practitioners seem to not find ways to conduct performance appraisal for these employees within such organizations.

In this paper, we have put emphasis on a broader insight into industrial practice on performance appraisal technique for Software Developers and Software QAs utilizing the already established software quality metrics.

Corresponding Author:- Fariha Al. Ferdous
Address:- Dhaka, Bangladesh.

Methodology:-

Search strategies:

Several keywords and strategies have been used to conduct the research on web libraries. Google Scholar search engine has been used and keywords were following:

Performance Appraisal, Software Quality Metrics, Software Metrics for Software Developers, Software Metrics for Software QA, Performance Appraisal for Software Developers, Performance Appraisal for Software QAs, Performance Appraisal for Software Testers, Employee Evaluation Techniques.

Inclusion and exclusion criteria:

Priorities have been given to topic area of Performance Appraisal of Software Developers and Software QAs. Any other Performance Appraisal criteria have been excluded from this search query for this research. Depth in the topic of Performance Appraisal techniques has also been given with caution.

Theoretical framework of Software Quality Metrics:

Agile Process Metrics for Software Developers

Velocity:

Velocity is measurement of how many features got done during a period. Most often, velocity is measured by sum of story points but it can also be measured by number of feature tickets delivered within a period. This period can be a sprint, such as, if 20 story points done within 2 weeks and that equals to 1 sprint, then average velocity will be 10 of that team individually. There is another measurement which is the average of story point. From theory we can say that, falling velocity proves there is a bottleneck that the management should investigate to find out the cause behind it. If velocity is stable means target achieving process is working right and an increasing velocity means performance quality increased. If an employee is consistently working on any lower level average story point then it shall mean s/he is not capable of taking on tasks or projects of higher complexity.

Hypothesis:

Velocity gives us insight on measuring the team's efficiency and capability accomplishing a target of story points.

Throughput:

The delivery rate; conceptually, this is the total work done by a team or an individual within a period of time. Here, work could be any task, milestone, feature or bug etc. Average throughput can also be measured. This metric helps to find out which department of software projects is getting their daily workload done mostly, whether, new features, support tickets or bugs etc. Analyzing the theories from the references, the average throughput is the threshold since it is the ideal delivery rate, so if the throughput is closer to average throughput then it shall define a higher delivery rate. Throughput can also give insight into the capacity planning of the team whether to adjust workload or not by measuring average throughput against current workload.

Hypothesis:

The closer throughput to the average throughput, the higher delivery rate it is.

Open Pull Requests:

These are the pull requests with status open under repositories waiting to be pulled by senior team members or managers. These requests stay there with this status until the code gets reviewed. Once the code is reviewed and necessary changes are made the requests are then put as closed. This metric can be measured from tools like GitHub.

Hypothesis:

Analyzing the theories, tracking Open Pull Requests may help in identifying bottlenecks as this impacts on iteration flow and throughput of a team or individual for any amount of work within a period of time. This may indicate whether the team requires additional time or resources in order to complete and deliver the current projects.

WIP:

Work in Progress metric gives us the number of how many tickets or works are currently in progress. An increasing WIP with a low Resolved Count means there must be some bottleneck in the process or the processes are inefficient. So, ideally, this number should be stable over timeline. On average, the number of WIP will be close to one to one ratio which suggests proceeding with one thing at a time.

Hypothesis:
This metric gives us insight of the team ‘s or individual’s capacity of handling current pressure or workload.

Cumulative flow:
This metric shows the time delivery, status of work from one state to another, the repeatability and the health of the process being followed for the development of the software projects. This shows whether there is a bottleneck in any state, this will be indicated when there shall be a stack of large number of tickets in that state. If tickets are stacked in Resolved state, not making it to Delivered state then it means there is a bottleneck at QA. So, ideally, the state to state transition should be smooth and stable to indicate a healthy process is being followed for the resolution of the existing workload. This metric can also show us tickets reverting back to the previous state which proves repeatability and needs to be noticed by the manager. This way work becomes predictable and easy to manage for maintaining a smooth working process.

Hypothesis:
This metric shows if there is bottleneck in any state that is hampering the regular workflow.

Lead time:
Lead time is from developing an idea of a software product or even the initialization of the development of software product to its output into delivery. This software metric may give an insight into the trend of our typical software delivery process, that whether to bring a change or not in it for a faster and more efficient delivery. Higher Lead time may require proper investigation. Lead time may also vary on other factors like resource count, velocity or even WIP. Variants in those factors may lead to variety in Lead time for any size of project.

Hypothesis:
Ideally, lower Lead time with high velocity or WIP proves fluency in responsiveness of software developers to customer demands.

Cycle time:
This is either by status or type. Cycle time per type means how long a bug or a feature took to transform from one status to another. Cycle time per status means how long a feature or a bug took in a particular state. How a particular module or a change in a module takes time and this time is called Cycle time. So, ideally, Cycle time may show us the turnaround time of a bug by measuring how much time it took to resolve it from open state to close state.

Hypothesis:
A hike in cycle time graph may help in identifying bottlenecks in that particular state, so we understand that it is time to take efficient steps in order to get back the quality in effort in that state.

Code churn:
Code churn represents the rate of lines of a code added, modified or deleted in the same file. Theoretically, software developer with lower churn is an efficient coder. Higher code churn will also result in missing deadlines, lack of quality and inefficiency as it resembles that the individual is probably struggling in providing efficient contribution in coding.

Hypothesis:
Code churn shows the efficiency of software developers.

MTBF:
From the theory of references, Mean time between failures is the mean time between consecutive failures of service or system. Theoretically, a higher MTBF rate is an ideal rate.

Hypothesis:
This software metric shows the consistency of software systems, whether operation and maintenance department need checkup or not.

MTTR:
Mean time to recover; this is the mean time before recovering a system after failure takes place. This software metric resembles the efficiency of software developers in recovering back a system after a period of service failure. So ideally, the time has to be low to ensure a well-run system.

Hypothesis:
The lower MTTR, the higher the team efficiency is.

ACR:
This actually means the amount of time the app is not functioning and fails to perform. Availability, as a percentage, is often spoken of "nines", as in how many "nines" in the decimal point after 99%. For a month, 3 "nines", or 99.999% availability would mean up to about 40 minutes of downtime. 5 "nines" or 99.99999% availability is less than a minute of downtime each month. So ideally, like MTTR, this has to be low. Application Crash Rate= (Times Down time/ Total period of time) *100

Hypothesis:
From the theory we can say that, having lower ACR means system running smoothly,

DRE:
This KPI metric actually proves how efficient the dev team is in removing defects before delivery. So theoretically, this has to be high. From theory, the closer DRE to 1 the fewer bugs found after delivery. DRE rate acknowledges the defect removal process efficiency, the planning of software project development and delivery. So, from theory, how much efficient the team is in removing defects before the customers find them is very important. This metric may affect the bug count of a software project. Here, Defect Removal Efficiency Rate= (Total Defects resolved/ (Bugs found before + after delivery)) *100

Hypothesis:
A higher DRE ensures lower Bug count or Bug Escape Rate.

Defect Density:
This is the number of defects or bugs per LOC where LOC is typically measured as KLOC (Thousands of Lines of Code).

Defect Density = Total Defect/KLOC:
From theory, if defect density is high that means QA team is efficient enough to reduce bug count to zero.

Hypothesis:
A higher number of Defect Density means lower number of Escaped Bugs as well as an inefficient software developer but an efficient QA team.

Front-end app Response Time:
This is basically the client-side page transaction time. So, by definition, this value has to be low.

Hypothesis:
A higher Front-end app Response time basically proves inefficiency in the development work of the front-end team. This may result in negative feedback from the users.

Burn down Chart:
This is the chart with two lines. One is the ideal graph other is the actual burn down graph line. Ideally, the graph line starts with values and ends at 0 meaning all work count has finished to zero. And by actual graph line, it determines whether in real, all work has finished to zero or there have been some delays, whether projects underestimated or overestimated.

Hypothesis:
A lagging graph means the projects overestimated and line leaving the graph line showing pending work left means, projects underestimated. Moreover, in an iterative work system, since new tasks and supports or bugs arise every

day. So, in such a system, a Burn-Down graph may fail to show the actual workflow of a software development team as work status will vary each day.

Backend Response time:
This is the server response time to requests made from client side. By definition this value has to be low for a better user experience.

Hypothesis:
Higher Backend Response time may result in faulty back end user experience. If the server doesn't respond in time as expected negative customer feedback shall arise.

Percentage number of Incidents:
This provides insight into the percentage of service interruptions or incidents happened within a period of time.

Hypothesis:
Having a high percentage of Incidents proves a faulty management system and maintenance support.

Spent time:
Spent time is the amount of time spent per resource within working days of a sprint. From theory, if time spent regularly with a potential delivery rate may result in maximum output.

Hypothesis:
Higher throughput as well as higher spent time may ensure higher number of Resolved count where the opposite may lead to lower resolved issues.

Resolved Count:
This is the amount of work delivered per spent time. Higher throughput as well as higher spent time may ensure higher number of resolved count and vice versa. But the Resolved count may also fluctuate with velocity or throughput. From theory, if a project of a high story point is attempted to resolve in minimal amount of days may output into an unresolved issue.

Hypothesis:
Higher velocity may lead to lower Resolved Count. Also, tasks with lower velocity with lower throughput may also lead to lower resolved count.

State Transition-Code review to Code in progress:
This verifies the quality of written code. Bouncing back and forth in states may prove bottlenecks in resource availability and quality.

Hypothesis:
Higher rate of going back to code in progress state from code review state determines inefficiency in quality of a software developer.

Agile Process Metrics for Software QAs
Bug Escape Rate:
This is the count of bugs or bug tickets generated and reported by customers. Having count of escaped bugs gives an insight into the QA process, that is, how much effective the QA process is. Escaped bugs = Escaped bugs / (Defects found in process + Escaped bugs)

Hypothesis:
From theory, an increasing number of Bug Escape Rate proves an ineffective QA process and inefficiency in QA resource quality and vice versa.

High Priority Bugs:
This is the percentage of priority of bugs over the number of total bugs.

Hypothesis:
Theoretically, high percentage of High priority bugs means, QA department is proving to be inefficient and this needs to be noticed.

Test Case Effectiveness:
This is analyzing the test cases written and run by QA engineers and finding out the effectiveness of them. This can be measured by the number of defects found by a particular test case run by the QA.

Hypothesis:
A higher effective test case will generate lower number of bugs but higher number of defects.

Conclusion:-

Utilizing the Software Quality Metrics should be a vastly adopted technique in many organizations for evaluating the performance of software developers and software QAs. These should not only be followed, but also KPIs can be set as a standard for utilizing the performance appraisal technique properly.

In this conceptual paper, such software quality metrics have been discussed and a comparative study has been conducted dividing them into two categories. One is for Software Developers and another is for Software QAs. For QA Engineers it matters how effective are the test cases with a low percentage of Bug Escape Rate. There is also other software quality metrics for QA Engineers like, High Priority Bug rate and Spent time. Utilizing these may result in bug-free delivered projects with an effective QA process.

For Software Developers, it matters how effectively and efficiently spent time is utilized by spending maximum time with high delivery rate touching all level of story points, that is, with a standard average velocity generating zero defects with lowest amount of code churn.

Maintaining an appropriate value of all the ACR, MTTR, MTBF, Percentage of Incidents, Front end loading and Back end response time are also a priority for ensuring a healthy software system.

References:-

1. Tanjila Kanij and John Grundy (2014). Performance Appraisal of Software Testers. Information and Software Technology, Volume 56, Issue 5, May 2014, Pages 495-505, https://doi.org/10.1016/j.infsof.2013.11.002
2. Angelo S. DeNisi and Robert D. Pritchard (2015). Performance Appraisal, Performance Management and Improving Individual Performance: A Motivational Framework. Cambridge University Press: 02 February 2015, https://doi.org/10.1111/j.1740-8784.2006.00042.x
3. 9 metrics that can make a difference to today's software development teams, https://techbeacon.com/app-dev-testing/9-metrics-can-make-difference-todays-software-development-teams (accessed in December, 2019)
4. 8 Essential Software Development metrics for team productivity, https://blog.usenotion.com/8-essential-software-development-metrics-for-team-productivity-273737868960 (accessed in December, 2019).