International Journal of
INTELLIGENT SYSTEMS AND APPLICATIONS IN ENGINEERING
ISSN:2147-6799 www.ijisae.org Original Research Paper

Performance Measurement Framework for Agile Scrum Software Engineering Environment

Deepa T.[1], Dr. Thaddues S.[2], Dr. Dhamodharan G.[3]

Submitted: 01/10/2023 **Revised**: 21/11/2023 **Accepted**: 01/12/2023

Abstract: Performance of individual developers and effort on reworks in agile scrum environments has an impact on the productivity and quality of the developed software. The traditional agile metrics do not consider reworks in measuring productivity. Yet, an enhanced methodology to compute individual performance and quantify reworks in agile scrum contexts is required. By introducing Earned Value Analysis based Productivity Ratio and Blended Quality Metric based on Code Maintainability Index, this gap is bridged. Applying the metrics to ten real time software projects over eighteen months, having twelve sprints each, a framework is decided for performance management in agile scrum development. The findings indicate the efficacy of the metrics in providing valuable insights to improve productivity and quality measures in software development environments.

Keywords*: Agile Scrum, Performance Management, Software Metrics, Software Engineering Environment*

1. Introduction

Measuring individual performance and rework in an agile scrum environment is discussed much in terms of productivity and quality in software development environments. While it is argued based on an agile manifesto that focus on individual performance goes against the principles of agile and scrum, others believe that understanding individual contributions provide valuable insights into team dynamics, skill gaps, and potential areas for improvement [1]. Tracking and quantifying rework can help identify issues with the quality of work, the clarity of requirements, or the effectiveness of testing procedures [2].

Studying various productivity and quality metrics, it is identified that all categories of metrics do not consider productivity from the aspect of reworks, change requests or defects. They are treated as part of the next sprint. To be precise with regard to productivity from a business perspective, an enhanced productivity measurement framework is required. This paper proposes a methodology to compute individual performance and rework in agile scrum contexts. The key metrics used to incorporate rework and maintenance are improved earned value analysis and code maintainability index.

Presenting the collection of metrics for productivity and quality in agile scrum environments systematically from various research works, the limitations are highlighted. To improve upon these limitations, Earned Value Analysis based Productivity Ratio (EVAPR) and Blended Quality Metrics based on Code Maintainability Index (BQMCMI) are proposed as productivity and quality metrics respectively. Defining an evaluation methodology, the metrics are applied to ten projects over a period of eighteen months of twelve sprints. The results are tabulated and findings are deliberated as a framework for performance management in agile scrum development.

[1]Research Scholar, [2]Associate Professor, [3]Assistant Professor
[123]PG & Research Department of Computer Science,
Don Bosco College (Co-Ed), Guezou Nagar, Yelagiri Hills,
Tamilnadu-635854.
(Affiliated to Thiruvalluvar University)
[1]margaret.deepa23@gmail.com,[2]thad@boscoits.com,
[3]haidhamo@gmal.com

2. Agile Scrum Software Metrics

Many authors have proposed, and various industries have implemented, a wide range of agile metrics. Team velocity, posited by Ahmed et al. [3],Hayes et al. [4], Rajani Dixit et al. [5], and Fernando Almeida et al. [6], measures the amount of work a team can tackle in a single sprint, offering an accurate gauge of productivity over time

In addition to team velocity, agile project management also relies heavily on sprint burndown charts. This tool provides a graphical representation of the remaining work within a sprint. The significance of the sprint burndown chart has been acknowledged [3][4][5]. Release burndown[8] illustrates the progress of a team towards the completion of a product release, offering a broader view of progress across multiple sprints. A few researchers emphasize the relevance of monitoring completed stories and tests in agile project management sprints and the consequences of violating standards [7][9][10]. They advocate for the assessment of team satisfaction and turnover rates as vital indicators of productivity and team cohesion. A few authors [12],[7], in contrast, discuss the metric of failed deployments, which enumerates unsuccessful software deployments, providing a perspective on deployment procedures and software dependability .

The concepts of lead time and work in progress, emphasized by [4], [7] provide a comprehensive understanding of the total duration and the pending tasks within a project. Tracking business value delivered, sprint goal success, total project duration, time to market, total product cost, and return on investment by [9],[13]. These metrics collectively offer a comprehensive view of the financial and operational performance of a project. The issue of lack of cohesion in methods within a software class or module as an essential metric to assess software design quality [7],[9]. Cycle time is discussed in [14],[7]. Table 1 summarizes productivity metrics for agile scrum.

Table 1: Agile Scrum Productivity Metrics

Category	Metric	Description
Effort Metrics	Team Velocity [3][4][5][6]	Measures the amount of work a team can tackle in a single sprint.
	Sprint Burndown [4][5][6]	Represents the remaining work in a sprint over time.
	Release Burndown [3][8][17]	Shows progress towards the completion of a product release across multiple sprints.
	Number of Stories [7][9][16]	Counts the total user stories completed in a specific time frame.
	Work in Progress [4][13]	Represents the total number of tasks or projects that are currently being worked on but not yet completed.
Team Dynamics Metrics	Team Satisfaction [6][11]	Measures the overall happiness and morale of a team.
	Team Member Turnover[6][11]	Measures the rate at which team members leave and are replaced within a given time period.
Efficiency Metrics	Lead Time [3][13][14]	Measures the total time from the moment a new task is requested until it is completed and delivered.
	Cycle Time [14]	Measures the total time from the beginning to the end of a process or task, including process time, delay time, and inspection time.
Business Impact Metrics	Business Value Delivered [9]	Measures the total value or benefits a completed project brings to a business.
	Sprint Goal Success [9]	Evaluates whether or not the team was able to achieve the predefined goals set for a specific sprint.
	Total Project Duration [9]	Measures the entire length of time from the start to the completion of a project.
	Time to Market [9]	Measures the total time it takes from the inception of a product or service idea to its official release in the market.
	Total Product Cost [9]	Calculates the overall costs involved in developing, producing, and maintaining a product from inception to discontinuation.

	Return on Investment [9]	Calculates the profitability of an investment or project by dividing the net profit by the total investment cost.
Quality Metrics	Standard Violation [16]	Measures non-adherence to established guidelines, conventions, or best practices.
	Defects per Interaction [7][16]	Calculates the average number of errors found per interaction with a software feature.
	Defect Density [6]	Measures the number of confirmed defects per unit size of a software product.
	Failed Deployments [12]	Counts the number of unsuccessful software deployments.
	Number of Tests [9][7][16]	Counts the total test cases designed and executed in a given timeframe or project phase.

Although velocity is used much in agile scrum environments, to measure team and individual performance, velocity isn't suitable for individual or external evaluations due to its team-specific and context-dependent nature. Burndown charts assume a linear work progression, which may not reflect real-world challenges or changes. These charts also focus on remaining work and can lack granularity, making it difficult to identify specific task progress or bottlenecks. By emphasizing the number of stories completed, this metric fails to consider the complexity or scope of individual stories. Number of Tests values the quantity of tests conducted over their effectiveness, potentially overlooking the quality of the testing process. Defects per Interaction metric doesn't consider the severity or complexity of defects, only their number.

3. Earned Value Analysis based Productivity Ratio (EVAPR)

Earned Value serves as a project management technique utilized to assess the progress and performance of a project in relation to its planned objectives at a given point in time. It also aids in forecasting future performance. The method involves analyzing three key dimensions: planned, actual and budgeted expenditures for completed work. By considering these aspects, Earned Value provides a comprehensive and insightful overview of the project's status and trajectory [15]

Through earned value analysis, this metric offers an accurate measure of team and individual productivity, including rework efforts for user stories. It provides a comprehensive view of the real effort required for user story completion, encompassing the work necessary to tackle unclear requirements, insufficient testing, or human errors.

1. **Initial EV (after UAT and release):** Typically, in agile projects, Earned Value is calculated using Story Points. So, for each user story, once it has been accepted in the User Acceptance Testing (UAT) and subsequently released, it has "earned" its value. Therefore, the Story Points (or equivalent) of all such completed and released user stories are summed up.

Let's denote the Story Points of user story i as SP_i. Then the initial EV for a sprint with n completed user story is given by:

$$\boxed{\textit{Initial EV (Before Rework)} = {}_{i=1}\sum^{n} SP\_i}$$

Initial EV (after UAT and release) = ∑ Story Points of all completed tasks in a sprint

2. **Re-evaluated EV :** This is more subjective and depends on how the re-evaluation is being performed. If it's a simple adjustment based on customer feedback, the user story is a revised Story Point value based on customer feedback, then this has been summed up.

$$\boxed{\textit{Re-evaluated EV (with Reworks)} = {}_{i=1}\sum^{n} SP'\_i}$$

Re-evaluated EV = Σ Revised Story Points of all completed tasks over sprints

3. **Change in EV:** This is calculated by subtracting the Initial EV from the Re-evaluated EV. A negative value would indicate a decrease in perceived value based on customer feedback.

$$\boxed{\textit{Change in EV = Re-evaluated EV - Initial EV (after UAT and release)}}$$

4. **Thus, the EV Ratio:**

EV Ratio = Re-evaluated EV / Initial EV

The EV-based Productivity Ratio serves as a lens focusing specifically on the impact of reworks in an agile setting. It helps teams understand how post-release modifications (in terms of reworks based on feedback or quality issues) affect the overall value delivered and can drive process improvements.

4. Blended Quality Metrics (BQM) Based on Code Maintainability

5.1 Excluding Line of Code (LoC) in CMI Computation

An important change has been made in the Code Maintainability Index (CMI) calculation, the CMI has been modified by excluding the Lines of Code (LoC) from the formula. This modification is aimed at achieving a more accurate representation of the Maintainability Index. When the Lines of Code (LoC) increase, it naturally leads to a rise in the Code Maintainability Index (CMI), even if other variables remain constant. The Code Maintainability Index (CMI) was conceived to incorporate multiple factors, among which the size of the codebase is significant. Yet, it's worth noting that Lines of Code (LoC) already influence both the Halstead Volume and Cyclomatic Complexity. Therefore, its separate inclusion in the CMI calculation seems to be redundant.

1. **Halstead Volume**: This takes into account the size and complexity of the code, as it considers the number of operators and operands (elements of the code). So, indirectly, it's related to the size of the codebase.

2. **Cyclomatic Complexity**: This measures the number of linearly independent paths through the code, which is related to decision points in the code and thereby indirectly to the size of the code.

Larger codebases can inherently be harder to maintain, all else being equal, due to factors like more places for bugs to hide, more effort needed for understanding the code, longer build times, etc. It's important to remember *that not all lines of code contribute equally to maintainability difficulties. Comments, whitespace, and lines containing only braces (in languages that use them) do not add to complexity or difficulty of understanding. Therefore, removing LoC from the CMI is considered as a reasonable adaptation.*

Thus, traditional CMI = 171 - 5.2 * log2(V) - 0.23 * (G) - 16.2 * log2(LOC) is modified as

CMI'=k*(171 - 5.2 * log2(V) - 0.23 * (G))

Here k is the constant value, and it is called a scaling factor to fit the value within the range of 0 to 100.

k=100/Max value of CMI (Excluding the LoC)

5.2 Defining Blended Quality Metric (BQM)

The BQM intends to assess the quality of code with an emphasis on maintainability. The maintainability of code has a direct impact on the volume of rework required. High-quality, maintainable code tends to require less rework and, therefore, less time and effort to modify or enhance to an individual developer and team. The BQM considers factors such as Code Maintainability Index, Defect Density and Test Coverage Metrics that provides a comprehensive measure of the quality of the code and, indirectly, the skill level and work quality of the development team and an Individual. The blending of Code Maintainability Index (CMI), Defect Density, and Test Coverage is in the ratio of 4:3:3, respectively. The resulting Blended Quality Metric (BQM) can be calculated using the following formula:

BQM = 0.4**CMI + 0.3****Defect Density + 0.3*Test Coverage**

Since these three metrics are combined and normalized to a scale between 0 and 1, the value 1 is considered as the highest possible quality.

5.2.1 Quality Prediction Criteria for BQM

The coefficient of correlation measures the linear relationship between two datasets. The value of the coefficient ranges between -1 and +1. In the context of this study, BQM is the quality metrics and the Defect Density is the number of defects, a coefficient nearing -1 would suggest that as the quality improves (BQM increases), the number of defects decreases, which is an expected relationship in most quality scenarios. This relatively strong negative correlation supports the idea that BQM is a good measure of quality as the quality metric value goes up, defects go down.

5. Evaluation Methodology

An empirical evaluation has been carried out to validate the effectiveness of these two proposed metrics. The assessment encompassed the scrutiny of 10 distinct software projects across two organizations. The evaluation underscored the improved accuracy these metrics brought to measuring both team and individual productivity, as well as the quality of the final product. Earned Value Analysis based Productivity Ratio and Blended Quality Metrics driven by Code Maintainability have demonstrated significant potential in enhancing project outcomes.

6.1 Computational Approach to EVAPR

In the initial phase of the study, the Earned Value (EV) for each user story across 12 sprints is measured. This provided a set of initial EV metrics, indicative of the perceived value at the end of each sprint. It's essential to note that these initial EV measurements did not account for any potential rework.

Following each sprint release, in accordance with agile principles, customer feedback is gathered for the completed sprint and the requirements are prioritized for the subsequent sprint. During this process, some user stories that have been previously released may require rework and, therefore, they are included in the upcoming sprint.

The reasons behind this necessary rework are analyzed and the Earned Value (EV) points of the respective user story from the previous sprint are appropriately adjusted. This adjustment ensures that the actual productivity of the preceding sprint is accurately assessed. The revision of EV story points predominantly arises from developmental issues, primarily due to misinterpretation of requirements and insufficient testing. The reasons for each instance of rework is tracked, associating it with the respective developers.

6.2. Computational Approach to BQM-CMI

In the quality aspect, the proposed Code Maintainability Index (CMI') was computed across 12 sprints for 10 projects, with Lines of Code being intentionally excluded. Alongside this, two other key metrics, namely Defect Density and Test Coverage, were quantified for the identical set of projects and sprints. BQM-CMI formula was applied. The resulting value, falling within a range from 0 to 1, serves as a measure of quality, with a score of 1 indicating the highest possible level of quality. A consistent pattern is identified that when rework was correctly tracked and addressed, individual productivity improved, leading to enhanced team productivity.

6. Case Study

7.1 Productivity Measurement

The study encompasses an empirical evaluation of ten software development projects, five each from two different companies, spanning a period of one year. The projects evaluated include a diverse array of products and systems. The projects were based on different technologies as shown in table 2. The development teams were of varied size. Our evaluation process comprised an analysis of 12 sprints from each project. Each sprint lasted 15 days, culminating in approximately 18 months of sprints for each project. Earned Value was gathered after the 12th sprint, marking the end of the 6th month for each project. The number of user stories varied depending on the team size for each project, providing a diverse dataset for our analysis.

Table 2: Overview of Projects for Productivity Measure

Project Name	Type	Technology	Team Size	Company
Project 1	Hospital Management System	Dot Net	3	B
Project 2	University/College management System	Python	6	A
Project 3	Church Management System	PHP	4	A
Project 4	School Ranking and Accreditation	Angular JS	3	B
Project 5	Yoga Class Booking and Teaching	React JS	4	B
Project 6	School Management System	DotNet	8	A
Project 7	HR Management ERP	Python	5	B

Project 8	Accounts Management System ERP	DotNet	2	B
Project 9	Online Class Solution	PHP	4	A
Project 10	MIS for Education Institutions	PHP	5	A

In the evaluation process, our initial step involves scrutinizing the Planned and Earned Story Points over a consecutive series of 12 sprints for each project. Table 3, illustrates the Project 1 Earned Values of 12 sprints.

Table 3: Earned Value for Project 1, 12 Sprints

Sprint	Planned Story Points	Earned Story Points	Sprint	Planned Story Points	Earned Story Points
1	92	81	**7**	228	170
2	90	81	**8**	172	130
3	138	114	**9**	330	244
4	126	100	**10**	282	218
5	276	200	**11**	122	88
6	216	148	**12**	128	104

Table 4 : Ripple Effect of a Sample 3 Sprints and User Story

User Story-ID	Sprint 1	Sprint 2	Sprint 3
MSB OPD Reg 05	12 SP	6 SP	
MSB OPD Reg 09		12 SP	7 SP
MSB OPD Reg 04		12 SP	5 SP
MSB OPD Reg 07		18 SP	12 SP

Table 4 illustrates the 'Ripple Effect' of selected User Stories which required revisiting and reworking in later sprints. The table comprehensively tracks the necessary 'rework' for every User Story over multiple sprints, portraying how changes in requirements, initial misunderstandings, or inadequate testing can lead to varied estimates across sprints.

7.1.1 Re-calculating EVAPR

The user story 'MSDOPD Reg 05' started in sprint-1 with 12 Story Points and was completed earning 10 points. However, due to requirement misinterpretation, it was revisited in sprint-2 with 6 points assigned.

Since this rework was a development issue, we adjust the Earned Value of sprint-1 for an accurate productivity assessment. The initial 10 points are reduced by the 6 points reassigned, leaving an actual earned value of 4 points. Rework story points are thus adjusted as per the cause. Table 5 displays the original and re-evaluated Story Points after rework adjustment for the 12 sprints.

Table 5 : Difference between Initial Earned Value and Re-Evaluated Earned Value

Sprint	Initial EV	Re-Evaluated EV	Sprint	Initial EV	Re-Evaluated EV
1	77	71	**7**	162	146
2	81	57	**8**	130	130
3	104	104	**9**	236	200
4	76	70	**10**	212	204
5	204	180	**11**	88	80
6	144	144	**12**	104	94
	Initial EV = 1618			**Re-evaluated EV = 1475**	

Table 6 displays the Initial and Re-Evaluated Earned Values.

Table 6 : 10 Projects: Initial and Re-Evaluated Earned Values

Projects	Initial Earned Value	Re-Evaluated Earned Value	Projects	Initial Earned Value	Re-Evaluated Earned Value
Project 1	1618	1475	**Project 6**	2245	1645
Project 2	1842	1455	**Project 7**	1608	1100
Project 3	1870	1270	**Project 8**	964	896
Project 4	2100	1500	**Project 9**	1050	600
Project 5	1060	910	**Project 10**	1457	958

The Re-Evaluated EV reflects the precise Earned Value for each sprint release, which is calculated by taking into account the rework assigned to the subsequent sprint

7.1.2 Re-Evaluating Individual Productivity: Incorporating Rework into EV

Understanding individual contributions in agile methodologies can identify challenges and foster teamwork, thereby reducing rework and enhancing overall productivity projection. Instances where a novice developer's productivity affects team output can spotlight opportunities for mentorship. The data below illustrates each individual's initial and adjusted Earned Value contributions, providing insight into this aspect.

Table 7 : Individual Productivity: Initial and Re-Evaluated Earned Values by Sprint of a Project

Person - A			Person - B			Person - C		
Sprint	Initial Earned Value	Re-Evaluated Earned Value	Sprint	Initial Earned Value	Re-Evaluated Earned Value	Sprint	Initial Earned Value	Re-Evaluated Earned Value
1	30	30	1	22	22	1	25	25
2	23	6	2	24	24	2	34	37
3	50	50	3	22	22	3	29	29
4	36	30	4	22	22	4	18	18
5	102	94	5	60	60	5	42	42
6	54	54	6	68	68	6	22	22
7	74	66	7	26	26	7	62	54
8	54	54	8	20	20	8	56	56
9	84	78	9	78	74	9	74	70
10	128	124	10	74	70	10	10	10
11	34	30	11	34	30	11	20	20
12	38		12	40		12	26	

Table 7 presents the productivity of three persons A, B and C considering EVAPR calculation for the individuals drawn from the respective sprint logs.

7.2 Quality Measurement

7.2.1 Computation of Blended Quality Metrics

The calculation of the Code Maintainability Index (CMI), excluding Lines of Code, was executed across 12 sprints for 10 distinct projects. Simultaneously, the metrics of Defect Density and Test Coverage were also quantified for the identical dataset of projects and sprints. All these operations were facilitated using the SonarQube tool.

As part of the continual evaluation process, the Weekly Assessment Report plays a pivotal role. It acts as an instrument for quantifying key metrics such as the number of defects, Defect Density, and Test Coverage for each sprint. These metrics provide a comprehensive understanding of the quality status of the project, helping teams to identify areas of improvement and initiate necessary corrective actions.

Table 8, presents a detailed examination of a sample sprint, meticulously analyzed and documented.

Table 8 : CMI’ of a Sprint - Project 5

CMI	HV	CC	LoC	Log2(LoC)	16.2 * Log2(LOC)	0.23* CC	Log2(HV)	5.2 *LOG2(HV)	MI	MI'	k*MI’ (k=100/max val of MI')
50	48	14	320	5	81	3.22	5.58	29.04	57.73	138.73	94.34
43	575	17	410	5.3575	86.79	3.795	9.16	47.66	32.74	119.54	81.28758096

49	118	8	440	5.45	88.44	1.725	6.88	35.80	45.02	133.46	90.75831489
55	29	2	400	5.32	86.21	0.46	4.83	25.13	59.18	145.40	98.87285457
74	233	18	390	5.28	85.62	4.09	7.86	40.87	40.40	126.02	85.69617704
55	45	3	340	5.08	82.41	0.575	5.50	28.62	59.38	141.79	96.42353877
48	49	3	390	5.28	85.62	0.69	5.62	29.26	55.42	141.04	95.91055049
47	53	4	420	5.39	87.35	0.805	5.71	29.71	53.12	140.48	95.52702785
53	34	3	270	4.75	77.02	0.575	5.09	26.47	66.91	143.94	97.88474939
58	33	3	160	4	64.8	0.69	5.05	26.28	79.22	144.02	97.93992613
67	290	20	640	6	97.2	4.6	8.18	42.53	26.66	123.86	84.22731376
46	53	4	420	5.39	87.35	0.805	5.71	29.71	53.12	140.48	95.52702785

k is the scalar factor that maintains the range of the CMI' value within 0 to 100 range

7.2.2 Defect Density and Test Coverage of Sprint

The following are the metrics related to Defect Density and Test Coverage for the given Sprint: This is calculated by using the Formula: Defect Density = No of Defects / Line of Code and Test Coverage = (Line of code covered by tests / Total Line of code) * 100

Table 9 : Defect Density & Test Coverage of Project 5 - Sprint 1

#	No of Defect	Defect Density	Lines of Code Executed by Tests	Test Coverage (%)	#	No of Defect	Defect Density	Lines of Code Executed by Tests	Test Coverage (%)
1	2	0.0062	320	87.5	**6**	1	0.0029	340	91.1
2	7	0.0170	410	90.2	**7**	2	0.0051	390	89.7
3	3	0.0068	440	90.9	**8**	1	0.0023	420	90.4
4	1	0.0025	400	90.0	**9**	1	0.0037	270	88.8
5	6	0.0153	390	89.7	**10**	2	0.0125	160	87.5

These metrics illuminate code quality within the sprint. Defect Density gauges the number of software defects, indicating potential areas needing more testing. Conversely, Test Coverage measures testing thoroughness, with high coverage suggesting better code quality.

7.2.3 BQM Computation in a 4:3:3 Ratio of Sprint

The presented data represents the computed Build Quality Metric (BQM) for various code projects. The BQM is calculated using a specific formula: BQM = 0.4*k*MI' + 0.3*Defect Density + 0.3*Test Coverage.

Table 10 : Computation of Blended Quality Metrics for Sprint 1 of Project 5

#	0.4*k*MI'	0.3*Defect Density	0.3*Test Coverage	BQM	#	0.4*k*MI'	0.3*Defect Density	0.3*Test Coverage	BQM
1	37.7367	0.001875	26.25	**63.98866404**	7	38.3642202	0.00153	26.91	**65.2757586**

2	32.51503238	0.005121	27.06	**59.58015433**	8	38.21081114	0.000714	27.12	**65.33152543**
3	36.30332595	0.002045	27.27	**63.57537141**	9	39.15389975	0.001111	26.64	**65.79501087**
4	39.54914183	0.00075	27.00	**66.54989183**	10	39.17597045	0.00375	26.25	**65.42972045**
5	34.27847081	0.00461	26.91	**61.1930862**	11	33.6909255	0.003281	26.7	**60.39420675**
6	38.56941551	0.000882	27.33	**65.90029786**	12	38.21081114	0.000714	27.12	**65.33152543**

Each project's quality is represented through four lines of information: MI' for code complexity, Defect Density for defects per unit size, Test Coverage for the extent of tested code, and the computed BQM as an overall quality measure is projected in the above table Table 10.

7. Results and Discussion

8.1 Earned Value based Productivity Analysis

In Figure 1, the line graph exhibits the Initial and Re-evaluated Earned Value Comparison. The difference between the two values for each sprint gives a visual representation of the impact of rework.

Rework Impact Across Sprints: For Sprints 2, 4, 5, 7, 9, 11, and 12, the re-evaluated EV is lower than the initial EV. This indicates that these sprints had some level of rework due to factors like misunderstanding of requirements or testing defects which led to a reduction in the actual earned value. The degree of the drop from the initial EV to re-evaluated EV gives an idea of the extent of rework in those sprints.

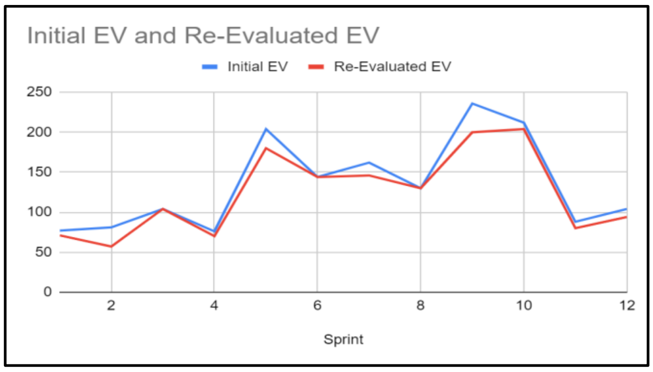

Fig 1 : Comparison of Initial EV and Re-Evaluated EV of Project - 1

Thus, figure 1 provides a clear visualization of the impact of rework on productivity across the 12 sprints. It highlights the importance of accurately understanding requirements and rigorous testing to minimize rework, thereby maximizing the earned value and productivity of each sprint. Tracking this information over time can help the team to identify patterns, implement improvements, and continually boost productivity.

8.1.1 Re-Evaluating the Earned Value of Projects

Trend Line and Project Comparison: Each project in figure 2 displays two trend lines showing initial and re-evaluated earned values, enabling clear comparison. A substantial decrease from initial to re-evaluated value suggests significant rework, likely due to misinterpreted requirements or development obstacles.

Variation across Projects: The distance between the initial and re-evaluated earned value lines in figure 2 signifies the extent of the rework or adjustments needed for each project. For example, the project -3 shows a considerable drop from 2100 to 1500, signaling a high degree of rework was necessary. Conversely, 'Project 10 reveals a smaller decrease from 1060 to 910, indicating that less rework was required.

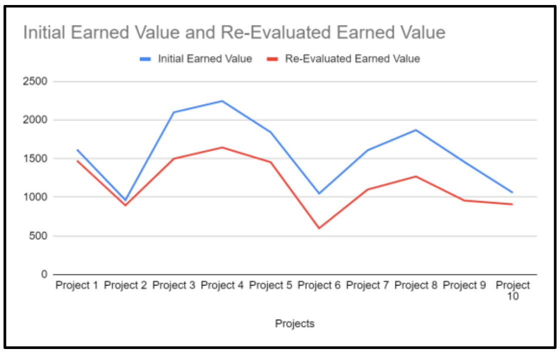

Fig 2: Comparison of Initial EV with Re-Evaluated EV

A general downward trend from the initial to re-evaluated earned value for most projects could suggest a systemic issue in estimation practices or project execution. Thus, the line figure 2 provides clear visual insights into the impact of rework on project performance. The ability to identify the extent of this impact enables teams to work towards mitigating similar issues in the future. This, in turn, contributes to improving productivity and reducing the risk of cost and time overruns.

8.1.2 Individual Productivity

Developer A Productivity Developer B Productivity Developer C Productivity

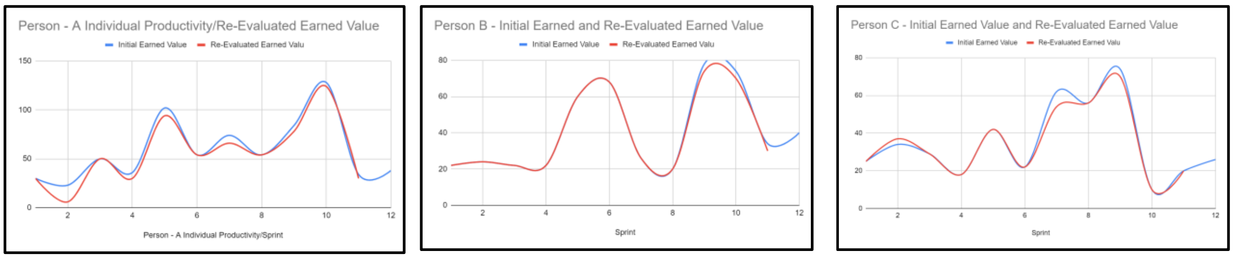

Fig 3: Comparison of Initial and Re-Evaluated EV for Individuals

Based on Figure 3, we have three developers namely A, B and C with their respective experience levels as 2 Yrs-7 Months, 5Yrs-8Months and 2Yrs-11months respectively.

Upon scrutinizing the productivity graph, it's evident that the re-evaluated earned value consistently trails the initial planned value. This suggests the project's progress consistently misses initial expectations, irrespective of the developers' experience levels.The similarity between Developer B's initial and reevaluated EV with little variance could signal the developer's experience

8.2 Efficacy of BQM-CMI

To validate the efficacy of BQM-CMI, BQM value was correlated with the number of defects found post-release across 12 sprints of a project (Project 5). The result was a high negative correlation coefficient of -0.98.

This high negative correlation indicates that as the BQM value increases (indicating higher quality), the number of post-release defects decreases. This correlation strongly supports the argument that BQM is a reliable and insightful quality indicator in the context of software development. It suggests that focusing on improving the BQM during the development phase could lead to a significant reduction in defects after release, thus enhancing the overall quality of the software product.

Table 11 : BQM-CMI for 10 Projects

Project	Coefficient of Correlation(r)	Project	Coefficient of Correlation(r)
Project 1	-0.7272931102	**Project 6**	-0.7106356486
Project 2	-0.9284964516	**Project 7**	-0.9775780372
Project 3	-0.8808605311	**Project 8**	-0.911622708

Project 4	-0.724413879	**Project 9**	-0.9713667718
Project 5	-0.9773438887	**Project 10**	-0.9727630083

Table 11, presents the correlation coefficients between BQM-CMI and the number of defects identified post-release for the ten projects. The negative values indicate a reverse relationship, where a high BQM (indicating good quality) is associated with a lower number of defects post-release, and vice versa.

8. Conclusion

This study underscores the value of Earned Value Analysis-based Productivity ratio in software development process assessment, especially regarding rework and individual productivity. It promotes process improvement, issue resolution, and enhances team productivity. Recognizing frequent rework in user stories and understanding the 'Ripple Effect' aids in realistic future sprint planning.

The research also underlines the positive impact of reduced rework on client satisfaction, facilitating faster delivery times, more predictable schedules, and products that align closely with client expectations.

This study also attests to the effectiveness of BQM-CMI as a comprehensive indicator of software and code quality, suggesting its use as a predictive tool for quality. The research paves the way for improving quality assurance in software development, reducing rework and promoting efficiency. Further studies could expand on BQM's applicability and explore other factors impacting software quality.

References

[1] Individual performance in the Scrum environment. (2017, November 7). Individual performance in Scrum environment. Online Forum Post.

[2] https://www.scrum.org/forum/scrum-forum/13381/individual-performance-scrum-enviornment

[3] Rubin, K. S.. (2012). Essential Scrum: A Practical Guide to the Most Popular Agile Process.

[4] Ahmed, A, Tayyab, M, Bhatti, S, Alzahrani, A, & Babar, M (2017). Impact of story point estimation on product using metrics in scrum development process. 8(4), 385–391.

[5] Hayes, W, Miller, S, Lapham, M. A., Wrubel, E, & Chick, T (2014). Agile Metrics: Progress Monitoring of Agile Contractors.

[6] Dixit, R, & Bhushan, B Scrum: An Agile Software Development Process and Metrics. 7(1), 73–87.

[7] Almeida, F, & Carneiro, P (2023). Perceived Importance of Metrics for Agile Scrum Environments. 14, 327. https://doi.org/10.3390/info14060327

[8] Almeida, F, & Carneiro, P (2021). Performance metrics in scrum software engineering companies. 14(327). https://doi.org/10.3390/info14060327

[9] Mahnic, V, & Zabkar, N (2012). Measuring progress of scrum-based software projects. 18(8), 73–76

[10] Ifra, I. and Bajwa, J. (2016) ‘Metrics of scrum methodology’, International Journal of Computer Applications, Vol. 149, No. 2, pp.24–27.

[11] Bitla, K. S, & Veesamsetty, S. S (2019). Measuring Process Flow using Metrics in Agile Software Development A Systematic Literature Review and a Case Study.

[12] Tripp, J., Riemenschneider, C. and Thatcher, J. (2016) ‘Job satisfaction in agile development teams: agile development as work redesign’, Journal of the Association for Information Systems, Vol. 17, No. 4, pp.267–307.

[13] Tanner, M. and Mackinnon, A. (2015) ‘Sources of interruptions experienced during a scrum sprint’, The Electronic Journal of Information Systems Evaluation, Vol. 18, No. 1, pp.3–18.

[14] Kupiainen, E, Mäntyla, M, Mika, V. and Itkonen, J. (2015) ‘Using metrics in agile and lean software development – A systematic literature review of industrial studies’, Information and Software Technology, Vol. 62, pp.143–163

[15] Eduard Budacu (2018) Real Time Agile Metrics for Measuring Team Performance. Informatica Economica

[16] Cabri, A, & Griffiths, M. (2006). Earned value and agile reporting. In AGILE 2006 (pp. 10-pp). IEEE. https://doi.org/10.1109/AGILE.2006.21

[17] Agarwal, M., & Majumdar, R.. (2012). Tracking scrum projects tools, metrics and myths about agile. 2(3), 97–104.

[18] Alegria, J., Bastarrica, M. C., & Bergel, A.. (2011). Is it safe to adopt the scrum process model?. 14(3), 1–11.