QUALITY PAPER

Optimizing defect rework for a software start-up’s schedule variation: a Six-Sigma-based approach

Satyajit Mahato and Supriyo Roy

Department of Management, Birla Institute of Technology, Ranchi, India

Abstract

Purpose – Managing project completion within the stipulated time is significant to all firms’ sustainability. Especially for software start-up firms, it is of utmost importance. For any schedule variation, these firms must spend 25 to 40 percent of the development cost reworking quality defects. Significantly, the existing literature does not support defect rework opportunities under quality aspects among Indian IT start-ups. The present study aims to fill this niche by proposing a unique mathematical model of the defect rework aligned with the Six Sigma quality approach.

Design/methodology/approach – An optimization model was formulated, comprising the two objectives: rework “time” and rework “cost.” A case study was developed in relevance, and for the model solution, we used MATLAB and an elitist, Nondominated Sorting Genetic Algorithm (NSGA-II).

Findings – The output of the proposed approach reduced the “time” by 31 percent at a minimum “cost.” The derived “Pareto Optimal” front can be used to estimate the “cost” for a pre-determined rework “time” and vice versa, thus adding value to the existing literature.

Research limitations/implications – This work has deployed a decision tree for defect prediction, but it is often criticized for overfitting. This is one of the limitations of this paper. Apart from this, comparing the predicted defect count with other prediction models hasn’t been attempted. NSGA-II has been applied to solve the optimization problem; however, the optimal results obtained have yet to be compared with other algorithms. Further study is envisaged.

Practical implications – The Pareto front provides an effective visual aid for managers to compare multiple strategies to decide the best possible rework “cost” and “time” for their projects. It is beneficial for cost-sensitive start-ups to estimate the rework “cost” and “time” to negotiate with their customers effectively.

Originality/value – This paper proposes a novel quality management framework under the Six Sigma approach, which integrates optimization of critical metrics. As part of this study, a unique mathematical model of the software defect rework process was developed (combined with the proposed framework) to obtain the optimal solution for the perennial problem of schedule slippage in the rework process of software development.

Keywords Integrated Six Sigma, Defect rework, Software startup, Cost and time, Mathematical model, NSGA-II, Multiobjective optimization

Paper type Research paper

1. Introduction

Today’s business is technology-enabled. In an increasingly technological world, software solutions provide support at several business levels, from operational transactions to strategic decisions. Success in software projects is fundamental to any organization (Kobus et al., 2018). Evaluating success can be a massive endeavor because there are many technical and behavioural variables to consider due to the complexity of projects (Ibrahimovic and Franke, 2017).

Rework in software projects is considered undesirable work triggered to correct problems or tune an application. The same is regarded as an ongoing problem in the software development process (Panwar et al., 2022). Many software firms need clarification about isolating the rework. Working to solve the prevailing situation is part of routine maintenance for them. The critical point is differentiating the rework (Pai et al., 2021). Rework in software development is the undesirable effort of redoing a process or activity that was incorrectly
implemented in the first instance or due to modifications in requirements from customers. Unfortunately for start-ups, small businesses, and even multi-million dollar firms, tightening costs and rising competition imply a desperate scramble to find opportunities to slash expenses (Sharma et al., 2020). In financial aspects, the rework, cost, and time are experienced as expense-related heads (Kundu et al., 2015).

The planning and execution of any software project are complex and involve many risk elements. The estimated budget spread over time is based on the scheduled time and costs during the execution of the project. It is subject to constant modifications resulting from time, cost, and organizational factors (Maglyas et al., 2017).

Financial and non-financial losses generated due to missing schedules due to defects rework have been a matter of concern in the software development industry. Fixing software defects after project completion is costly, so the defect rework phase is intrinsic to the software development life cycle (Palanisamy et al., 2015). However, an ineffective rework phase increases the schedule deviation of project delivery and the cost of rework. Therefore, most software companies spend considerable effort on the rework phase. Up to 40% of the development time is spent repairing the defects (Shishodia et al., 2018). This staggering amount of “cost” and “time” spent due to the schedule variation in the rework phase has a vast implication on the competitive potential of the start-ups (Czajkowski, 2017). Despite this, there is minimal research on process improvement solutions to reduce the schedule deviation of the rework phase.

Literature shows that Six Sigma studies have demonstrated ways to reduce the defects in the development and maintenance phases (Patel and Patel, 2021). But still, no specific work has been reported so far that deals with the problem of schedule variation in the rework phase. Due to the lack of a systematic approach to deal with schedule deviation in defect rework, a start-up’s financial and other impacts are even worse because they keep losing competitive potential compared to large enterprises (LEs) (Raja Sreedharan and Raju, 2016). Therefore, these small start-ups look for solutions to the rework problem, but at the same time, they are sensitive to the cost involved in implementing the solution (Mahato et al., 2016).

The rationale of the problem has initiated the industry outlook, which always demands better answers.

The problem thus identified attempted to make a unique contribution to the theory by providing a mathematical formulation of the defect rework phase. The proposed formulation has been used in a Six Sigma platform and implemented in a software start-up to solve the problem of high rework “time” at an optimum cost. An unsupervised decision tree model was developed in the “Analyze” phase under the Six Sigma approach to predict defects in software modules. The proposed mathematical model fed the predicted defects to deduce the rework problem into a multi-objective optimization problem between rework “costs” and “time”. NSGA II was applied through the “gamultiobj” solver of the MATLAB platform.

The remainder of the paper is under Section 2, which discusses the literature support, significant learning, gap(s), and objective(s). Research methods, model development, and application of Six Sigma under quality are discussed in Section 3. Section 4 is the discussion of the result and output. Managerial implications are discussed in Section 5. Lastly, Section 6 summarizes the work, novelty, and future directions.

2. Literature review
The ineffective defect rework planning is the primary source of schedule variation in software development projects. Most software development companies face the challenge of paying high costs during the defect rework phase because of defects’ unpredictability (Hashmi et al., 2022). Keeping human resources issues aside, on-time project delivery is also significant. The uncertainty in the scheduled delivery due to the ill-planned defect rework
phase leads to inaccurate estimation of “time” and “cost”, resulting in schedule variation and loss of competitive potential (Palanisamy et al., 2015). Literature shows that Six Sigma may work as a solver for this problem as a process improvement framework, especially in manufacturing, and its efficacy is relevant (Ahmed et al., 2020). We review the literature on defect rework, schedule variation, and the application of Six Sigma in software development enterprises.

Software development is an intricate process where many factors throughout the development cycle lead to some defects. Therefore, defect rework has become integral to software development (Pai et al., 2021). The changing requirements are one of the reasons for software defects; however, the dynamic nature of conditions is inherent in the software development business (DeFranco and Laplante, 2018). Therefore, the defect rework phase shall be treated as an integral part of software development, and adequate planning is required in rework management to mitigate schedule variation (Sekgweleo and Iyamu, 2022). The manufacturing and software industries have different approaches to dealing with defects. The manufacturing processes are repetitive and machine-driven (Shamsi and Alam, 2018).

In contrast, software development processes are highly intellectual. A significant amount of errors are due to the cognitive nature of software development. Hence, it is natural to have a defect rework phase as part of the project plan (Mahfuz, 2015).

Although many researchers have underscored the criticality of the defect rework phase, they still make terrible management decisions. An ill-managed rework process aggravates the “time” and “cost” of rework, which leads to schedule deviation and customer complaints (Papatheocharous et al., 2017). Excessive rework time and cost are prevalent problems in software projects’ rework phase. Six Sigma and its application in the software domain do not have sufficient research work to deal with this repetitive issue. Some researchers have highlighted similar views on the lack of comprehensive studies to address the defect rework “time” (John and Kadadevaramath, 2020). They reported that Six Sigma projects in the software development arena emphasized reducing defects during product development (Karout and Awasthi, 2017). High direct and indirect costs are incurred due to missing project delivery deadlines caused by a flawed defect rework process. The application of DMAIC in the software development process has so far taken a narrow view of reducing defects induced during different development phases (Lamine and Lakhal, 2018). However, the mismanagement of the rework phase itself has remained unanswered.

The current approach of Six Sigma studies in the software arena focuses more on eliminating the source of errors. This approach replicates manufacturing studies (Raja Sreedharan and Raju, 2016). A framework like DMAIC will be more effective in the software domain if applied with other analytical techniques (Purushothaman and Ahmad, 2022). Suitable customization is needed to make it effective in the software industry.

There are few research studies where Six Sigma has been customized as per the suitability of the software process, such as attempting to integrate simulation to predict defects early in development. Researchers suggested conjugating defect prediction models with DMAIC as it helps in planning the rework phase (Karout and Awasthi, 2017). The researcher presented a system that integrates it with Monte-Carlo simulation to assess the results of the continuous improvement activities with the help of historical data. Integrating Six Sigma with a flexible strategy framework in the software domain was proposed to benefit organizations and reduce errors and overhead costs (Sreeram and Thondiyath, 2015). Even recent studies on applying DMAIC in the software sector have a myopic view. These research studies are either on software quality or simply discussing Six Sigma’s integration issues with other process improvement methods (Raja Sreedharan and Raju, 2016). The perpetual problem of excessive rework “time” and the associated “cost” still needs to be answered.
The lack of a systematic approach has created a vast disparity between start-ups and established enterprises. Even the Six Sigma solutions to reduce rework “time” come at a “cost”, which is not affordable for small enterprises (Czajkowski, 2017). It is also reflected in the number of research papers since not even 2% were related to software start-ups (Pai et al., 2021). This low penetration of Six Sigma in software start-ups is due to the one-sided approach to defects without considering the trade-off needed between rework “time” and “cost”. Due to this lopsided approach, start-ups are reluctant to implement Six Sigma to improve their rework management.

The LEs spend massively on DMAIC projects without accounting for the “cost”. Start-ups, however, with meager resources, are sensitive to the cost involved in implementing any solution generated by the DMAIC framework (Czajkowski, 2017). The reason for this apprehension is the lopsided approach of executing Six Sigma without considering the “cost” factor. There are instances when the benefit of deploying a solution does not justify the cost (Shishodia et al., 2018). Start-ups cannot spend such an unbridled amount on implementing costly solutions (Shamsi and Alam, 2018). Therefore, they look for cost-optimized solutions from the DMAIC framework, particularly to address the high rework “time” (Mahato et al., 2017). A similar research gap in the application of Six Sigma in software start-ups has also been underscored by other authors. A review of the research papers of the last decade concluded that the focus is still on peripheral issues like the applicability of DMAIC in software processes and integration with ISO, CMMI, and other frameworks. The perennial problem of the chaotic defect rework phase remains there, hurting most software start-ups still looking for a solution that can take care of both “time” and “cost”.

2.1 Significant learning(s), gap(s), objective(s), and research statement

We conducted a comprehensive study on literature surveys using the keywords in Table A6 in Appendix 3. Significant learning from the literature review lacks studies on a comprehensive approach to address the problem of excessive “time” and “cost” during the defect rework phase of software development projects. Due to the inherent defects, it extracts a perennial research gap. The present work aims to address the gaps by implementing a comprehensive framework to reduce rework “time” at an “optimum cost”.

The following objective(s) have been framed accordingly.

(1) **Objective 1:** Identify analysis schema for the defect rework (quality) problem.

(2) **Objective 2:** Propose an integrated quality management framework under the Six Sigma quality approach framed into a multi-objective problem.

(3) **Objective 3:** Develop a mathematical model of the defect rework phase as a function of the software development parameters to be integrated with the proposed framework (Objective 2) to optimize the rework process at an optimum cost. Validation of the model through the proposed framework contributes to framing a benchmark in the theory of quality development.

(4) **Objective 4:** Validation of the mathematical model through a case study’s proposed quality management framework demonstrates its suitability and applicability.

(5) **Objective 5:** The multi-objective problem, as developed, gives a solution through NSGA-II to the objective functions: rework “time” and “cost”.

The case study was designed for an IT start-up engaged in software development. The company had been facing the persistent issue of excessive “time” during defects rework. Lastly, inferences have been highlighted in line with the quality perspective.
2.2 Research statement
We frame our research statements accordingly with the objective(s) as framed.
Defect rework in a software development process is resource-intensive. Literature suggests that even large enterprises (LEs) have faced schedule slippage during defect rework. However, start-ups are more adversely affected due to their limited developer resources. Hence, there is a need to solve this persistent problem of schedule slippage in the defect rework phase. Accordingly, we frame our research question:

RQ1. Is there any solution for optimized defect rework management (by effective deployment of developers) to complete the rework process on time with a minimum cost?

3. Research methodology
The study follows a step-by-step procedure: First, identify the analysis schema for the defect rework (quality) problem. Secondly, propose an integrated quality management framework under the Six Sigma quality approach framed into a multi-objective problem. In the next stage, we develop a mathematical model of the defect rework phase as a function of the software development parameters to be integrated with the proposed framework (step 2) to optimize the rework process at an optimum cost. Validation of the model contributes to framing a benchmark in the theory of quality development. Lastly, we solve the multi-objective problem, as developed (which gives a solution through NSGA-II to the objective functions: rework “time” and “cost”). Notably, the case study was designed for an IT start-up engaged in software development. Finally, inferences have been highlighted, aligning with the quality perspective.

The research methodology consists of the following steps: research design, mathematical model development, and case study development based on the proposed model. First, we frame a research design. For this, we draw a figure showing the study’s overall process and flow diagram in Appendix 2 (Figure A1). The following section describes the proposed integrated quality management framework. The abbreviations and notations used throughout the study have been summarized and shown in Appendix 1.

3.1 Integrated quality management framework (DMAIOC)
This paper proposes a novel framework for quality management depicted in Figure A2 (Appendix 2). The authors have developed the optimization view with the conventional DMAIC approach of Six Sigma in the context of the manufacturing process (Mahato et al., 2017). This paper enhances the previous work by building a formal framework to integrate the “Optimize” phase with the conventional phases of the Six Sigma quality approach, then validate the proposed framework in a software defect rework process. In short, the framework is denoted as DMAIOC, where “O” stands for Optimize. The conventional phases of Define (D), Measure (M), Analyze (A), and Improve (I) are useful for transforming the business issue of schedule slippage into a quantifiable problem of defect rework management. A quantitative defect prediction model is the expected outcome of the Improve (I) phase. The novelty of the framework is the “Optimize” phase, which is integrated with the conventional DMAIC framework through four sub-steps (O1-O2-O3-O4). In this case, the “Optimize” phase aims to optimize the key metrics, rework “time”, and rework “cost”. The first step of Optimize (O1) formulates the two stated objectives. The second step (O2) describes the limitations of the rework process into mathematical constraints. This is followed by the mathematical formulation of the optimization problem in (O3) and then the application of a suitable algorithm in (O4) to obtain the optimal solution, which is maintained and monitored in the Control (C) phase.
The proposed DMAIOC approach provides an end-to-end guiding framework to address the issue, from identifying and quantifying the problem to obtaining an optimal solution. However, to get an optimal solution for any Engineering process, the Optimization algorithm needs to define it precisely mathematically. Next is developing a mathematical model to formulate the defect rework management of a software development process.

3.2 Development of a mathematical model

This research has attempted to advance the theory of Software defect rework by developing a unique mathematical model to formulate the rework issue as a multi-objective problem between its two fundamental aspects, “cost” and “time”. The model was integrated with DMAIOC in the “Optimize” phase and applied to obtain an empirical equation and a Pareto-front of the two stated objectives: rework cost and rework time. The Pareto front was executed to make trade-off decisions between cost and time and devise a rework strategy for the ongoing projects. The derivation of the proposed model starts by expressing the total development effort, which includes the effort required in coding, peer review, and unit testing (Maglyas et al., 2017). Therefore, the measure of total development effort can be described as follows:

$$ E_D = E_C + E_{REVE} + E_T $$

Here, $E_D$ represents the total development effort and $E_C, E_{REVE},$ and $E_T$ represent the coding, review, and testing efforts, respectively. As the defect rework is an intrinsic part of software development, a fraction of development effort is kept for reworking the defects, i.e. $E_R$ (Pillai et al., 2012).

$$ E_R = \vartheta E_D $$

Therefore, the overall project effort is,

$$ E_P = (1 + \vartheta)E_D $$

Here, $E_P$ represents the total project effort, and $\vartheta$ represents the planned rework-effort percentage. At the same time, the organizations are preparing for a 10 percent rework effort, which currently ends at 40 percent (Tandon et al., 2022). All effort terms are measured in man-hours.

A software development project is divided and managed into multiple modules. Each module has an estimated line of codes. Based on efficiency, skill, and cost, the pool of developers is split into “n” quantiles (Shishodia et al., 2018). The average code development time per developer for a small pool of developers can be estimated as the ratio of coding time and the product of the number of developers and lines of code (Panwar et al., 2022).

$$ \bar{t}_c, \bar{t}_{rev} = \frac{\sum_j t_{\bar{c}}, \sum_j t_{\bar{rev}}}{p_j, \sum_j l_d} $$

Here, $(\bar{t}_c)_j$ represents the average code development time per line of code, $p_j$ stands for the number of developers and $L_d$ indicates lines of code developed by a developer of the $j^{th}$ quantile. In the same way,

$$ (\bar{t}_{rev})_j = \frac{\sum_j t_{\bar{rev}}}{p_j, \sum_j t_{\bar{rev}}} $$
Here, \((I_{rev})_j\) and \((I_t)_j\) represent the average code review and testing time per line of code, \(L_{rev}\) and \(L_t\) indicate the lines of code reviewed and tested developed by a developer of the \(j^{th}\) quantile. The effort spent in the software development phase is a product of the number of developers and the time spent (Li et al., 2016; Sekgwele and Iyamu, 2022).

Hence, the coding review and testing effort can be estimated as follows:

\[
EC = \sum_{j=1}^{p} p_j \left( \frac{\sum_{j=1}^{p} (I_{rev})_j (l_{rev})_j}{p_j} \right)
\]

\[
EREVE = \sum_{j=1}^{p} p_j \left( \frac{\sum_{j=1}^{p} (I_{REV})_j (l_{REV})_j}{p_j} \right)
\]

\[
ET = \sum_{j=1}^{p} p_j \left( \frac{\sum_{j=1}^{p} (I_t)_j (l_t)_j}{p_j} \right)
\]

Where \(p_j\) represents the number of developers planned from the \(j^{th}\) quantile of a software development project. The total effort is the sum of the phase-wise action (Papatheocharous et al., 2017). Hence, the expression for the development effort can be rewritten as,

\[
ED = \sum_{j=1}^{p} p_j \left\{ \sum_{j=1}^{p} (I_{rev})_j (l_{rev})_j + \sum_{j=1}^{p} (I_{REV})_j (l_{REV})_j + \sum_{j=1}^{p} (I_t)_j (l_t)_j \right\}
\]

The average rework time/defect for a developer of the \(j^{th}\) quantile is the ratio of total rework time and the product of the number of developers and actual defects resolved (Kula et al., 2013)

\[
(I_r)_j = \frac{\sum_{j=1}^{p} t_r}{\left( \frac{\sum_{j=1}^{p} d}{p_j} \right)}
\]

Here, \((I_r)_j\) represents the average defect rework time per defect, and \(d\) indicates the number of errors. Hence, the weighted defect rework time/defect for the pool of developers belonging from various skill quantiles is estimated as,

\[
\bar{T}_r = \frac{\sum_{j=1}^{k} (\xi_j \times (I_r)_j)}{\sum_{j=1}^{k} \xi_j}
\]

Here, \(\bar{T}_r\) indicates the weighted defect rework time/defect and \(\xi_j\) the number of developers picked from the development phase to the rework phase in the rework phase for the \(j^{th}\) quantile.

A project is managed by splitting it into multiple modules. The rework time for a module depends on the number of defects detected in the testing phase, the average defect resolution time per developer, and the number of developers assigned in the rework phase (Zhang et al., 2018). The estimated total rework time for a project is expressed as,
Substituting the expression of $T_R$ in equation (11).

\[
T_R = \frac{\sum_{i=1}^{m} d_i \cdot \left( \frac{T_R}{\xi_j} \right)}{\sum_{j=1}^{k} \xi_j}
\]

Equations (12) and (14) represent the expression for rework “time”, $T_R$, and rework “cost” $Z$, the two objectives for minimization. The total defects resolved by each group of developers shall be at least equal to the summation of the defect count from all modules (Sun et al., 2017). Therefore, the inequality relation exists as follows:

\[
\sum_{j=1}^{k} d_j \geq \sum_{i=1}^{m} d_i
\]
A project manager aims to reduce the “rework time” in the defect rework phase, so it has to be maintained below a predefined proportion of the development time (Ibrahimovic and Franke, 2017). Therefore, in line with equation (2), the condition imposed on rework “time” can be expressed as,

\[
T_R \leq \left\{ \sum_{j=1}^{k} (\bar{l}_j l_j) + \sum_{j=1}^{k} (\bar{l}_{rev} l_{REV})_j + \sum_{j=1}^{k} (\bar{l}_j l_j) \right\}
\]

(19)

Besides, the number of allocated developers from any category can only be a smaller subset of the total number of developers in that category:

\[
0 \leq \xi_j \leq p_j
\]

(20)

A software development project could have multiple modules employing different teams (Ghobadi and Mathiassen, 2020). These project teams constitute developers with varying levels of experience. Therefore, they have varying “costs” and the ability to resolve defects in a shorter “time”. The more experienced developers come at a higher cost but take less time to fix errors (Sun et al., 2017). A critical rework “time” was chosen to be 10% of the overall development time, based on acceptable industry practice (Zhang et al., 2018). Every software project has its composition of developers. Hence, a user needs to generate results using the general methodologies described here as required.

The following section illustrates the case study to solve the problem of high rework “time” at an optimum “cost” through the integrated DMAIOC Six Sigma approach (Figure A2). The mathematical model developed here has been used to obtain the optimum solution.

### 3.3 Development of a case study based on the proposed model

**ABC Limited** (name changed due to data security problem) has been a software development start-up in the market for the last two years. It is based in Bangalore, the IT city of India. It struggles to consistently pay its customers the penalty for not meeting the agreed delivery timelines. The “schedule variation” is caused by the unpredictable and long rework “time” in the defect rework phase. There is a need to find out an approach that can be implemented in any software development project to ensure that the rework “time” is reduced from 44% to 10% of the development “time”, and the new rework “time” shall be achieved at the minimum rework “cost”.

Six Sigma is a cutting-edge philosophy for breakthrough improvement. Different industrial sectors have successfully applied the Six Sigma methodology for performance excellence (Shamsi and Alam, 2018). The concept behind this work came from the service-providing industry when the firm’s management wanted to improve its overall performance. An integrated DMAIOC framework of Six Sigma, depicted in Figure A2 (Appendix 2), is implemented in this case study to meet this stated purpose. The conventional phases, Define-Measure-Analyze-Improve, helped to develop an unsupervised Decision-tree model to predict module-wise defects. The mathematical model developed is then integrated to express the defect rework process as a multi-objective problem for “cost” and “time”. The problem was solved by applying the Genetic Algorithm toolbox of MATLAB software.
In the “Define” stage, the Voice of Customers (VOC) was captured reviewing the feedback from customer and finance managers, which indicates that the, 

(1) The poorly managed defect rework phase leads to very high schedule variation (SV), which causes financial losses for the customers.

(2) ABC Limited pays a colossal penalty to its customers due to the excessive rework “time”, which leads to high schedule variation.

Schedule variation is defined as non-adherence to the planned delivery time and measured as a percentage of deviation from the standard delivery schedule. In the contracts agreed with the customers, the acceptable limit is

\[ \text{Schedule variation (SV)} \leq 10\% \] (21)

The company is liable to pay the penalty if the SV > 10%. The amount paid due to the delay in delivery is calculated as a linear proportion.

\[ \text{Penalty amount } \alpha (\text{Actual SV} \% - 10\%) \] (22)

We have collected the historical data for 737 software modules of the last two financial years (2019-21) to validate the problem. Historically, 63.7% of the projects have breached the SV limit given in equation (22). The average rework “time” % for the penalized projects is four times that of the non-penalized projects. The data support the VOC and views expressed in literature that the defect rework phase has the severest and unpredictable SV due to excessive “time”. The target is to reduce it from 44% to 10% of the overall development time at an optimum rework “cost”.

The “Define” phase was concluded with a supplier, input, process, output, and customer (SIPOC) diagram of the defect rework phase, as depicted in Table A1 of Appendix 3.

In the “Measure” phase, the sequence of steps followed during the defect rework was noted down to create a process flow diagram, depicted in Figure A3 of Appendix 2. The retention of the original developer makes a significant change in the defect rework process. The developer assignment process for defect rework is illustrated in Figure A3. The planning and execution of the defect rework phase depend on the various parameters of the software development process. Such variables are identified from the literature and enlisted in Table A2 of Appendix 3. Data was collected for these variables from the software modules.

In the “Analyze” phase, failure mode effect analysis (FMEA) is a risk assessment tool to identify and prioritize the process areas prone to failures by estimating the risk priority number (RPN). The RPN is evaluated on the failure’s severity, frequency, and early detectability. FMEA for the defect rework process, documented in the process flow map (Figure A3), was done and depicted in Table A3 of Appendix 3. The RPNs indicate that the first step, i.e. estimating the number of defects, is highly error-prone. This has a cascading effect on the subsequent stages related to effort planning and developer retention. If some original developers are not retained for the rework, a new developer is assigned to fix the defects. Identifying a new developer with an appropriate skill set is itself highly prone to failure. Even if a new developer is attached, understanding the requirements and design documents takes much time. Chances of the misinterpretation of test results are very high, and this leads to high rework “time”. Historical data validated the findings of the FMEA. For the last six months of defect data, the pattern of developer alignment indicated that new developers had handled 64% of defects.

A hypothesis test evaluated the effect of the original developer on the rework time. Defects were randomly selected from a defect database. The hypothesis was framed as follows: “New developers take equal rework time to fix faults as the original developers”. The test results in
Table A4 of Appendix 3 indicate that the original developer takes less time to rework the defect than a new developer.

A “Why-Why” analysis was done to determine the root cause of the high rework. The outcome of the “Why-Why” report is presented in Figure A4 of Appendix 2. The “Improve” phase took the cues from the conclusion of the “Why-Why” analysis from the “Analyze” phase and then improved the defect rework planning by developing a model and a method to predict the number of defects in a module. This study has implemented an “unsupervised decision-tree model” to indicate errors in the ongoing software modules, which is our novel contribution. Data from more than 14,000 modules were obtained on the dependent variable, i.e. defect (DC) and independent variables.

Figure A5 depicts the SPSS 25 output of a decision tree model. Through this tool, the defect count of a module can be predicted by locating its terminal node, which is decided by looking at the values of independent variables. The prediction model’s accuracy was validated by applying it to the unseen test data. The model’s accuracy for the training and test dataset was 93.30% and 89.60%, respectively. It confirms the accuracy of the prediction model. These predicted defects of an ongoing project were now used in solving the multi-objective problem.

3.3.1 Multi-objective problem formulation from the case study. The mathematical model proposed in Section 3.2 is now integrated with the DMAIOC (Figure A1) to obtain the multi-objective problem of the case study. The two stated objectives of this problem were reworked time and cost. The exact formulation of the problem of this case study, including the constraints, is described below: The software development project in this case study had four modules. The summation of the predicted defects was obtained by using the Decision-tree for predicting defects (Figure A4), which yields,

$$\sum_{i=1}^{m} d_i = 497$$
for $i = 1$ to $4$  \hspace{1cm} (23)

In this case study, as the researchers prescribed, the developers’ span is divided into four quantiles based on skill and cost (Sun et al., 2017; Ghobadi and Mathiassen, 2020). The empirical data was then used to obtain the average defect rework time for each quantile of developers by using equation (10),

$$\begin{align*}
\bar{T}_{rj} &= 17 \frac{Hrs}{\text{defect}} \\
\bar{T}_{rj+1} &= 20 \frac{Hrs}{\text{defect}} \\
\bar{T}_{rj+2} &= 29 \frac{Hrs}{\text{defect}} \\
\bar{T}_{rj+3} &= 41 \frac{Hrs}{\text{defect}}
\end{align*}$$
for $j = 1$  \hspace{1cm} (24)

The software project under this study’s scope had several developers allocated from each quantile during the development phase. The quantile-wise distribution of the developer is given below,
Putting the empirical data obtained from equations (23) and (24) into (12), the expression for the rework “time” for the problem under consideration becomes,

\[
Rework\, time' (T_R) = 497 \left\{ \frac{(17 \xi_i + 20 \xi_{i+1} + 29 \xi_{i+2} + 41 \xi_{i+3})}{\left( \sum_{i=1}^{4} \xi_i \right)^2} \right\}
\]

The cost of developers from each quantile was also taken from the organization’s database. The quantile-wise cost/hour in INR is given below.

\[
Z_j = \begin{cases} 
121 & \text{for } j = 1 \\
82 & \text{for } j = 1 + 1 \\
60 & \text{for } j = 1 + 2 \\
40 & \text{for } j = 1 + 3 \\
\end{cases}
\]

The expression of the rework “cost” for the specific problem of the case study was obtained by substituting equations (23) and (27) in equation (14),

\[
Rework\, Cost' (Z) = 497 \left\{ \frac{(121 \xi_i + 82 \xi_{i+1} + 60 \xi_{i+2} + 40 \xi_{i+3})(17 \xi_i + 20 \xi_{i+1} + 29 \xi_{i+2} + 41 \xi_{i+3})}{\left( \sum_{i=1}^{4} \xi_i \right)^2} \right\}
\]

The non-linear constraints can be expressed for the problem under consideration by substituting equation (10) in (18)

\[
(-1) \cdot \left( \frac{\xi_j}{17} + \frac{\xi_{j+1}}{20} + \frac{\xi_{j+2}}{29} + \frac{\xi_{j+3}}{41} \right) \left( \frac{(17 \xi_j + 20 \xi_{j+1} + 29 \xi_{j+2} + 41 \xi_{j+3})}{\left( \sum_{i=1}^{4} \xi_i \right)^2} \right) - 1
\]

It is desired to restrict the rework “time” to 10% of the total development time. Another constraint is obtained by putting the empirical values calculated from equations (4–6) in (19),

\[
497 \left\{ \frac{(17 \xi_j + 20 \xi_{j+1} + 29 \xi_{j+2} + 41 \xi_{j+3})}{\left( \sum_{i=1}^{4} \xi_i \right)^2} - \frac{18967}{100} \right\} \leq 0
\]

The bounds for the stated problem are defined by putting the quantile-wise number of developers during the development phase (equation (25)) into equation (20),
\[ 0 \leq \xi_j \leq 14 \\
0 \leq \xi_{j+1} \leq 19 \\
0 \leq \xi_{j+2} \leq 35 \\
0 \leq \xi_{j+3} \leq 75 \\
\text{For } j = 1 \] 

(31)

The subsequent section describes the application of GA to solve the problem formulated in this section.

### 3.4 Genetic Algorithm to solve the framed multi-objective problem

This study’s algorithm applied for optimization is a controlled, elitist, non-dominated sorting algorithm, a variant of NSGA-II (Katoch et al., 2021). An elitist GA always favors individuals with better fitness values (Goldberg, 1989; Roy et al., 2005). In contrast, a controlled elitist GA supports individuals who can help increase the diversity of the population even if they don’t have better fitness (Arasteh et al., 2021). The best values of the computational parameters used in the algorithm for solving the problem have been obtained after preliminary numerical experimentation (Afrouzy et al., 2016). The impact of variations in the GA parameters, viz., the number of generations, the population size (Np), the crossover probability (P\text{cross}), and the mutation probability (P\text{mute}), was studied as per the guidelines provided by (Deb et al., 2002). Each parameter was varied once a time, keeping all others fixed at their reference values in Table A5 of Appendix 3.

The “gamultiobj” solver creates an initial default population using a uniform random number generator in every trial. Since the current problem has only four variables, the tests started with the default population size of 50 and then increased to 300. It is essential to maintain the diversity of the population for convergence to an optimal Pareto front (Arasteh et al., 2021). The variety is achieved by restricting the elite members of the population as the algorithm progresses. Two options, “Pareto Fraction” and “Distance Fcn”, were applied to manage the elitism. The Pareto fraction option controls the number of elite members on the Pareto front, and the distance function helps maintain diversity by favoring relatively far away individuals (Deb et al., 2002). The non-dominated and optimal solutions for a few trials for brevity are shown in Figure A6 of Appendix 2. Uniform mutation function, Single point crossover, and tournament selection method size of 2 were adopted.

In every trial, 18 solutions were obtained. The first trial converged in 800 iterations and had a smaller average distance (Figure A6). The population size increased to 200. At the same time, P\text{cross} and P\text{mute} were increased to 0.80 and 0.45, respectively. The average distance and spread measure were identical, but there was a discontinuity in the Pareto optimal front in the second, third, and fourth trials. The discontinuity diminished in the fifth instance as P\text{mute} was increased to 0.04. However, an increase in the scatter is visible at the same time. The final (6th) iteration (Figure A6) provides the best possible combination of distance and spread measures. The Pareto front of Figure A6 provided eighteen non-dominated solutions. In this case study, the selected solution has a rework time” of 163.5 h or 27 days at a “Cost” of INR 8,67,965.00. The chosen solution was proposed for six working hours per day following the developer allocation strategy in the rework phase for the First quantile: 1 developer for 1.2 h per day. In the second quantile, 4 developers were allocated for the entire day, but the 5th developer was only for 2.4 h per day. For the third quantile, 33 developers were earmarked for the day, but the 34th developer was only for 3.6 h per day. In the fourth quantile, 73 developers were allocated for the whole day.

The proposed solution completed the rework phase in 27 days. In the “Control” phase, control charts were used to monitor the success rate for the completion of the rework phase and the rework “time” of the projects being monitored.
4. Results and discussion

As per the first research objective, an analysis schema for the defect quality rework problem is created, represented in the research design (Figure A1). The research design outlines the multi-objective nature of the rework management issue. Subsequently, according to the second research objective, this paper proposed a novel DMAIOC framework (Figure A2) under the Six-sigma quality approach to integrating the “Optimize” phase, thereby solving the perennial problem of high rework “time” at an optimum cost. The “Optimize (O)” phase in the proposed DMAIOC framework provides the steps from the implementation standpoint, a novel contribution to the quality management theory in software development. Any optimization algorithm needs a mathematical description of the engineering problem under consideration. Hence, following the third research objective, a mathematical model for the software defect rework process is developed in Section 3.2.

A case study on implementing the proposed framework is done in Section 3.3 to meet the fourth research objective. The conventional “Define”, “Measure”, “Analyze”, and “Improve” transformed the issue of schedule slippage into a quantified rework management problem. The “Improve” phase created a Decision-tree model to predict the number of defects in software modules. The forecast defects were later used in the “Optimize” phase of DMAIOC with the mathematical formulation to deduce the rework process as a multi-objective optimization problem between rework (1) “time” and (2) “cost”. The “gamultiobj” solver of MATLAB applied NSGA II to solve the deduced problem following the fifth research objective.

A set of non-dominated optimal solutions was obtained with a Pareto front. The Pareto show revealed essential insights into the relationship between the time and cost of the defect rework phase. While any software project aims to reduce the rework “time”, there is a considerable increase in rework “cost” and vice-versa. However, the non-linear nature of the relationship suggests that the strategy to retain developers cannot be uniform.

The findings of this study indicate that the “Cost” to improve per unit time is less when the estimated rework time is close to 250 h. These are generally ill-managed projects. Significant improvement can be achieved at a small cost. The price to improve per unit rework time increases as the estimated time reduces. There is a stiff increase in “Cost” if the time is reduced to a value lesser than 140 Hrs.

These results are significant managerial implications in defect rework planning regarding managing the cost and schedule of software development projects. The case study aimed to restrict the rework “time” to 10% of the overall development time. The selected solution from the Pareto optimal front resulted in a rework “cost” of INR 8,67,963.00 to achieve the stated target. Implementation of the solution decreased the rework “time” to 27 working days.

5. Managerial implications

5.1 Theoretical as well as academic contributions

This research has attempted to fill this niche by (1) proposing an integrated quality management framework, DMAIOC and (2) a novel mathematical model for the defect rework process to address the high rework “time” at an optimum cost. The mathematical model describes the defect rework process as a function of the software development parameters. The proposed model formulated a multi-objective problem between the two objectives, reworking “time” and “cost”. This formulation is integrated with the DMAIOC framework’s “Optimize” phase and then implemented in a real-time case study conducted in a software development start-up. This approach solves the multi-objective problem by applying NSGA-II in MATLAB. A Pareto optimal front between the two objectives, “cost” and “time”, was obtained. The set of non-dominated solutions is a valuable aid for rework planning as it can be applied to determine the rework “cost” for a pre-determined “time” and vice-versa. The
Pareto front visually represents the increase in “cost” for the desired decrease in “time”; therefore, it has significant theoretical implications in understanding the interaction between the two in rework management.

5.2 Managerial contributions
To contain the schedule slippage, a project manager seeks to reduce the “time” of defect rework; it happens at the “cost” of adding more developers, but reducing “cost” is also critical. Implementing the DMAIOC framework (Figure A2) led to the development of a defect prediction model. Prior knowledge of the number of defects the decision tree provides enables estimating the effort required (man-hr). The Pareto front obtained through NSGA-II is a valuable aid for the managers to negotiate between the “cost” and “time” of the rework phase with their customers.

5.3 Policy formulations
The contractual agreements between the managers and their customers for software development projects have two key aspects: cost and time. Managing the cost and time in the defect rework phase is tricky because of the complex relationship between the two; hence, it is difficult to formulate a rework strategy. This paper proposes a framework (Figure A2) implemented in a case study to obtain a set of non-dominated optimal solutions from a Pareto front, which is used to estimate the “cost” for a contractually agreed “time” of the rework. The estimated “time” formed the basis of deploying the number of developers in the rework phase. The software development start-ups can implement the proposed DMAIOC framework and the novel mathematical model to obtain a Pareto front, which can be used to calibrate the “cost” for a contractually agreed “time” of the rework. On the other hand, if the customers are more focused on the economic aspects of the project, then the start-up managers can estimate and negotiate the “time” required to complete the rework phase. Either way, it enables the managers to better negotiate for effective defect rework management.

6. Concluding remarks
This study aims to reduce the “time” of the rework phase at an optimum “cost”. A unique scientific contribution to the theory of operational excellence in the software domain has been attempted by developing a mathematical model of the defect rework process in Section 3.2. The proposed mathematical formulation deduced the perennial issue of high rework “time” and “cost” into a multi-objective optimization problem. Software start-ups specifically ask for cost optimization while looking for solutions to their operational issues. The mathematical model is integrated with the DMAIOC framework under the Six Sigma approach, another novel contribution to the quality management theory in the software development domain. It is then implemented in a real-time case study. The conventional stages of DMAIOC analyzed the rework problem to make it quantifiable. By the end of the “Improve” phase, a Decision-tree model was created to predict the defect counts. The mathematical model with the expected defect count was then applied in the “Optimize” phase of the DMAIOC to obtain the multi-objective formulation for the case study’s problem. The formulated problem was later solved with MATLAB’s “gamultiobj” solver. The proposed solution reduced the rework “time” in the case study by 31%.

This work has deployed a Decision tree for defect prediction, but it is often criticized for overfitting. This is one of the limitations of this paper. Apart from this, comparing the predicted defect count from other prediction models hasn’t been attempted. NSGA-II has been applied to solve the optimization problem; however, the optimal results obtained have yet to be compared with other algorithms. Further study is envisaged.
Future work can include more sophisticated prediction models like Random Forest to address the overfitting. Other Heuristic algorithms capable of solving non-linear optimization problems, such as Ant colony, shall be used to verify the optimal solutions. This study has taken equal weightage for all defects for the rework effort estimation. In future studies, the defects can be categorized according to their varying complexity to resolve while estimating their work effort.

References


Appendix 1
Abbreviations and notations:

- $E_D$: Total development effort in man-hours
- $E_C$: Total coding effort in man-hours
- $E_{REV}$: Total code review effort in man-hours
- $\theta$: Ratio of rework time and development time
- $E_p$: Total project effort in man-hours
- $n_j$: No. of developers from the $j^{th}$ quantile in empirical data
- $(\bar{t}_c)_j$: Average coding time of a $j^{th}$ quantile developer
- $(\bar{t}_{rev})_j$: Average code review time of a $j^{th}$ quantile developer
- $(\bar{t}_t)_j$: Average testing time for the code developed by the $j^{th}$ quantile developer
- $L_d$: Lines of code developed from the empirical data
- $L_{rev}$: Lines of code reviewed from the empirical data
- $L_t$: Lines of code tested from the empirical data
- $p_j$: No. of developers from the $j^{th}$ quantile in the development phase
- $(l_c)_j$: Lines of code developed by $j$th quantile developers
- $(l_{REV})_j$: Lines of code reviewed by $j$th quantile developers
- $(l_t)_j$: Lines of code tested, which $j$th quantile developers developed
- $m$: Number of modules
- $d_i$: Number of defects in the $i^{th}$ modules
- $\xi_j$: Number of developers allocated in the rework phase by $j$th quantile developers
- $T_R$: Total rework time
- $C_j$: Cost/Hr for the $j$th quantile developers
- $Z$: Total Rework Cost
- $k$: Total number of quantiles
- $SV$: Schedule variation

Source(s): Authors’ work
Figure A1.
Research design with the flow diagram

Source(s): Authors’ creation
Converted Business problem into a Software defect rework problem

Variable identification and data collection for the rework process

Prepare control plan, Monitor by control chart

Build a defect prediction model

Causal analysis and FMEA to validate identified variables

Obtain optimal solution using a suitable Optimization algorithm

Apply mathematical model to derive rework cost and time in terms of rework parameters

Formulate a multi-objective problem with Rework cost and Rework time as its two objectives

Formulate the limitations of rework process as constraints

Source(s): Authors’ creation

Figure A2. Integrated DMAIOC framework

Schedule variation in a software start-up
Figure A3. Process flow map of the defect rework phase

Source(s): Authors’ creation
High ‘rework time’ and ‘rework cost’ in the defect rework phase

Longer time taken by developers to understand the requirement, design and code of the module being reworked

(1). Only 10%~15% of the developers of the ‘Coding’ phase are retained in the ‘Defect rework’ phase due to cost constrains
(2). Developers working during rework are not associated with the module, during its development phases

The decision to retain developers in the rework phase is-
(3). Not based on the susceptibility of the defective modules
(4). Skill of developers not considered during retention
(5). Cost factor not considered during retention

The Existing ‘Defect rework’ planning does not have -
1. A mechanism to predict the number of defects in modules
2. A method to retain developers in a way that minimizes the rework time and cost of rework

Source(s): Authors’ creation
Figure A5.
Decision tree model for defect prediction

Source(s): Authors’ compilation
Figure A6. Nondominated solutions for varying combinations of GA parameters (output)
### Appendix 3

<table>
<thead>
<tr>
<th>Supplier</th>
<th>Input</th>
<th>Process</th>
<th>Output</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Team of code developers</td>
<td>Completed modules</td>
<td>Unit testing of delivered</td>
<td>Reworked module</td>
<td>Project manager</td>
</tr>
<tr>
<td>Project manager</td>
<td>Defined test cases</td>
<td>completed modules</td>
<td></td>
<td>Integration team</td>
</tr>
<tr>
<td>Team of business analysts</td>
<td>Rework schedule</td>
<td>Defects listing</td>
<td></td>
<td>Client</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Assigning developers to defects</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Defect rework</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Source(s):** Authors’ creation

### Table A1.
Supplier, input, process, output and customer (SIPOC) of defect rework phase

<table>
<thead>
<tr>
<th>Description</th>
<th>Variable</th>
<th>Unit of measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defects</td>
<td>DC</td>
<td>Number</td>
</tr>
<tr>
<td>Number of modules</td>
<td>m</td>
<td>Number</td>
</tr>
<tr>
<td>Programming experience</td>
<td>PE</td>
<td>Months</td>
</tr>
<tr>
<td>Kilo lines of code</td>
<td>KLOC</td>
<td>Number</td>
</tr>
<tr>
<td>Requirement stability index</td>
<td>RSI</td>
<td>Percentage</td>
</tr>
<tr>
<td>Requirement effort</td>
<td>RE</td>
<td>Man-Hr</td>
</tr>
<tr>
<td>Design effort</td>
<td>DE</td>
<td>Man-Hr</td>
</tr>
<tr>
<td>Configuration management effort</td>
<td>CM</td>
<td>Man-Hr</td>
</tr>
<tr>
<td>Knowledge and training effort</td>
<td>KTE</td>
<td>Man-Hr</td>
</tr>
<tr>
<td>Programming language</td>
<td>Lang</td>
<td>–</td>
</tr>
<tr>
<td>Requirement development method</td>
<td>RM</td>
<td>–</td>
</tr>
<tr>
<td>Engineering tool adaption level</td>
<td>ETAL</td>
<td>–</td>
</tr>
<tr>
<td>Program complexity index</td>
<td>PCI</td>
<td>–</td>
</tr>
<tr>
<td>Requirement review method</td>
<td>RM</td>
<td>–</td>
</tr>
<tr>
<td>Design review method</td>
<td>DM</td>
<td>–</td>
</tr>
<tr>
<td>Code review method</td>
<td>CM</td>
<td>–</td>
</tr>
</tbody>
</table>

**Source(s):** Authors’ creation

### Table A2.
List of variables in the software development process

<table>
<thead>
<tr>
<th>Description</th>
<th>Variable</th>
<th>Unit of measurement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Defects</td>
<td>DC</td>
<td>Number</td>
</tr>
<tr>
<td>Number of modules</td>
<td>m</td>
<td>Number</td>
</tr>
<tr>
<td>Programming experience</td>
<td>PE</td>
<td>Months</td>
</tr>
<tr>
<td>Kilo lines of code</td>
<td>KLOC</td>
<td>Number</td>
</tr>
<tr>
<td>Requirement stability index</td>
<td>RSI</td>
<td>Percentage</td>
</tr>
<tr>
<td>Requirement effort</td>
<td>RE</td>
<td>Man-Hr</td>
</tr>
<tr>
<td>Design effort</td>
<td>DE</td>
<td>Man-Hr</td>
</tr>
<tr>
<td>Configuration management effort</td>
<td>CM</td>
<td>Man-Hr</td>
</tr>
<tr>
<td>Knowledge and training effort</td>
<td>KTE</td>
<td>Man-Hr</td>
</tr>
<tr>
<td>Programming language</td>
<td>Lang</td>
<td>–</td>
</tr>
<tr>
<td>Requirement development method</td>
<td>RM</td>
<td>–</td>
</tr>
<tr>
<td>Engineering tool adaption level</td>
<td>ETAL</td>
<td>–</td>
</tr>
<tr>
<td>Program complexity index</td>
<td>PCI</td>
<td>–</td>
</tr>
<tr>
<td>Requirement review method</td>
<td>RM</td>
<td>–</td>
</tr>
<tr>
<td>Design review method</td>
<td>DM</td>
<td>–</td>
</tr>
<tr>
<td>Code review method</td>
<td>CM</td>
<td>–</td>
</tr>
</tbody>
</table>

**Source(s):** Authors’ creation
<table>
<thead>
<tr>
<th>Process step</th>
<th>Description</th>
<th>Responsibility</th>
<th>Failure mode</th>
<th>Failure effect</th>
<th>Severity</th>
<th>Occurrence</th>
<th>Detection</th>
<th>RPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Estimate the number of defects</td>
<td>Project manager</td>
<td>Lesser defects estimated compared to actual defects</td>
<td>Inadequate effort and resources are allocated in the rework phase. Chances of high SV and penalty</td>
<td>9</td>
<td>7</td>
<td>9</td>
<td>567</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>More defects estimated compared to actual defects</td>
<td>More resources are allocated, leading to the misallocation of resources</td>
<td>4</td>
<td>4</td>
<td>9</td>
<td>144</td>
</tr>
<tr>
<td>2</td>
<td>Estimate defect rework effort</td>
<td>Project manager</td>
<td>Incorrect man-hr planning to fix all defects</td>
<td>High SV in the defect rework phase</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>315</td>
</tr>
<tr>
<td>3</td>
<td>Decide the developers to be retained for rework</td>
<td>Project manager</td>
<td>Inadequate resources to rework all defects on time</td>
<td>Delay in analyzing and fixing defects</td>
<td>7</td>
<td>7</td>
<td>5</td>
<td>245</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Developers of defective modules not retained</td>
<td></td>
<td>9</td>
<td>6</td>
<td>9</td>
<td>486</td>
</tr>
<tr>
<td>4</td>
<td>Release other developers</td>
<td>Project manager</td>
<td>Did not release another developer on time</td>
<td>Resource misutilization</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>Receive the defect list from the testing team</td>
<td>Project manager</td>
<td>Failed to receive complete defect report</td>
<td>Delay in starting the defect rework</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>Identify the module of defect</td>
<td>Technical manager</td>
<td>Unable to identify the module of the defect</td>
<td>High SV</td>
<td>7</td>
<td>1</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>7</td>
<td>The original developer is retained?</td>
<td>Project manager</td>
<td>Failed to identify the original developer of the module</td>
<td>Unnecessarily, a new developer deployed. Resource misutilization</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>Developer assigned to rework</td>
<td>Project manager</td>
<td>Original developer could not be assigned</td>
<td>Unnecessarily, a new developer deployed. Resource misutilization</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>The developer reviews the test result</td>
<td>Original developer</td>
<td>Misinterpreted the test results</td>
<td>Delay in finding a solution for the defect</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>10</td>
<td>Rework the defect and validate</td>
<td>Original developer</td>
<td>Unable to fix the defect</td>
<td>Requirement for a new developer. Cost overshots</td>
<td>8</td>
<td>1</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>11</td>
<td>Identify a new developer</td>
<td>Project manager</td>
<td>Delay in finding a developer with the proper skill set</td>
<td>Delay in starting the defect rework. High SV</td>
<td>9</td>
<td>9</td>
<td>7</td>
<td>567</td>
</tr>
</tbody>
</table>

(continued)
<table>
<thead>
<tr>
<th>Process step</th>
<th>Description</th>
<th>Responsibility</th>
<th>Failure mode</th>
<th>Failure effect</th>
<th>Severity</th>
<th>Occurrence</th>
<th>Detection</th>
<th>RPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>Developer assigned to rework</td>
<td>New developer</td>
<td>The new developer could not begin work</td>
<td>Delay in starting the defect rework.</td>
<td>9</td>
<td>3</td>
<td>1</td>
<td>27</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>High SV</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>Review requirements</td>
<td>New developer</td>
<td>Delay in understanding requirements</td>
<td>Delay in finding a solution for the defect</td>
<td>9</td>
<td>7</td>
<td>9</td>
<td>567</td>
</tr>
<tr>
<td>14</td>
<td>Review design documents</td>
<td>New developer</td>
<td>Delayed and misinterpreted design documents</td>
<td>Delay in finding a solution for the defect</td>
<td>9</td>
<td>7</td>
<td>9</td>
<td>567</td>
</tr>
<tr>
<td>15</td>
<td>Review test results</td>
<td>New developer</td>
<td>Misinterpreted the test results</td>
<td>Delay in finding a solution for the defect</td>
<td>9</td>
<td>5</td>
<td>9</td>
<td>405</td>
</tr>
<tr>
<td>16</td>
<td>Rework the defect and validate</td>
<td>New developer</td>
<td>Unable to fix the defect</td>
<td>Requirement for a new developer. Cost overshoots</td>
<td>9</td>
<td>5</td>
<td>5</td>
<td>225</td>
</tr>
</tbody>
</table>

**Source(s):** Authors’ compilation
### Table A4.
Hypothesis test to evaluate effect of developer selection on rework time

<table>
<thead>
<tr>
<th></th>
<th>N</th>
<th>Mean</th>
<th>StDev</th>
<th>SE Mean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rework Time – Original Dev</td>
<td>60</td>
<td>18.68</td>
<td>5.31</td>
<td>0.68</td>
</tr>
<tr>
<td>Rework Time – New Dev</td>
<td>60</td>
<td>60.39</td>
<td>6.71</td>
<td>0.87</td>
</tr>
</tbody>
</table>

Difference = μ (Original Dev) – μ (New Dev)

Estimate for difference: $-41.7141$

95% upper bound for difference: $-39.8832$

t-test of difference = 0 (vs <); t-value = $-37.77$ p-value = 0.000 DF = 118

**Source(s):** Authors’ compilation

### Table A5.
Failure mode effect analysis of the defect rework process

<table>
<thead>
<tr>
<th>Population size</th>
<th>P_c</th>
<th>P_m</th>
<th>Pareto Fraction</th>
<th>No. of optimal solutions</th>
<th>Generations</th>
<th>Average distance</th>
<th>Spread</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.65</td>
<td>0.01</td>
<td>0.30</td>
<td>18</td>
<td>800</td>
<td>0.0133193</td>
<td>0.141171</td>
</tr>
<tr>
<td>100</td>
<td>0.70</td>
<td>0.01</td>
<td>0.35</td>
<td>18</td>
<td>231</td>
<td>0.0162068</td>
<td>0.114833</td>
</tr>
<tr>
<td>150</td>
<td>0.75</td>
<td>0.02</td>
<td>0.40</td>
<td>18</td>
<td>231</td>
<td>0.0162068</td>
<td>0.114833</td>
</tr>
<tr>
<td>200</td>
<td>0.80</td>
<td>0.03</td>
<td>0.40</td>
<td>18</td>
<td>231</td>
<td>0.0162068</td>
<td>0.114833</td>
</tr>
<tr>
<td>200</td>
<td>0.80</td>
<td>0.04</td>
<td>0.45</td>
<td>18</td>
<td>231</td>
<td>0.0162068</td>
<td>0.114833</td>
</tr>
<tr>
<td>300</td>
<td>0.80</td>
<td>0.05</td>
<td>0.50</td>
<td>18</td>
<td>249</td>
<td>0.0104878</td>
<td>0.096018</td>
</tr>
</tbody>
</table>

**Source(s):** Authors’ compilation

### Table A6.
Comprehensive literature support

<table>
<thead>
<tr>
<th>SL No.</th>
<th>Topics</th>
<th>Definition</th>
<th>Literature support</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Schedule variation</td>
<td>Schedule variance calculates actual progress against expected progress Indicates whether the performance (the authorized work performed) exceeds, falls below or is equal to the planned performance</td>
<td>Maglyas et al. (2017), Shisodia et al. (2018)</td>
</tr>
<tr>
<td>2</td>
<td>Defect rework</td>
<td>Rework in software projects is considered undesirable work triggered to correct problems or tune an application</td>
<td>Kula et al. (2013), Subrahmaniam et al. (2015), Mahato et al. (2023), Sharma et al. (2020)</td>
</tr>
<tr>
<td>4</td>
<td>Genetic algorithm</td>
<td>A metaheuristic inspired by the process of natural selection. An adaptive heuristic search algorithm that belongs to the more significant part of evolutionary algorithms</td>
<td>Goldberg (1989), Deb et al. (2002), Roy et al. (2005, 2017), Afrouzy et al. (2016), Arasteh et al. (2021), Katoch et al. (2021)</td>
</tr>
</tbody>
</table>

**Source(s):** Authors’ creation
About the authors
Satyajit Mahato is working as Assistant professor in Birla Institute of Technology, Mesra, Ranchi, India. He has received his Ph.D. from IIT (ISM) Dhanbad, India. He is also a certified Lean Six Sigma Black Belt and an experienced Industrial Engineering professional. In a span of 16 years, he has worked in areas of lean manufacturing, supply chain, operational analytics, IT and business process services. He has a special interest in process excellence techniques, quantitative methods, operations research, process re-engineering, optimization, 3D printing and sustainable supply chain.

Supriyo Roy has worked as Associate Professor at Birla Institute of Technology, Mesra, Ranchi. A Post doctoral Fellow from EADS-SMI Center for Sourcing and Management, IIM, Bangalore, Dr Roy has a rich experience of 4 years in Industry and 21 years of Post graduate Teaching with premier Institutes like BIT-Mesra, SOMS-IIEST Shibpur. His teaching and research interests comprise interdisciplinary areas of management science, production and operations management, multivariate data analysis and business forecasting. He has to his credit, published two books, fifteen international book chapters, twelve international conference papers and forty-five research papers in leading International and National Journals of repute. He has guided ten research scholars toward the Ph.D. program. Dr Supriyo is empaneled Reviewer and Editorial member of International journals of repute. In 2020, he conferred with D. Litt. (Management) from Sambalpur University-Burla, Odisha. Supriyo Roy is the corresponding author and can be contacted at: supriyo.online@gmail.com