Towards automated prevention of rework in software development

VIKTORIIA KONASHCHUK, University of Twente, The Netherlands

Redoing tasks in software development significantly impacts project costs and efficiency and can take up to 50% of a project team's time. Rework may appear due to a variety of reasons like poor planning or incomplete requirements, and the longer root cause issues go undetected, the greater the price for resolving the consequences. Information technologies (IT) are known for assisting humans reliably thanks to their strict logic-based operating and, therefore, may offer promising solutions for automating rework detection and prevention at various stages of the software development cycle. This paper aims to compose an approach that helps IT organizations identify how can the most frequently occurring rework in their operations be minimized. The approach is presented in the shape of a framework that corresponds rework cause categories with their early indicators, operational mitigation strategies and potential automation solutions. The framework is developed based on the academic literature review and validated by applying it to the Dutch IT company context and interviewing IT experts. The results demonstrate that the proposed framework has the potential to reduce rework load by around 9.67% and help organizations identify and prioritize the most effective automation solutions. As a result, this may help businesses shorten the development cycle and improve sustainability through optimised resource use.

Additional Key Words and Phrases: Automation, Rework, Software Development, Software Projects

1 INTRODUCTION

Rework in software development typically implies additional efforts required for redoing incorrectly implemented tasks. These tasks can take up to 50% of a company's development expenditures as well as may negatively affect project outcomes, productivity, team motivation, and customer satisfaction [3, 9]. Rework triggers can appear at any stage of the software development life cycle (SDLC) which consists of planning, defining, designing, building, testing and deployment of the project [27]. This means that there are many possible ways where the execution of the task may go wrong, resulting in an extended list of rework root causes. While rework cannot be fully eliminated due to the dynamic and iterative nature of software projects, its impact can be reduced by early fault detection and appropriate mitigation strategy enforcement [7]. To minimize rework and focus more on value-added tasks, information technology (IT) companies are constantly seeking ways to optimize their workflows [4]. Wide adoption of agile and lean methodologies is a good example for demonstrating efforts of IT businesses to make teams more responsive to changes and eliminate waste in workflow, therefore decreasing the probability of errors [10, 23].

At the same time, due to the abundance of possible pitfalls, humans may struggle to constantly monitor the accuracy of all processes. This is where information technologies come into play. By 2020, around 67% of companies worldwide have already started adopting automation technologies [4] and according to [3], up to 30-50% of rework can be prevented through automation. In software projects, nowadays the majority of error-prevention automations are focused on testing and integration stages. For instance, Jenkins helps to avoid problems with synchronization by establishing automatic deployment [26], while Selenium helps to ensure that applications are bug-free via automated testing [34]. Fast technological advancements over the past decade have expanded the range of IT tools, therefore potentially creating more automation options for rework prevention, which might ultimately help IT companies decrease rework load.

This paper attempts to systematically map the common categories of rework in software projects, their early indicators, corresponding prevention strategies and possible automated prevention solutions into a framework. To note, the framework focuses on operational improvements, emphasizing immediate, actionable steps rather than long-term strategic solutions. The main research question (RQ) of this work is:

"How can an approach be developed that effectively contributes to proactive automated minimization of rework in software development?"

This RQ can be broken down into the following sub-questions (SQ):

- SQ1: What factors contribute to rework in software development?
- SQ2: What early operational indicators signal potential rework?
- SQ3: What operational mitigation strategy can be applied at each rework identification stage?
- SQ4: What automation solutions can help proactively address rework-causing factors?

The framework abridges the answers to the research sub-questions with the aim of presenting the approach for assisting IT businesses in exploring possible automation opportunities for rework minimization. As an additional benefit, through categorization of company rework issues, a framework may help organizations define major causes of rework, their frequency and, therefore, the priority for addressing them.

The effectiveness and applicability of this framework are validated in the context of the Dutch IT company but with the potential to be applied to a wider range of software projects. This is because the framework is based on academic research on global software development processes, which often follow similar SDLC stages and have common rework causes. Despite the differences in methodologies adopted by IT organizations, the underlying processes are generally comparable, therefore sharing the framework's applicability.

The paper is organised as follows. Section 2 demonstrates the research methodology of this work. Section 3 is split into five subparts each addressing one of the steps towards automated rework prevention with the final part presenting a total overview of the framework with the guideline for its application. Section 4 showcases the framework application on the example of the software development company. Section 5 includes the results of validating the framework along with a discussion of its implications and practical limitations. Lastly, a conclusion can be found in Section 6.

2 METHODOLOGY

Figure 1 outlines the research methodology adopted for answering research sub-questions.

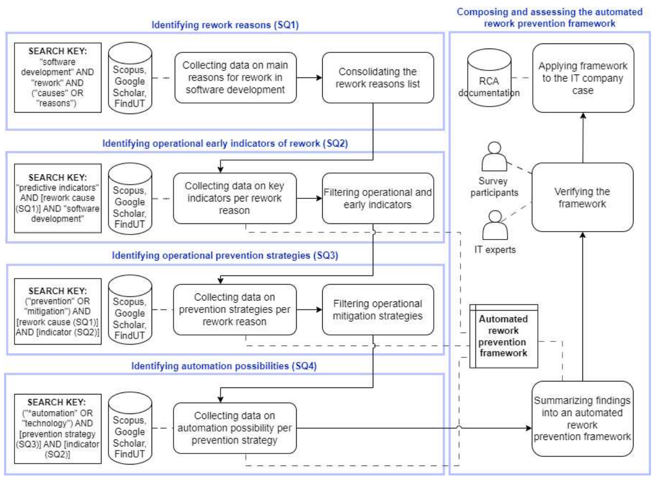
Identifying rework reasons (SQ1)
SEARCH KEY: "software development" AND "rework" AND ("causes" OR "reasons")
Scopus, Google Scholar, FindUT
Collecting data on main reasons for rework in software development
Consolidating the rework reasons list
Identifying operational early indicators of rework (SQ2)
SEARCH KEY: "predictive indicators" AND [rework cause (SQ1)] AND "software development"
Scopus, Google Scholar, FindUT
Collecting data on key indicators per rework reason
Filtering operational and early indicators
Identifying operational prevention strategies (SQ3)
SEARCH KEY: ("prevention" OR "mitigation") AND [rework cause (SQ1)] AND [indicator (SQ2)]
Scopus, Google Scholar, FindUT
Collecting data on prevention strategies per rework reason
Filtering operational mitigation strategies
Identifying automation possibilities (SQ4)
SEARCH KEY: ("*automation" OR "technology") AND [prevention strategy (SQ3)] AND [indicator (SQ2)]
Scopus, Google Scholar, FindUT
Collecting data on automation possibility per prevention strategy
Composing and assessing the automated rework prevention framework
RCA documentation
Applying framework to the IT company case
Survey participants
IT experts
Verifying the framework
Automated rework prevention framework
Summarizing findings into an automated rework prevention framework

Fig. 1. Research methodology. Search key values mentioned in [brackets] are to be replaced with actual corresponding values.

The literature review shows that studies often address rework causes and mitigation strategies jointly [5, 9, 14, 27, 28]. Papers [14] and [27] are comprehensive summaries of existing research on rework causes. This paper builds upon a combination of work by Ramdoo and Huzooree who examine rework causes in an IT company case [28] and a summary by Jain and Suman reviewing existing academic literature on rework [14]. [14] was chosen over [27] due to a more extensive rework causes list and demonstrating mapping to mitigation strategies. The answer to the first research sub-question consolidates the findings into a list of rework causes. Similar rework reasons from both studies are merged to create a more streamlined yet comprehensive scope.

The outline of early rework is based on [12], a summary of error indicators identified through interviews with IT experts. Although not focused on early indicators, [15, 18, 19, 33, 35] provide additional insights by exploring recommendations for the good practices of process arrangement in software projects.

The majority of the suggested prevention strategies included in the framework are derived from the papers exploring a wide range of mitigation techniques for problems in software projects[1, 14, 28], with additions from the more narrow mitigation strategies search for specific scenarios i.e. [16, 29, 32].

Thereafter, the ways for automation are explored. Papers that were used for automation possibilities analysis can generally be classified into the ones that focus on a single SDLC stage or a single rework cause [6, 25, 26, 31, 34], and the ones that explore the benefits of a single technology for rework prevention (i.e artificial intelligence (AI) [17], machine learning [21] etc.). [30] has the most comprehensive summary of how all stages of SDLC can be automated yet limited to the possibilities of AI, which might not always be the most optimal solution. However [25] focuses mostly on the development stage of the project, it provides insights into error prevention approaches that are already utilized by development teams and may be of benefit for the mitigation of a wider range of mistakes. To conclude this step, the outcome list of relevant IT solutions is categorised into easy-to-implement off-the-shelf fixes (category I), more complex but readily available solutions (category II), and solutions that require more advanced technologies (i.e. models with NLP capabilities) for rework prevention (category III). In summary, while existing research effectively addresses specific areas, this paper bridges the gap by connecting the insights from these works.

After the findings are summarised into a comprehensive framework, the framework is presented to IT field experts for review and rectified accordingly. Representatives of IT experts have more than 5 years of experience in software projects and occupy different positions. Next, the framework's effectiveness and applicability are provisionally validated in an IT company operating in the finance sector in the Netherlands.

Finally, the survey consisting of two parts was conducted. The first part includes questions about the applicability, usability and expected effectiveness of the framework and was distributed among audiences with different levels of experience in IT such as students, IT professionals, and researchers. The second section contains similar questions but was distributed only among the IT professionals from the company the framework was tested on. Questions were answered after outcomes of the framework's application in their company's context are presented to survey participants.

3 REWORK PREVENTION FRAMEWORK

Appendix B depicts the research results, the correlation chain of rework root causes - early rework indicators - prevention strategies - and automation possibilities. These tables form the automated rework prevention framework. The framework expands when SQ2 findings are examined. This is because each rework cause may have multiple diverse early indicators. In the later stages, while the table could be expanded further, this research attempted to match findings one-to-one and generalize the findings to keep the outcome comprehensive but to limit the work scope.

The following sections describe the necessary details of each section.

3.1 Rework root causes

Through refining the works [28] and [14], the SQ1 is answered, which forms Column 1 of the framework (Appendix B). The possible root causes were classified into 11 types:

(1) Inadequate requirements: Unclear or shifting project needs (usually from the customer).
(2) Poor communication within the team: Weak information sharing internally.

(3) Poor communication with a client: Lack of clarity with the customer.
(4) Synchronisation and integration challenges: Difficulties in aligning different components or systems.
(5) Internal changes: Changes inside of the company i.e. team composition changes, transfer to a new tool etc.
(6) Poor planning: Inadequate or unclear initial project strategy.
(7) Inadequate domain knowledge: Lack of understanding in relevant subject areas.
(8) Poor technical implementation: Defective execution of technical solutions.
(9) Unreliable documentation: Information that does not reflect current project status.
(10) Insufficient testing: Ineffective testing procedures.
(11) Poor risk identification and management: Lack of preparedness to mitigate potential issues in the project.

To reduce the scope, further sections are explored on the example of the “Poor communication with a client” rework reason.

3.2 Early operational indicators

This part of the framework aims to answer how each of the rework strategies from the previous part could be identified early in the software development process (SQ2). In other words, what can be an early trigger showing that the project or its part might require rework in the future? For example, for the rework reason “Poor communication with a client”, the early might be:

- Inconsistent meeting attendance, client responses taking long [24]
- Lack of domain knowledge (from any of the sides) [22, 33]
- No feedback or review of the main milestones of feature development from the client [16, 18]
- Changes in requirements without notice [27, 35]

Apart from the operational indicators mentioned above, an early signal for the current rework reason can be “Poor communication channel choice”, but due to its strategy-orientedness, it is out of scope for this work. To add, early triggers of other rework cases may also apply to this rework cause. Hence, for the best effectiveness, it is recommended to review also indicators of other rework reasons.

3.3 Operational prevention strategies

For each of the identified indication moments, mitigation strategies are mapped to illustrate how the issues can be prevented from being amplified into a rework task. For instance, when a customer changes requirements without notice, the optimal preventive action would be to get a review from the client on a full subject scope before proceeding with implementation [14, 32]. Another example is when a lack of domain knowledge (from any of the sides) is spotted, it is advised to conduct a knowledge-sharing meeting with domain experts [1]. Similarly, prevention strategies are mapped to each of the identified early operational indicators, making Column 3 address research sub-question 3. While there may be more than one relevant prevention strategy, this work limits the scope to the one most frequently mentioned mitigation strategy per trigger and keeps prevention strategies applicable to a wider range of problems.

3.4 Automation of prevention strategies

Column 4 of the framework suggests an automated way to prevent rework at its early indication moment (SQ4). After examining the capabilities of various existing information technologies and existing good practices on automation, possible automation solutions are mapped to each rework trigger. If existing, an example of the fitting tool is also named. Potential automation solutions can be divided into three categories (Column 5):

- Category I: Readily implementable off-the-shelf solutions. These are the solutions that are already present on the market and commonly used by the software project team. This can include Continuous Integration systems like Jenkins for building, compiling and testing software [8], or project management tools like Jira or Trello for standardizing processes, therefore, decreasing the error rate [31]. While existing tools help to organize processes and to process structured information, they are not suitable for working with unstructured data.
- Category II: Existing solutions requiring advanced configurations. These are usually add-ons on the Category I solutions, that allow a higher level of customisation, hence normally requiring technical skills for implementation [6]. An example of a Category II solution is Jira Automation, which allows users to add custom functionalities by building up custom code parts on top of basic Jira functionality.
- Category III: Solutions requiring intelligence. This type of tools has the capability to deal with a high amount of unstructured data, including the biggest challenge - understanding the intricacies of the human language. According to Krutilla, artificial intelligence models with NLP capabilities excel in this area [17]. NLP models can execute in-depth analysis for textual information such as requirements or context of subject analysis. Although Category III solutions can tackle a wide range of advanced problems, their implementation is typically resource-demanding, which may not always justify the cost. Companies might be willing to opt for simplified versions of solutions from Categories I or II instead.

Continuing the previous example from the framework it can be concluded that a solution is needed that detects a lack of domain knowledge and recommends a team member to conduct a knowledge-sharing meeting with domain experts. Expertise tracking software such as Workday would be a potential solution for this, as it allows deriving the expertise level of one based on the historical projects the employee engaged in. It can also be configured to trigger an alert once it suspects a lack of expertise in the formed team for a new project. This automation solution falls under category III. Category I alternative to it would be a task in workflow to verify the solution with a domain expert on each project stage.

3.5 Total overview

To sum up, the framework in Appendix B integrates all steps of defining the advised automation per rework cause. To apply the framework, an IT company can follow three steps:

(1) Gather historical data on rework tasks, focusing on frequency and root causes. More than one rework reason is possible per rework issue.
(2) Map out causal chains from rework causes to potential early indicators, then to the prevention strategies and potential automation tools for them suggested by the framework.
(3) Implement suggested tools and monitor their effectiveness, adjusting them to the company's needs. The priority for implementing the automation should correspond to the frequency of the rework cause. The higher the frequency - the higher the potential expected benefit from implementing the automation.

To validate the applicability, usability and expected effectiveness of the framework the survey was conducted among people related to IT but with different proficiency levels (see Table 3). The average number of years of experience with software projects among participants is 6.14 years. Participants were asked to choose a score from 1 (strongly disagree) to 5 (strongly agree) for the given statements. Validating the framework in a real-world context can yield more accurate estimates; thus, the following section illustrates its application and assesses the resulting outcomes.

4 FRAMEWORK APPLICATION TO IT COMPANY

To demonstrate the applicability of the framework, a Dutch company that builds a financial IT platform was selected. Documentation from root-cause analysis (RCA) sessions held in the first half of 2024 was used as a basis for analyzing the historical reasons for rework in the organization. Documentation files include a problem summary, the adverse created for the customer, the root causes of the issue, and an elaborated analysis of the rework impact on people, processes, environments or configurations. After applying the framework to the company context following the steps from the previous section, the framework's usability and applicability are assessed.

In the first step, rework reasons were identified from 51 RCA documents and classified into 11 categories. Results are depicted in Table 1. An AI assistant with NLP capabilities, specifically ChatGPT, was utilized for categorization. Although the AI's internal logic is not fully transparent and may limit exact replication, the process involved uploading the anonymized documents to the AI model and instructing it to classify the root causes into 11 predefined categories suggested by the framework. Confidential information or information that may help to identify the company was manually removed beforehand. One-fourth of root-cause analysis documents were also reviewed manually to help validate that the outcomes of AI analysis are accurate enough could reflect reality. To achieve the highest accuracy, the manual review of RCA cases is advised, yet might not be feasible due to the high volume of documentation in combination with time constraints. The outcome showed that "poor technical implementation" is the most frequent cause for rework in the company, followed by "poor communication with a client" and "Inadequate domain knowledge".

Given that these three root causes of rework occur with the greatest frequency, addressing them is likely to yield the most significant impact in reducing rework. Thus, subsequent analysis focuses on these three factors.

Table 1. Rework causes frequency in the studied IT company context.

Rework cause	**Frequency**
Inadequate requirements	4
Poor communication within the team	5
Poor communication with a client	9
Synchronization and integration challenges	7
Internal changes	2
Poor planning	4
Inadequate domain knowledge	9
Poor technical implementation	11
Unreliable documentation	3
Insufficient testing	7
Poor risk identification and management	5

In step 2, relevant early indicators from Column 2 of the framework were pinpointed based on the organizational expertise. Only the selected indicators were included in Table 2.

Consequently, in steps 3 and 4, selected early indicators corresponded with mitigation strategies and suggested automation solutions.

As can be concluded from Column 4 of Table 2, setting up code quality gates, making technical and functional reviews by the team and client review stage mandatory, and setting up the expertise tracking software may contribute the most towards minimization of the rework in the organization. Thus, it is recommended that the company implements the suggested automation solutions to achieve a potential decrease in rework.

Notably, for better results, the project team should also consider implementing the solutions suggested for the rework causes with lower frequency scores following the same steps. In this way, the percentage of the rework prevented is expected to be higher due to more comprehensive coverage of rework issues.

The survey was conducted with software project team members of the Dutch IT company to estimate the expected impact upon framework recommendations implementation and the general applicability of the framework. The survey has a Likert scale format from 1 (strongly disagree) to 5 (strongly agree) with the exception of the last question that estimates the average of the expected decrease in rework after implementing the five top priority automations at the company (in percentage). The results of the survey are presented in Table 4.

The precision of the estimation could be enhanced through the practical implementation of rework prevention automation and subsequent impact tracking. However, given time constraints, the estimate relies on anticipated impacts derived from prior experience.

5 DISCUSSION

5.1 Performance of the framework

Responses to the similar survey statements asked in theory (Table 3) and after a pilot study (Table 4) show a similar pattern. Results reveal descent potential for the framework to minimize rework. From the lowest yet still adequate score for easiness of rework categorization, it can be concluded that it might be challenging to associate a rework reason with certain categories. The easiness and applicability

Table 2. Framework application to IT company context.

Rework cause	Early indicator	Prevention strategy	Automation solution	Type
Poor technical implementation	Poor coding practices	Conduct a knowledge-sharing meeting with domain experts	Set up code quality gates (i.e. SonarQube)	III
	No review of work done by a domain professional	Ask domain expert to review the work piece	A review by a domain expert as a task in a workflow (i.e. Trello)	I
Poor communication with a client	Lack of domain knowledge (from any of the sides)	Conduct a knowledge-sharing meeting with domain experts	Expertise tracking software (i.e. Workday)	III
	No feedback or review of main milestones of feature development from the client	Before proceeding with implementation, get a review from the client	Adding the client review step to the checklist, automated reminder if missing (i.e. Trello)	I
Inadequate domain knowledge	No verification of analysis outcomes with domain professional	Ask domain expert to review the work piece	A review by domain expert as a task in workflow (i.e. Trello)	I

Table 3. Survey results on general expected framework performance.

Question	Average (st.dev.)
General information	
I have experience with identifying rework	3.57 (1.27)
I have experience with mitigating rework	3.57 (1.27)
I have experience with automating rework prevent.	3.14 (1.57)
Framework performance questions	
The framework is easy to understand	4.71 (0.49)
It's easy to categorize rework reasons to fit in the framework	3.43 (0.98)
The framework is useful for identifying early indicators of potential rework	3.71 (0.95)
The framework is useful for defining the prevention strategy for identified rework causes	4.29 (0.49)
The framework suggests reasonable automation solutions	4.14 (0.38)

Table 4. Survey results on framework performance in IT company context.

Question	Average (st.dev.)
Application of the framework may benefit my organization	4.67 (0.58)
It is clear how framework applies to my organization	5.00 (0)
The framework can be useful for categorizing and prioritizing the rework in my organization	4.67 (0.58)
The framework can be useful for defining rework mitigation strategies for my organization	3.67 (1.15)
The framework provides reasonable suggestions for automated rework prevention solutions	4.33 (0.58)
It can be beneficial to implement suggested automations of Category I in my organization	4.67 (0.58)
It can be beneficial to implement suggested automations of Category II in my organization	3.67 (0.58)
It can be beneficial to implement suggested automations of Category III in my organization	2.67 (0.58)
If based on personal estimates, implementing the five automations recommended by the framework could lead to a potential decrease in rework within my organization, what percentage reduction can be expected?	9.67% (5.03%)

of further steps of the framework application demonstrate higher levels of expected effectiveness. The acceptability of the proposed solutions is around 82%. The expected percentage of potential rework decrease is 9.67%, which shows a space for improvement as according to [3] up to 50% of rework is feasible to be prevented.

Among the solution categories, Category I has a comparable expected impact to Category II. Category III, despite having the highest share among the proposed automation on a framework (45% comparably to 32.5% and 22.5% for categories I and II respectively), has a lower score, which might be due to its cost-benefit ineffectiveness.

5.2 Practical Limitations and Implications

The company's historical data on rework tasks of good enough quality is required for proper analysis and rework cause classification in the first step of the framework application. However, unlike [13], which requires a sufficient number of records to identify action patterns, this framework does not necessitate a minimum amount of documentation for its successful application. Having met this requirement, this framework is possibly generalizable to be applied to various IT companies although the framework's effectiveness may vary based on project complexity, technology infrastructure, and organizational culture. Sometimes the framework might not account for the specifics of a certain company, for example, when none of the key early indicators are feasible. In such cases, the following questions can guide in defining possible automated rework prevention solutions:

(1) What is the root cause?
(2) What can be an early indicator that this cause is about to happen?
(3) Having it early identified, what would be the most effective preventive action?

(4) How can that action be automated? Are there any existing solutions that can be utilized?

To add, the current framework is mostly beneficial for projects when no consistent and quantifiable data is available. The current framework also strives to offer solutions that require manual input only when a rework threat is identified, but not for constant monitoring as proposed by [20]. This is because having a structured input event stream, a solution like Complex Event Processing might be more beneficial [36]. Moreover, rework prevention contributes to sustainability by optimizing resource use. This means it does not only boost team morale and customer satisfaction but also reduces the environmental impact of software development activities.

6 CONCLUSION

To finalise, this paper presents the approach aiming to help software companies proactively reduce the amount of rework by advising the automation to be implemented based on the frequency of rework types. The study identifies eleven causes of rework and maps each category to their relevant early operational indicators, operational mitigation strategies and possibilities for automation for the prevention of the rework reason. Suggested information technologies for automation represent three types: readily implementable off-the-shelf solutions (32.5%), existing solutions requiring advanced configurations (22.5%) and solutions requiring intelligence (45%). The findings are summarised into a framework, validated in the Dutch IT company context. Results show that the proposed framework has decent potential for minimization of rework with around 82% of automation solutions acceptability rate, therefore, potentially helping spare the organization's resources and improve project metrics.

Due to research time and scope constraints, this paper tends to synthesize findings. In the future, the study can be extended to provide a more comprehensive list of scenarios, make the scenarios more narrow, and include a strategy-oriented direction for rework prevention. The accuracy of framework performance assessment can also be improved by increasing the number of survey participants, or for even higher precision, implementing and monitoring the framework's impact on various IT organizations. Finally, the framework can be extended to focus on strategical prevention for rework to contribute to more long-term solutions.

REFERENCES

[1] José L. Barros-Justo, Fabiane B. V. Benitti, and Jefferson S. Molleri. 2021. Risks and risk mitigation in global software development: An update. *Journal of software* 33, 11 (2021). https://doi.org/10.1002/smr.2370

[2] Saskia Bick, Kai Spohrer, Rashina Hoda, Alexander Scheerer, and Armin Heinzl. 2018. Coordination Challenges in Large-Scale Software Development: A case study of planning misalignment in hybrid settings. *IEEE transactions on software engineering* 44, 10 (2018), 932–950. https://doi.org/10.1109/tse.2017.2730870

[3] B.W. Boehm and P.N. Papaccio. 1988. Understanding and controlling software costs. *IEEE transactions on software engineering* 14, 10 (1988), 1462–1477. https://doi.org/10.1109/32.6191

[4] Camunda. 2023. The State of Process Automation Report 2020. https://camunda.com/state-of-process-automation-2020/

[5] Aaron G. Cass, Stanley M. Sutton, and Leon J. Osterweil. 2003. *Formalizing rework in software processes.* 16–31 pages. https://doi.org/10.1007/978-3-540-45189-1

[6] S. Chandrashekar, B. Mayfield, and M. Samadzadeh. 1993. Towards automating software project management. *International journal of project management* 11, 1 (1993), 29–38. https://doi.org/10.1016/0263-7863(93)90007-a

[7] R.N. Charette. 2005. Why software fails [software failure. *IEEE spectrum* 42, 9 (2005), 42–49. https://doi.org/10.1109/mspec.2005.1502528

[8] Levi Connelly, Melody Hammel, Benjamin Eger, and Lan Lin. 2022. Automated Unit Testing of Hydrologic Modeling Software with CI/CD and Jenkins. *Proceedings of the International Conference on Software Engineering and Knowledge Engineering* (2022). https://doi.org/10.18293/seke2022-074

[9] R.E. Fairley and M.J. Willshire. 2005. Iterative rework: the good, the bad, and the ugly. *Computer* 38, 9 (2005), 34–41. https://doi.org/10.1109/mc.2005.303

[10] Simon Hacks, Hendrik Hofert, Johannes Salentin, Yoon Chow Yeong, and Horst Lichter. 2019. Towards the Definition of Enterprise Architecture Debts. https://doi.org/10.1109/edocw.2019.00016

[11] Muhammad Hamid, Furkh Zeshan, Adnan Ahmad, Saadia Malik, Muhammad Saleem, Nadia Tabassum, and Muhammad Qasim. 2022. Analysis of software success through structural equation modeling. *Intelligent automation and soft computing* 31, 3 (2022), 1689–1701. https://doi.org/10.32604/iasc.2022.020898

[12] Douglas Havelka, T. M. Rajkumar, and Patrick Serve. 2004. Early indicators of troubled IS development projects. *Americas Conference on Information Systems* (2004), 115. https://aisel.aisnet.org/amcis2004/115/

[13] Fuqun Huang and Lorenzo Strigini. 2023. HEDF: a method for early forecasting software defects based on human error mechanisms. *IEEE access* 11 (2023), 3626–3652. https://doi.org/10.1109/access.2023.3234490

[14] Ritu Jain and Ugrasen Suman. 2021. Root causes and reduction techniques for rework in global software development. *International journal of computer applications* 183, 43 (2021), 40–44. https://doi.org/10.5120/ijca2021921840

[15] Leon A. Kappelman, Robert McKeeman, and Lixuan Zhang. 2006. Early warning signs of IT project failure: the dominant dozen. *Information systems management* 23, 4 (2006), 31–36. https://doi.org/10.1201/1078.10580530/46352.23.4.20060901/95110.4

[16] Artem Katasonov and Markku Sakkinen. 2005. Requirements quality control: a unifying framework. *Requirements engineering* 11, 1 (2005), 42–57. https://doi.org/10.1007/s00766-005-0018-1

[17] Zsolt Krutilla. 2023. Developing of NLP models by model based software development. *Gradus* 10, 2 (2023). https://doi.org/10.47833/2023.2.csc.024

[18] Lina Abu Lebdeh, Amer Qasim, and Faten Kharbat. 2020. Implementing agility in large software development projects. *TEM Journal* (2020), 1285–1294. https://doi.org/10.18421/tem93-58

[19] Timo O.A. Lehtinen, Mika V. Mäntylä, Jari Vanhanen, Juha Itkonen, and Casper Lassenius. 2014. Perceived causes of software project failures – an analysis of their relationships. *Information and software technology* 56, 6 (2014), 623–643. https://doi.org/10.1016/j.infsof.2014.01.015

[20] Shaoying Liu. 2021. A three-step hybrid specification approach to error prevention. *Journal of systems and software/The Journal of systems and software* 178 (2021), 110975. https://doi.org/10.1016/j.jss.2021.110975

[21] Lukas Longard, Felix Brungs, Christian Hertle, Jochen Roeth, and Joachim Metternich. 2021. Reduced rework through data analytics and machine learning – a three level development approach. *Social Science Research Network* (2021). https://doi.org/10.2139/ssrn.3859900

[22] David B. Lowe and John Eklund. 2002. Client needs and the design process in web projects. *Journal of web engineering* 1, 1 (2002), 23–36. https://doi.org/10.5555/2011098.2011103

[23] Peter Middleton, Philip S. Taylor, Amy Flaxel, and Ammon Cookson. 2007. Lean principles and techniques for improving the quality and productivity of software development projects: a case study. *International journal of productivity and quality management* 2, 4 (2007), 387. https://doi.org/10.1504/ijpqm.2007.013334

[24] Nils Brede Moe and Darja Šmite. 2008. Understanding a lack of trust in Global Software Teams: a multiple-case study. *Software process improvement and practice* 13, 3 (2008), 217–231. https://doi.org/10.1002/spip.378

[25] Bhaveet Nagaria and Tracy Hall. 2022. How software developers mitigate their errors when developing code. *IEEE Transactions on Software Engineering* 48, 6 (2022), 1853–1867. https://doi.org/10.1109/TSE.2020.3040554

[26] P. Narang. 2023. Automated continuous deployment of software projects with Jenkins through devops-based hybrid model. (2023). https://doi.org/10.21203/rs.3.rs-3205341/v1

[27] Shiza Nawaz, Anam Zai, Salma Imtiaz, and Humaira Ashraf. 2022. Systematic Literature Review: Causes of Rework in GSD. *The international Arab journal of information technology* (2022). https://doi.org/10.34028/iajit/19/1/12

[28] Vimla Devi Ramdoo and Geshwaree Huzooree. 2015. Strategies to reduce rework in software development on an organisation in Mauritius. *International journal of software engineering and applications* 6, 5 (2015), 9–20. https://doi.org/10.5121/ijsea.2015.6502

[29] Kenneth H. Rose. 2013. A Guide to the Project Management Body of Knowledge (PMBOK® Guide)-Fifth Edition. *Project management journal* 44, 3 (2013). https://doi.org/10.1002/pmj.21345

[30] Hazrina Sofian, Nur Arzilawati Md Yunus, and Rodina Ahmad. 2022. Systematic mapping: artificial intelligence techniques in software engineering. 10 (2022), 51021–51040. https://doi.org/10.1109/access.2022.3174115

[31] Carsten Stechert and Hans-Patrick Balzerkiewitz. 2020. Digitalization of a lean product development organization. *Procedia CIRP* 91 (2020), 764–769. https://doi.org/10.1016/j.procir.2020.02.232
[32] A. Takats and N. Brewer. 2006. Improving communication between customers and developers. In *Agile Development Conference (ADC'05)*. https://doi.org/10.1109/adc.2005.30
[33] June Verner, Jennifer Sampson, and Narciso Cerpa. 2008. What factors lead to software project failure? *IEEE Xplore* (2008). https://doi.org/10.1109/rcis.2008.4632095
[34] Fei Wang and Wencai Du. 2012. A test automation framework based on WEB. In *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*. 683–687. https://doi.org/10.1109/ICIS.2012.21
[35] Terry Williams, Ole Jonny Klakegg, Derek H. T. Walker, Bjørn Andersen, and Ole Morten Magnussen. 2012. Identifying and acting on early warning signs in complex projects. *Project management journal* 43, 2 (2012), 37–53. https://doi.org/10.1002/pmj.21259
[36] Fuyuan Xiao, Cheng Zhan, Hong Lai, Li Tao, and Zhiguo Qu. 2017. New parallel processing strategies in complex event processing systems with data streams. *International journal of distributed sensor networks* 13, 8 (2017). https://doi.org/10.1177/1550147717728626

A USE OF AI

During the preparation of this work the author(s) used (1) Grammarly, (2) chatGPT, (3) Scite and (4) Quillbot to (1) correct grammar and spelling mistakes and paraphrase sentences for better readability, (2) paraphrase sentences, help to define general concepts, categorize the input data from the company into rework causes proposed by the framework, (3) assist in finding academic sources and (4) conduct AI detection check. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the work.

B AUTOMATED REWORK PREVENTION FRAMEWORK

See following pages.

Table 5. Automated rework prevention framework.

Rework cause	Early indicator	Prevention strategy	Automation solution	Type
Inadequate requirements [14, 28]	Major solution direction or requirements switching [19, 27, 28]	Refer to change control process [29]	Requirements analysis software with NLP capability (i.e. Jama Connect) [30]	III
	The customer is not involved in the process [28, 33]	Establish regular communication checkpoints [1, 32]	Client involvement communication multi-channel monitoring	III
	Ambiguous, vague language of requirements [35]	Review of requirements for clarity, nonduality and completeness [16]	Requirements analysis software with NLP capability (i.e. Jama Connect) [30]	III
	Very different scope/tasks compared to the scope of similar projects	Review completeness of gathered requirements with stakeholders including owners of similar projects [1, 14, 16, 28]	Projects comparison software integratable with company wiki	III
	Impact on pipeline applications is not inspected	Discuss impact on pipeline with relevant stakeholders [1, 14, 16, 28]	Requirements analysis software with NLP capability (i.e. Jama Connect) [30]	III
	No proper pre-analysis	Gather requirements and discuss solution direction with domain experts	A pre-analysis as a task in workflow (i.e. Trello) [25]	I
	Missing functional or non-functional requirements (i.e. performance metrics)	Contact domain experts [1, 14, 16, 28]	Requirements analysis software with NLP capability (i.e. Jama Connect) [30]	III
Poor communication within the team [14, 28]	Frequent missing of goals (i.e. deadlines) [35]	Labelling and correction actions during sprint reviews and retrospectives [1]	Missed deadlines alerting script monitoring through the project planning tool (i.e. Trello bot)	II
	Conflicting changes in requirements (within the story or within multiple subjects that are currently in development) [16]	Review the story for cohesion; contact team members working on intersecting work pieces [14]	Requirements analysis and tracking software integrated with company wiki	III
	Low participation in meetings, long team members' response times [35]	Label it and clarify expectations with people involved [1]	Client involvement tracker for multi-channel communication monitoring	III
	Insufficient documentation [15, 22]	Conduct a meeting with domain professionals to document knowledge [1]	Automation script in the project management tool for tracking the presence of documentation (i.e. Jira Automation)	II
	Multiple teams working simultaneously on closely related tasks [2]	Contact team members working on intersecting work pieces	Requirements analysis software for detecting interdependent tasks (i.e. Jama Connect)	III
Poor communication with a client [14, 28]	Inconsistent meeting attendance, client responses taking long [24]	Label it and clarify expectations with people involved [1]	Client involvement tracker for multi-channel communication monitoring	III
	Lack of domain knowledge (from any of the sides) [22, 33]	Conduct a knowledge-sharing meeting with domain experts [1]	Expertise tracking software (i.e. Workday)	III
	No feedback/review of main milestones of feature development from the client [16]	Before proceeding with implementation, get a review from the client [1, 14, 28, 32]	Adding the client review step to the checklist, automated reminder if missing (i.e. Trello) [25]	I
	Changes in requirements without notice [27, 35]	Get a review from the client on a full subject scope [1, 14, 28, 32]	Approval of final subject scope signed by a client as a task in workflow (i.e. Trello) [25]	I
Synchronization and integration challenges [14]	Absence of an integration plan [29]	Make creation of a plan a mandatory step in project step execution	A check for integration plan creation as a task in a workflow (i.e. Trello) [25, 29]	I
	No intermediate integrations [18]	Schedule code integration or split integrations into smaller parts to fit sprints for early validation	Automated merge checks (i.e. Jenkins)	I
Internal changes [14]	Received announcements of changes	Analyze and note the impact of changes on the work piece	Intelligent alerting of potentially affected people	III

Rework cause	**Early indicator**	**Prevention strategy**	**Automation solution**	**Type**
Poor planning [14]	Missed deadlines early on [12, 35]	Review and adjust the planning or scope with stakeholders [28]	Automation script in the project management or planning tool (i.e. Jira Automation or Trello bot)	II
	Lack of detailed project plan [35]	Make creation of a plan a mandatory step in project step execution [29]	Project plan creation as a task in workflow (i.e. Trello) [25, 29]	I
	Frequent scope changes [12]	Refer to change control process [29]	Automation script in the project management tool for scope tracking (i.e. Jira Automation)	II
	Unrealistic project timelines or deadlines [11, 33]	Review and adjust the planning with stakeholders [1]	Planning feasibility assessment tool	III
	Unreasonable planning in comparison to past projects	Review and adjust the planning with stakeholders [1]	Planning feasibility assessment tool	III
Inadequate domain knowledge [14, 28]	No verification of analysis outcomes with domain professionals [33]	Ask domain expert to review the work piece [28]	A review by domain expert as a task in workflow (i.e. Trello) [25]	I
	Frequent corrections that majorly change the solution direction [35]	Conduct a knowledge-sharing meeting with domain experts [1]	Requirements analysis software with NLP capability (i.e. Jama Connect) [30]	III
	Vague answers to critical questions [35]	Conduct a knowledge-sharing meeting with domain experts [1]	Requirements analysis software with NLP capability (i.e. Jama Connect) [30]	III
	No kick-off done with stakeholders who authorize solution (architects etc.)	Conduct a meeting with stakeholders to verify solution	A review by domain expert as a task in workflow (i.e. Trello) [25]	I
Poor technical implementation [14]	An unusually high number of bug reports early in the development process [12]	Conduct code review [1, 27]	Automated alerts when bug tracking tool detects high number of bugs (i.e. Bugzilla) [25]	II
	Poor coding practices [18]	Conduct a knowledge-sharing meeting with domain experts	Set up code quality gates (i.e. SonarQube)	I
	High amount of technical debt [10]	Address critical areas of technical debt before new feature development [1]	Automate static code analysis to monitor technical debt (i.e. SonarQube) [25]	II
	No review of work done by a domain professional [19]	Ask domain expert to review the work piece [1, 28]	A review by domain expert as a task in workflow (i.e. Trello)	I
	No toggles to turn functionality off to test it on acceptance environments before going to production	Check whether toggles might be beneficial to implement	A check for toggles applicability as a task in a workflow (i.e. Trello) [25]	I
Unreliable documentation [14]	Discrepancies between the documentation and the actual functionality [12, 27]	Conduct a meeting with domain professionals to document latest knowledge [1]	Company wiki and code base analysis for correspondence, intelligent search for existing documentation	III
	Old modification dates of the documentation or missing documentation	Conduct a meeting with domain professionals to document latest knowledge [1]	Automation script in the project management tool for tracking presence of documentation (i.e. Jira Automation)	II
	Limited documentation [15, 22]	Conduct a meeting with domain professionals to document knowledge [1]	Automation script in corporate wiki to alert about outdated documentation (i.e Confluence automation)	II
Insufficient testing [28]	Lack of test coverage for requirements [19]	Verify completeness of tests for covering all the functional requirements [1]	Requirements analysis and test scenarios tracking software for coverage tracking (i.e. Cucumber)	III
	Inconsistent or unstable testing environments	Standardize and automate environment provisioning	Automated environment provisioning with infrastructure as code (i.e. Terraform)	I
Poor risk identification and management [14]	Absence of risk register [12, 15, 29]	Develop a risk register [29]	A check for risk register creation as a task in workflow (i.e. Trello) [25]	I
	Old modification dates of risk register [15, 33]	Conduct a risk review and update the register [29]	Automation script in the corporate wiki (i.e Confluence automation)	II