SOFTWARE QUALITY PLAN

Process Mining-Based Failure Prediction and Prevention in IKEA's Order Lifecycle Process

VERSION 2

12/07/2023

DOCUMENT CONTROL

FILE NAME		DOCUMENT FINAL VERSION
Software Quality Plan		2
DOCUMENT OWNER	**ISSUE DATE**	**LAST SAVED DATE**
Elham Honarvar		12/07/2023

VERSION HISTORY

VERSION	REVISION DATE	DESCRIPTION OF CHANGE	AUTHOR	REVIEWER
0.1	29/05/2023	The first draft of the Quality Plan.	Elham Honarvar	
0.2	06/06/2023	The requirements are added to this document.	Elham Honarvar	Gijs Walravens Ger Cloudt
0.3	16/06/2023	• New requirements added to this document. • The quality assurance strategy is added. • Scrum development strategy is added. • Bug lifecycle is added.	Elham Honarvar	Navoneel Chakrabarty Gijs Walravens Mihai Zelina
1.0	18/06/2023	• Quality control testing strategy is added • Design and implementation strategy is added.	Navoneel Chakrabarty	Elham Honarvar
1.1	22/06/2023	Design and implementation strategy is edited.	Mihai Zelina	Elham Honarvar
1.2	27/06/2023	• New bugs added • The *Quality Assurance table for design and implementation strategy is edited.*	Elham Honarvar	Ger Cloudt Mihai Zelina
1.3	11/07/2023	The static code analysis result is added	Elham Honarvar	TIOBE company
1.4	12/07/2023	The recommendation regarding quality improvement is added	Elham Honarvar	

DOCUMENT APPROVALS

ROLE	NAME	DATE
Project Sponsor	IKEA	
Project Review Group	JADS and TU/e	14/07/2023
Project Manager	Gijs Walravens	14/07/2023
Quality Assurance Manager	Elham Honarvar	12/07/2023
Test Manager	Navoneel Chakrabarty	14/07/2023
Software Architect	Mihai Zelina	14/07/2023
Scrum master	Elham Honarvar	12/07/2023
Team Leader	Saba Abdian	14/07/2023
Quality Coach	Ger Cloudt	14/07/2023

QUALITY CONTROL PLAN OVERVIEW

1. Project Goal
2. Quality Control Objectives
3. Verification of Requirements
4. ISO/IEC 25010 Quality Model
5. Design Methodology
6. Bug Management
7. Testing Strategy
8. Code analysis and review
9. Code analysis result and recommendations

TABLE OF CONTENTS

1. PROJECT GOAL

The goal of the project titled "Process Mining-Based Failure Prediction and Prevention in IKEA's Order Lifecycle Process" is to create a technique, for identifying critical characteristics or trends in order lifecycle records, enabling the early detection of probable issues. By doing this, we hope to increase customer satisfaction, lower expenses, and improve the order fulfillment process.

2. QUALITY CONTROL PLAN OBJECTIVES

The purpose of this quality plan is to guarantee a high-quality and dependable solution for IKEA since it offers an organized way to verify requirements, monitor quality attributes, evaluate deliverables, manage issues, and conduct code reviews.

3. VERIFICATION OF REQUIREMENTS

Deliverables	Requirements	Priority
Working prototype of crucial attributes/patterns detection	• The solution shall detect crucial activity patterns for imperfect orders. • The solution reports the probability of a given order being imperfect. • The solution shall provide a visualization that explains the prediction of the order being perfect or imperfect. • The solution shall be able to handle one country's worth of data.	**High**
	The solution might be integrated to provide real-time alerts for orders that are likely to become imperfect.	**Low**
Code of the prototype	• The solution shall be implemented in Celonis ML Workbench. • The solution shall be created using Python. • The solution shall only be accessible via Celonis ML Workbench. • The solution shall ensure code readability and maintainability.	**High**
User manual on how to run the prototype	• The team shall create a user manual instruction on how to use the delivered solution. • The documentation of the solution shall be done in English.	**High**
Report on the design and implementation of the prototype	The whole design process will be documented in the System Architecture document.	**High**
Report on any findings	• The team shall provide a literature study document. • The team shall provide a technical report that documents any findings, recommendations, and observations.	**Medium**

4. ISO/IEC 25010 QUALITY MODEL

The widely acknowledged standard ISO/IEC 25010 served as the foundation for the choice of quality criteria in this project. This standard offers a thorough framework for defining and assessing the quality of software products. We guarantee that the project's quality qualities are clearly defined, measurable, and in line with best practices in the industry by aligning with ISO/IEC 25010.

Quality Attributes	Quality Assurance
QA1. Portability	The solution shall be implemented only in Celonis so portability is not a primary focus in this quality plan.
QA2. Security	Security is not a primary focus in this quality plan as the solution in Celonis does not have access to IKEA customer-sensitive data, ensuring the protection of software and data against unauthorized access.
QA3. Compatibility	• Perform a detailed analysis of the system requirements and specifications to identify any potential compatibility constraints • Adopt the solution to any additional requirement • Track changes made to the solution and its dependencies • Foster open communication and collaboration with stakeholders
QA4. Performance Efficiency	• High performance of the predictive model for order imperfection • Optimized computational time for the predictive model to predict imperfection (if any) • The generation of the visualization explaining the model prediction should be self-explanatory as well as computationally optimized as much as possible • Assess stakeholders' satisfaction with the solution's performance
QA5. Functional Suitability	• Conduct comprehensive requirement analysis and validation. Each requirement should be validated using various methods, including comparing the performance of different models and employing diverse visualization approaches. • Perform thorough functional testing. • Approve the functionality of the code through stakeholder's meeting.
QA6. Usability	• Provide clear documentation and user manual • Ensure that the code is written in a clear and understandable manner, accessible to both technical and non-technical teams. This should be done by incorporating comments and diagrams in to the code.
QA7. Reliability	• Identify and address bugs or issues in the code as much as possible • Apply appropriate fixes to resolve the bugs and ensure that the code functions as intended • To ensure the reliability of the results several modeling approaches will be implemented by the design team and will be approved by the quality manager, test manager, and stakeholders

QA8. Maintainability	• Ensure proper documentation and code commenting • Conduct regular code reviews • Re-training the model based on predictive performance with future orders

5. DESIGN METHODOLOGY

The project follows the SCRUM lifecycle, allowing for iterative and incremental development. To ensure compatibility throughout the development process, the following strategies will be employed:

Scrum Methodology	Quality assurance	Responsible Person
Sprint Planning	During daily sprint sessions, the compatibility of requirements and constraints will be analyzed and prioritized	Scrum master Team Leader
User Stories and Acceptance Criteria	• Clear Definition of Acceptance Criteria for each user story and a test method that clearly defines the acceptance criteria for each user narrative and verifies it. • Review and validate that the acceptance criteria are realistic and achievable within the project constraints • Encourage open communication and regular discussions within the team to clarify requirements, resolve ambiguities, and ensure shared understanding of the user stories and acceptance criteria • Continuously monitor the quality and effectiveness of the user stories and acceptance criteria throughout the project • Promote knowledge sharing and provide training if needed to ensure everyone understands the quality expectations	Scrum master Team Leader
Continuous Integration and Testing	While unit testing and integrated testing were not performed on the code due to some time limitations, we implemented a comprehensive set of tests that focused on content verification and data consistency, and update verification.	Test Manager
Compatibility Testing Sprint	In each sprint, compatibility checks will be performed and approved to ensure that the solution remains adaptable to any new changes in the requirements.	Quality manager Test manager
Collaboration and Communication	Regular collaboration and communication will be encouraged with stakeholders to address compatibility concerns	Project manager Scrum master Team Leader

From an architectural point of view, the software architect devises a Design and Implementation Strategy during the software development lifecycle. This strategy explains the high-level design of the end product based on the architectural specifications and requirements. In our case, the

(software) deliverables mainly revolve around a demo/prototype that incorporates the key functionality of a predictor model, meaning that the product has a more compact, straightforward structure. In the table below, we describe, for each architecture component, the means through which it is accomplished and the personnel responsible for quality.

System Architecture Document Components	Method	Tool	Review by
Introduction and Context	• Review project requirements and goals • Determine business, information systems, and technology context • Determine key stakeholders and their required views		Project Manager Quality Manager Stakeholders
Requirements	• Extract key solution responsibilities • Determine key requirements that influence architectural decisions		Quality Manager Team Leader Stakeholders

Key Design Decisions and Concerns	• Determine the key architectural decisions, together with rationale and impact
	• Determine remaining architectural decisions
	  ○ Monitor new concerns
	  ○ Integrate new decisions
	  ○ Update architecture
		Celonis ML Workbench	Design team
			Test Manager
			Quality Manager

Operational View	• Determine stakeholder view needs. • Identify involved applications, entities, and infrastructure. • Display run-time interaction and functionality	Celonis ML Workbench StarUML	Test Manager

Delivery Breakdown View	• Determine high-level delivery strategy.
	• Create Solution Breakdown Structure (SBS):
	  ○ Determine member responsible for each deliverable element
	  ○ Determine how element will be validated/tested
	  ○ Ensure elements are structured such that they represent deliverables ready to be sent to customer
	• Determine development approach for each deliverable
	• Determine integration plan
		Celonis ML Workbench
		StarUML	

Specialized Views	• Describe other possible views • Relevant: use-case views, activity diagram	Celonis ML Workbench StarUML	

It is crucial to check that the quality needs and considerations are adequately handled during the design and execution phases in the quality plan.

System Architecture Document Components	Quality Assurance Method
Introduction and Context	• Ensure that the content is well-organized and easy to understand • Ensure that the introduction and context section provides a comprehensive overview of the project and its context • Engage technical teams of stakeholders in reviewing the introduction and context section • Continuous improvement of the System Architect document: Collect feedback from stakeholders, monitor the effectiveness of the document, and update it when necessary
Requirements	• Review the requirements to ensure they cover all essential aspects of the system • Verify that there is no contradictory or overlapping requirements • Identify the key requirements that have a significant impact on architectural decisions. • Key requirements should be specific, measurable, and traceable • Discuss requirements and their implications on architectural choices with technical teams of stakeholders • Requirements should aim to be SMART: Specific, Measurable, Achievable, Relevant, and Time-Bound
Key Design Decisions and Concerns	• Verify that the decisions are aligned with the project goals, requirements, and stakeholder expectations • Monitor and identify new concerns that may impact the system architecture through stakeholders meeting and internal meetings with the design team • If new decisions are integrated, make sure that the system architecture document is updated accordingly
Operational View	• Ensure that the document captures and addresses the view needs of the stakeholders and validates the alignment of it • Ensure that the document provides a comprehensive representation of all elements involved in the software's operation • Ensure that the operational view is consistent with logical view • Assess the operational view for clarity and understandability • Review the operational view to ensure that it provides a complete and accurate representation of the system's operational aspects
Delivery Breakdown View	• Review the SBS to check whether it provides a clear hierarchy and breakdown of the solution components and check its consistency with the project's scope and stakeholder needs • Identify the specific validation method for the solution • Determine the development approach for each deliverable element in the SBS and check that the selected approaches align with the project requirements • Engage stakeholders to validate the quality of the delivery breakdown view
Specialized Views	• Ensure that the document provides a comprehensive description of different specialized views that can improve the understanding of the system architecture • Verify that the use-case view provides valuable insights into the system's behavior • Ensure that the use-case view depicts the interactions between different components of the system • As the document and project evolve, re-assess whether other possible relevant views might need to be added

6. BUG MANAGEMENT

Bug management refers to the systematic process of identifying, documenting, tracking, and resolving software bugs or defects throughout the development lifecycle. It involves capturing, prioritizing, and managing reported issues to ensure their timely resolution and to maintain the overall quality of the software product. The severity level of the bug is decided by System Architect and System designers.

Bug number	Reason	Severity	How to deal with it
B.1	The amount of data that can be extracted directly relates to the RAM space in Celonis	**Critical**	Increase the Memory in Celonis
B.2	Data inconsistency due to data loss and new data push in the Data Pool.	**Major**	Checking the consistency of the data using the old class labels and the newly available set of activities
B.3	Order creation time and the time of the 1st activity of the order lifecycle are unequal	**Minor**	
B.4	In LIME explainability, the contribution of the features involved in imperfection/perfection is changing every time it is executed	**Major**	Change the explainable method to SHAP

All members of the design team will actively track the bug lifecycle to ensure timely resolution.

Bug lifecycle	New	In analysis	In resolution	In verification	closed
B.1	✓	✓	✓	✓	✓
B.2	✓	✓	✓	✓	✓
B.3	✓	✓	✓		
B.4	✓	✓	✓	✓	✓

7. TESTING STRATEGY

In the quality plan, our testing strategy employed an approach to ensure the reliability and accuracy of the code's output. While unit testing and integrated testing were not performed on the code, we implemented a comprehensive set of tests that focused on content verification and data consistency, and update verification.

Test objective	Test Type	Test environment	Test Status	Quality Attributes coverage
To find the number of unique orders for each salesSetId	Order-based Univariate EDA Verification	Celonis ML Workbench	**Approved**	QA4, QA7
To extract the salesSetIds with all imperfect orders that are more than one	Order-based Bivariate EDA Verification	Celonis ML Workbench	**Approved**	QA4, QA7
To explore the positions of the first imperfect activity in every order lifecycle	Event-time-based EDA (Investigative Data Analysis)	Celonis ML Workbench	**Approved**	QA4, QA7
To find the minimal position of the first imperfect activity across all order lifecycles	Event-time-based EDA (Investigative Data Analysis)	Celonis ML Workbench	**Approved**	QA4, QA7
To find the % of unique orders with different positions of occurrences of the first imperfect activity	Event-time-based EDA (Investigative Data Analysis)	Celonis ML Workbench	**Approved**	QA4, QA7
To compare the order creation datetime and the time of the first activity of the order lifecycle	Event-time-based EDA (Investigative Data Analysis)	Celonis ML Workbench	**Approved**	QA4, QA7
To check the data inconsistencies and data update	Data Maintenance	Celonis ML Workbench	**Approved**	QA3, QA4, QA7
To merge the notebooks performing similar tasks into one	Code Maintenance	Celonis ML Workbench	**Approved**	QA6, QA8
To perform feature engineering with the sequence of activities for each order lifecycle	Predictive Process Mining	Celonis ML Workbench	**Approved**	QA4, QA5
To perform different visualization approaches to check the consistency of the explainability diagram.	Explainability AI	Celonis ML Workbench	**Approved**	QA4, QA5

8. CODE ANALYSIS AND REVIEW

We want to perform a thorough static code analysis on our final codebase using the knowledge and resources offered by **TIOBE** company, a prominent software analysis business. Finding potential flaws, inefficiencies, and adherence to coding standards all depend heavily on static code analysis. We want to make sure that our software is of the best caliber and dependability, thus we have asked TIOBE to conduct this analysis. According to TIOBE Quality Indicator to obtain a systematic way of measuring and qualifying these quality attributes, the 8 most commonly used software code quality metrics have been selected that can be measured in an automated way. These are:

1. **Code coverage:** The code coverage metric indicates how many lines of code or executable branches in the code have been touched during the unit test runs. The lower the coverage, the lower the quality of the performed unit tests. Code coverage is an indicator of both "Functional Suitability" and "Reliability" of the ISO 25010 standard.

2. **Abstract interpretation:** This is used to check the possible reliability issues in software programs. It can detect all kinds of programming errors related to the control flow of a program. This metric is mapped to the "Reliability" characteristic of the ISO 25010 standard.

3. **Cyclomatic complexity:** Cyclomatic complexity counts the number of independent paths through a program.

4. **Compiler warnings:** Compilers/interpreters generate errors and warnings. Errors must be fixed otherwise the program cannot run. Warnings on the other hand do not necessarily need to be solved. However, some compiler warnings indicate serious program flaws. Leaving these unresolved has probably impact on the "Reliability" of the code. So this metric can also mapped to "Portability" in most cases.

5. **Coding standards:** A coding standard is a set of rules that engineers should follow. Since coding standards usually contain many different rules they can be mapped to most quality characteristics. Most rules concern "Maintainability" and "Reliability", but there are also rules available for "Portability" and "Performance Efficiency" of the ISO 25010 standard.

6. **Code duplication:** The drawback of code duplication is that if one part of the code must be changed for whatever reason (solving a bug or adding missing functionality), it is very likely that the other parts ought to be changed as well. This has a negative effect on the "Maintainability" characteristic of the ISO 25010 standard.

7. **Fan out:** The fan out metric indicates how many different modules are used by a certain module. If modules need a lot of other modules to function correctly (high fan out), there is a high interdependency between modules, which makes code less modifiable. Hence, fan out is mapped to the "Maintainability" ISO 25010 characteristic.

8. **Security:** Security of software is about how vulnerable code is to get unauthorized access to data and how easy it is to make changes to the software by exploiting security leaks.

Here is the mapping between TQI metrics and ISO 25010 quality attribute:

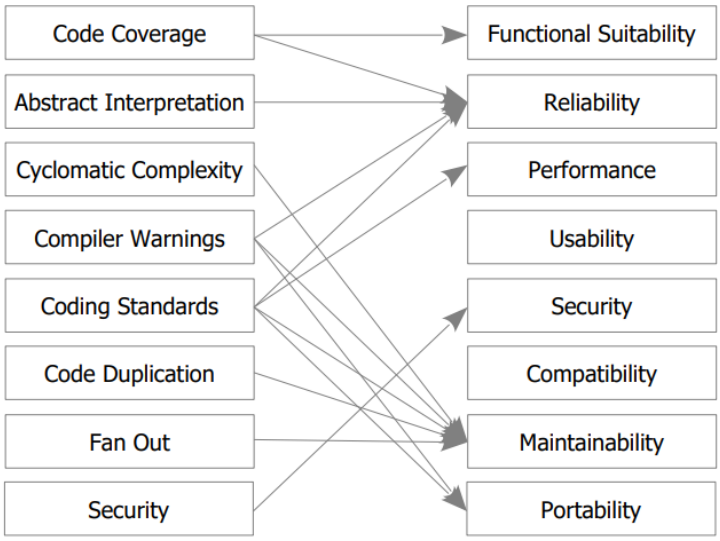


Figure 1- Mapping TQI metrics to ISO 25010

Metric	Weight
Code Coverage	20%
Abstract Interpretation	20%
Cyclomatic Complexity	15%
Compiler Warnings	15%
Coding Standards	10%
Code Duplication	10%
Fan Out	5%
Security	5%

Figure 2- Weights of TQI metrics

Category	Name	TQI Score
A	Outstanding	>= 90%
B	Good	>= 80%
C	Fairly Good	>= 70%
D	Moderate	>= 50%
E	Weak	>= 40%
F	Poor	< 40%

Figure 3- TQI score

9. CODE ANALYSIS RESULT AND RECOMMENDATIONS

To facilitate a comprehensive analysis, a dashboard has been made available by TIOBE Company. Access to this dashboard can be obtained using the information provided below:

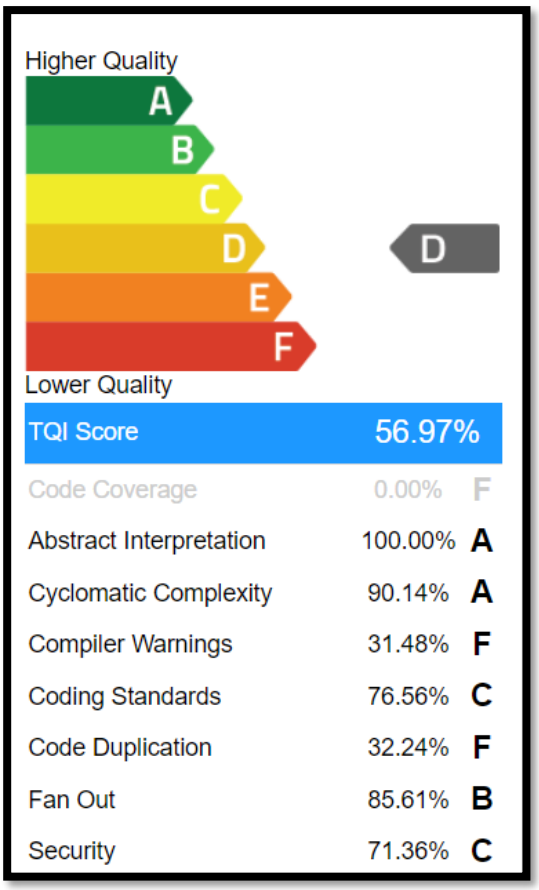
By utilizing the details mentioned, you can gain entry to the dashboard and delve into the intricacies of this analysis.

URL:
https://assessments.tiobe.com/tiobeweb/TICS/TqiDashboard.html#axes=Owner(TU%20Eindhoven),Project(IKEA),Sub()&metric=tqi

User: tu-eindhoven-ikea

Pass: aHQDvUr^Ep3DDTor%pz547N6Srn5KZDN

Here is the summary of the results provided by TIOBE software:



QUALITY LEVEL	PERCENTAGE	A	B	C	D	E	F
TQI SCORE	56.97%						
CODE AVERAGE	Not considered in the evaluation as we didn't have any but it is highly recommended to have						
ABSTRACT INTERPRETATION							
CYCLOMATIC COMPLEXITY							
COMPILER WARNINGS							
CODING STANDARDS							
CODE DUPLICATION							
FAN OUT							
SECURITY							

TIOBE Quality Indicator (TQI): The TIOBE Quality Indicator for our project is 56.97% and it means that it is in category D which is a Moderate level.

Security: We have got C for this metric which is acceptable for us because this quality attribute was not the focus of our project but in case this attribute becomes the main focus so it needs some improvement.

Fan out: It is an indicator of modularity we got B which means that it is good in modularity.

Code Duplication: We received a grade of F for this metric, signaling a substantial opportunity for improvement by systematically eliminating duplicate code and replacing it with a reusable function that can be utilized as required.

Coding Standards: We achieved a C grade for this metric; however, based on our results, the extent of these standard violations is not considered severe. Nevertheless, it is recommended to make necessary changes and enhance this metric as it can potentially impact various quality attributes.

Compiler warnings: F level for this metric is not desirable. It is essential to address and rectify some of these warnings since they have the potential to impact the maintainability attribute significantly.

Maintaining high levels of maintainability is one of the primary objectives of our project, making it crucial to make the necessary improvements in this metric. Some tools can be used to check the quality within Jupyter Notebooks such as pylint or nbQA.

Cyclomatic Complexity: The complexity of our code is very low and the score is A for this metric. It means that it is easily understandable for people with different backgrounds.

Abstract Interpretation: No issue is found in our code regarding this metric and we got A for this metric which is acceptable.

Code Coverage: No code coverage is provided for this project but it is highly recommended to have one as it affects functionality and reliability of our code.

Suggestion: There is a todo list provided by the TIOBE tool and it is recommended to start with solving these important issues.