Automating Requirement Quality Standards with QVscribe

Olivia Kenney
Marketing Dept.
Halifax, NS
olivia.kenney@qracorp.com

Matthew Cooper
Product Dept.
Halifax, NS
matt.cooper@qracorp.com

Abstract

QVscribe is a requirement authoring and analysis tool that harnesses Natural Language Processing (NLP) to proactively check for compliance of the best requirements analysis practices identified by associations such as INCOSE and against generally accepted industry frameworks such as Easy Approach to Requirements Syntax (EARS). It has been proven to reduce time and cost of reviewing requirements by 50% or more (QRA19).

1 Overview

QVscribe's automated requirements analysis enables engineering teams to build faster by identifying errors in the first stages of the development and design process, where they matter most, cost the least to fix, and reduce future rework. Its latest updates include improvements to Quality Analysis, INCOSE guideline compliance detection, EARS templating, and semantic similarity. QVscribe also offers customizable configurations to meet the unique needs and standards of any team and can generate actionable reports to facilitate effective collaboration and review.

1.1 Quality Analysis

QVscribe's Quality Analysis assists users with authoring and quality review of requirements by automating best practices and compliance standards. The tool checks for proper use of imperatives, optional escape clauses, superfluous infinitives, data-driven quality indicators, immeasurable quantifications, and weak, vague, and subjective words. Based on those factors as well as on its preset or customized configuration, QVscribe analyzes the requirement document and provides a quality score for the entire document in addition to one for each specific requirement. The quality score is from one to five with one indicating a high risk of error and five indicating a low risk of error. Additionally, it offers quality warnings for factors such as universal quantifiers and the use of passive voice to warn users of common quality issues. Quality warnings do not affect the quality score, as some of these issues may be intentional, giving users the flexibility to determine which requirements need the most attention.

1.2 EARS Templating

The EARS methodology was first presented at the IEEE International Requirements Engineering (RE) Conference in 2009. It has since been widely adopted across the RE community to help users author clear, concise, natural language requirements easily, and improve RE workflows.

EARS classifies requirements into five EARS syntax patterns. These include ubiquitous, event-driven, optional feature, unwanted behaviour, and complex requirements. All of the syntax options have been formatted into automated templates within QVscribe (as seen in Figure 1) to create a fill-in-the-blank authoring process that is clear and compliant to EARS best practices. The templates are as follows:

Ubiquitous Requirements: are not invoked by an event or input, nor are they limited to a subset of the system's operating states.
The <system name> shall <system response>.

In: M. Sabetzadeh, A. Vogelsang, S. Abualhaija, M. Borg, F. Dalpiaz, M. Daneva, N. Fernández, X. Franch, D. Fucci, V. Gervasi, E. Groen, R. Guizzardi, A. Herrmann, J. Horkoff, L. Mich, A. Perini, A. Susi (eds.): Joint Proceedings of REFSQ-2020 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track, Pisa, Italy, 24-03-2020, published at http://ceur-ws.org

State-Driven Requirements: are active throughout the time a defined state remains true.
While <in a specific state> the <system name> shall <system response>.

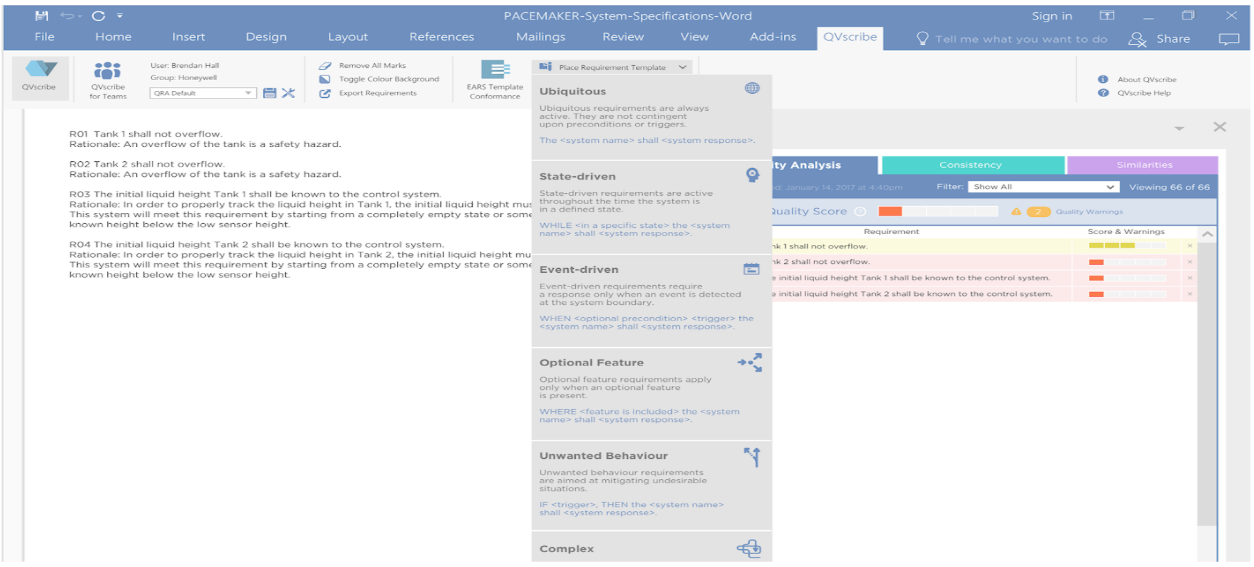
Event-Driven Requirements: require a response only when an event is detected at the system boundary.
When <trigger> the <system name> shall <system response>.

Optional Feature Requirements: apply only when an optional feature is present as a part of the system.
Where <feature is included> the <system name> shall <system response>.

Unwanted Behaviour Requirements: are often imposed when the system must respond to a trigger under less than optimum conditions.
If <trigger>, then the <system name> shall <system response>.

Complex Requirements:
While <precondition(s)> when <trigger> the <system name> shall <system response>.

This feature allows users to standardize and automate their authoring process so that teams with multiple authors can write consistent requirements while also reducing misinterpretation. This consistency also creates an increase in efficiency in both the authoring and review processes. Within QVscribe's Quality Analysis, the tool will check each requirement to see whether or not it is EARS conformant and which syntax is being used. This automated compliance check allows teams to ensure they are



User: Brendan Hall
Group: Honeywell
QRA Default

R01 Tank 1 shall not overflow.
Rationale: An overflow of the tank is a safety hazard.

R02 Tank 2 shall not overflow.
Rationale: An overflow of the tank is a safety hazard.

R03 The initial liquid height Tank 1 shall be known to the control system.
Rationale: In order to properly track the liquid height in Tank 1, the initial liquid height mus
This system will meet this requirement by starting from a completely empty state or some
known height below the low sensor height.

R04 The initial liquid height Tank 2 shall be known to the control system.
Rationale: In order to properly track the liquid height in Tank 2, the initial liquid height mu
This system will meet this requirement by starting from a completely empty state or some
known height below the low sensor height.

Ubiquitous
Ubiquitous requirements are always active. They are not contingent upon preconditions or triggers.
The <system name> shall <system response>.

State-driven
State-driven requirements are active throughout the time the system is in a defined state.
WHILE <in a specific state> the <system name> shall <system response>.

Event-driven
Event-driven requirements require a response only when an event is detected at the system boundary.
WHEN <optional precondition> <trigger> the <system name> shall <system response>.

Optional Feature
Optional feature requirements apply only when an optional feature is present.
WHERE <feature is included> the <system name> shall <system response>.

Unwanted Behaviour
Unwanted behaviour requirements are aimed at mitigating undesirable situations.
IF <trigger>, THEN the <system name> shall <system response>.

Complex

d: January 14, 2017 at 4:40pm
Filter: Show All
Viewing 66 of 66
2 Quality Warnings

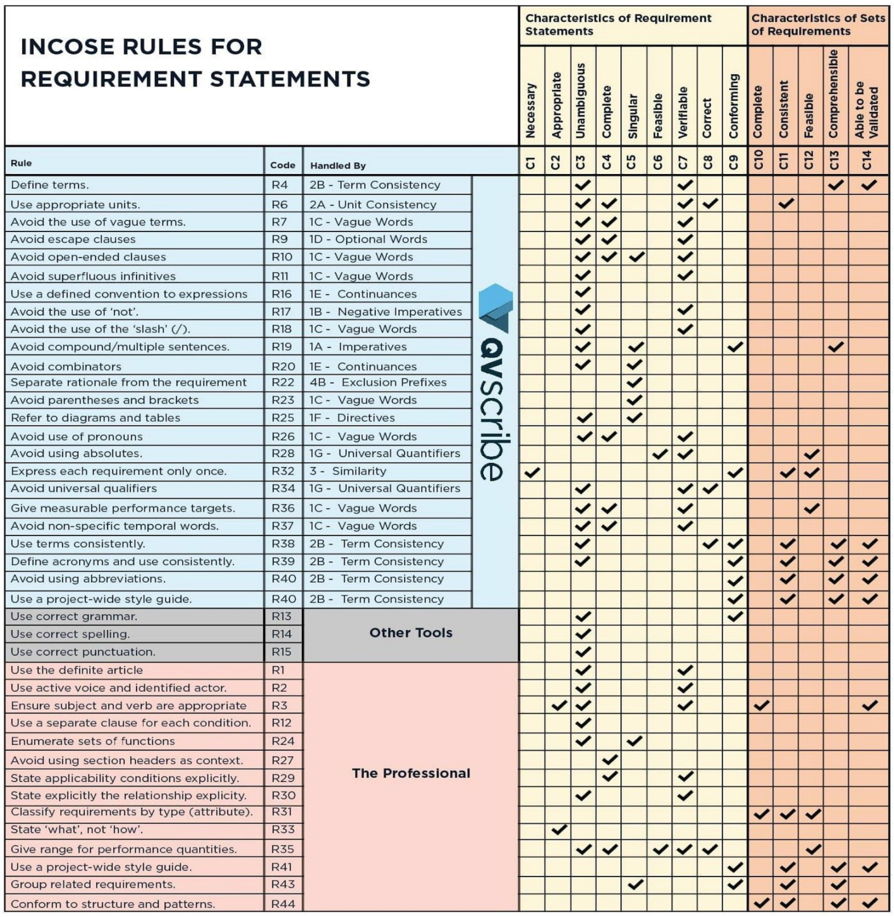
Requirement	Score & Warnings
nk 1 shall not overflow.	
nk 2 shall not overflow.	
e initial liquid height Tank 1 shall be known to the control system.	
e initial liquid height Tank 2 shall be known to the control system.	

formatting requirements uniformly, avoiding ambiguities and misinterpretations.

Figure 1: EARS Templating Feature

1.3 Automating the INCOSE Guidelines

The INCOSE *Guide for Writing Requirements* (*GFWR*) is one of the most widely used and highly respected references in RE (INCOSE 2017). It provides a comprehensive set of 44 rules (as seen in Table 1) for helping users author clear and concise requirements. Previously, RE teams have had to manually train themselves and check for compliance to INCOSE which is a tedious, error-prone process.



INCOSE RULES FOR REQUIREMENT STATEMENTS

			Characteristics of Requirement Statements									Characteristics of Sets of Requirements				
			Necessary	Appropriate	Unambiguous	Complete	Singular	Feasible	Verifiable	Correct	Conforming	Complete	Consistent	Feasible	Comprehensible	Able to be Validated
Rule	Code	Handled By	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14
Define terms.	R4	2B - Term Consistency			✓				✓						✓	✓
Use appropriate units.	R6	2A - Unit Consistency			✓	✓			✓	✓			✓			
Avoid the use of vague terms.	R7	1C - Vague Words			✓	✓			✓							
Avoid escape clauses	R9	1D - Optional Words			✓	✓			✓							
Avoid open-ended clauses	R10	1C - Vague Words			✓	✓	✓		✓							
Avoid superfluous infinitives	R11	1C - Vague Words			✓				✓							
Use a defined convention to expressions	R16	1E - Continuances			✓											
Avoid the use of 'not'.	R17	1B - Negative Imperatives			✓				✓							
Avoid the use of the 'slash' (/).	R18	1C - Vague Words			✓				✓							
Avoid compound/multiple sentences.	R19	1A - Imperatives			✓		✓				✓				✓	
Avoid combinators	R20	1E - Continuances			✓		✓									
Separate rationale from the requirement	R22	4B - Exclusion Prefixes					✓									
Avoid parentheses and brackets	R23	1C - Vague Words					✓									
Refer to diagrams and tables	R25	1F - Directives			✓		✓									
Avoid use of pronouns	R26	1C - Vague Words			✓	✓			✓							
Avoid using absolutes.	R28	1G - Universal Quantifiers						✓	✓					✓		
Express each requirement only once.	R32	3 - Similarity	✓								✓		✓	✓		
Avoid universal qualifiers	R34	1G - Universal Quantifiers			✓				✓	✓						
Give measurable performance targets.	R36	1C - Vague Words			✓	✓			✓					✓		
Avoid non-specific temporal words.	R37	1C - Vague Words			✓	✓			✓							
Use terms consistently.	R38	2B - Term Consistency			✓					✓	✓		✓		✓	✓
Define acronyms and use consistently.	R39	2B - Term Consistency			✓						✓		✓		✓	✓
Avoid using abbreviations.	R40	2B - Term Consistency									✓		✓		✓	✓
Use a project-wide style guide.	R40	2B - Term Consistency									✓		✓		✓	✓
Use correct grammar.	R13	Other Tools			✓						✓					
Use correct spelling.	R14				✓											
Use correct punctuation.	R15				✓											
Use the definite article	R1	The Professional			✓				✓							
Use active voice and identified actor.	R2				✓				✓							
Ensure subject and verb are appropriate	R3			✓	✓				✓			✓				✓
Use a separate clause for each condition.	R12				✓											
Enumerate sets of functions	R24				✓		✓									
Avoid using section headers as context.	R27					✓										
State applicability conditions explicitly.	R29					✓			✓							
State explicitly the relationship explicity.	R30				✓				✓							
Classify requirements by type (attribute).	R31											✓	✓	✓		
State 'what', not 'how'.	R33			✓												
Give range for performance quantities.	R35				✓	✓		✓	✓	✓				✓		
Use a project-wide style guide.	R41										✓		✓		✓	✓
Group related requirements.	R43						✓				✓		✓		✓	
Conform to structure and patterns.	R44											✓	✓		✓	✓

Table 1: Coverage of Rules and Characteristics for Requirements and Sets of Requirements

There are nine characteristics that every requirement should possess. They are necessary, appropriate, unambiguous, complete, singular, feasible, verifiable, correct, and conforming. In addition, there are five characteristics that every set of requirements should possess. They are complete, consistent, feasible, comprehensible, and able to be validated. QVscribe uses NLP to assess if each characteristic is present in each requirement and identifies weaknesses to be corrected. Identifying these errors early on help to massively reduce risk later on in the project.

The latest version of QVscribe uses NLP to automatically measure compliance against these and is preconfigured to check for 24 out of 44 of the *INCOSE Rules for Requirement Statements*. The only rules that are not automatically checked by QVscribe are rules that require human action such as "R41: Use a project-wide style guide". QVscribe will identify and alert authors to the use of passive voice, it is then up to the author to correct the error. Overall, it helps users automate the majority of requirements quality assurance tasks, reduces requirement review and correction time by as much as 50% or more, as seen by the RCAF and Ultra Electronics Marine Systems (QRA19), and allows RE professionals more time to focus on content rather than syntax.

1.4 Similarity

QVscribe uses NLP to compare each requirement to all the others in the document and assesses requirement similarity on two levels; lexical and semantic. Lexical similarity compares requirements based on characters. Semantic similarity compares requirements based on their meaning. For example, "What is your age?" and "How old are you?" are typed quite differently but essentially have the same meaning therefore their lexical similarity would be low, but their semantic similarity would be high. This feature enables requirements engineers to ensure there are no redundant or duplicate requirements. This is especially valuable to teams with multiple authors collaborating on one document where different authors could have expressed the same requirement in different ways. Avoiding redundancies is crucial to improving efficiency and reducing rework which inevitably increases over time as they lead to additional time spent correcting errors along in the development process.

2 Plan for Demo

This demo will be an overview of how QVscribe functions and improves RE workflows. The demo will use an existing requirement document example to show a variety of issues and demonstrate how QVscribe addresses them.

2.1 Quality Score

The presenter will demonstrate that when the user clicks on a requirement, QVscribe highlights any issues and gives suggestions based on industry best practices so that authors can easily improve their requirements prior to review. With QVscribe, users can now edit and receive feedback in real time, aiding significantly in the training and knowledge transfer process. Next, the presenter will explain QVscribe's automated EARS templating feature which allows engineers to use fill-in-the-blank templates to standardize their authoring process and reduce time and misinterpretation. They will show an example of a couple of the templates, when to use them and how to fill them in, as well as showing that the Quality Analysis will identify when requirements are EARS conforming. Then they will show examples of poorly authored requirements and how to correct them. Firstly, the presenter will show a requirement with a quality issue, for example, no imperatives, they will show how QVscribe highlights that error, then they will fix the error and reanalyze the requirement to show a new requirement with a high quality score.

2.2 Consistency Analysis

Next, the presenter will show an example of a unit consistency issue and explain the value of this analysis and how to utilize it in one's workflow. For example, requirements listing some speeds in km/h and some in mph, including an explanation on how QVscribe identifies these issues and how to fix them if/when necessary.

2.3 Similarity Analysis

The presenter will go into Similarity Analysis and provide an explanation of lexical and semantic similarity, including the difference and importance of both. This includes an example of requirements that are not lexically similar but have a significant semantic similarity.

2.4 Generating a Report

Finally, the presenter will generate a PDF report which includes all of QVscribe's analysis, including a breakdown of all the Quality Indicators, each requirement highlighted to show any issues, Unit Consistency, Term Consistency, and a list of all the terms in the document. The presenter will also explain the value of having these easily shareable reports for collaboration across teams.

References

[QRA19] QRA. How Ultra Electronics Maritime Systems Reduced the Time and Cost to Revise Requirements by 75% or More, February 2019.

[INCOSE17] INCOSE. Guide for Writing Requirements, June 2017.