qvscribe

EARS

Easy Approach to Requirement Syntax

Part 2

Introduction

Unconstrained natural language requirements can be vague, ambiguous, verbose, and confusing. In many cases, these requirements can lead to unexpected interpretations, erroneous implementations, costly scrap and rework, and – in the worst cases – disaster.

The Easy Approach to Requirements Syntax - EARS, helps solve that problem by bringing just enough rigor to the process of writing requirements in natural language. Simplifying the requirements writing process and supporting authors in writing clear and consistent requirements.

EARS is available in QVscribe as templates, and in the following three part series, we will outline how you can optimize the EARS format to enhance and simplify your requirement writing personally, in a team, and company-wide.

Please note that in certain circumstances using EARS templates is not ideal. Please refer to the following blog on when not to use EARS.

In Part 2 of the EARS Templates Series, we are focusing on using EARS format to avoid the use of high risk problems commonly found in natural language requirements and how you can optimize EARS Complex templates in your requirements and projects.

Use EARS to Reduce Major Problems in Natural Language Requirements

Providing authors with a clear and easy framework allows them to focus more on the content and intent rather than the structure of their requirements.

Having that structure not only helps with quicker reviews, but it leaves little room for common requirements pitfalls like ambiguity, wordiness, and lack of singularity.

EARS along with QVscribe helps authors reduce the following common errors:

- Unclear – Risk of being misunderstood by a reader
- Not Testable – Too vague to be proved
- Non-Compliant Structure – Poorly formatted requirements
- Uncontrolled Terms – Multiple names for the same object
- Improper Units – Different systems of units used
- Duplication – The same information repeated
- Contradiction – Conflicting information in the same document

Using EARS to author your requirements from the start can limit the chances of running into these common errors. You can use EARS to help rewrite requirements during your review process and translate legacy data. Requirements are commonly reused from previous projects. Using EARS to rewrite historical data will ensure your requirements are of good quality from project to project. You can also compare the quality of the historical requirements with the success of your projects to really understand the impact of clear, atomic and consistent requirements.

EARS can turn a high-risk and wordy requirement into a clear, consistent, and atomic requirement.

Rewriting High Risk Requirements

Let's set the stage.

You are using QVscribe to review the requirements you wrote for a Cruise Control system. You just ran the QVscribe analysis. In the QVscribe Quality Analysis, you have a few requirements that are scored as very high-risk - 1 out of 5 and high-risk - 2 out of 5. You pull up the requirements to see how you can fix them. However, you are really struggling to make changes that would give these requirements a better score.

Whether you have never written requirements before or you have been writing them for years, you may need some guidance in certain circumstances. Maybe the context is complicated, you are using other information as support or you simply do not know the best way to phrase your requirement - EARS can help with rewriting and restructuring.

EARS templates set a clear structure. They can help you consider WHAT you want the system to do and IF, WHEN, WHILE, and WHERE the conditions surrounding the system behavior will occur. Using EARS templates makes it easy to be clear about your pre-conditions and triggers, and guides you to quickly define the states of your system.

Let's walk through a couple of examples.

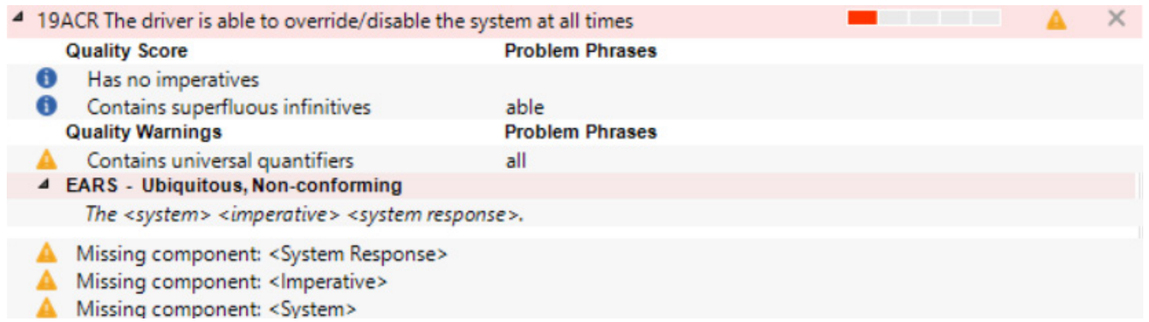
Example 1

Our first example is a requirement that has a 1 out of 5 score in the QVscribe Quality analysis and does not follow EARS format.

Original requirement:

Key: Quality Alert Quality Warning

The driver is able to override/disable the system at all times.



This requirement is very high-risk and has critical problems. Specifically, no imperative, a superfluous infinitive, and a possible issue, a universal quantifier. This requirement needs some fundamental changes to make it well-written and clear.

QVscribe has also provided an EARS conformance analysis, which has determined that this requirement might fit into ubiquitous pattern, but given the fact that it is non-conforming, we need to analyze the intent of the requirement to decide what template format is most appropriate.

Let's consider the word 'all' are we really going to give the driver the ability to override/disable the system at all times? To ensure we are being accurate and clear about when this is supposed to happen, it would make sense to write the requirement from the perspective of the system using an **Event-Driven Template**.

An Event-Driven Template requires a response only when an event is detected at the system boundary. Using this template can make it clear when this requirement applies. The template will also guide us to write a requirement from the perspective of the system, which aligns with system requirement best practices.

Let's use the Event-Driven Template to rewrite your very high-risk requirement.

When <trigger>, the <system> <imperative> <system response>.

If you want more information on writing event-driven requirements, please refer to EARS Use Case Part 1.

What is the <trigger>?

Original requirement:

Key: Quality Alert Quality Warning

The driver is able to override/disable the system at all times.

A trigger causes something to happen. The trigger specifies events that are desired. In this example, we are not clear on what the trigger is. We need to specify what trigger will allow the driver to override/disable the system.

Let's say we want the driver to be able to disable/override the system when they turn the cruise control off.

When the driver sets the cruise control toggle to off, the <system> <imperative> <system response>.

This outlines a clear trigger.

What is the <system>?

Original requirement:

Key: Quality Alert Quality Warning

The driver is able to override/disable the system at all times.

Edits in progress:

When the driver sets the cruise control toggle to off, the <system> <imperative> <system response>.

In this case, it is fairly simple, the system is the Cruise Control System.

Based on the Event-Driven Template, we should write the requirement from the perspective of the system. Therefore, our revised requirement will flow a bit differently.

When the driver sets the Cruise Control Toggle to off, the Cruise Control System <imperative> <system response>.

What is the <imperative>?

Original requirement:

Key: Quality Alert Quality Warning

The driver is able to override/disable the system at all times.

Edits in progress:

When the driver sets the Cruise Control Toggle to off, the Cruise Control System <imperative> <system response>.

Either the writer has completely omitted the use of the imperative or is trying to use the word **able** to describe the action that needs to be taken. A requirement must contain an imperative to assert that action is taken by the subject in the requirement. Let's take a look at the lack of an imperative as a problem type. *(Refer to Issue One on the Following Page)*

Is this really a requirement? Without an imperative, how can the reader be sure that what they are reading is actually a requirement? How can the author be sure that each reader will understand that this is required? Is the requirement describing the current state or a new function that needs to be implemented?

In this case, we suggest adding an imperative to this requirement.

When the driver sets the Cruise Control Toggle to off, the Cruise Control System shall <system response>.

By adding the shall, we are also removing the word **able**.

Issue One:

Issue	**No Imperative**
Issue Type	**Quality Alert -** The presence of Quality Alert issues will have a negative impact on the Quality Score of requirements and the document. A Quality Alert will also be present if a requirement lacks an imperative or has multiple imperatives.
Insight/ Recommendations	Enhance requirement completeness by including an acceptable imperative such as "shall", "must" and "will" in between the entity responsible and the action that is required.
Description	A requirement must contain an imperative to assert that action is taken by the subject in the requirement. Without an imperative, it can be difficult to assert that what is being described is actually required. Using the same imperative consistently is the best way to reduce risk in this area.
Possible Exceptions	Some organizations will employ specific terms to differentiate between types of clauses such as requirements, recommendations, and allowances. Your organization can modify the imperatives list to accommodate any allowable terms it employs.

Issue Two:

Problem Word	**able**
Issue	**Superfluous Infinitives**
Issue Type	**Quality Alert -** The presence of Quality Alert issues will have a negative impact on the Quality Score of requirements and the document. A Quality Alert will also be present if a requirement lacks an imperative or has multiple imperatives.
Insight/ Recommendations	Improve requirement verifiability by removing unnecessary verb phrases such as “allow”. These phrases make it unclear under which circumstances the intended action needs to be taken.
Description	Infinitives are a type of verb that makes the action passive. In requirement writing superfluous infinitives can make it difficult to know under which circumstances the requirement applies and how to know when the requirement has been met.
Possible Exceptions	If you’re defining user needs or high level requirements it is acceptable to use infinitives to describe certain capabilities, but these must be broken down into specific actions in the later stages. Your organization can create a QVscribe Configuration for high level requirements which allows a specific set of these phrases for use in these cases.

Would it be sufficient to say that the system will provide the driver with the ability to take action or do we need to know that the system will actually perform as expected? Could this be rephrased to describe what action will be taken when a particular event takes place rather than explaining what needs to be possible?

Clearly, removing the word able will make our requirement easier to understand.

What is the <system>?

Original requirement:

Key: Quality Alert Quality Warning

The driver is able to override/disable the system at all times.

Edits in progress:

When the driver sets the Cruise Control Toggle to off, the Cruise Control System shall <system response>.

The system response is the result that the system must produce. In this case, it would override/disable the system. Within the system response, there is a problem word flagged, we should work through this issue to ensure we have the right information.

Issue Three:

Problem Word	**all**
Issue	**Universal Quantifier**
Issue Type	**Quality Warning -** The presence of Quality Warning issues will not impact the Quality Score of requirements and the document. The proper use of Quality Warning issues depends on the context in which they are used. They can cause scope creep and can distract from the meaning.
Insight/ Recommendations	Reduce ambiguity by ensuring the scope of the requirement is clear. Consider replacing universal quantifiers such as 'all', 'any', and 'no' with the specific entities or values that are being referenced.
Description	A universal quantifier is a reference without boundaries. When used in a requirement, universal quantifiers can make the meaning unclear or expand the scope beyond what is intended.
Possible Exceptions	A universal quantifier can be used in certain situations where a clear boundary is defined, for example, "when all of the following are true: (followed by a finite list)". They can also be used when there are in fact no limits "if no signal is detected". For these reasons, universal quantifiers are categorized as Quality Warnings and do not impact the Quality Score.

In this context, do we really mean at **all** times? How do we account for all times?

Given that we are now using an Event-Driven Template, we have defined the clear boundary at the start of the requirement, by saying 'when the driver sets the Cruise Control Toggle to off'.

In this case, we just need to be clear about how the system will respond to this trigger. We want the system to override/disable. This can be more easily described using a mode. A mode is a collection of settings, which we can outline outside of the requirement statement. This allows us to easily and clearly define a system response across different requirements.

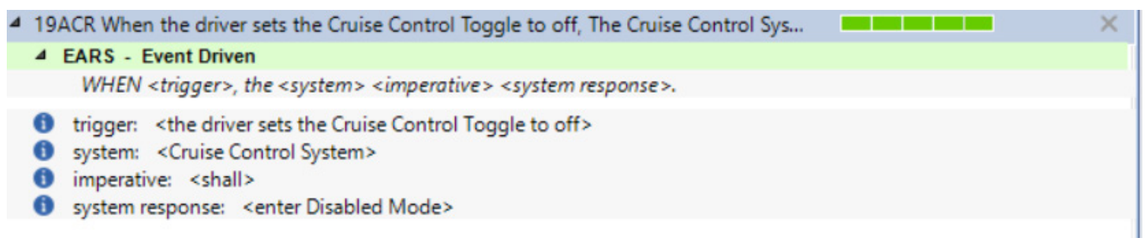
Let's say in a mode we want multiple actions to occur, we don't have to list them all in the requirement and can simply refer to the mode. In this case, we will call this system response the Disabled Mode.

Revised Requirement:

> When the driver sets the Cruise Control Toggle to off, the Cruise Control System shall enter Disabled Mode.

Using an Event-Driven Template to rewrite our very high-risk requirement along with QVscribe recommendations for resolving problem words has made our requirement singular, clear, and verifiable.

Our new score in the QVscribe Quality Analysis is 5 out of 5.



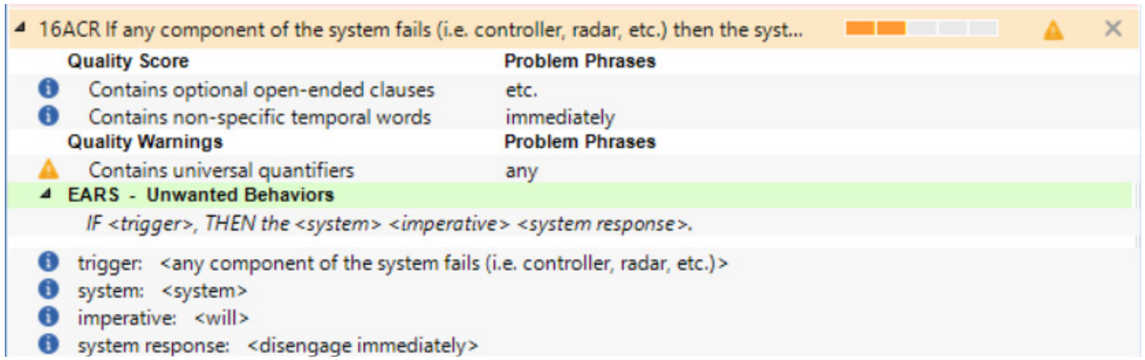
Example 2

Our next example is a requirement that has a 2 out of 5 score in the QVscribe Quality Analysis but does follow EARS format.

Original requirement:

Key: Quality Alert Quality Warning

If any component of the system fails (i.e controller, radar, etc.) then the system disengages immediately.



This requirement is high-risk and has critical problems. Specifically, no imperative, an optional open-ended clause, a non-specific temporal word, and a possible issue, a universal quantifier. Looking at this requirement, there are quite a few changes that need to be made to make it well-written and clear.

QVscribe has also provided an EARS conformance analysis, which has determined that this requirement already fits into the **Unwanted Behavior Template**.

An Unwanted Behavior Template can support authoring requirements that cover all undesirable situations. It is good practice to anticipate undesirable situations and author requirements to mitigate them. They are often a major source of omissions in requirements that can lead to costly rework. EARS best practices recommend writing unwanted behavior requirements on the second pass after you have written your normal condition requirements. This allows you to use your normal condition requirements as a reference to see if and how any unwanted situations need to be mitigated.

Let's use the Unwanted Behavior Template to rewrite your high-risk requirement.

IF <trigger>, THEN the <system> <imperative> <system response>.

If you want more information on writing event-driven requirements, please refer to EARS Use Case Part 1.

What is the <trigger>?

Original requirement:

Key: Quality Alert Quality Warning

If any component of the system fails (i.e controller, radar, etc.) then the system disengages immediately.

The trigger in this case is describing the undesired event that is occurring - 'when the components fail'. Within the trigger, there are two problem words flagged, we should work through those issues to ensure we have the right information and details to proceed.

Issue One:

Problem Word	**any**
Issue	**Universal Quantifier**
Issue Type	**Quality Warning -** The presence of Quality Warning issues will not impact the Quality Score of requirements and the document. The proper use of Quality Warning issues depends on the context in which they are used. They can cause scope creep and can distract from the meaning.
Insight/ Recommendations	Reduce ambiguity by ensuring the scope of the requirement is clear. Consider replacing universal quantifiers such as 'all', 'any', and 'no' with the specific entities or values that are being referenced.
Description	A universal quantifier is a reference without boundaries. When used in a requirement, universal quantifiers can make the meaning unclear or expand the scope beyond what is intended.
Possible Exceptions	A universal quantifier can be used in certain situations where a clear boundary is defined, for example, "when all of the following are true: (followed by a finite list)". They can also be used when there are in fact no limits "if no signal is detected". For these reasons, universal quantifiers are categorized as Quality Warnings and do not impact the Quality Score.

In this context, do we really mean 'any' component? Does this mean any component of the vehicle or any component of the Cruise Control System? How do we account for all components? Is the reader expected to list out all of the components and hope they don't miss any?

The problem word 'any' is also working in conjunction with the other problem word 'etc.'

Issue Two:

Problem Word	**etc.**
Issue	**Optional Open-Ended Clauses**
Issue Type	**Quality Alert -** The presence of Quality Alert issues will have a negative impact on the Quality Score of requirements and the document. A Quality Alert will also be present if a requirement lacks an imperative or has multiple imperatives.
Insight/ Recommendations	Enhance requirement completeness by removing the open-ended clause such as 'including' and 'etc', and using one or more requirements to clearly define the scope.
Description	Optional open-ended clauses allude to additional information without defining exactly what is included. This leaves the scope of the requirement up to the interpretation of the reader, which can lead to important items being missed, or unnecessary work being done.
Possible Exceptions	Sometimes an open-ended clause is used to give examples to try to ensure clarity. While this information can be helpful, it is better suited for a note or comment rather than the requirement itself. If the language in the requirement is clear enough to establish the scope, remove the examples or move them to another section. If the scope cannot be definitively established without the examples, be sure to include every item that is required and eliminate the open-ended clause.

Again, how do we account for all components included in 'etc.'? Are there 3 components, 10 components, or 100 components? Will all components be accounted for without additional expense or wasted time? Is the scope of all components clearly understood by the stakeholders?

It is vital to replace the universal quantifier and optional open-ended clause with the specific components that need to be accounted for.

Moreover, based on requirement fundamentals and EARS format, each component should be its own singular requirement. Making each component its own requirements ensures that all requirements are allocated properly, testable, clear, and verifiable.

Here is how we would suggest writing the trigger for this Unwanted Behavior requirement.

If the controller fails, then the <system> <imperative> <system response>.

Each component that needs to be accounted for should be listed as its own requirement. For this requirement, we will focus on the controller and will make the appropriate changes to the other components at the end.

What is the <system>?

Original requirement:

Key: Quality Alert Quality Warning

If any component of the system fails (i.e controller, radar, etc.) then the system disengages immediately.

Edits in progress:

If the controller fails, then the <system> <imperative> <system response>.

In this case, it is fairly simple, the system is the Cruise Control System

If the controller fails, then the Cruise Control system <imperative> <system response>.

What is the <imperative>?

Original requirement:

Key: Quality Alert Quality Warning

If any component of the system fails (i.e controller, radar, etc.) then the system disengages immediately.

Edits in progress:

If the controller fails, then the <system> <imperative> <system response>.

Either the writer has completely omitted the use of the imperative or is trying to use the word **disengages** to describe the action that needs to be taken. A requirement must contain an imperative to assert that action is taken by the subject in the requirement. Let's take a look at the lack of an imperative as a problem type.

Issue Three:

Issue	**No Imperative**
Issue Type	**Quality Alert -** The presence of Quality Alert issues will have a negative impact on the Quality Score of requirements and the document. A Quality Alert will also be present if a requirement lacks an imperative or has multiple imperatives.
Insight/ Recommendations	Enhance requirement completeness by including an acceptable imperative such as “shall”, “must” and “will” in between the entity responsible and the action that is required.
Description	A requirement must contain an imperative to assert that action is taken by the subject in the requirement. Without an imperative, it can be difficult to assert that what is being described is actually required. Using the same imperative consistently is the best way to reduce risk in this area.
Possible Exceptions	Some organizations will employ specific terms to differentiate between types of clauses such as requirements, recommendations, and allowances. Your organization can modify the imperatives list to accommodate any allowable terms it employs.

Is this really a requirement? Without an imperative, how can the reader be sure that what they are reading is actually a requirement? How can the author be sure that each reader will understand that this is required?

In this case, we suggest adding an imperative to this requirement.

If the controller fails, then the then the Cruise Control system shall <system response>.

What is the <system reponse>?

Original requirement:

Key: Quality Alert Quality Warning

If any component of the system fails (i.e controller, radar, etc.) then the system disengages immediately.

Edits in progress:

If the controller fails, then the Cruise Control system shall <system response>.

The system response is the result that the system must produce. In this case, it would be notifying the driver and disengaging. Within the system, there is a problem word flagged, we should work through this issue to ensure we have the right information.

Issue Four:

Problem Word	**immediately**
Issue	**Non-Specific Temporal Word**
Issue Type	**Quality Alert -** The presence of Quality Alert issues will have a negative impact on the Quality Score of requirements and the document. A Quality Alert will also be present if a requirement lacks an imperative or has multiple imperatives.
Insight/ Recommendations	Improve requirement verifiability by using time-specific constraints to indicate when the action needs to be taken. If timing is not in the scope of the requirement, remove phrases such as “after”, “immediately” and “temporarily” to improve clarity.
Description	Non-specific temporal words are expressions that indicate a timeline for when an action should take place but do not give a definite timeline. This leaves the timing open to interpretation and makes the requirement impossible to verify objectively. Often these phrases are also included in functional requirements that are only describing behaviors and not speaking to speed or performance. In these cases removing the phrase will reduce the risk of misinterpretation.
Possible Exceptions	Sometimes teams will want to document a time-based requirement before they’re able to define the exact timing constraint. In these cases, a specific phrase such as TBD can be used. This will make it clear that the requirement is not yet complete and reduces the risk that a weak requirement will be approved.

What exactly does the author mean by ‘immediately’? What if the reader’s definition of immediately in this context is different than the author intended? Is ‘immediately’ in this context 1 second, 5 seconds, half a second, or less? What will be the impact on the project if the reader interprets ‘immediately’ differently? How can it be proven that this requirement has been met?

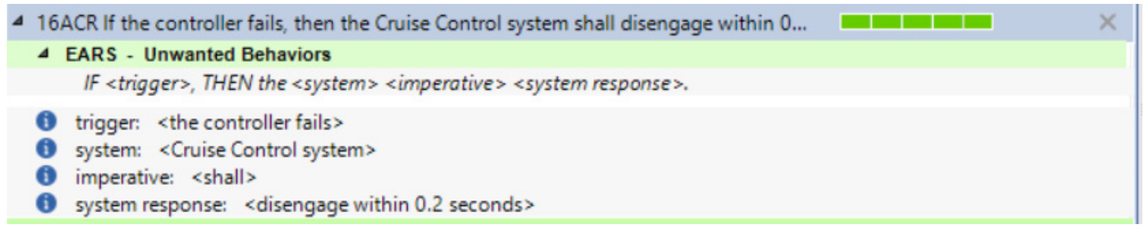
In this case, we suggest providing a clear and specific timing constraint.

Revised Requirement:

> If the controller fails, then the Cruise Control system shall disengage within 0.2 seconds.

Using an Unwanted Behavior template to rewrite our high-risk requirement along with QVscribe recommendations for resolving problem words has made our requirement singular, clear, and verifiable.

Our new score in the QVscribe Quality Analysis is 5 out of 5.



The next step in this requirement would be to write a requirement for each component, using the structure from our controller requirements. Here is an example of what the other component requirement would look like:

- **If the radar fails, then the Cruise Control system shall disengage within 0.2 seconds.**
- **If the x fails, then the Cruise Control system shall disengage within 0.2 seconds.**
- **If the xx fails, then the Cruise Control system shall disengage within 0.2 seconds.**

Ears Templates Make Authoring Complex Requirements Easy

Complex requirements are inevitable. As authors continue writing top-level requirements in more detail, the system definition gets more specific and well-defined. Often at this point, complex requirements become more common. We need to account for a specific set of one or more pre-conditions defining a system state which must exist prior to the event or unwanted behavior for it to trigger the required system response.

Both desired and unwanted system behaviors can be contingent upon a combination of initial conditions and optional features, with or without a trigger event.

However, sitting down to author these types of requirements can be overwhelming. What the result should be is usually very clear to the author, but how do they write it in a way that is clear and easy to understand by the reader?

We recommend EARS Complex Templates. There are 7 complex templates to choose from in QVscribe.

Name	Template
Optional Feature and Event-Driven	Where <feature is included>, when <trigger>, the <system> <imperative> <system response>.
Optional Feature and State-Driven	Where <feature is included>, while <pre-condition(s)>, the <system> <imperative> <system response>.
Optional Feature and Unwanted Behavior	Where <feature is included>, if <trigger>, then the <system> <imperative> <system response>.
Optional Feature, State-Driven, and Event-Driven	Where <feature is included>, while <pre-condition(s)>, when <trigger>, the <system> <imperative> <system response>.
Optional Feature, State-Driven and Unwanted Behavior	Where <feature is included>, while <pre-condition(s)>, if <trigger>, then the <system> <imperative> <system response>.
State Driven and Event-Driven	While <pre-condition(s)>, when <trigger>, the <system> <imperative> <system response>.
State Driven and Unwanted Behavior	While <pre-condition(s)>, if <trigger>, then the <system> <imperative> <system response>.

What makes these templates easy to understand and use is that it puts the elements in the most logical order. We can avoid potential wordiness or possibly omissions by following the template's logical order and flow.

Below we will walk through the order of complex requirements which take into account the logical flow of detail. Please note that a requirement can have multiple optional features or pre-conditions that need to be true - so multiple **Wheres** and **Whiles**. However, you should only account for one trigger event per requirement - so only one **When** or **If/Then.**

1. **Where <feature is included>**: If a requirement only applies when a specific optional feature is present, we should make that clear from the start. Optional feature requirements always start with 'Where <feature is included>', therefore any requirements that mention an optional feature will start with 'Where'.

2. **While <pre-condition(s)>**: Pre-conditions should come next. It must be clear that these pre-conditions must exist for the requirement to be applied before anything can happen. State-driven requirements begin with 'While <pre-condition(s)> and will always follow after an optional feature but before any trigger and the system response.

3. **When <trigger> or If <trigger> then**: A trigger causes something to happen only if any optional features are already listed and/or required pre-conditions are already true. Therefore, the trigger will always follow any optional features and pre-conditions but come before the system response. If the trigger is a desired event, the phrase 'When' will demonstrate that it is an event-driven requirement. If the trigger is an undesired event, the phrases 'If/Then' will demonstrate that is an unwanted behavior requirement.

4. **The <system>:** Next, the system being specified is what must provide the required response (once any required pre-conditions and/or trigger events have been detected). The system name appears in the requirement statement immediately before the imperative but after any pre-conditions and triggers.

5. **The <imperative>:** An imperative verb must be used to connect the subject of the requirement to the desired response. A requirement must contain an imperative to assert that it is mandatory for the subject to take this action.

6. **The <system response>:** Finally, the system response – being the result that the subject must produce – logically comes at the end of the requirement statement. A requirement may include multiple response elements if they are all caused by the same set of features, preconditions, and triggers.

This order of logical flow makes requirements easier to write. It can also help readers to quickly understand requirements. Having a standardized flow and process for how requirements are authored makes it effortless for authors to remember and master the templates.

Remember:

To successfully construct an EARS template requirement, you must completely understand the content and intent of the requirement before you begin authoring. Likely if you can't fit your requirement into an EARS template, you don't fully understand the requirement or have all the information on the subject that you need.

Conclusion

EARS templates can support authors and teams with authoring complex requirements, avoiding costly errors, and re-writing and revising high-risk requirements.

EARS templates are often used as a standardized structure for requirement writing across authors and teams. This makes requirements easier to understand and saves time and money. EARS can also support potential language barriers across organizations and teams by simplifying the language structure. Often, people writing requirements even in their non-primary language can write better requirements and comprehend EARS requirements easier due to their clarity and directness.

Due to the numerous benefits and advantages of using EARS format to support requirement authoring and reviewing, many teams across different industries, organizations, and teams are mandating the use of EARS templates.

This ends Part 2 of our 3 part series on EARS Templates. In the final part of our series on EARS Templates, we will outline how teams and organizations can standardize the requirement writing processes using EARS templates, and what benefits this can have on the success of their processes and projects.

EARS Use Case Series:

Part 1: Write natural language requirements that are clear, concise, unambiguous and testable

Part 2: Rewriting high-risk requirements and using complex EARS templates

Part 3: Learning requirement structure and standardizing requirement writing processes for speed and consistency

QRA

To learn more about QVscribe, visit qracorp.com/qvscribe

qracorp.com