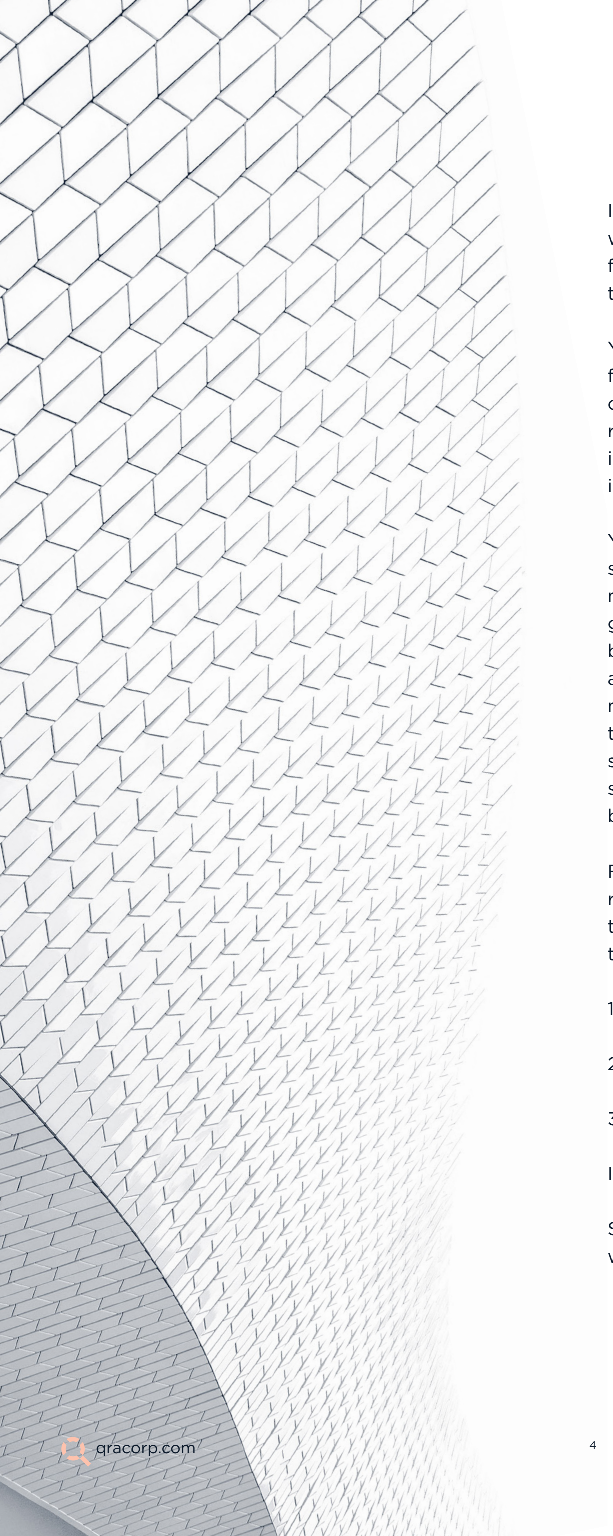
EARS

The Easy Approach to Requirements Syntax: The Definitive Guide

QRA

TABLE OF CONTENTS



Introduction

Imagine you've been chosen to lead a team that will create the system and subsystem specifications for a new product your company has decided to develop.

You've been handed the initial top-level requirements for the system. One glance tells you most of the contributors have never been trained in writing requirements. The product of several iterations of input from numerous stakeholders, the document is wordy, complex and confusing.

You and your team have been tasked with making sense of this document: finding the real requirements, organizing them, refining them, filling in the gaps. Your specifications must be easily understood by both the engineers who will develop the hardware and software for the system, and by the management stakeholders who must approve them. Many of the latter do not have technical backgrounds. And several of the subsystem specs will be sent to suppliers for bids. Thus, all the specifications must be in natural language.

Finally, your management has mandated that all requirements for the project must be clear, concise, testable, traceable and correct. The objectives they've set for you are to:

1. Significantly reduce requirements errors compared to previous projects
2. Lower the percentage of requirements errors that become implementation errors
3. Minimize error correction costs.

In other words, you've really got your work cut out.

So, what do you do? Where do you start? What tools will you use?

A TOOL FOR CLEAR REQUIREMENTS

Well, fortunately, requirements engineer Alistair Mavin and some of his colleagues faced a similar challenge just a few years ago while he was working at Rolls-Royce. Thanks to that challenge, they discovered and developed a methodology you can use to:

- Write clear, concise, unequivocal natural language requirements very easily
- Improve requirements engineering workflows
- Simplify your own life and the lives of those who work from your specifications

This methodology is called the Easy Approach to Requirements Syntax – or 'EARS' for short.

Mavin says EARS is not a template. It's a philosophy – a way of thinking about requirements. He characterizes EARS as a way to "gently constrain" natural language to write better requirements.

WHY EARS IS BECOMING POPULAR IN REQUIREMENTS ENGINEERING

Since the EARS methodology was first presented to the 17th IEEE International Requirements Engineering Conference in 2009, it has been adopted by numerous organizations and included in the requirements engineering (RE) curricula of many universities. Mavin believes this is because the EARS method imposes just a slight constraint on natural language, while providing a simple, logical method for constructing clear, concise, unambiguous requirements.

"People like natural language," says Mavin. "It's their natural way to communicate. It's familiar, easy to use and easy to understand. In contrast, people tend to dislike formalisms. Formalisms complicate things. Each new method is something more to learn, if you're not already familiar with it."

EARS, on the other hand, is easy to learn and apply. As we'll see later, EARS uses a few keywords in ways that are quite familiar. We're already used to using these keywords in the exact same ways in normal speech, as well as in pseudocode and other forms of logical expression. Most people find the EARS syntax constructions intuitively obvious.

Easy
Approach to
Requirements
Syntax

HOW EARS CAME TO BE

Alistair Mavin and his colleagues – though highly experienced in systems and requirements engineering – didn't set out to create a new requirements notation. Instead, the notation emerged from their work serendipitously.

Mavin's team were analyzing requirements from the European Aviation Safety Agency (EASA) Certification Specification for Engines (CS-E). Specifically, they were extracting requirements for a new engine control system. The CS-E has evolved through incremental updates over many years. These repeated updates have resulted in long paragraphs of prose containing a rich mixture of both complex and simple requirements, along with design and verification statements and supporting information. Much of the document is written at an abstract level, relying on lists and explanatory notes to add meaning.

As he and his colleagues were trying to put the extracted requirements in the simplest terms possible, Mavin noticed something interesting. He saw that all the requirements they were writing were falling into a very small number of patterns. Analyzing these patterns, he found their requirements tended to contain a handful of common elements in a common order.

Mavin and his colleagues refined these patterns, reducing them to five, and called them the Easy Approach to Requirements Syntax.

Thus, EARS was born out of engineering practice. This practice was then turned into research, rather than the other way around. Since then, thanks to the widespread enthusiasm for his research, Mavin has become an evangelist for EARS, conducting training sessions at companies around the globe.

Alistair "Mav" Mavin

What you'll learn in the remainder of this Guide

This guide is intended as a primer on EARS. The remainder of this document will cover:

1. Why you should use EARS
2. The EARS process
3. How to implement EARS in your organization
4. Challenges of using EARS
5. Advice from seasoned practitioners
6. Getting started with EARS

WHY USE EARS?

Despite the emergence of a range of formal, graphical and model-based approaches to requirement specification, the vast majority of requirements documents today are still written in natural language (NL). In fact, even when MBSE is used, the initial, high-level specifications for the system are always written in natural language.

Unfortunately, unconstrained natural language requirements can often be vague, ambiguous, overly wordy and confusing. Such requirements can lead to unexpected interpretations, erroneous implementations, costly scrap and rework and – in the worst cases – disaster.

EARS helps solve that problem by bringing just enough rigour to the process of writing requirements in natural language.

In his training courses, Mavin is sometimes asked, "Why use EARS?"

His answer: "Because people like natural language, and they like things easy. Most people don't want to learn a specialized notation for writing requirements. EARS uses natural language, and it's easy."

Besides being easy to use, EARS provides several other basic benefits.

First, EARS makes a big impact on requirements quality for very little overhead. With less than a day of training, most engineers' and analysts' skill in writing requirements improves dramatically. With some follow-on coaching, the techniques are quickly mastered. One training class – and some practice – can transform an organization's RE culture.

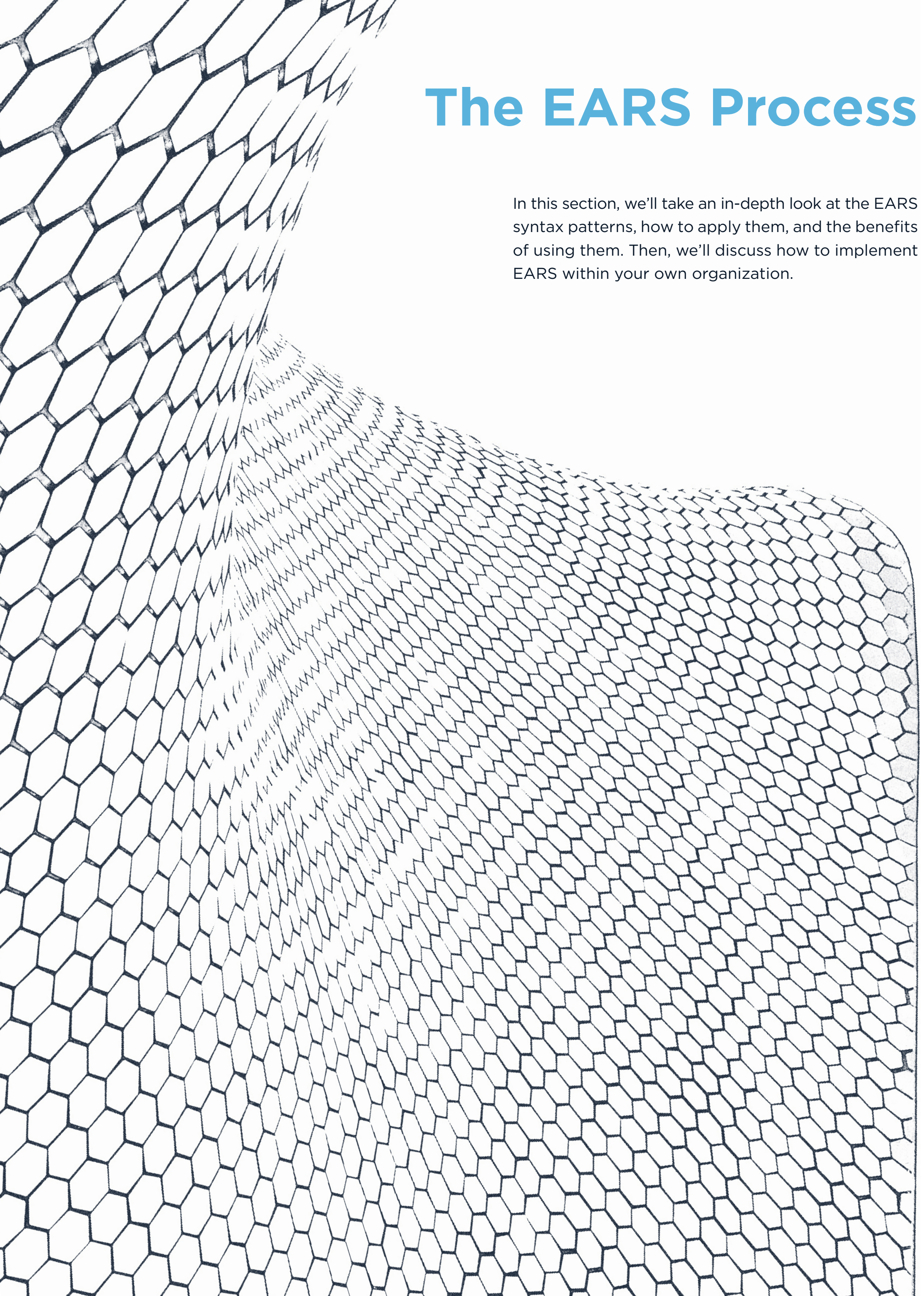
Second, the five, compact EARS syntax patterns – which we'll look at shortly – greatly simplify NL requirements. "In trying to improve something, we often add to it – make it larger," says Mavin. "Rarely do we take the time to remove. It's like that famous quote often attributed to Blaise Pascal: 'I have made this letter longer than usual, only because I have not had time to make it shorter.'" Such was the case of the oft-updated CS-E with which Mavin and his colleagues had to wrestle.

The EARS patterns force us to conform to a simple, efficient format. They tear away extraneous words. EARS virtually eliminates the temptation to add extra information, because the patterns don't allow it. This makes the resulting requirements much clearer and easier to understand, which saves the person reading or implementing the requirement much time and effort.

Finally, the resulting requirements are very similar to each other, even if they are written by different people. "I think the purpose of a written requirement is to get the meaning into the reader's head as quickly as possible," says Alistair Mavin. "If requirements have fewer words in them, and they are similar in form to requirements the reader is already familiar with, they are easier to understand and assimilate."

Some people claim this is a disadvantage of EARS. They say that because all EARS requirements look very much alike, they make it painful to read a lengthy specification. Mavin's reply: "Nobody ever read a requirements document for pleasure. The purpose of the document is not to be an entertaining read. The purpose is that each requirement is correct and states as simply as possible what the system must do.

"Besides that, nobody reads an entire specification in one go, unless they're reviewing it. More often, people will be reading the document a section at a time, trying to understand the requirements for a specific set of capabilities or circumstances."



The EARS Process

In this section, we'll take an in-depth look at the EARS syntax patterns, how to apply them, and the benefits of using them. Then, we'll discuss how to implement EARS within your own organization.

THE EARS PATTERNS

As mentioned, there are five fundamental EARS patterns. Four of these are used to author requirements for normal conditions, i.e., what we want the system to be and to accomplish. The fifth pattern is used for requirements to mitigate unwanted events or user behaviour.

The four types of "normal" requirements are:

1. Ubiquitous requirements
2. State-driven requirements
3. Event-driven requirements
4. Optional feature requirements

The fifth type is simply called:

5. Unwanted behaviour requirements

Ubiquitous Requirements
Ubiquitous requirements are called that because they are always active. They are not invoked by an event or input, nor are they limited to a subset of the system's operating states.

The EARS syntax for ubiquitous requirements is:

The <system name> shall <system response>.

(In the preceding and subsequent syntax definitions, clauses which change from requirement to requirement are indicated within arrow brackets, as shown above.)

So, using the EARS syntax, a ubiquitous requirement will look like the following:

The control system shall prevent engine overspeed.

A word of caution regarding ubiquitous requirements: Always question ubiquitous requirements when writing or reviewing them. What at first seems ubiquitous may be state-driven. Check carefully that the requirement is indeed true in all states in which the system must operate.

State-driven Requirements
State-driven requirements are active throughout the time a defined state remains true.

In EARS, state-driven requirements are identified by the keyword 'WHILE':

WHILE <in a specific state> the <system name> shall <system response>

An example of a state-driven requirement written in EARS syntax is:

While the aircraft is in-flight and the engine is running, the control system shall maintain engine fuel flow above ?? lbs/sec.

Event-driven Requirements
Event-driven requirements require a response only when an event is detected at the system boundary.

Event-driven requirements are identified by the keyword 'WHEN' and have the following syntax:

WHEN <trigger> the <system name> shall <system response>

Here's an example of a state-driven requirement written using EARS syntax:

When continuous ignition is commanded by the aircraft, the control system shall switch on continuous ignition.

Optional Feature Requirements
Requirements that apply only when an optional feature is present as a part of the system are indicated by the keyword, 'WHERE'. Optional feature requirements have the following syntax:

WHERE <feature is included> the <system name> shall <system response>

So, if an engine control system is to have a provision for an optional overspeed protection function, its specification may include a requirement like the following:

Where the control system includes an overspeed protection function, the control system shall test the availability of the overspeed protection function prior to aircraft dispatch.

It's possible, of course, that some optional features will have state-driven or event-driven requirements imposed upon them. We'll look at how to deal with such situations later, under Complex Requirements.

Unwanted Behaviour Requirements
'Unwanted behaviour' is a general term used to cover all situations that are undesirable. Since it is good systems engineering practice to anticipate such undesirable situations and impose requirements to mitigate them, EARS includes a provision for such requirements.

In EARS, unwanted behaviour requirements are indicated by the keyword combination 'IF/THEN'. They obey the following syntax, derived from the event-driven pattern:

IF <trigger>, THEN 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, as in the following example:

If the computed airspeed is unavailable, then the control system shall use modelled airspeed.

Unwanted behaviour is a major source of omissions in requirements, necessitating costly rework. Therefore, it's good RE practice to write unwanted behaviour requirements in a second pass, after you've written your requirements for normal conditions. On this second pass, examine the 'normal operation' requirements you've written to see if any unwanted conditions or inputs need to be mitigated.

The *If/Then* keywords provide a useful cue which tells readers this is a requirement for the system to mitigate some unwanted event.

Complex Requirements
As engineers decompose top-level requirements into more detail, pure event-driven and unwanted behaviour requirements become increasingly rare. Far more often, a specific set of one or more preconditions defining a system state which must exist prior to the event or unwanted behaviour for it to trigger the required system response.

Such complex requirements can be expressed using a combination of the EARS keywords Where, While, When and If/Then using, in general, the following pattern:

While <precondition(s)> when <trigger> the <system name> shall <system response>

Since the set of preconditions define a state in which the system must respond, the state-driven EARS keyword 'While' precedes the list of preconditions. This is followed by either the event-driven keyword 'When' or the unwanted behaviour keyword 'If/then' to identify the trigger event, as in the following example:

While the aircraft is on the ground, when reverse thrust is commanded, the control system shall enable deployment of the thrust reverser.

Using EARS, it is quite easy to see that in the preceding requirement the aircraft being on the ground is a prerequisite (precondition) for the system selecting deploying the thrust reverser, and that the reverse thrust command is the trigger that makes that happen.

Finally, there may be times where an optional feature is applied to a complex requirement. In such a requirement, the keyword 'where' should be used before that precondition, as we'll see next.

Want to learn and practise your EARS writing

This guide is intended as a primer on EARS. The remainder of this document will cover...

1. Why you should use EARS
2. The EARS process
3. How to implement EARS in your organization
4. Challenges of using EARS

What differentiates EARS from other notations, according to Mavin, is that it puts the elements of each requirement statement in the most logical order.

Each of the specific EARS syntax patterns we've just examined conform to either of two generic syntax patterns. The first, for desired behaviour requirements is:

Where <optional feature>, while <precondi-tion(s)>, when <trigger> ***the <system name> shall <system response(s)>***

The second, for unwanted behaviour requirements is:

Where <optional feature>, while <precondi-tion(s)>, ***if <trigger> then the <system name> shall <system response(s)>***

In both generic patterns, the **clauses in bold text** are mandatory, while those in underlined italic text are optional, depending upon whether a specific requirement is contingent upon features, preconditions or triggers.

Thus, except in the case of ubiquitous requirements, the first element is always the **keyword** which identifies the type of requirement. Keywords are an important feature of EARS. As we've seen, each syntax, except the ubiquitous, has a very specific keyword. The EARS keywords make it easy to identify the nature of each requirement.

What's more, the keywords always appear in the same, logical order.

If a requirement only applies when a specific **optional feature** is present, that should be made clear at the outset, so the 'Where' phrase always comes first in such requirements. In principle, there could be requirements that apply only when multiple optional features are present – which would necessitate multiple 'Where' elements – but in practice this is unlikely.

Next come any other **preconditions**, preceded by the keyword 'While'. Multiple preconditions may be prerequisites for activating a specific system response, so an EARS requirement may have multiple 'While' elements. Like the optional feature, these preconditions must exist for the requirement to be applied and before anything can happen. Thus, they appear in the requirement statement before any trigger and before the system response.

A **trigger** causes something to happen only if any required preconditions are already true. Therefore, the trigger should always follow any preconditions, and always precede the system response. A trigger will also be preceded by a keyword that indicates the type of trigger it is: 'When' in the case of a desired event, 'If/Then' in the case of unwanted behaviour.

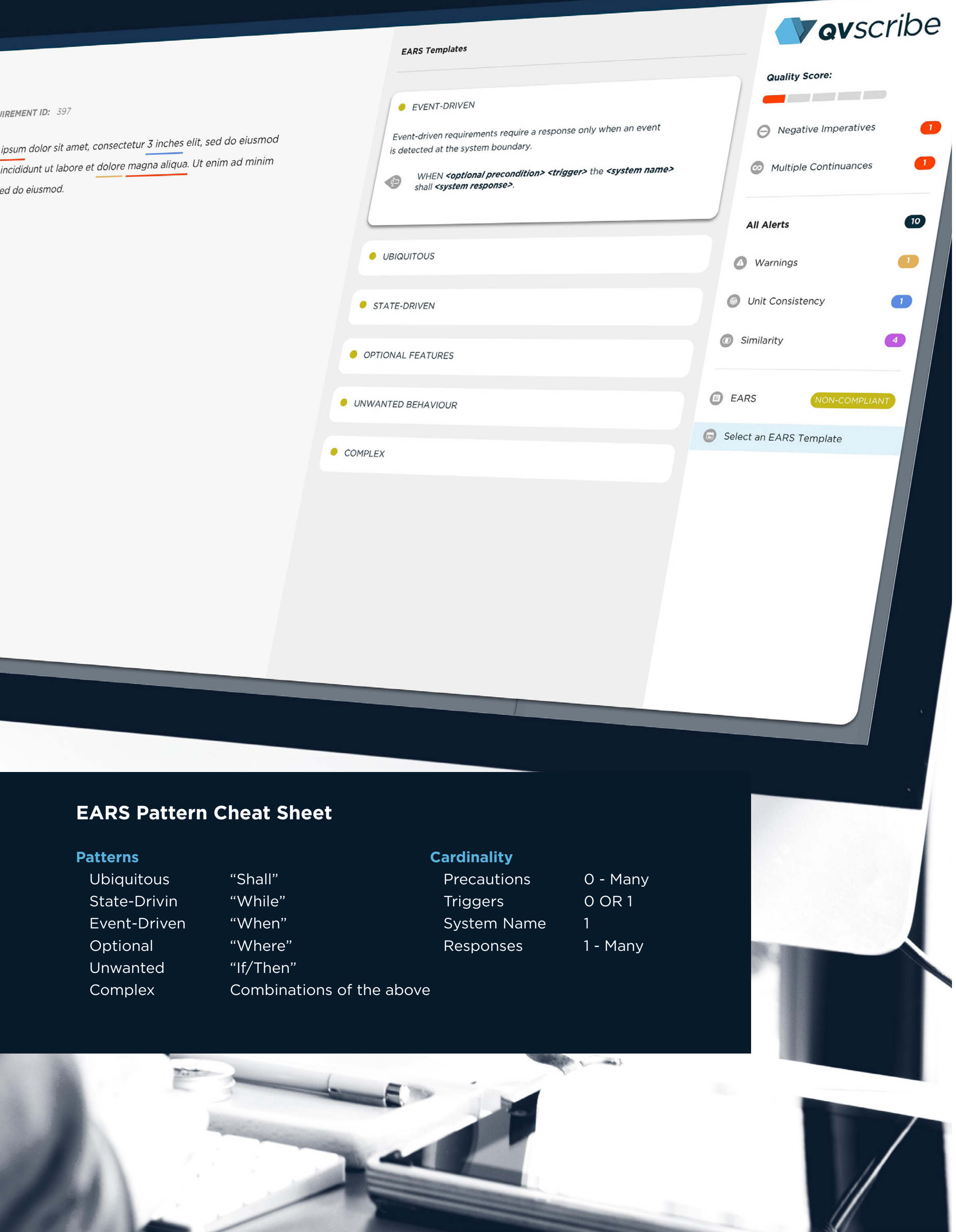
The system being specified is what must provide the required response (once any required preconditions and/or trigger events have been detected). So, the **system name** appears in the requirement statement *immediately before the word* **shall** (or another imperative verb, like must) but after any preconditions and trigger.

Mavin also recommends explicitly naming the system rather than using a generic name, like "system" or "pump". Such vague names can lead to confusion: which "system" or which "pump" must deliver this behaviour? The use of a specific system name is important, because requirements may be copied, pasted, used in multiple documents and sent to multiple suppliers. A specific system name therefore helps to avoid potential confusion, interface difficulties and problems of scope, and ultimately, helps lower program cost.

Finally, the **system response** – being the result of the requirement – 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 feature, preconditions and trigger.

This temporal order makes requirements easier to understand and helps specification users understand those requirements quickly. It helps engineers and analysts remember the EARS patterns and master them more easily.

What's more, the EARS syntax forces you to think carefully about the requirements you write. As many EARS practitioners and trainees have pointed out to Mavin, "If you can't write it in EARS, you probably don't fully understand it." You need to give it more research and thought. To successfully construct an EARS requirement, you need to completely understand the requirement before you write it down.


qvscribe
EARS Templates
IREMENT ID: 397
ipsum dolor sit amet, consectetur 3 inches elit, sed do eiusmod
incididunt ut labore et dolore magna aliqua. Ut enim ad minim
ed do eiusmod.
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>.
UBIQUITOUS
STATE-DRIVEN
OPTIONAL FEATURES
UNWANTED BEHAVIOUR
COMPLEX
Quality Score:
Negative Imperatives 1
Multiple Continuances 1
All Alerts 10
Warnings 1
Unit Consistency 1
Similarity 4
EARS NON-COMPLIANT
Select an EARS Template

EARS Pattern Cheat Sheet

Patterns

Ubiquitous	"Shall"
State-Drivin	"While"
Event-Driven	"When"
Optional	"Where"
Unwanted	"If/Then"
Complex	Combinations of the above

Cardinality

Precautions	0 - Many
Triggers	0 OR 1
System Name	1
Responses	1 - Many

EARS CARDINALITY

It should also be mentioned that the EARS patterns have an inherent cardinality governing their shared elements which must be observed and enforced. That is, the numbers of preconditions, triggers, system names and system responses specified within a given requirement are bounded as follows:

- Preconditions: zero to many
- Triggers: zero or one
- System name: one
- System responses: one to many

In more detail:

- Any number of preconditions may be specified, as necessary, for an action (response).
- No more than one trigger event may be specified; if a trigger is specified, any other specified preconditions must be met prior to the trigger event, or the system shall not provide the specified response.
- If no trigger is specified, the system shall carry out the specified response while all preconditions are met, regardless of the order in which they are satisfied.
- Each requirement must specify a single system name which provides the specified action or actions(system response[s]) under the requirement's prescribed conditions.
- For a single set of preconditions and/or trigger, a requirement must state at least one system response but may state multiple system responses.

So, in theory, it is possible for a requirement to require any number of preconditions for a response and one or any number more responses to a given set of conditions. In practice, however, one should limit the number of preconditions and system responses within a single EARS requirement to two or three at the most. Otherwise, the requirement can become difficult to understand. When preconditions or system responses are present in greater numbers, it is usually best to make use of alternative formats, such as tables or diagrams.

OTHER BENEFITS OF THE EARS FORMAT AND PROCESS

Makes Requirements Easy to Understand
The better people understand what a system's requirements are, the better the chance they have of building a system that meets those requirements. EARS is designed to make requirements easier to understand. That means organizations using EARS will:

- Spend less time trying to understand what requirements mean
- Have a better understanding of what the system they're building must do
- Waste less time and money fixing errors

Simplification of Requirement Structure makes it Easier to Consider Other Issues
EARS only addresses how to write individual requirements correctly. It doesn't directly address any other issues, like coverage or traceability. However, because the EARS syntax is very clear about preconditions and triggers, it helps you understand and define the states your system must support.

For example, if you have a requirement for "while in flight", what must the system do while on ground? If "at or above low idle" is mentioned, what should happen "while below low idle"? If you have a requirement triggered by an input in a given state, what should happen when a different input is received? The simple EARS formats make these issues more visible.

Ensures Requirements are Written in Active Voice
The use of active voice – where the entity performing the action - is the subject of the sentence is a standard best practice of requirements writing.

The widely-used INCOSE Guide for Writing Requirements, for example, warns, "The onus for satisfying the requirement is on the subject, not the object of the sentence. If the entity responsible for the action is not identified explicitly, it is unclear who or what should perform the action, making verification of that requirement very difficult… Often when the phrase "shall be" is used, the statement is in the passive voice."

EARS Writing Example:

Claim:
The software shall begin recording the call.

Claim rewritten with EARS:
When the user selects record, the mobile phone software shall begin a recording of the call screen **and** of the audio from all participants.

Requirements in active voice include the form 'the <system name> shall', as is specified in every EARS pattern. In EARS, there is always an actor performing an action; there is always an explicitly named system that must deliver the system response. Therefore, requirements that are written in EARS will always be in active voice.

Helps Assure Coverage
While EARS doesn't directly address the issue of requirements coverage, it does help requirements authors address not only the use cases directly implied (normal operation), but also those use cases where preconditions are missing, or the system receives undesirable inputs (unwanted behaviour).

Because EARS includes syntax patterns for both normal operation and unwanted behaviour, it encourages the writing of requirements in two passes. In the first pass, you write requirements for normal operation. In the second, you examine each requirement you wrote for normal operations, looking for any unwanted inputs that might impact the system under the same preconditions, and any unwanted behaviour that might result from the absence of a prescribed precondition, such as 'aircraft data unavailable'. You then write unwanted behaviour requirements to mitigate those situations.

This two-pass technique is an important skill that develops with practice. Learning it can be accelerated through training and coaching.

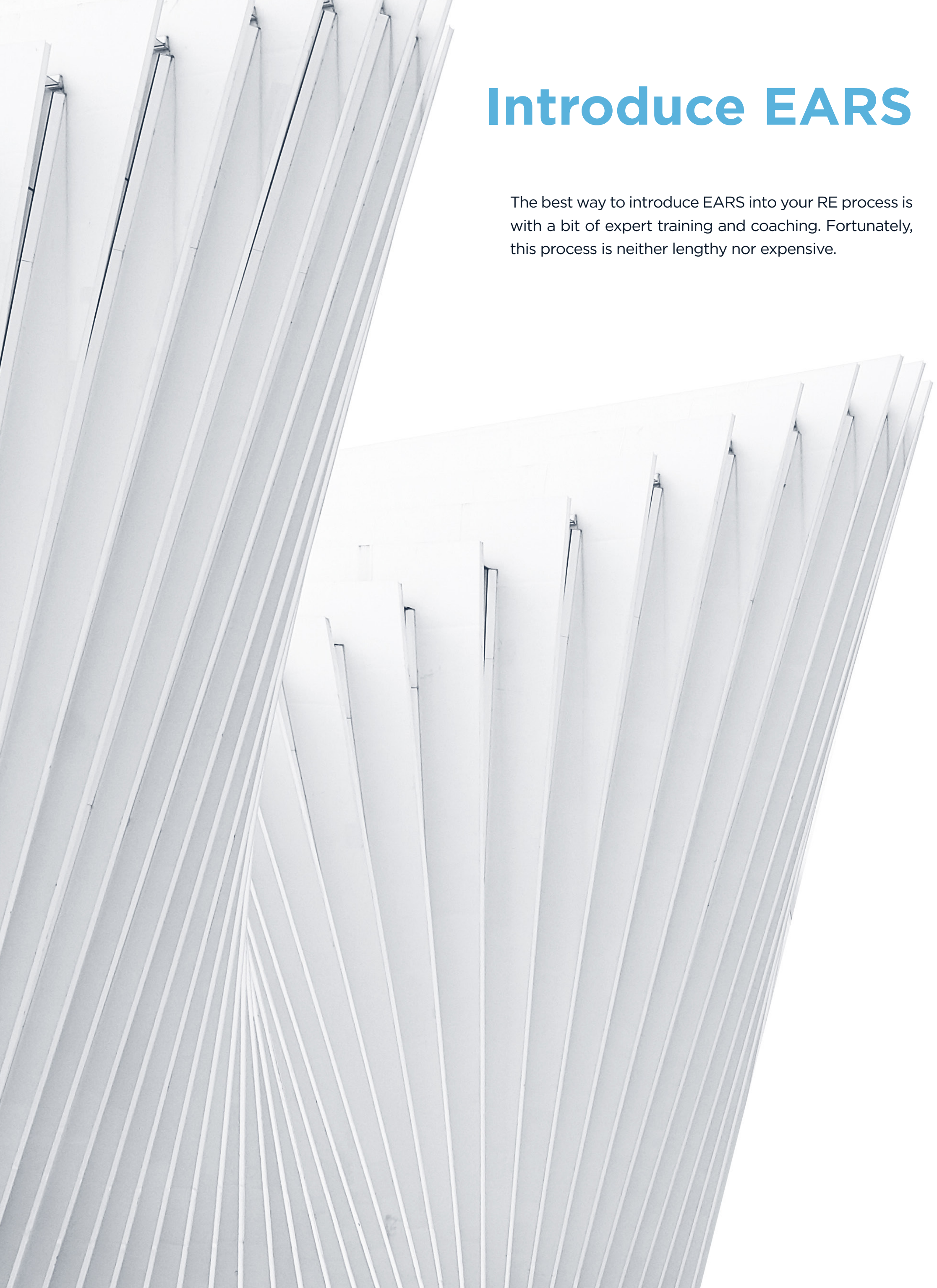
Helps with System Decomposition
Any system that operates as a state machine must be able to determine what operating state it is in. To do that, it must be able to detect the inputs that determine those states.

By forcing you to include only preconditions (states), triggers (inputs) and system responses (outputs) – by stripping away all unnecessary contextual material - EARS helps you make sure all the necessary inputs are available to fully define your preconditions and triggers. In other words, EARS reveals any missing elements that are needed for your requirement to work.

If you can't write a requirement in EARS, key information – a precondition or a trigger – is missing. You need to add that information to fully define your system.

If you can't write a requirement in EARS, key information – a precondition or a trigger – is missing.

You need to add that information to fully define your system.



Introduce EARS

The best way to introduce EARS into your RE process is with a bit of expert training and coaching. Fortunately, this process is neither lengthy nor expensive.

HOW TO IMPLEMENT EARS IN YOUR ORGANIZATION

The best way to introduce EARS into your RE process is with a bit of expert training and coaching. Fortunately, this process is neither lengthy nor expensive.

Ideally, training should begin just before starting the requirements definition phase of a new project. You don't want to train too far in advance of application, as trainees are likely to forget much of what they've learned. Of course, introducing EARS after requirements definition has already begun will result in rework of what has already been done, but many organizations have seen great benefit in rewriting existing requirements using EARS.

Alistair Mavin provides EARS training in half-day or one-day sessions, complemented by some individual coaching and follow-on support.

Mavin's half-day training session is a comprehensive examination of how to apply the EARS syntax patterns through numerous examples and hands-on practice, along with a few systems engineering and requirements engineering principles. Following the training, Mavin provides individual coaching by correspondence. Trainees send Mavin ten requirements they've written and apply his feedback to the next ten requirements they write. Two or three iterations are usually enough for students to gain a firm grasp of the EARS patterns. After that, Mavin can provide follow-on support via email to help trainees deal with any troublesome requirements they encounter.

In his full-day training program, the morning session consists of the same classroom presentation Mavin conducts in his half-day training program. The afternoon is dedicated to direct application, with hands-on coaching, using requirements from the trainees' current project. This leaves delegates with a working knowledge of the EARS approach, having directly applied it in their own domain. Follow-on support can be provided by email, as in the half-day program.

No matter how you introduce EARS into your RE process, you'll want some follow-on coaching. While EARS rapidly improves your requirements writing, it's not something you'll immediately master. Expert coaching will help you...

- Embed the EARS methodology
- Unlock the nuances of the EARS syntax
- Learn to deal with difficult requirements and unusual situations

CHALLENGES OF USING EARS

Objections to Adoption (and how to handle them)
When introducing EARS into their RE environment, some organizations may encounter objections from within. These are among the most frequent obstacles to adoption.

One objection Mavin hears often is that EARS is too simple. Some people seem to think something simple can't be effective. Mavin says one organization he worked with even wanted him to change the 'E' in EARS to 'effective' or 'efficient'. 'Easy', they claimed, made the method sound trivial.

But easy isn't a dirty word. Many great ideas are very simple. Think F = mA, V=IR, or e = mc2, to name just a few. In fact, simplicity is what gives EARS its power. Often – in writing requirements as elsewhere – less is more. Putting requirements in simple, easily recognized formats makes them much easier to understand, which greatly reduces the chances of error and misinterpretation.

Another frequent objection, especially engineers who want to use MBSE, is that natural language can't be sufficiently precise. Some may reason they can't do any real analysis with NL requirements.

Mavin reasons EARS meets people where most of them are today: writing NL requirements without a robust process or specialized tools. Many people who write requirements are uncomfortable with rigorous methods and specialized notations. Most of them are unfamiliar with MBSE tools, which require a considerable investment in both acquisition and training.

EARS adds just a small degree of formality to the RE process – an amount of rigour most people can feel comfortable with. EARS starts people thinking about states and triggers. Plus, it can be used alongside more abstract concepts – like state transition diagrams, activity diagrams and sequence charts, among others – in situations which warrant them.

In fact, EARS actually complements MBSE. "You can't specify a system using only boxes and arrows," says Mavin. "You need words within the boxes! EARS is a bridge from the informal to the formal, but it is also a good way to write the textual aspects of an MBSE specification."

> *"People like natural language. It's their natural way to communicate"*
>
> —Alistair Mavin

EARS represents a relatively painless way to improve your RE process. No special tools and only a small investment in training are necessary to start writing structured NL requirements in EARS syntax. The results will be clear, concise, testable requirements that are easy to read and understand, even for those untrained in EARS.

OBSTACLES ENCOUNTERED WHEN APPLYING EARS

Besides the obstacles to adoption, you may run into a few problems when first applying EARS. The following are the most common.

Confusing Goals with Requirements
In requirements engineering, confusing stakeholder goals with engineering requirements is a common error which can result in unachievable requirements and overdesigned systems. To avoid this pitfall, one must understand the difference between goals and requirements.

Goals, according to Mavin, are things stakeholders want. They're aspirational. They often represent ideal conditions which are unattainable. They may even be emotions. Mavin likens the elicitation of stakeholder goals to a Spice Girls song: "Tell me what you want, what you really, really want…"

A project's stakeholders represent diverse perspectives, both internal and external to an organization. Thus, goals will vary from stakeholder to stakeholder and many will conflict with one another. In fact, it's highly unlikely that all the stakeholder goals for a given project, if any, can be fully achieved.

Goals are also independent of the system to be created. They can be addressed to some degree by any number of solutions, but they are not altered by the chosen solution. Goals remain, regardless of which solution is chosen.

For example, let's say an aircraft manufacturer wants a new jet engine. If the engine builder were to ask the aircraft company what they want, the latter's goals for the new engine might be:

- Weigh 20% less than the previous engine
- Burn 20% less fuel per passenger mile
- Be 100% available

Meanwhile, the engine maker might have an internal goal of reusing as many parts as possible from the previous engine.

"Now, both parties really do want these things," Mavin says. "Each would love to have them if they could, but they both know – barring some game-changing, technological breakthrough – they can't have them. In designing a solution, however, these stakeholder goals must be taken into account. Goals indicate the desired direction. Each point toward an ultimate destination or achievement, albeit an unattainable one."

System requirements, on the other hand, are quite different from goals. System requirements must satisfy all stakeholders and must be agreed. There can be no conflicts between them. They must all be achieved, so it must be possible to verify they have been achieved. In short, requirements must be met.

A common mistake is to try to translate goals directly into requirements. "This is typically done in one of two ways," says Mavin. "First, some people – even some who understand the difference between goals and requirements – will simply shoehorn the word "shall" into a goal and declare it a requirement. They'll write, 'The engine SHALL weigh 20% less than the previous engine,' or 'The engine SHALL burn 20% less fuel per passenger mile than the previous engine,' without first determining what is truly achievable."

"The second way is to water down the goal into something that is achievable. The danger here arises when no one keeps a record of the original goal. It may have been in an email or meeting minutes, but if it never gets put into the requirements database or the specification, it gets lost. Now you have a 'shall' statement, a requirement, but you've lost track of why you wanted to achieve it in the first place. You've fixed something as absolute need, when it may only be a desire of one stakeholder."

Mavin recommends making a clear distinction and separation between goals and requirements. Make a list of all known stakeholder goals, both internal and external, and place it toward the beginning of the requirements document. Put requirements in a separate, subsequent section.

Managing stakeholder expectations is a large part of requirements management. You need to explain to stakeholders why each of their goals can't be fully achieved and how various goals conflict with one another. Then you need to explain how your requirements are going to address their goals, not satisfy them. You can't possibly satisfy all customer goals.

Early False Precision
Early false precision is a problem that can occur when rigour is introduced too early into the system design process, that is, before needed information becomes available. This can happen in MBSE when values need to be inserted in a model before the true values are known. It's easy to lose track of such place holders.

EARS allows you to postpone insertion of values that need to be verified. Don't be afraid to write 'TBD' or 'TBC' in early-stage EARS requirements until the actual values can be confirmed. However, it is good practice to have a documented plan for when and by whom these TBDs and TBCs will be replaced with actual values.

When to and when not to use EARS

✓
1. When writing for audiences with different skill levels
2. If you have less than three preconditions
3. Requirements are to be structured and clear

✕
1. When the requirement extends complexity
2. If you have more than three preconditions
3. When requirements are mathematical formulas

KNOWING WHEN NOT TO USE EARS

Use Notation that's Appropriate for the User
Requirements should be written in the notation that's most appropriate for the user of the specification.

For a software requirements specification, for example, the users are software developers. They're generally comfortable with abstract thinking, pseudo-code, state transition diagrams, etc. For this audience you might not use natural language requirements at all. If there is a more appropriate methodology that captures exactly what system must do and that ALL the document's users understand, use that method.

For a high-level system specification, however, you'll probably need to accommodate a mixed group of users. Some of these will be unfamiliar with specialized notations. In such cases, it's better to use only as much rigour as all users are comfortable with. EARS provides the rigour needed to make requirements clear, concise and testable in a form that's palatable to just about everyone.

Don't Require the Same Notation for Every Requirement
While EARS injects rigour into RE in a way that's accessible to everyone, not every requirement in a predominantly natural language specification should be written in EARS. Some requirements are simply too complicated for the EARS format. A specification may be 95% natural language requirements, but if the other 5% could be more clearly expressed in another format, then it is best to use a different format for such requirements. There is no value in forcing a requirement into text if it is simpler to convey the meaning in another format. It is perfectly reasonable to have some non-textual requirements within a predominantly textual requirements specification.

Alistair Mavin suggests the following rule of thumb when using EARS: If you've got more than three preconditions, consider writing the requirement in a different format. Four pre-conditions make for a very long sentence. A table or some other notation may make the requirement easier to understand.

There are also situations in which you would not use EARS, regardless of the audience. One such situation is where the requirement is best expressed as a mathematical formula. Another is where the requirement is inherently graphical, as in the case of a required flight envelope, for example.

Also note that EARS and modelling are potentially complementary. For example, if you were writing your requirements in EARS, you might use activity diagrams to check that you've got full coverage of your main usage scenarios. You might then choose to include some of those diagrams in your requirements document, depending on how comfortable your audience is with abstract thinking and modelling conventions. If a part of your audience is not comfortable with models, however, it's probably safer to use text.

ENCOURAGEMENT AND ADVICE FROM SEASONED PRACTITIONERS

Dr. Jane Cleland-Huang – Professor of Software Engineering, University of Notre Dame, US

Jane Cleland-Huang used to teach a graduate course in requirements engineering at DePaul University. Students had to do a large project which required them to choose a system to define, identify real stakeholders, and put together a requirement specification for it.

"I hated grading their projects, because the requirements were always such a mess," says Cleland-Huang. "In no way were they unambiguous, and they would have many pieces to them. It was clear that students were having great difficulty defining clear requirements."

Then, Cleland-Huang found EARS. The next year, instead of teaching the guidelines she had been teaching, she decided to teach the EARS method. "The quality of the requirements produced by each student, even those whose first language was not English, improved dramatically," she says.

In end-of-term course appraisals, students frequently mentioned that learning to use EARS was one of the things they appreciated most about the class. "Many students really liked the EARS method," says Cleland-Huang. "Several took it back into their work environments and began applying it on real projects. They found it a lot easier to write requirements that way."

Cleland-Huang cautions against going it alone. "EARS is easy to use, but there are certain ways in which you should use it," she warns. "It's very easy to get a huge improvement in the way you write requirements, but there are some subtleties that are easy to get wrong. So, training is important."

"It's easy to get started with some basic training. Put EARS to use," she recommends. "Then get some follow-up training, or coaching, where you analyze and evaluate your requirements with a qualified instructor and look at ways you could improve them.

"There's a very intuitive first step, but in looking back at requirements you've written and getting some expert feedback, you'll often see better ways in which you could have written the same requirements," concludes Cleland-Huang. She says she found ways she could improve her own requirements while leading an EARS workshop with Alistair Mavin.

Philip Wilkinson – Systems Engineer, Rolls-Royce, UK

Philip Wilkinson has co-authored four papers on EARS with Alistair Mavin. Coming from a safety background, his first interest in requirements stemmed from needing to know what safety requirements were being imposed on design engineers. Soon, he became interested in the general issue of requirements writing.

"I saw there was more to it than meets the eye; writing requirements isn't easy," says Wilkinson. "I wanted to know why a safety requirement I had written – one that was clear to me – might not be clear to someone else who had to implement it."

When asked what he likes most about EARS, Wilkinson cited three things.

First, it's portable. EARS can be used at various levels of requirements writing. No tools are needed other than pencil and paper (or a document processing program).

Second, it's flexible. The small number of simple patterns makes EARS easy to teach to business analysts to help them write clearer high-level business requirements. Meanwhile, engineers can use those same EARS patterns – either by themselves or in conjunction with a variety of other notations – to craft detailed engineering requirements.

Third, very little training is needed – although practice is essential and expert feedback is extremely beneficial, he adds.

Wilkinson recommends getting started with a simple course that introduces the importance of requirements, as well as the EARS method. It's also helpful to have a crib sheet on hand – like the one Mavin hands out in his training courses – when first trying to apply EARS.

Finally, students should discuss their first few EARS requirements with an EARS expert. "You need someone available on-site for a couple of days, and some on-going support afterward to help you make sure you really understand what you're doing," says Wilkinson. "The intuitive nature of the EARS patterns often gets people into trouble if they try to work with them strictly on their own, without expert feedback."

Eero Uusitalo – Partner and Consulting Engineer, Intoworks, Finland

As an engineering consultant, Eero Uusitalo helps his clients meet established quality standards. Besides reviewing their requirements documentation, he is also involved in coaching his customers' personnel, so they can achieve a high level of quality from the very beginning of a project.

Uusitalo has used EARS in situations ranging from helping utility companies develop multi-billion-dollar power plants to restating regulatory requirements so they could be clearly understood. Uusitalo's favourite thing about EARS is that it can be presented to a variety of stakeholders, and they'll all understand the resulting requirements without any training.

"In the context of designing a power plant, for example, there is a huge variety of stakeholders from several organizations. You need something that's readily understandable," says Uusitalo. "You can present EARS easily and show clear upside immediately. It's the easiest method I've encountered for truly improving your requirements documentation."

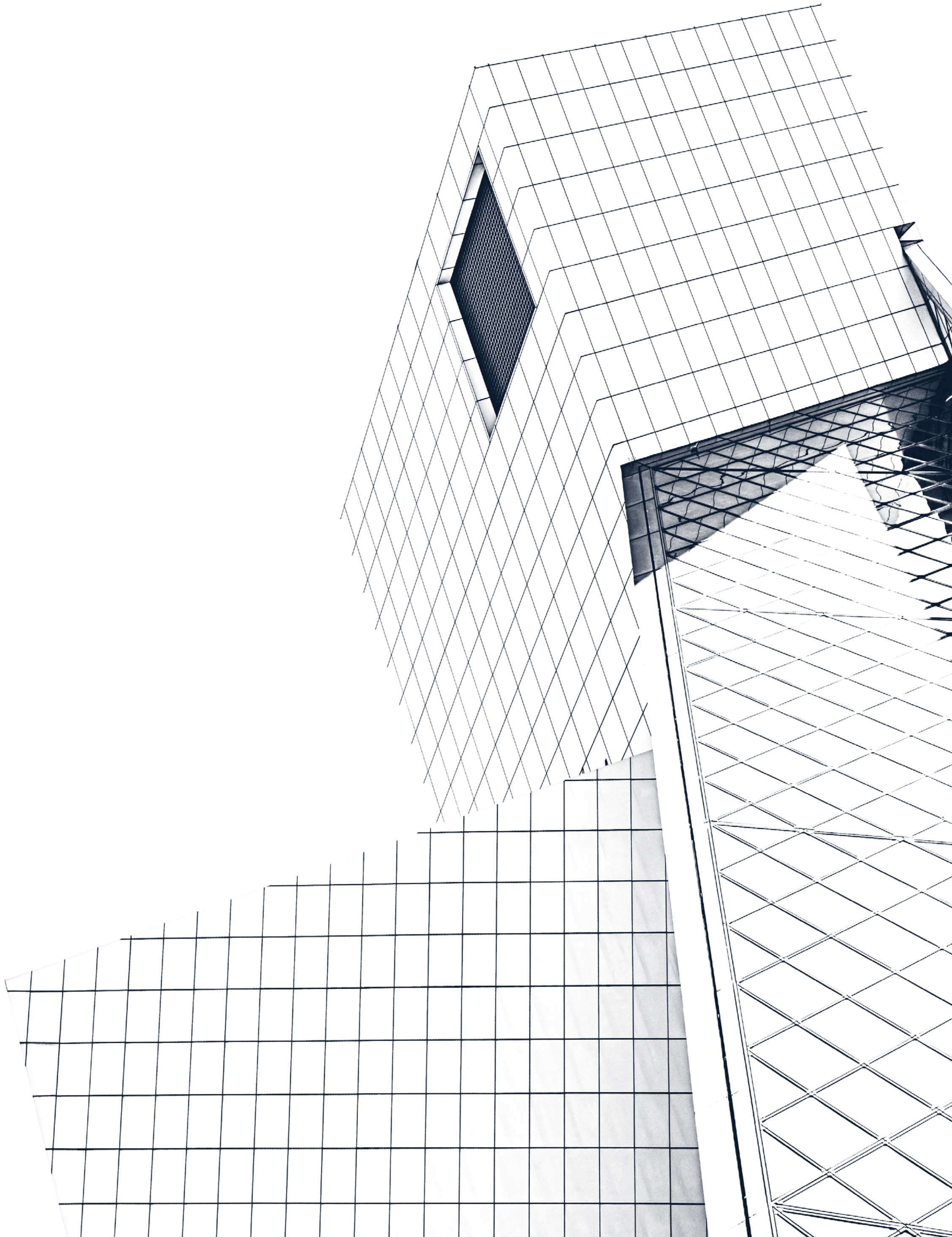
For team leaders considering EARS, Uusitalo offers several tips.

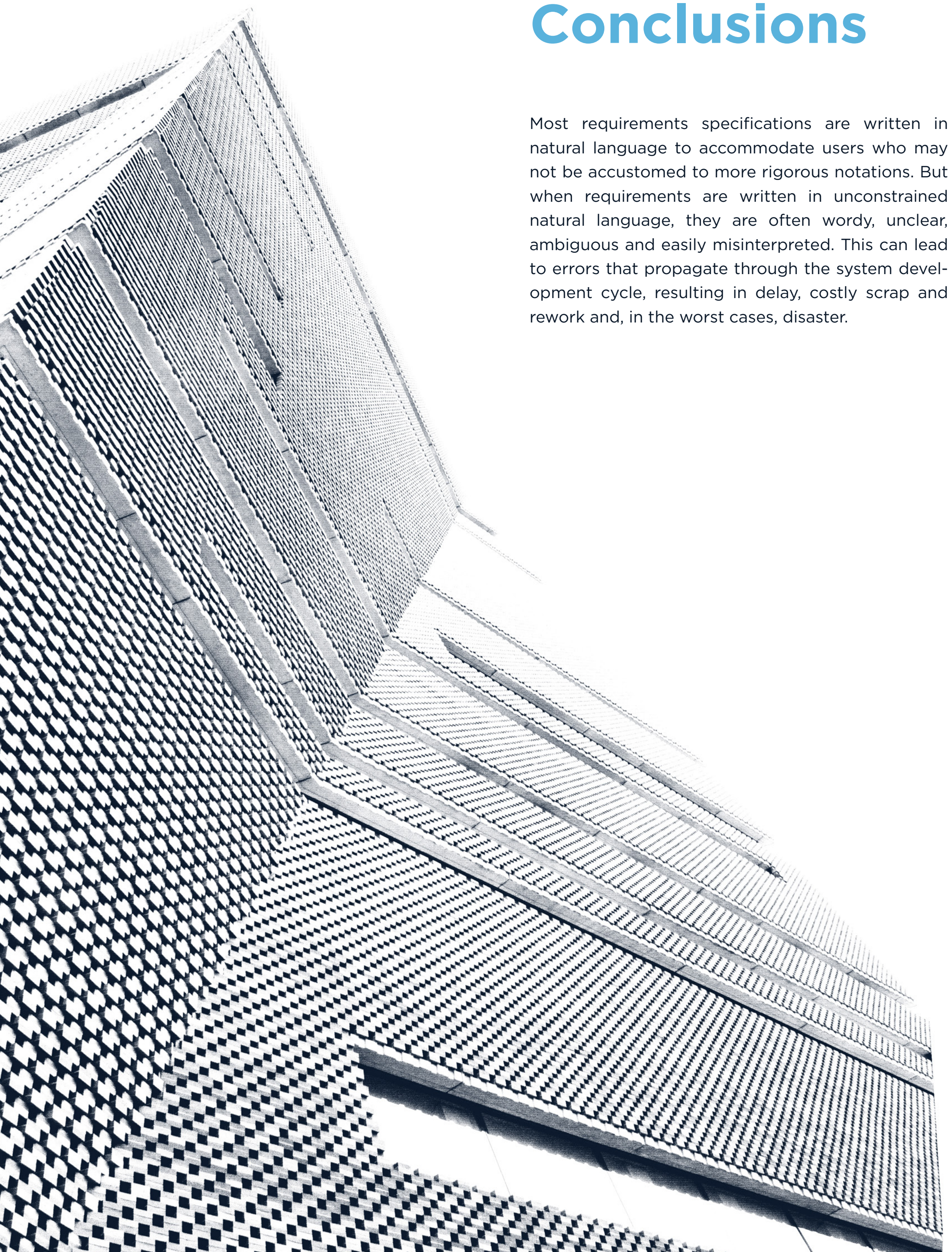
"First, it's important to remember that not all kinds of requirements are suited to expression in EARS," he says. "Requirements of some user-centric software applications might better be expressed using use cases, for example. Understand the context where you're applying EARS, and don't try to use it where it doesn't fit."

He also cautions that no method is a silver bullet. "EARS is a method that's easy to approach, but even though it's easy, it's not trivial. There is a learning curve involved. Progress along that curve can be accelerated with training and coaching. If you find that the premise is useful, but you're having some difficulties in applying it, it's best to seek expert help," he advises.

Next, Uusitalo reminds us, it's necessary to iterate and improve our use of the method. "You can't expect that simply applying the EARS patterns will magically make your requirements great. Writing good requirements with EARS requires that you do a thorough, up-front analysis of your system. In so doing, you will also begin to understand the strengths and weaknesses of the EARS method. That said, using EARS is much better than relying on unstructured natural language."

Finally, Uusitalo recommends cultivating an in-house expert or advocacy group. "As with any new tool or method, it's always good to have some champion within your organization who really understands the method and who can promote its use within the organization," he says.





Conclusions

Most requirements specifications are written in natural language to accommodate users who may not be accustomed to more rigorous notations. But when requirements are written in unconstrained natural language, they are often wordy, unclear, ambiguous and easily misinterpreted. This can lead to errors that propagate through the system development cycle, resulting in delay, costly scrap and rework and, in the worst cases, disaster.

IN SUMMARY

EARS – the Easy Approach to Requirements Syntax – helps engineers and business analysts write natural language requirements that are clear, concise, unambiguous and testable. EARS "gently constrains" natural language in ways that are familiar and comfortable to everyone. Plus, it requires no tools and only a small amount of training, so barriers to adoption are minimal.

Finally, EARS offers numerous benefits, including:

- Greatly improves requirements quality for very little overhead
- Makes requirements clearer, simpler, easier to understand
- Easy to learn
- Standardizes form of requirements across the organization with minimal effort
- Eliminates or greatly reduces most common types of requirements errors
- Helps assure requirements coverage of both normal and unwanted behaviour
- Assists system decomposition
- Reduces word count of requirements and specifications
- Injects rigour into the RE process in a palatable form
- Doesn't require a specialist tool or notation
- Good for people whose first language is not English

EARS Pro Tips

Remember the patterns:
- Ubiquitous requirements
- State-driven requirements
- Event-driven requirements
- Optional feature requirements

  And
- Unwanted behaviour requirements

Remember the cardinality:
- Preconditions: zero to many
- Triggers: zero or one
- System name: one
- System responses: one to many

Remember when NOT to use EARS:
- When the requirement extends complexity
- If you have more than three preconditions
- When requirements are mathematical formulas

Get Training and Coaching
- Visit **www.alistairmavin.com**

Author with QVscribe
- Visit **qracorp.com/qvscribe**

ABOUT ALISTAIR MAVIN

Alistair Mavin has been practicing requirements engineering for over 20 years. A chartered engineer, he is a member of IEEE and the British Computer Society Requirements Engineering Specialist Group committee.

Mavin is the lead author of the EARS notation. Of his 24 published papers, eight have been on EARS. He has provided training in requirements engineering and EARS to numerous companies, presented guest lectures on those topics at many universities, and delivered EARS tutorials at more than ten international conferences, including IEEE RE, REFSQ, INCOSE and Sophist. Mavin has worked with clients including BAE Systems, Bombardier, Daimler, Honeywell and Rolls-Royce. His EARS method is also used at Bosch, EADS (Airbus), Intel, Lockheed Martin and Siemens and many other companies, as well as being taught at several universities.

As an independent consultant, Mavin offers full-day and half-day training sessions in EARS, EARS+, and requirements engineering best practices. He also provides coaching and follow-on support to help trainees and organizations embed their learning and deal with obstacles.

For inquiries regarding his services, Mavin can be reached at mav@alistairmavin.com. Further details on his services can be found on his website at: **www.alistairmavin.com.**

ABOUT QRA CORP

QRA Corp's mission is to accelerate the design process across industries who are tackling the most complex systems by empowering them to build tomorrow's safe, secure, and incredibly powerful products. QRA's technology, patented toolsets and capabilities have been used to avoid stressful reworks, enable confident engineering, and find previously undetected catastrophic flaws.

QRA's requirements analysis tool, QVscribe, harnesses Natural Language Processing to automatically apply the best requirements analysis tactics by leading industry experts. Automated requirements analysis empowers engineering teams to build faster by identifying errors where they matter most - in the requirements. QVscribe and EARS complement each other in helping organizations craft clear, unambiguous requirements.

To discover how QVscribe can help your organization improve and accelerate its requirements definition and analysis processes, click here to schedule an online demonstration.

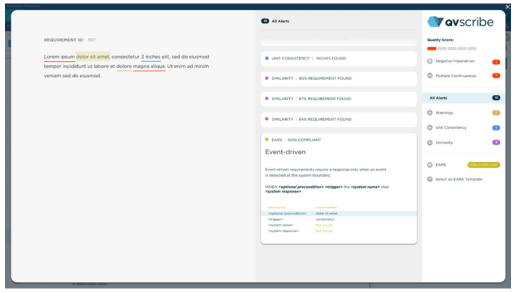
To learn more about QRA and find additional helpful resources for improving your requirements and your RE processes, visit **qracorp.com**

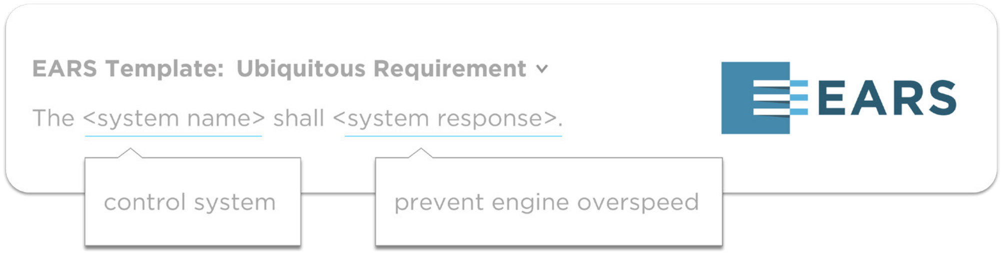
ABOUT QVSCRIBE

Jumpstart the authoring of your requirements with fill-in-the-blank templates that are pre-configured to follow the Easy Approach to Requirements Syntax (EARS).

Whether your requirement is ubiquitous, state-driven, event-driven, an optional feature, or an unwanted behaviour – you can choose amongst a subset of patterns that structure your requirement to be simple, clear, and a dream to work with.

Learn why you should be using EARS and how to get started right now! Visit **qracorp.com/qvscribe**




EARS Template: Ubiquitous Requirement
The <system name> shall <system response>.
control system
prevent engine overspeed
EARS

REFERENCES

Mavin, A., Wilkinson, P., Harwood, A.R.G. and Novak, M.: Approach to Requirements Syntax (EARS), 17th IEEE International Requirements Engineering Conference, Atlanta, GA, USA, 2009.

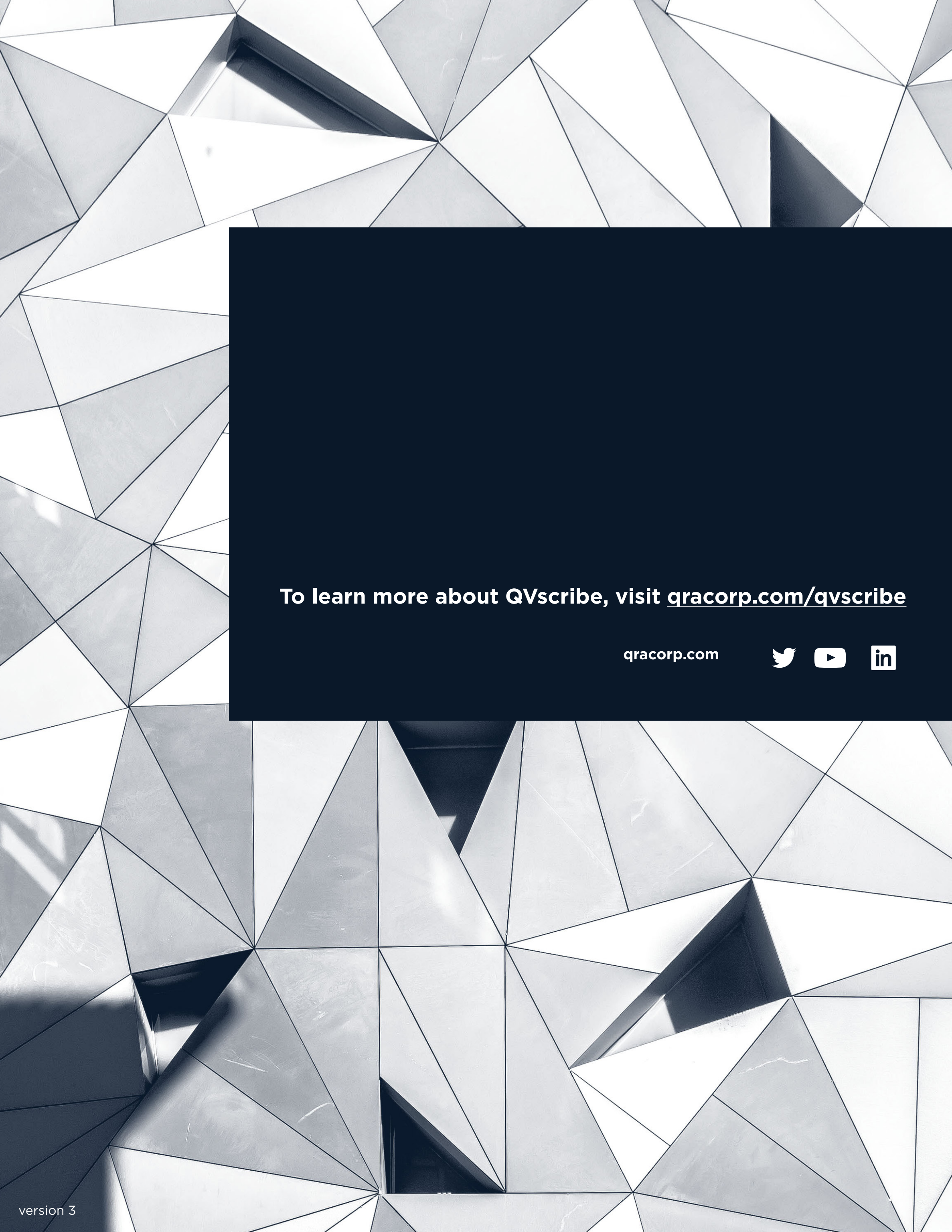
Mich, L., Franch, M., Novi, I.P.: Market research for requirements analysis using linguistic tools. Requirements Engineering, Vol. 9, pp. 4056, 2004.

INCOSE Requirements Working Group, Guide for Writing Requirements (Version 2.1), INCOSE, June 2017.

Mich, L., Franch, M., Novi, I.P.: Market research for requirements analysis using linguistic tools. Requirements Engineering, Vol. 9, pp. 4056, 2004.

Mavin, A., Wilkinson, P., BIG EARS: The Return of the Easy Approach to Requirements Syntax, 18th IEEE International Requirements Engineering Conference, Sydney, NSW, Australia, 2010.

Ibid.



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

qracorp.com

version 3