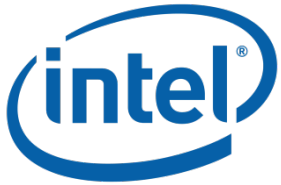


EARS at Intel

John Terzakis

Intel Corporation

john.terzakis@intel.com

March 1, 2016

REConf 2016

Munich, Germany

Version 1.0

Legal Disclaimers

Intel Trademark Notice:

Intel and the Intel Logo are trademarks of Intel Corporation in the U.S. and other countries.

Non-Intel Trademark Notice:

*Other names and brands may be claimed as the property of others

Agenda

- Requirements & their issues
- Overcoming issues with requirements
- Identifying ubiquitous requirements
- EARS and EARS examples
- Using EARS to rewrite requirement examples
- EARS at Intel
- Wrap up

Requirements & Their Issues

What is a Requirement?

A **requirement** is a statement of one of the following:

1. What a system must do
2. A known limitation or constraint on resources or design
3. How well the system must do what it does

The first definition is for **Functional Requirements**

The second and third definitions are for **Non-Functional Requirements (NFRs)**

This session will focus on improving requirements defined by 1 & 2

Issues with Requirements

- Authors often lack formal training on writing requirements
- Authors write requirements using **unconstrained** natural language:
  - Introduces ambiguity, vagueness and subjectivity
  - Not always clear, concise and coherent
  - Often not testable
  - Logic is not always complete ("if" but no "else")
  - Some times missing states, triggers or error conditions
- Authors "copy and paste" poor requirements, which multiplies the number of requirements with defects

Examples of Issues with Requirements

1. The software shall support a water level sensor.
   - What does the word “support” mean?

2. The thesaurus software shall display multiple alternatives for the requested word.
   - How many is “multiple”? Three? Four? Six? Ten? More?
   - Under what conditions are they displayed ?

3. If a boot disk is detected in the system, the software shall boot from it.
   - What if a boot disk is not present? The logic is incomplete.

4. The software shall blink the LED on the adapter using a 50% on, 50% off duty cycle.
   - Does the software blink the LED at all times? Or is there a trigger that initiates the blinking?

Overcoming Issues with Requirements

Tools to Overcome Requirements Issues

- Training
- Check requirements against “goodness” criteria
- Use a consistent syntax
- Use a constrained natural language (e.g., Planguage)

See backup for more information

- Express requirements using EARS

Express Requirements Using EARS

EARS: Easy Approach to Requirements Syntax

- Is an effective method of expressing requirements
- Was created by Alistair Mavin and others from Rolls-Royce PLC and presented at the 2009 Requirements Engineering (RE 09) conference[1]
- Differentiates between five types (or patterns) of requirements:
  - Ubiquitous (always occurring)
  - Event-driven
  - Unwanted behaviors
  - State-driven
  - Optional features

[1] *EARS (Easy Approach to Requirements Syntax),* Alistair Mavin et al, 17th IEEE International Requirements Engineering Conference (RE 09),

EARS Background

- Case study involved Rolls-Royce group working on aircraft engine control systems
- Software was safety critical, contained thousands of components and involved up to twenty different suppliers
- Rolls-Royce identified 8 major problems with existing natural language requirements:
  - Ambiguity
  - Vagueness
  - Complexity
  - Omission
  - Duplication
  - Wordiness
  - Inappropriate implementation
  - Untestability
- Rewriting requirements using EARS "…demonstrated a significant reduction in all eight problem types…" *

(From: EARS (Easy Approach to Requirements Syntax),* Alistair Mavin et al, 17th IEEE International Requirements Engineering Conference (RE 09), page 321)

Identifying Ubiquitous Requirements

Ubiquitous Requirements

Ubiquitous Requirements:

- State a fundamental system property
- Do not require a stimulus in order to execute
- Are universal (exist at all times)

Requirements that are not ubiquitous do not occur at all times. They

- Require an event or trigger in order to execute, or
- Denote a state of the system, or
- Denote an optional feature

Most requirements are not ubiquitous

About Ubiquitous Requirements

Question ubiquitous requirements: Things that may *seem* universal are often subject to unstated triggers or preconditions

Most legitimate ubiquitous requirements state a fundamental property of the software:

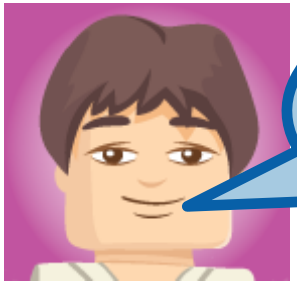
- The software shall be distributed on CD-ROM and DVD media.
- The software shall prevent Unauthorized Access to patient data.

Software functions that appear ubiquitous are often not:

- The software shall wake the PC from standby
- The software shall log the date, time and username of failed logins.

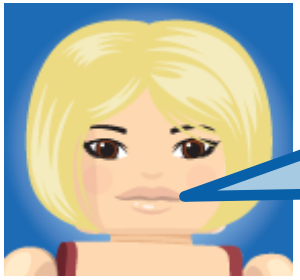
The last 2 requirements are missing a trigger

Ubiquitous Requirements or Not?



The installer SW shall be available in Greek

The SW shall display a count of the number of participants.



The SW shall phone the Alarm Company

The SW shall mute the microphone

The SW shall download the book without charge

The SW shall warn the user of low battery

Which requirements are ubiquitous?

Ubiquitous or Not?

1. The installer software shall be available in Greek.
   - Yes. This is a fundamental property of the installer software
2. The software shall display a count of the number of participants.
   - No. There is likely an event that causes the count to be displayed.
3. The software shall phone the Alarm Company.
   - No. We don't want the software phoning the Alarm Company unless there is a fault or problem.
4. The software shall mute the microphone.
   - No. The muting occurs during some state the system is in
5. The software shall download the book without charge.
   - No. There is apt to be an optional condition under which books are downloaded for free.
6. The software shall warn the user of low battery
   - No. There is a state or an event needed for this warning to occur.

EARS and EARS Examples

EARS Patterns

Pattern Name	Pattern
Ubiquitous	The <system name> shall <system response>
Event-Driven	WHEN <trigger> <optional precondition> the <system name> shall <system response>
Unwanted Behavior	IF <unwanted condition or event>, THEN the <system name> shall <system response>
State-Driven	WHILE <system state>, the <system name> shall <system response>
Optional Feature	WHERE <feature is included>, the <system name> shall <system response>
Complex	(combinations of the above patterns)

Ubiquitous Requirements

- Define a fundamental property of the system
- Have no preconditions or trigger
- Do not require a pattern keyword
- Have the format:

  The <system name> shall <system response>

Ubiquitous Examples

- The software package shall include an installer.
- The software shall be written in Java.
- The software shall be available for purchase on the company web site and in retail stores.

Event-Driven Requirements

- Are initiated *when and only when* a trigger occurs or is detected
- Use the “When” keyword
- Have the format:

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

Event-Driven Examples

- When an Unregistered Device is plugged into a USB port, the OS shall attempt to locate and load the driver for the device.

- When a DVD is inserted into the DVD player, the OS shall spin up the optical drive.

- When the water level falls below the Low Water Threshold, the software shall open the water valve to fill the tank to the High Water Threshold.

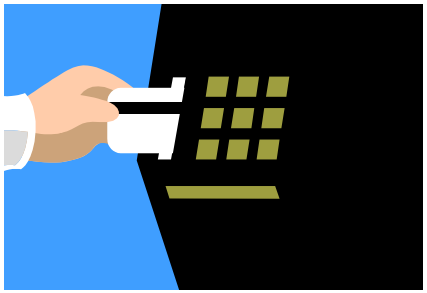
Unwanted Behavior Requirements

- Handle unwanted behaviors including error conditions, failures, faults, disturbances and other undesired events
- Use the “If” and “then” keywords
- Have the format:

  IF <unwanted condition or event>, THEN the <system name> shall <system response>

Unwanted Behavior Examples

- If the measured and calculated speeds vary by more than 10%, then the software shall use the measured speed.
- If the memory checksum is invalid, then the software shall display an error message.
- If the ATM card inserted is reported lost or stolen, then software shall confiscate the card.



State-Driven Requirements

- Are triggered while the system is in a specific state
- Use the “While” keyword (or optionally the keyword “During”)
- Have the format:

  WHILE <system state>, the <system name> shall <system response>

State-Driven Examples

- While in Low Power Mode, the software shall keep the display brightness at the Minimum Level.

- While the heater is on, the software shall close the water intake valve.

- While the autopilot is engaged, the software shall display a visual indication to the pilot.

Optional Feature Requirements

- Are invoked *only* in systems that include the particular optional feature

- Use the “Where” keyword

- Have the format:

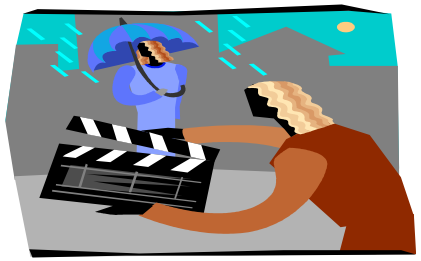
  WHERE <feature is included>, the <system name> shall <system response>

Optional Feature Examples

- Where a thesaurus is part of the software package, the installer shall prompt the user before installing the thesaurus.

- Where hardware encryption is installed, the software shall encrypt data using the hardware instead of using a software algorithm.

- Where a HDMI port is present, the software shall allow the user to select HD content for viewing.



Complex Requirements

- Describe complex conditional events involving multiple triggers, states and/or optional features
- Use a combination of the keywords When, If/Then, While and Where
- Have the format:
    - <Multiple Conditions>, the <system name> shall <system response>

Complex Examples

- When the landing gear button is depressed once, if the software detects that the landing gear does not lock into position, then the software shall sound an alarm.

- Where a second optical drive is installed, when the user selects to copy disks, the software shall display an option to copy directly from one optical drive to the other optical drive.

- While in start up mode, when the software detects an external flash card, the software shall use the external flash card to store photos.

Using EARS to Rewrite Requirement Examples

Example 1

Original:

The installer software shall be available in Greek.

Rewritten using EARS (no change):

The installer software shall be available in Greek.

What type of EARS Pattern? Ubiquitous

Example 2

Original:

The software shall display a count of the number of participants.

Rewritten using EARS:

When the user selects the caller count from the menu, the software shall display a count of the number of participants in the audio call in the UI.

What type of EARS Pattern? Event Driven

Example 3

Original:

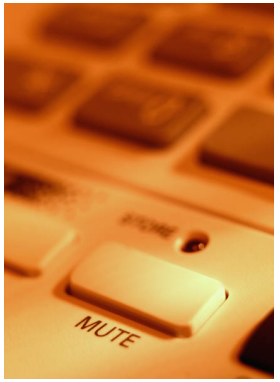
The software shall phone the Alarm Company.

Rewritten using EARS:

If the alarm software detects that a sensor has malfunctioned, then the alarm software shall phone the Alarm Company to report the malfunction.

What type of EARS Pattern? Unwanted behavior

Example 4



Original:

The software shall mute the microphone.

Rewritten using EARS:

While the mute button is depressed, the software shall mute the microphone.

What type of EARS Pattern? State-driven

Example 5

Original:

The software shall download the book without charge.

Rewritten using EARS:

Where the book is available in digital format, the software shall allow the user to download the book without charge for a trial period of 3 days.

What type of EARS Pattern? Optional feature

Example 6

Original:

The software shall warn the user of low battery.

Rewritten using EARS:

While on battery power, if the battery charge falls below 10% remaining, then the system shall display a warning message to switch to AC power.

What type of EARS Pattern? Complex

EARS at Intel

History of EARS at Intel

- EARS was introduced at Intel in 2010
- Although originally developed for the aircraft industry, it was easily adapted to a wide variety of projects at Intel
- Integrated into existing Requirements Engineering training materials
- Rapidly adopted by requirements authors due to its power and simplicity
- Praised by developers & validation teams for providing clarity and removing ambiguity from requirements.

EARS Requirements Examples from Intel

- The software shall include an online help file.
  - Type: Ubiquitous

- When the software detects the J7 jumper is shorted, it shall clear all stored user names and passwords.
  - Type: Event-driven

- If the software detects an invalid DRAM memory configuration, then it shall abort the test and report the error.
  - Type: Unwanted behavior

Note: Requirements slightly modified from original form

EARS Requirements Examples from Intel

- While in Manufacturing Mode, the software shall boot without user intervention.
  - Type: State-driven
- Where both 3G and Wi-Fi radios are available, the software shall prioritize Wi-Fi connections above 3G.
  - Type: Optional feature
- While on DC power, if the software detects an error, then the software shall cache the error message instead of writing the error message to disk.
  - Type: Complex

Note: Requirements slightly modified from original form

Wrap up

Session Summary

In this session we have:

- Provided a brief overview of requirements
- Discussed issues with requirements and listed tools to overcome them
- Taught how to identify ubiquitous requirements vs. those that are not
- Introduced the concept of EARS
- Reviewed examples using the EARS template
- Rewrote requirements using the EARS template
- Observed practical applications of EARS at Intel

Final Thoughts

- EARS is a structured aid to writing better requirements

  Focuses on the different patterns for requirements using keywords (When, If-Then, While, Where and combinations)

- EARS helps to identify which requirements are truly ubiquitous

  Some requirements, written as if they were ubiquitous, are really not

- EARS is beneficial to both developers and testers in understanding the intent of requirements

  Removes ambiguity, improves clarity and properly identifies underlying conditions or triggers

EARS is *Powerful*. Start Using it Today!

Contact Information

Thank You!

For more information, please contact:

John Terzakis

john.terzakis@intel.com

Backup

Papers on EARS

EARS (Easy Approach to Requirements Syntax), Alistair Mavin et al, 17th IEEE International Requirements Engineering Conference (RE 09)

Big EARS: The Return of Easy Approach to Requirements Syntax, Alistair Mavin et al, 18th IEEE International Requirements Engineering Conference (RE 10)

Training

- Develop internal training
- Attend external training
- Partner requirements authors with requirements subject matter experts
- Create “communities of practice” to share best known requirements methods internally

Requirements Syntax

Here is a generic syntax for functional requirements (optional items are in square brackets):

[Trigger] [Precondition] Actor Action [Object]

Example:

When an Order is shipped and Order Terms are not "Prepaid", the system shall create an Invoice.

- Trigger: *When an Order is shipped*
- Precondition: *Order Terms are not "Prepaid"*
- Actor: *the system*
- Action: *create*
- Object: *an Invoice*

Good Requirements Checklist

Utilize a checklist that defines the attributes of a well written requirement. For example, a Good Requirement is:

- ✓ Complete
- ✓ Correct
- ✓ Concise
- ✓ Feasible
- ✓ Necessary
- ✓ Prioritized
- ✓ Unambiguous
- ✓ Verifiable
- ✓ Consistent
- ✓ Traceable

Most of these attributes apply equally to a single requirement and the entire set of requirements

Planguage: A Constrained Natural Language

Many requirements defects can be eliminated by using a constrained natural language. Planguage is an example:

- Developed by Tom Gilb in 1988 and explained in detail in his book *Competitive Engineering* *
- Is a combination of the words *Plan*ning and Lan*guage*
- An informal, but structured, keyword-driven planning language (e.g., Name, Description, Rationale)
- Can be used to create all types of requirements

* *Competitive Engineering,* Butterworth-Heinemann, 2005

Planguage Keywords for Any Requirement

Name	A short, descriptive name
Requirement	Text that defines the requirement, following EARS*
Rationale	The reasoning that justifies the requirement
Priority	Sets the importance of the requirement relative to others in the product
Priority Reason	A brief justification for the assigned priority level
Status	The current state of the requirement
Contact	The person to contact with questions about the requirement
Source	The original inspiration for the requirement

Planguage Example

Name: Create_Invoice

Requirement: When an Order is shipped and Order Terms are not “Prepaid”, the system shall create an Invoice.

Rationale: Task automation decreases error rate, reduces effort per order. Meets corporate business principle for accounts receivable.

Priority: High.

Priority Reason: If not implemented, business process reengineering will be necessary and program ROI will drop by $400K per year. ← Finance study

Status: Committed

Contact: Hugh P. Essen

Source: I. Send, Shipping

Created by: Julie English

Version: 1.1, **Modified Date**: 20 Oct 11


intel®
Leap ahead™