# EARS: The Easy Approach to Requirements Syntax

John Terzakis

Intel Corporation

john.terzakis@intel.com

July 21, 2013
ICCGI Conference
Nice, France

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 overview
- Issues with requirements
- Overcoming issues with requirements
- Identifying ubiquitous requirements
- EARS and EARS examples
- Using EARS to rewrite requirement examples
- EARS at Intel
- Wrap up

# Requirements Overview

# What is a Requirement?

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

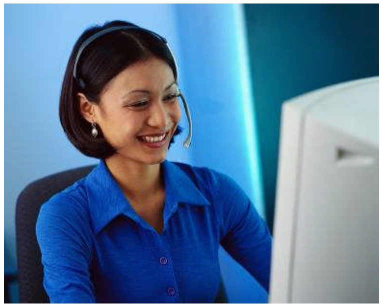
1. What a system must do
2. A known limitation or constraint on resources or design
3. How well the system must do what it does

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**

# Examples of Functional and Non-Functional Requirements



Video over IP
Conference Calling

**Functional Requirements**

- Add Participant
- Count Participants
- Drop Participant
- Lock Call to New Participants
- Summon Operator
- Mute voice

**Non-Functional Requirements**

- Voice and Video Quality
- Reliability
- Availability
- Ease of Use
- Cost
- Localization

# Issues with Requirements

# 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
  - Some times missing triggers
  - Logic is not always complete (“if” but no “else”)

- 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 about five alternatives for the requested word.
   - How many is “about five”?   Three?  Four?  Six? Ten?  More?
   - Under what conditions are they displayed ?

3. 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?

4. 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.

# Overcoming Issues with Requirements

# Tools to Overcome Requirements Issues

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

# 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

# Consistent Requirements Syntax

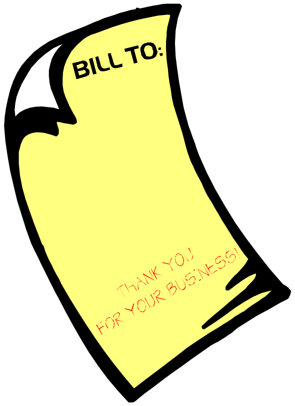
Here is a generic syntax for functional requirements (optional items are in square brackets):

[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

See Backup for details

# 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

# 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
- Differentiates between five types (or patterns) of requirements:
  - Ubiquitous (always occurring)
  - Event-driven
  - Unwanted behaviors
  - State-driven
  - Optional features

# 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:

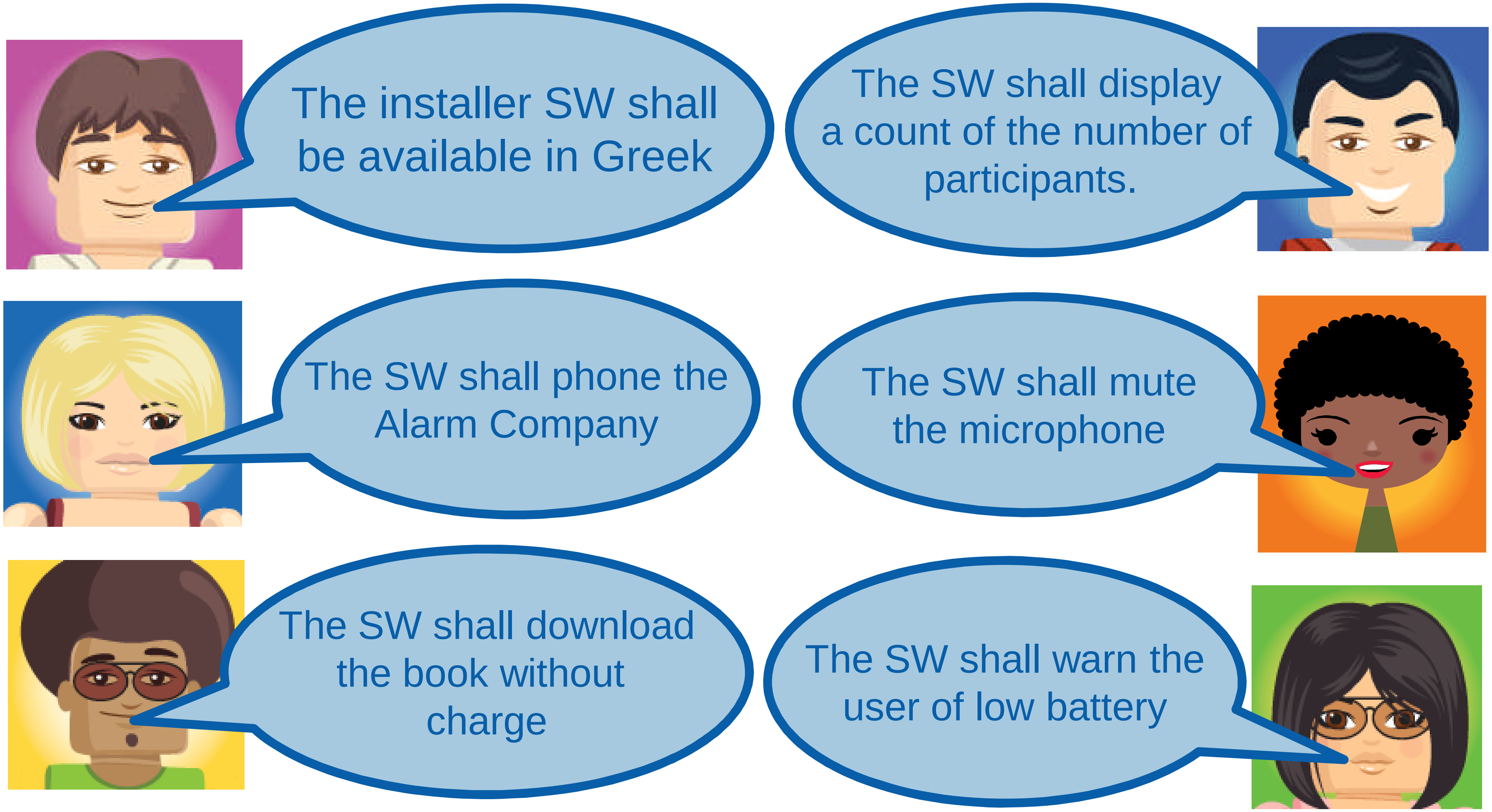
- The software shall be distributed on CD-ROM and DVD media.
- The software shall prevent Unauthorized Access to patient data.

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:

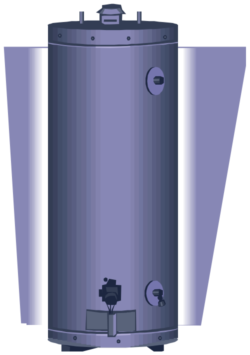
  WHEN <trigger> <optional precondition> the <system name> shall <system response>

# 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.



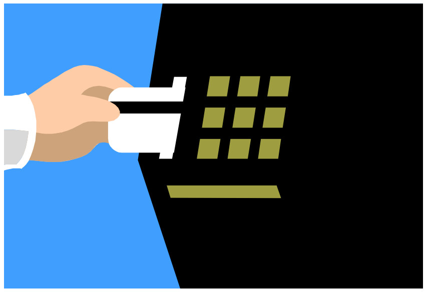
# Unwanted Behavior Requirements

- 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:

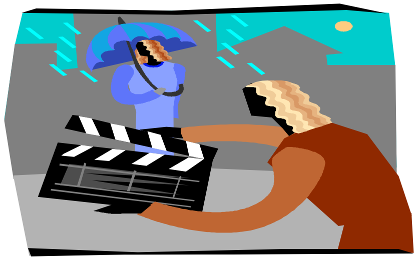
  WHERE <feature is included>, the <system name> shall <system response>

# 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:

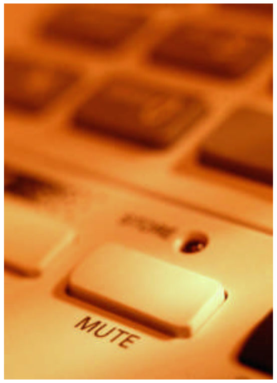
The software shall phone the Alarm Company.

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)

# Complete

A requirement is “complete” when it contains sufficient detail for those that use it to guide their work

Every gap forces designers and developers to guess –*who do you want specifying your product?*

**Not Complete:**

**The software must allow a TBD number of incorrect login attempts.**

**Complete:**

**When more than 3 incorrect login attempts occur for a single user ID within a 30 minute period, the software shall lock the account associated with that user ID.**

# Correct

A requirement is correct when it is error-free

Requirements can be checked for errors by stakeholders & Subject Matter Experts (SMEs)

Requirements can be checked against source materials for errors

Correctness is related to other attributes – ambiguity, consistency, and verifiability

**Not Correct:**

**The 802.3 Ethernet frame shall be 2048 bytes or less.**

**Correct:**

**The 802.3 Ethernet frame length shall be between 64 and 1518 bytes inclusive.**

# Concise

A requirement is concise when it contains just the needed information, expressed in as few words as possible

Requirements often lack conciseness because of:

- Compound statements (multiple requirements in one)
- Embedded rationale, examples, or design
- Overly-complex grammar

**Not concise:**

**The outstanding software written by the talented development team shall display the current local time when selected by the intelligent and educated user from the well designed menu.**

**Concise:**

**The software shall display the current local time when selected by the user from the menu.**

# Feasible

A requirement is feasible if there is at least one design and implementation for it

Requirements may have been *proven* feasible in previous products

Evolutionary or breakthrough requirements can be *shown* feasible at acceptable risk levels through analysis and prototyping

**Not Feasible:**

**The software shall allow an unlimited number of concurrent users.**

**Feasible:**

**The software shall allow a maximum of twenty concurrent users**

# Necessary

A requirement is necessary when at least one of the following apply:

- It is included to be market competitive
- It can be traced to a need expressed by a customer, end user, or other stakeholder
- It establishes a new product differentiator or usage model
- It is dictated by business strategy, roadmaps, or sustainability needs

**Not Necessary:**

**The software shall be backwards compatible with all prior versions of Windows®**

**Necessary:**

**The software shall be backwards compatible with Windows® Vista SP2 and SP1, and Windows® XP SP3.**

# Prioritized

A requirement is prioritized when it ranked or ordered according to its importance.

All requirements are in competition for limited resources.  There are many possible ways to prioritize:

- Customer Value, Development Risk, Value to the Company, Competitive Analysis, Cost, Effort, TTM

Several scales can be used for prioritization:

- Essential, Desirable, Nice to Have
- High, Medium, Low
- Other ordinal scales based on cost, value, etc.

**Not Prioritized:**

**All requirements are critical and must be implemented.**

**Prioritized:**

**80% of requirements High, 15% Medium and 5% Low.**

# Unambiguous

A requirement is unambiguous when it possesses a single interpretation

Ambiguity is often dependent on the background of the reader

Reduce ambiguity by defining terms, writing concisely, and testing understanding among the target audience

Augment natural language with diagrams, tables and algorithms to remove ambiguity and enhance understanding

**Ambiguous:**

**The software must install quickly.**

**Unambiguous:**

**When using unattended installation with standard options, the software shall install in under 3 minutes 80% of the time and under 4 minutes 100% of the time.**

# Verifiable

A requirement is verifiable if it can be proved that the requirement was correctly implemented

Verification may come via *demonstration*, *analysis*, *inspection*, or *testing*.

Requirements are often unverifiable because they are ambiguous, can't be decided, or are not worth the cost to verify.

**Not Verifiable:**

**The manual shall be easy to find on the CD-ROM.**

**Verifiable:**

**The manual shall be located in a folder named User Manual in the root directory of the CD-ROM.**

# Consistent

A requirement is consistent when it does not conflict with any other requirements at any level

Consistency is improved by referring to the original statement where needed instead of repeating statements.

**Inconsistent:**

**#1: The user shall only be allowed to enter whole numbers.**

**#2: The user shall be allowed to enter the time interval in seconds and tenths of a second.**

**Consistent:**

**#1: The user shall only be allowed to enter whole numbers except if the time interval is selected.**

**#2: The user shall be allowed to enter the time interval in seconds and tenths of a second.**

# Traceable

A requirement is traceable if it is uniquely and persistently identified with a Tag

Requirements can be traced to and from designs, tests, usage models, and other project artifacts.

Traceability enables improved

- Change impact assessment
- Schedule and effort estimation
- Coverage analysis (requirements to tests, for example)
- Scope management, prioritization, and decision making

**Not Traceable:**

**The software shall prompt the user for the PIN.**

**Traceable:**

**Prompt_PIN: The software shall prompt the user for the PIN.**

intel®

Leap ahead™