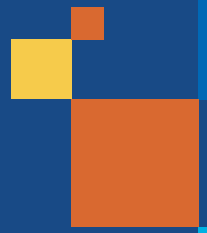
Leading in Complexity

Requirements at Intel

Distance Learning – Short Course

Version 9.2
October 2020

sarah.c.gregory@intel.com



intel®

Licensing

Licensed under Creative Commons: **Attribution-NonCommercial 4.0 International (CC BY-NC 4.0)**

You are free to:

Share — copy and redistribute the material in any medium or format

Under the following terms:

Attribution — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.

NonCommercial — You may not use the material for commercial purposes.

No additional restrictions — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.

You do not have to comply with the license for elements of the material in the public domain or where your use is permitted by an applicable exception or limitation.

No warranties are given. The license may not give you all of the permissions necessary for your intended use. For example, other rights such as publicity, privacy, or moral rights may limit how you use the material.

OpenRE – Notes about this course

- Intel Corporation has a robust internal curriculum of Systems and Requirements Engineering training and education materials.
- Most training has been delivered F2F. Distance Learning options have existed since ~2012, but COVID accelerated the need to make more distance sessions available.
- In 2021, we are exploring development of web-based versions of lecture content to enable instructors to better scale across the company for hands-on coaching.
- In 2021, we also plan to establish IREB Foundation Level RE Certification as an expectation for business unit RE leads. We will develop, pilot, and deploy an internal IREB-compliant class.

Welcome! What this class is NOT:

1. Tool training: The training is intentionally *tool-agnostic*. The practices are relevant and beneficial regardless of any tool(s) used.
2. A prescriptive "how to" job aid: You'll learn many good practices to add to your toolkit, but you won't be told how to be a Requirements Engineer. Task-relevant experience is required.
3. A Rulebook for "how to do requirements" for any business unit, product, project, or program – the course teaches *heuristics*.
4. A prep class for the IREB® CPRE™ Foundation Level certification exam. (ETA for this – Q2'2021)

A *heuristic* is anything that provides a plausible aid or direction in the solution of a problem.

* For intact team training, your instructor ***may*** discuss the rules that your particular team or business unit has chosen to follow, but will distinguish those from general RE heuristics.

Contents

Objectives

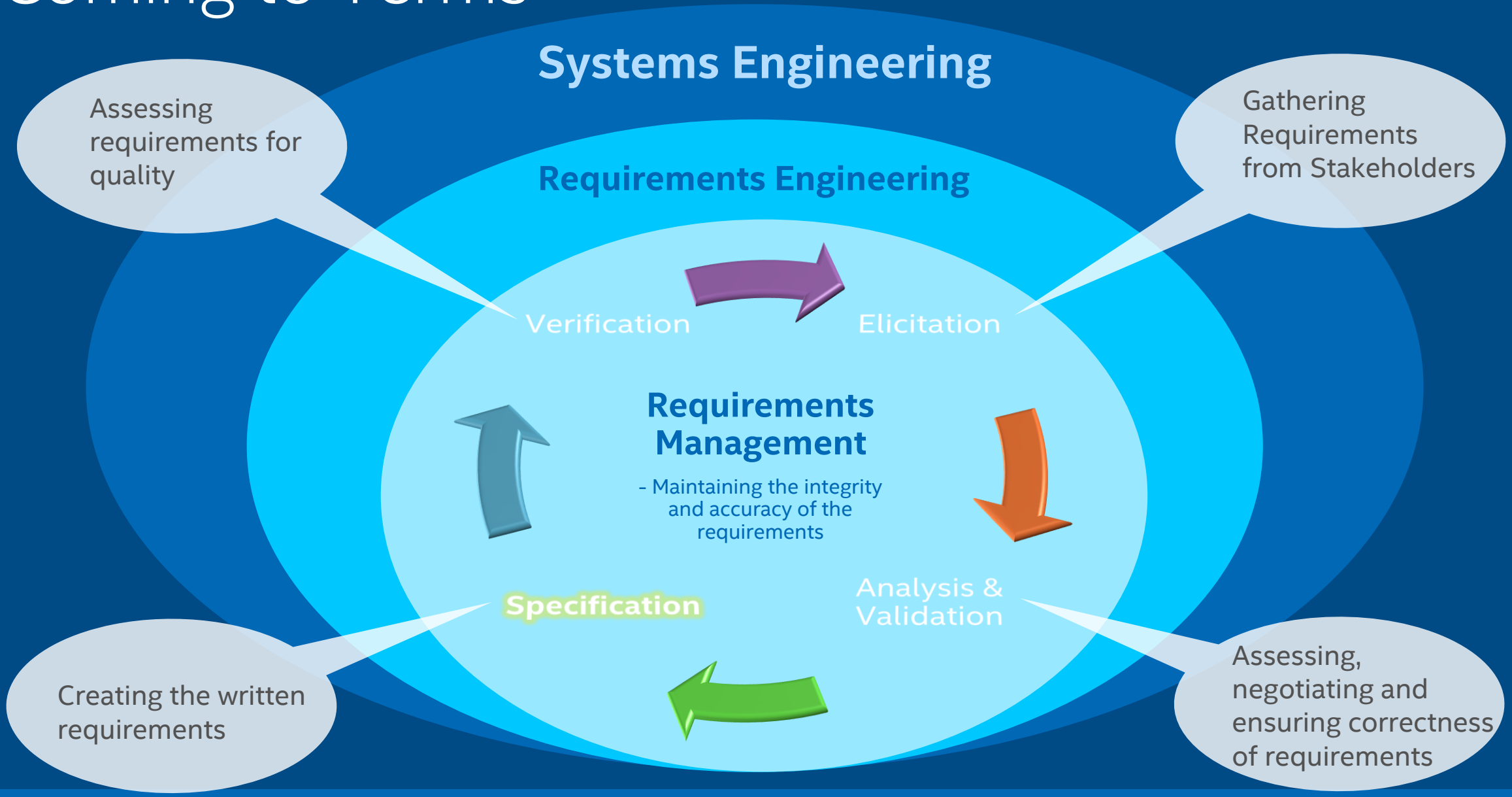
Upon completing this course, you should be able to:

- Identify the most common problems with natural language requirements
- Understand several fundamental best practices in requirements specification.
- Write functional requirements and constraints using a simple syntax
- Write quantified, unambiguous quality and performance requirements
- Describe additional techniques besides natural language that can be used to specify requirements.
- Describe *objective* and *subjective* techniques to assess requirements quality
- Describe key practices in Requirements Management, and how they're enabled
- Know where to find more information on the topics presented

Introduction

Requirements Engineering – an Engineering Discipline

Coming to Terms



What is a Requirement?

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

1. Functional Requirement: A result or behavior that shall be provided by a function of a system, including requirements for data or the interaction of a system with its context.
   ("What the system must do")
2. Quality Requirement: A statement of a quality concern such as performance, availability, security, reliability, and usability, not covered by Functional Requirements. ("How well the system does what it does")
3. Design Constraint: A statement that limits the solution space beyond what is necessary for meeting given functional, quality, and performance requirements ("What the system must be")

Source: International Requirements Engineering Board (IREB) – CPRE Foundation Level Syllabus

The Purpose of Requirements

Requirements help establish a **clear**, **common**, and **coherent** understanding of what the system must accomplish.

Clear: All statements are unambiguous, complete, and concise

Common: All stakeholders share the same understanding

Coherent: All statements are consistent and form a logical whole

Requirements are the foundation upon which systems are built.

Well written requirements increase the probability that we will release a successful system (low defect, high quality, on time)

Ways to Specify Requirements

Natural Language: Use of English as a means of specifying requirements

Diagrams and Models: Visual specification using formats such as state models, informal diagrams, and components of the Unified Modeling Language

Lightweight Formal Methods: Specification using defined objects and states in semi-formal notation, including full use of UML

Formal Methods: Mathematically-precise specification using rigid syntax, semantics, and theorem proving, can include SysML models

Natural language and formal methods are endpoints of a spectrum of formality. Use the methods that are appropriate for your project or program.

Example: The Specification Spectrum

Unconstrained natural language

The display should go off when the user has the phone by her face during a call.

Constrained natural language

While a CDMA or VoLTE call is active, when the device senses the user's face in _proximity_ to the display, the device shall turn off the display.

Lightweight formal methods

While (_CDMACall_ or _VoLTECall_) and _faceProximity; _devDisplayOff_

Formal specification

$\forall$ callCDMA: CDMA Call, callVoLTE: Voice over LTE Call, dev: Device, face: Face

(Active(call.CDMA) $\vee$ Active(call.VoLTE)) $\wedge$ At(dev, face) $\rightarrow$ dev.Display = 'off' *W* ($\ulcorner$ At(dev, face))

The Right Tool for the Job

There is no single "right way" to do Requirements Engineering – not at Intel, not in industry.

Some projects are well-suited for Agile methods with minimal documentation, while others may require formal specification methods, including models. Many will be in-between these endpoints.

This class focuses on **Constrained Natural Language (CNL)** as a foundational practice for Requirements Engineering. Regardless of other methods used, CNL is a valuable skill to have in one's repertoire.

On many projects, additional specification techniques, including Model Based Systems Engineering (MBSE) or mathematics-based specification languages, may be helpful or even necessary, but CNL will be applicable as well.

Section Summary

A **requirement** is a statement of something a system must do, how well it must do what it does, or a constraint that must be met.

The purpose of requirements is to help ensure a **clear, common, and coherent** understanding of the system among stakeholders.

Requirements Engineering consists of five activities: **elicitation, analysis & validation, specification, verification, and management.**

Requirements can be specified using many different formats that span a range of formality, rigor, and ease of reading.

No single “right way” exists to do Requirements Engineering. **Constrained Natural Language**, a foundational technique for this class, is useful across a wide spectrum of products and programs.

Detail Level and Timing Issues

When to Specify What Gets Specified

How Much Detail is Enough?

The correct detail level, like the correct investment in requirements activities overall, must balance risk and investment:

The acceptable risk-investment region depends on several different factors.

How Much Detail is Enough?

The correct level of detail in requirements depends on factors that include:

- Precedented vs. unprecedented product or feature
- Development team experience, size, and distribution
- Acceptable risk level during development
- Domain, organizational, and technical complexity
- Solution, Platform, Component, Subcomponent, or other hierarchy
- Product Line Engineering considerations (reuse)
- Need for regulatory compliance
- Current location in the life cycle

Requirements completeness is judged *continually*, based on the changing needs of the project and team.

Requirements must guide the **current** activities of all team members ***at an acceptable risk level.***

How Much Detail is Enough?

No requirements specification is ever truly complete.

There isn't enough time (or resource) available to write them all – *and you shouldn't have to anyway*...

Provide detail where it's needed most: **risky, unprecedented, not previously specified, or complex areas.**

Writing thousands of requirements may feel like productivity, but:

- If what gets documented is what everyone already understood, *what is the effect on project risk?*
- Large specifications can lead to a false sense of security.

Make a conscious decision on WHAT not to write

A Flexible Approach to Scope and Details

Regardless of what type of system you are building, use an *evolutionary* approach to requirements engineering :

1. Start by generating requirements that define the known scope of the system, but at minimum depth.
2. Decide *what not to write.*
3. Decide *when to write* what you will write
4. Create the necessary details at the right time, always using business value and risk reduction as guides.
5. Revisit steps 1-4 often based on what you learn as you make progress and the requirements evolve.

Make a conscious decision on WHEN to write what you do write

Requirements vs. Design

"Requirements are the *what*, design is the *how…*"

This is true – to a point, but the main difference between requirement and design is one of perspective:

How you look at a statement dictates whether it is a requirement or a design.

Requirements vs. Design

Many products carry the majority of their design specifications forward from previous versions as constraints on current requirements.

Therefore,

> It's not whether a statement is a "requirement" or a "design" that matters, but whether the statement places *appropriate constraints* on the people that will read it.

If the system *must be or act a certain way*, say so… If not, leave the people downstream as much freedom as possible to do their jobs

Section Summary

The correct level of detail is a matter of **balancing cost and risk.**

Many factors influence the choice of how much detail is enough, including attributes of both the team and the product.

Make a conscious decision up front on **what not to write.**

Make a conscious decision up front on **when to write** what you must write.

Use an **iterative, incremental, learning-based** approach to requirements specification.

It is not whether a statement is a “requirement” or “design” that matters, but whether it places **appropriate constraints** on those that use it to guide their work.

There is nothing intrinsically incompatible between good requirements engineering and agile development practices.

Common Problems with Natural Language

Specifying What We Mean, Meaning What We Specify

Common Problems with Natural Language

While useful in everyday interactions, natural language is fertile ground for a number of problems related to requirements, including:

- Weak words
- Unbounded lists
- Implicit collections
- Issues around verb choice and semantics
- Poor or complicated grammar

Let's examine each of these to see how they cause problems...

Discussion – Fun with Ambiguity

Analyze the following sentences for ambiguity. What possible meanings can you find?

1. Mary had a little lamb.
2. No one has seen a pig with a magnifying glass.
3. All engineers prefer a working prototype to a theory.
4. Shut down the pumps if the water level remains above 100 meters for more than 4 seconds.
5. When we finally reached the bank, we were impressed with how green it was.

Weak Words

Weak words are subjective or lack precise meaning.

- Quickly
- Easy
- Timely
- Before, after
- User-friendly
- Effective
- As possible
- Appropriate
- Normal, usual, regular
- Support, Capability
- Reliable
- State-of-the-art

And *many* more…

Don’t use weak words – define what you mean using precise, measurable terms

The use of “Support”

“Support” is both one of the most common weak words used in requirements specifications, and *potentially adds the greatest risk.*

- Level of completeness is generally undefined – what is enough “support”?
- Requirements that include “support” are often not verifiable as written.
- “Support” is presumptively a failure of the Unambiguous item on the Good Requirements Checklist; all review tools will flag the word as an issue.
- With practice, alternatives to “support” are easier to identify and use.
- Exceptions to its ambiguity exist, primarily in structural engineering.

Do requirements containing “support” need to be rewritten?
Risk vs. Reward: Assess carefully, and choose wisely!

Discussion – The use of “Support”

While use of “support” in requirements cannot be completely avoided, it is often used when other terms would be preferable.

Analyze the following uses of the term “support” - how are they different?

What terms might be substituted to improve clarity and reduce ambiguity?

1. The system shall support 802.11n.
2. The system shall support Windows* 10.
3. The system shall support 250 concurrent users.
4. The system shall support a static load of 4 metric tons.

Unbounded Lists

An Unbounded List is one that lacks a starting point, an ending point, or both.

Classic examples include:

- At least
- Including, but not limited to
- Or later
- Such as
- Etc.

For example, how would you design and test a system that “must support at least 250 users”?

Unbounded lists are impossible to design for or to test against

Implicit Collections

An Implicit Collection is a collection of objects within requirements that are not explicitly defined anywhere.

Without a definition, readers may assume an incorrect meaning.

Example:

“The software must support 802.11 and other network protocols supported by competing applications under Linux.”

- What is counted as a “competing application”?
- What belongs to the collection of “other network protocols”?
- What specific protocols of 802.11 are included?
- “Linux” is also a collection of OS vendors, versions, and revision levels

The Trouble with “And”

The word “and” in a requirement statement may signal an error:

A qualifier before a phrase with “and” can result in **conflicting interpretations** of a requirement: “authorized nurses and doctors”

“And” may indicate the presence of **more than one requirement**:

- The tool shall identify and prioritize functional safety requirements.

“And” is acceptable in a few circumstances:

- When sensor detects water level below the measurement midpoint, water management system shall simultaneously pause the pump and open the water tap.

The presence of more than one verb in a requirement is a strong indicator – but not an absolute rule - that “AND” signals more than one requirement.

Other Common Issues

Be careful with verb choice

- Systems shall manage, enable, allow, support, ensure, permit, assist, provide the capability to…

Be careful with ***each***, ***all***, ***every***, and ***only***

- How is “each user” different from “all users”?
- The placement of “only” can completely change the meaning of a sentence

Avoid grammatical issues

- “The system shall report/log improper access attempts and notify administrators if a user does not respond to warning messages or lock out the account.”

Section Summary

Natural language is prone to ambiguity.

Weak words leave the interpretation of the requirement open to each individual.

Unbounded lists make it impossible to adequately design for and test a requirement.

Implicit collections create multiple interpretations of group membership.

Poor grammar, complicated sentence structure, semantic issues, and other language use issues also cause requirements problems.

Exercise 1

Identifying Issues with Natural Language

Instructions: Assess a sample of product usability requirements.

Locate weak words, unbounded lists, implicit collections and other problems in the natural language of the sample.

Identify and categorize the errors that you see.

BONUS – can you identify the product from which this exercise was drawn? ☺

Skill Developed: Reinforce the ability to spot issues with natural language while reviewing requirements

Exercise 1

"The usability objective of the Broadcast Plus client is to be usable by the intended customer at a 5' distance. The client should be an integrated system that is both reliable and responsive. Reliability and responsiveness are more critical for this device than for PC desktop systems. Reliability should be as good as that of consumer home entertainment devices (e.g., TV or VCR) and response to user interaction should be immediate.

The applications should provide an easy-to-learn, easy-to-use, and friendly user interface, even more so than PC desktop applications. Users should be able to start using the application immediately after installation. Users should be able to satisfactorily use the device with little instruction.

Friendly means being engaging, encouraging, and supportive in use. Users must feel comfortable with the client and must not be given reason to worry about accidentally initiating a destructive event, getting locked into some procedure, or making an error. Feedback for interactions should be immediate, obvious, and appropriate."

End of Distance Learning Session #1

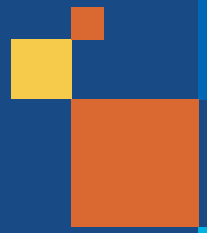
Systems and Requirements Engineering

Requirements Specification

Distance Learning Session #2

Version 9.2
October 2020

sarah.c.gregory@intel.com



intel®

Contents

Objectives

Upon completing this course, you should be able to:

- Identify the most common problems with natural language requirements
- Understand several fundamental best practices in requirements specification.
- Write functional requirements and constraints using a simple syntax
- Write quantified, unambiguous quality and performance requirements
- Describe additional techniques besides natural language that can be used to specify requirements. (TBD)
- Describe *objective* and *subjective* techniques to assess requirements quality
- Understand key concepts and capabilities needed for requirements management
- Know where to find more information on the topics presented

Beginning Distance Learning Session #2

Specification Basics

“Requirements 101”

Specification Basics

These basic practices have a high return on investment:

- **Use a template** for requirements specification (present in RM/SE tools).
- Move from **unconstrained natural language** to **constrained natural language** to reduce ambiguity and improve completeness with minimal effort.
- Do not include **design statements** in the requirements unless they are there as intentionally-imposed constraints.
- **Supplement natural language** where needed with other representations to improve comprehension and reduce ambiguity.
- Write requirements for **reuse** to create **known-good datasets**.

Specification Basics

- **Quantify qualitative requirements** so they are verifiable.
- **Define terms in a glossary** with an accessible location to ensure accurate use across projects and product lines. (Use standards-based terms where applicable and available.)
- **Validate requirements with stakeholders frequently** as a test of understanding.
- **Rigorously verify requirements** to prevent defects and maximize requirements quality.
- Make **reviews and approvals** of requirements essential lifecycle milestone steps

Using Imperatives

Use Shall or Must to indicate requirements.

Should and *May* are not used for requirements, but to specify design goals or options that will not be validated.

Will and *Responsible for* are not used for requirements, but may be used to refer to external systems or subsystems for informational purposes.

Imperatives are an example of Constrained Natural Language. We restrict the definition of these words in order to achieve shared *understanding* of their meaning.

Using Imperatives

Examples:

- The system shall conform to ISO 14825:2011, Intelligent transport systems - Geographic Data Files (GDF) - GDF5.0
- Design Goal: The data cache should occupy as little memory as possible, and may use lossless compression to achieve this
- Note: Accurate _accountBalance_ for the user's accounts will be supplied by the _hostFinancialInstitution_ systems. The ATM is not responsible for validating the reported _accountBalance_

Negative Specification

It is appropriate to state what the system shall not do, but keep in mind that *the system shall not do much more than it shall do.*

- Use negative specification sparingly, for emphasis.
- Don't use negative specification for requirements that could be stated in the positive.
- Avoid double negatives altogether.

NO: "Users shall not be prevented from deleting data they have entered"

YES: "The system shall allow users to delete data they have entered"

Discussion – Negative vs. Non-specification

Compare the requirements below. How would each guide the work of those that read it to implement a software installer?

What similarities and differences can you find between negative specification and non-specification?

Be prepared to share your insights with the class at the end of the discussion

1. The system shall not support Windows* 8.
2. There is no requirement to support Windows 8.
3. The system shall support Windows 10, all service packs and revisions

Good Requirements Checklist (1 of 2)

- Complete: A requirement is complete when it contains sufficient detail for those that use it to guide their work at an acceptable level of risk.
- Correct: A requirement is correct when it is error-free.
- Concise: A requirement is concise when it contains just the necessary information, expressed in as few words as possible.
- Feasible: A requirement is feasible if there is at least one design and implementation for it.
- Necessary: A requirement is necessary when it:
  - Is included to be market competitive
  - Can be traced to a regulatory, stakeholder, or approved customer need
  - Establishes a new strategic differentiator or usage model
  - Is dictated by business strategy, roadmaps, or sustainability

Good Requirements Checklist (2 of 2)

- Prioritized: A requirement is prioritized when it is ranked or ordered according to its importance.
- Unambiguous: A requirement is unambiguous when it possesses a single interpretation.
- Verifiable: A requirement is verifiable if it can be proved that the requirement was correctly implemented.
- Consistent: A requirement is consistent when it does not conflict with any other requirements at any level.
- Traceable: A requirement is traceable if it is uniquely and persistently labeled with an individual ID.

Natural Language Processing (NLP):

NLP tools are in the Intel environment and are accessible to authors and reviewers of requirements to help with quality improvements.

Used early in requirement authoring, these tools can provide an author with areas where their requirements skills can be improved.

Used during the requirements creation process, these tools can provide early indicators of requirements written by untrained authors, which thus may contain many defects.

Used prior to peer reviews or inspections, these tools can determine whether the requirements are of sufficient quality to spend person hours on detailed analysis.

A separate one-hour training and license purchase may be required to use these tools

Guidance for the Good Requirements Checklist

Remember – this course teaches *heuristics*, not rules.

- Apply the GRC checklist *heuristics* carefully, as learning aids, not just boxes to check.
- Focus on the checklist as a means of requirements defect prevention, rather than just as a tool to identify and remove requirements defects.
- Metrics based on the checklist may be helpful, but not sufficient. Subjective requirements quality assessment matters at least as much as objective defect data.

Section Summary

Several basic requirements practices have a high return on a relatively small investment; start using those practices soon if you are not doing so already.

Shall and must are the proper imperatives for a requirement statement – don't use *should* and *may*.

Use negative specification sparingly, for emphasis.

There are 10 Attributes of a Good Requirement that must be met – for individual requirements, collections of requirements, and the overall specification(s).

Natural Language Processing (NLP) requirements evaluation tools can potentially add value by identifying some issues, but use with caution.

Specifying Functional Requirements & Design Constraints

Constrained Natural Language
“What Shall the System Do? What Shall the System Be?”

Functional Requirements and Constraints

Functional requirements and constraints are measured on a Boolean (yes/no) scale – they are either present or absent in the system.

Functional requirements and constraints are captured with a single imperative statement containing “shall” or “must”.

Examples:

> The system shall permit the user to create playlists of songs.
>
> The system must comply with ISO 26262-1:2018 Road Vehicles – Functional Safety.

Easy Approach to Requirements Syntax (EARS)*

Pattern Name	Pattern
Ubiquitous	The <system|actor> shall <action> <object>.
Event-Driven	**When** <trigger> <optional precondition>, the <system|actor> shall <action> <object>
State-Driven	**While** <system state|actor state>, the <system|actor> shall <action> <object>
Unwanted Behavior	**If** <unwanted state|unwanted event>, **then** the <system|actor> shall <action> <object>
Optional Feature	**Where** <feature is included>, the <system|actor> shall <action> <object>
Compound	(combinations of the above patterns, usually “while” with “when” or “if/then”)

The *Easy Approach to Requirements Syntax* (EARS) consists of a set of patterns for specific types of functional requirements and constraints. The EARS syntax is used with kind permission of Alistair Mavin and Phil Wilkerson, and the RE Methods Group of Rolls Royce, PLC.

Examples of EARS Syntax

The user interface shall conform to Intel branding guidelines.

When a user commands installation of an _application_ that accesses _communicationsFunctions_, the system shall prompt the user to acknowledge the access and agree before continuing installation.

While the _phoneFunction_ is active and _speakerMode_ is off, **when** the system detects the user's face in _proximity_ to the display, the system shall turn off the display and deactivate the display's touch sensitivity.

While in _standby_, **if** the battery capacity falls below _lowBatteryThreshold_ remaining, **then** the system shall change the _systemLED_ to flashing red.

Where the system contains two SIM cards, the system shall allow the user to assign a default network to each contact in the address book.

About Ubiquitous Requirements

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

Most legitimate ubiquitous requirements state a fundamental property of the system

- The system shall meet RoHS standards
- The system case shall be available in both matte and gloss finishes

System functions that appear ubiquitous are often not

- The system shall warn the user of a low battery

Requirements *Completeness*

It takes much more than a single imperative sentence to satisfy all 10 Attributes of a Good Requirement for a single statement.

Other evidence can be provided through structural data model elements, or textual adds to the specification of the statement or collection of requirements.

> For example, a defined link from a requirement to an approved higher-level statement that it Satisfies is presumptive evidence of Necessity.
>
> Necessity can also be demonstrated by a “Rationale” attribute (“field”) attached to a requirement.

Structural arrangements of requirements may reasonably differ under different circumstances (product type, business unit, database functionality vs. Word/Excel, etc.) but strive to *meet the intent* of the Good Requirements Checklist items.

Essential Requirements Information

Every requirement needs to include key infomation

The items below represent the minimum list of attributes (“fields”) for any individual requirement statement:

Name	A short, descriptive name
Requirement	The requirement statement itself, in EARS format
Intel Classification	Classification of the requirement according to Intel information security guidelines (Intel Confidential / Intel Top Secret)
Notes	Any explanatory or informational comments

How to Demonstrate “Goodness”

Complete	All attributes (“fields”) are complete; review by Consumer(s) indicates that sufficient information is present
Correct	Technical accuracy; Producer of source content reviews and indicates correctness.
Concise	Requirement contains only one independent idea, defined and tested as a whole
Feasible	Review and Approval indicates requirement can be completed on the project with available time and resource
Necessary	Traceability to the content that requirement Satisfies – or – Statement of necessity (“Rationale” or “Motivation”) in an attribute or field.

How to Demonstrate “Goodness”

Prioritized	Inclusion in a baselined dataset - OR - Attribute or Field with “Priority” defined
Unambiguous	Cross-functional review or inspection indicates objective lack of ambiguity, agreement on content of requirement
Verifiable	Unambiguous; Test team member of cross-functional team approves requirement as testable as written
Consistent	Traceability and review or inspection identifies no conflicting requirements or gaps
Traceable	Unique and persistent ID; bidirectional traceability enabled for content prior to its relevant lifecycle milestone

“Housekeeping” details for Any Requirement

Created By	The person who first authored or entered the requirement
Creation Date	The date the requirement was first authored or entered
Modified By	The person who modified this requirement
Modified Date	The date the requirement was modified
ID	A unique, persistent identifier for the requirement

And some useful non-keyword devices:

< >	“Fuzzy brackets” used to mark terms requiring more details
abcde	Dual underscores to denote use of a _definedTerm_

Completeness through Context, or Metadata?

How to best specify information for a requirement or set of requirements? Is it better to use fields or other per-requirement attributes, or to arrange content structurally in a database or series of documents?

Metadata	Pro: Fields are intuitive, reflect standard office application use. They're easy to adapt, add, remove, and customize. Con: Rarely, if ever, used completely and correctly. Significant contributor to poor data integrity.
Structural Arrangement (Configuration)	Pro: Configuration Management of requirements supports reuse, product line engineering, dataset integrity, context delivery. Con: Requires RM-specific tool capabilities, user learning.

Example (textual): Invoice Creation

Name: InvoiceCreation

Requirement: When an _order_ is shipped and _orderTerms_ are “Credit”, 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.

Status: Committed

Contact: Hugh P. Essen

64

Source: I. Send, Shipping

Created by: Julie English

Version: 1.1, Modified Date: 20 Oct 20

Exercise 2

Writing Functional Requirements and Constraints

Instructions:

Choose one of the two items below – OR an item of your choice! - and write 5-10 functional requirements and constraints that it must satisfy using Constrained Natural Language (including each EARS pattern). Feel free to collaborate on this with your peers!

Smart phone

Drone

Section Summary

Functional requirements and constraints are measured on a yes/no (Boolean) basis.

EARS is strongly recommended as the default method for capturing natural language functional requirements,and constraints.

Question ubiquitous (universally stated) requirements; many have latent triggers or conditions.

Traceability, structural arrangement of requirements, and if needed, additional fields help ensure requirements satisfy the 10 Attributes of a Good Requirement. Use the best practice for the tool you are using, and for objectives you intend to enable (reuse, product line definition, etc.)

Specifying Quality Requirements

“How Well Does the System Do What It Shall Do?”

About Quality Requirements

Quality and Performance Requirements (IREB: “Quality Requirements”) are measured on some interval, such as more versus less, or better versus worse.

Because they are not measured on a yes/no basis like functional requirements and constraints, Quality Requirements are specified a little differently. The single imperative statement is supplemented by several keyword statements as shown on the next slide.

You can think of these new keywords as a more granular way to specify the requirement itself – functional requirements and constraints are simpler than quality requirements, and need only the imperative in the Requirement keyword.

Additional attributes of Quality Requirements

Scale	The scale of measure used to quantify the requirement
Meter	The process, device, or benchmark used to establish location on a Scale
Minimum	The minimum level of quality or performance that is permitted to avoid failure; worst-possible result that is still allowed
Target	The level at which good success can be claimed
Maximum	Upper limit of a stretch goal if everything goes perfectly; should not be exceeded. (Allocate resources elsewhere if trend is better than Maximum)

Example Quality Requirement (textual)

Name: Learnable

Requirement: The system shall be easy to learn

Rationale: Learnability issues are among the top 3 complaints from users, and upcoming hiring makes system learnability critical.

Priority: High

Scale: Average time required for a _novice_ to complete a 1-item order using only the online help system for assistance.

Meter: Measurements obtained on 100 _novices_ during user interface testing.

Minimum: No more than 7 minutes

Target: No more than 5 minutes

Contact: B. Bedderson

Source: Flo Larner

Defined: _novice: A person with less than 6 months of web application use.

Created By: Julie English, Version: 1.1, Modified Date: 20 Oct 2018

Finding Scales for Quality Requirements

There are three types of scales:

1. Natural: Scales with obvious association to the measured quality
2. Constructed: A scale built to directly measure a quality
3. Proxy: An indirect measure of a quality

Examples:

Natural	Temperature measured in degrees Celsius
Constructed	A 7-point scale created to measure environmental impact
Proxy	An in-field MTTF goal measured using pre-release reliability test results

Finding Meters

First, study the scale carefully. If no meter comes to mind:

- Look at references and handbooks for examples for ideas
- Ask others for their experience with similar methods
- Look for examples within test procedures

Once you have a candidate, check to see that:

- The meter is adequate in the eyes of all stakeholders
- There is no less-costly meter available that can do the same job (or better)
- The meter can be employed *before* product release or completion of the deliverable

Remember: Scale = scale of measure,
Meter = Device or process to measure position on the Scale

Examples of Scales and Meters

ID: **Environmental Noise**

Scale: dBA at 1 meter

Meter: Lab measurements performed according to a <standard environmental test process>

ID: **Software Security**

Scale: System resilience under attack

Meter: Time that a system remains accessible via a remote administrator while experiencing a DDoS attack

ID: **System Reliability**

Scale: The time at which 10% of the systems have experienced a <failure>

Meter: Highly-Accelerated System Test (HAST) performed on a sample from early production

Exercise: Quality Requirements

Revisit the requirements you wrote in Exercise 2.

1. Identify at least two quality requirements that can be derived from the functional requirements previously written by your group.
2. Specify these requirements, using the template on the next slide.

Smart phone

Drone

Requirement (primary text)
Rationale
Scale
Meter
Minimum
Target
Maximum

Section Summary

Quality and performance requirements are measured on some interval rather than a yes/no basis, so they need additional keywords and data to be measurable and unambiguous.

Capture quality and performance requirements using the Scale, Meter, Minimum, Target, Maximum keywords.


intel

Beginning Distance Learning Session #3

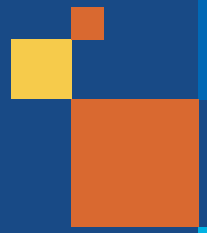
Systems and Requirements Engineering

Requirements Specification

Distance Learning Session #3 (Verification Overview)

Version 9.2
October 2020

sarah.c.gregory@intel.com



intel®

Contents

- The Cost of Quality
- Objective Requirements Quality (“Good Requirements”)
- Subjective Requirements Quality (Producer/Consumer Quality eXchange - PCQX)

This is a VERY HIGH LEVEL overview of Requirements Verification. (full-day class)
Training and experience are needed to be able to successfully verify requirements.

Objectives

After completing this session, you should be able to:

- Understand the Cost of Quality
- Use a checklist to assess objective requirements quality
- Describe how requirements producers and consumers ensure subjective requirements quality

Questions to Consider

1. What types of specification does your team or group use today (requirements, stories, use cases, architecture specs, etc.)?
2. How is this information captured, stored, and related?
3. What forms of review is used on the information? What is the purpose of those reviews? Who decides if the information is “good enough” to become the plan of record?
4. What is the most significant challenge you face in reviewing requirements and other product definitional data?
5. What was the quality level of your last program’s requirements?

Assessing Requirements Quality

Most teams *do not* systematically assess the quality of their requirements deliverables

- Requirements practice maturity is rarely sufficiently mature to understand the practices needed to evaluate requirements quality
- Requirements quality assessments take time, and require both cultural and behavioral changes for individuals, teams, and business units
- Failure to systematically perform Requirements (and Systems) Engineering activities leads to a hidden pile of work that shifts rightward across the product lifecycle.
- No bright line rule exists to define “how much” Requirements Verification is needed to ensure product quality, and how to evaluate cost versus benefit of this work. Analyze risk vs. benefit carefully.

The Cost of Quality

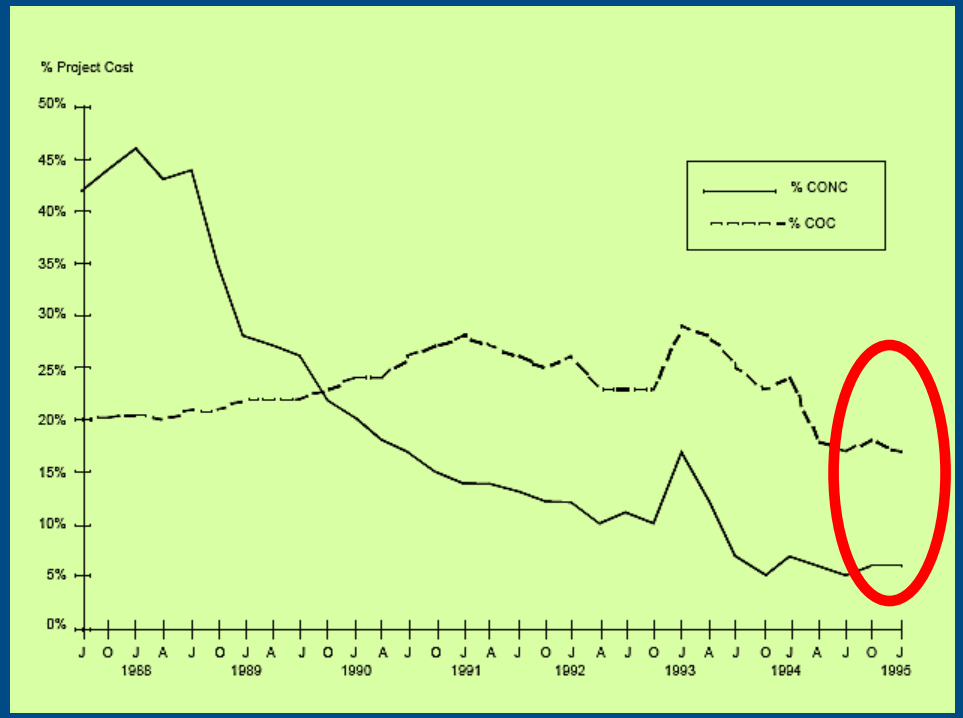
The cost of quality is often divided into two categories: *Conformance* and *Non-Conformance*

Cost of Conformance	Cost of Non-Conformance
• Reviews • Inspections • Training • Testing	• Rework and Delays • Customer support • Product updates • Recalls

84

Spending in these areas tends to be inversely related, but what's the most efficient balance?

The Cost of Quality | What Raytheon* Achieved



Over a seven-year program:

- Rework dropped from 41% to under 10% of budget
- Productivity increased by a factor of 2.8
- Budget/schedule variance reduced to +/- 3%
- Total cost of quality reduced from 61% to under 30%

85

Lessons:

- Large-scale improvement is possible, *but it takes time*
- Even so, early improvement is significant (~50% reduction in 2 years)

*Third-party brands and names are property of their respective owners

Requirements Review Challenges

Requirements reviews are often challenging because:

- Many authors are unable to separate themselves from their work product
- Monolithic documents create confusion about who needs to review what content, and which feedback matters
- Potential reviewers are often unable to dedicate the time needed to do the review correctly
- Reviews are often a step scheduled just before a milestone, with time and resource limitations, and no prescribed method or process
- Many reviews are conducted via email with limited participation and follow-up, and no record of comments, changes, and final content.

Requirements Verification – a separate RE activity. Additional training is available *and necessary.*

Objective Requirements Quality Assessment

Shared criteria build a shared understanding of “goodness”

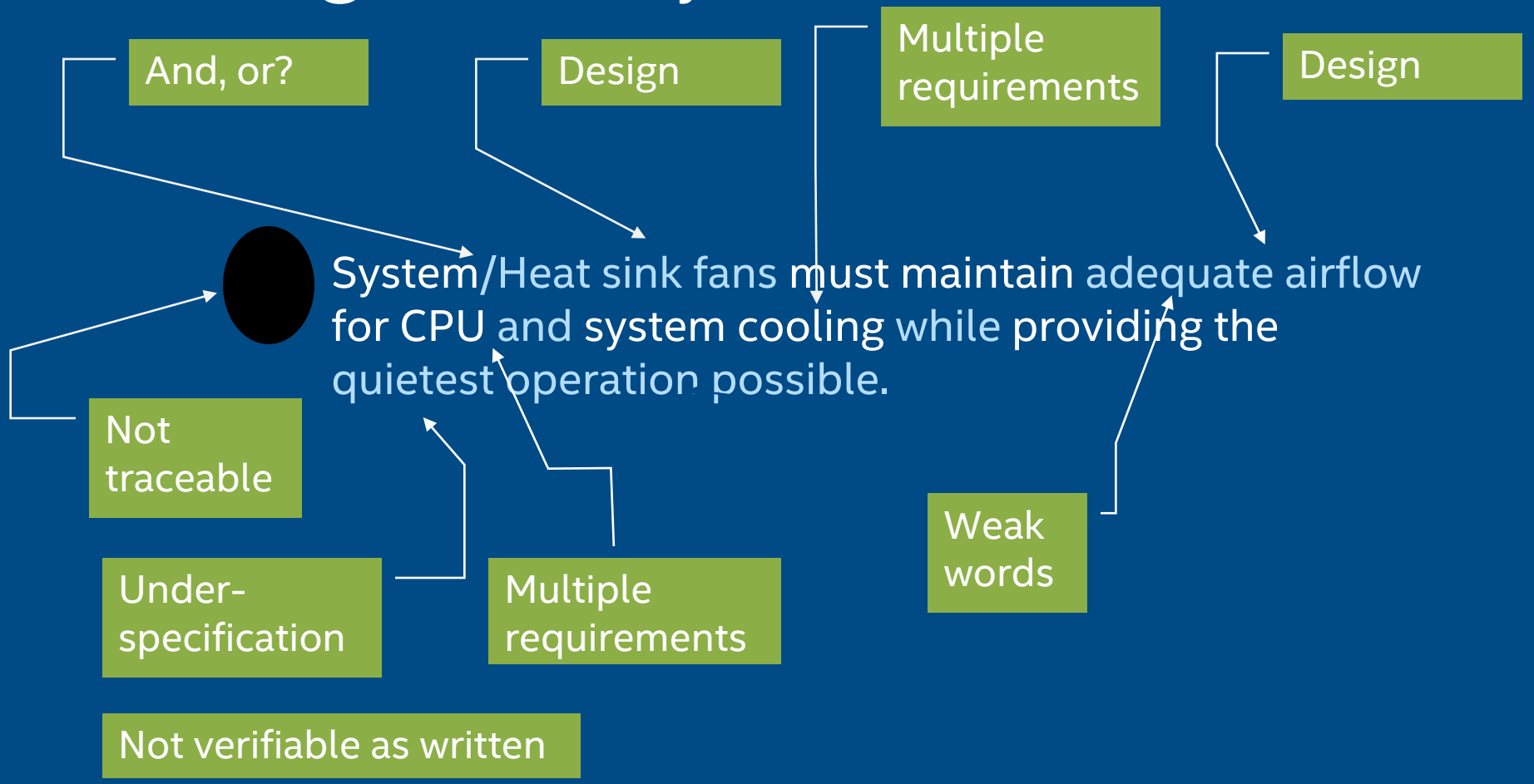
- The Good Requirements Checklist provides a objective guidelines for authors and reviewers to follow
    - Defect Prevention: Authors write requirements with the principles in mind
    - Defect Removal: Reviewers use the checklist to identify rule violations
- Techniques such as formal inspection, or Specification Quality Control (SQC) add more rigor and discipline to quality assessment (Gilb, 2006.)
- Metrics can track a team or organization’s requirements maturity and continuous improvement
- Warning: A perfectly-crafted requirement may nevertheless be incorrect, inaccurate, or otherwise not appropriate for a specification

What's Wrong With My Requirements?

System/Heat sink fans must maintain adequate airflow for CPU and system cooling while providing the quietest operation possible.

See anything wrong?...

A Lot is Wrong, Actually…



Objective Requirements Quality Criteria

- Complete: A requirement is complete when it contains sufficient detail for those that use it to guide their work
- Correct: A requirement is correct when it is error-free
- Concise: A requirement is concise when it contains just the necessary information, expressed in as few words as possible
- Feasible: A requirement is feasible if there is at least one design and implementation for it
- Necessary: A Requirement is necessary when it:
  - Is included to be market competitive
  - Can be traced to a stakeholder need
  - Establishes a new product differentiator or usage model
  - Is dictated by business strategy, roadmaps, or sustainability
- Prioritized: A requirement is prioritized when it is ranked or ordered according to its importance
- Unambiguous: A requirement is unambiguous when it possesses a single interpretation
- Verifiable: A requirement is verifiable if it can be proved that the requirement was correctly implemented
- Consistent: A requirement is consistent when it does not conflict with any other requirements at any level
- Traceable: A requirement is traceable if it is uniquely and persistently identified

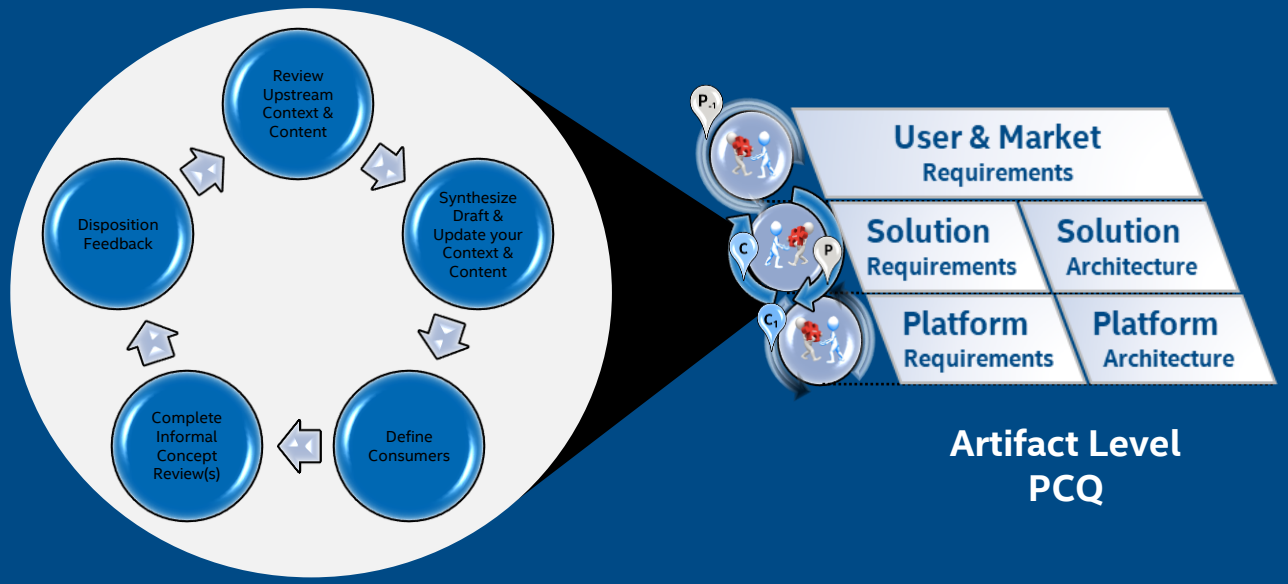
Basic Practice: Count rules that are violated. **Intermediate Practice:** Count every individual issue.

Subjective Requirements Quality

Requirements **completeness** is assessed by the consumers of the work products - *not by the authors*!

- It is impossible to automatically or objectively determine “doneness”
- Requirements must be sufficient to drive work forward at an acceptable level of risk for any given project or program.
- Subjective requirements quality requires recognition of the interdependence between the producers and the consumers of data.
- Subjective requirements quality assessments implicate both good authoring practices and strong requirements management.
- Metrics can measure and guide continuous improvement efforts with subjective Verification practices as well.

Producer-Consumer Quality Exchange (PCQX)



Detailed Continuous Informal Feedback Cycle

Artifact Level PCQ

- What is the source content? Who produces it, where, and in what format?
- Preconditions: Who produces the data that I consumed to produce my requirements? Who consumes what I produce to do their job?
- Postconditions: What did I produce? Who reviewed or approved it? Did all named reviewers complete a review? Was any feedback dispositioned? Was a final version of my content Approved?
- Recorded: Reviewers and Approvers are accountable to the Producers and program for their work.

Informal feedback early in requirements generation increases the likelihood of a better formal review and approval process later.

Techniques for Requirements Verification

Both objective and subjective requirements assessment can lead to improved requirements quality

- Many review techniques can be employed for either Objective or Subjective Requirements Verification.
- Which technique(s) to choose depends on factors including team maturity, data maturity, and risk tolerance for errors or lack of shared interpretation
- Almost any review *practice* – checklist or PCQX – is better than NO requirements review
- “Please review this and send me your feedback” – with no required reviews, no accountability for reviewers, no record of disposition of feedback, and no approval – can be worse than no review at all!

Review Methods: Pros and Cons

	Pros	Cons
Informal Review	• Flexible • Least threatening	• Finds fewer defects than other types • Variable, inconsistent results
Walkthrough	• More systematic than reviews • Identifies defects reviews miss	• May lack follow-up • More time intensive and inconvenient than reviews
Inspection	• Most defects located • Controlled, repeatable • Industry proven practice	• Intimidating to some • Requires training [94] • Can be too much effort without sampling

Inappropriate Data Use

There is one simple way to make any Requirements Verification process fail: Use the data for performance appraisal

When quality improvement data is misused in this way, teams will work around the review system by reporting issues privately to authors, or skipping reviews completely

Once someone loses trust in the peer review process, it is extremely hard to restore

Use of peer review data for performance appraisal is unacceptable – find other ways to measure individual performance

Beginning Distance Learning Session #4

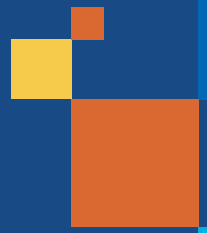
Systems and Requirements Engineering

Requirements Engineering

Distance Learning Session #4 – Requirements Management Overview

Version 9.2
October 2020

sarah.c.gregory@intel.com



intel®

Contents

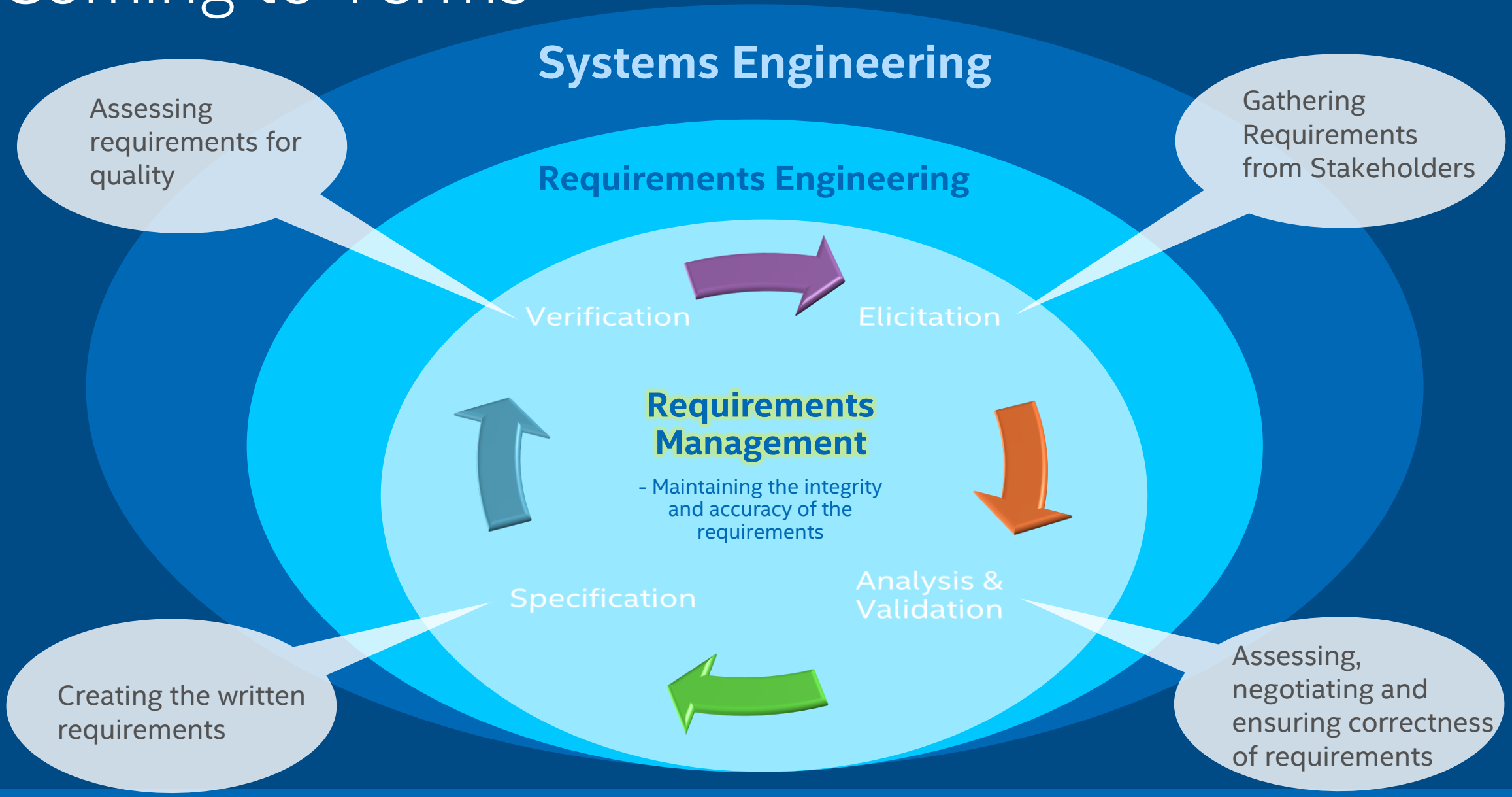
This is NOT the complete RM training, but an overview of the Activity.
Complete RM training is ~1-2 days long, hands-on with requirements.

Objectives

At the end of this session, you should be able to:

- Define and understand requirements management and requirements baselines
- Understand the basics of configuration management and how requirements management fits within the process
- Describe the essential capabilities of RM tools
- Know where to find more information on the topics presented

Coming to Terms



What is Requirements Management?

Requirements management encompasses those tasks that record and maintain the evolving requirements and associated context and historical information from the requirements engineering activities. Requirements management also establishes procedures for defining, controlling and publishing the baseline requirements for all levels of the system-of-interest.
ISO/IEC/IEEE 29148-2018

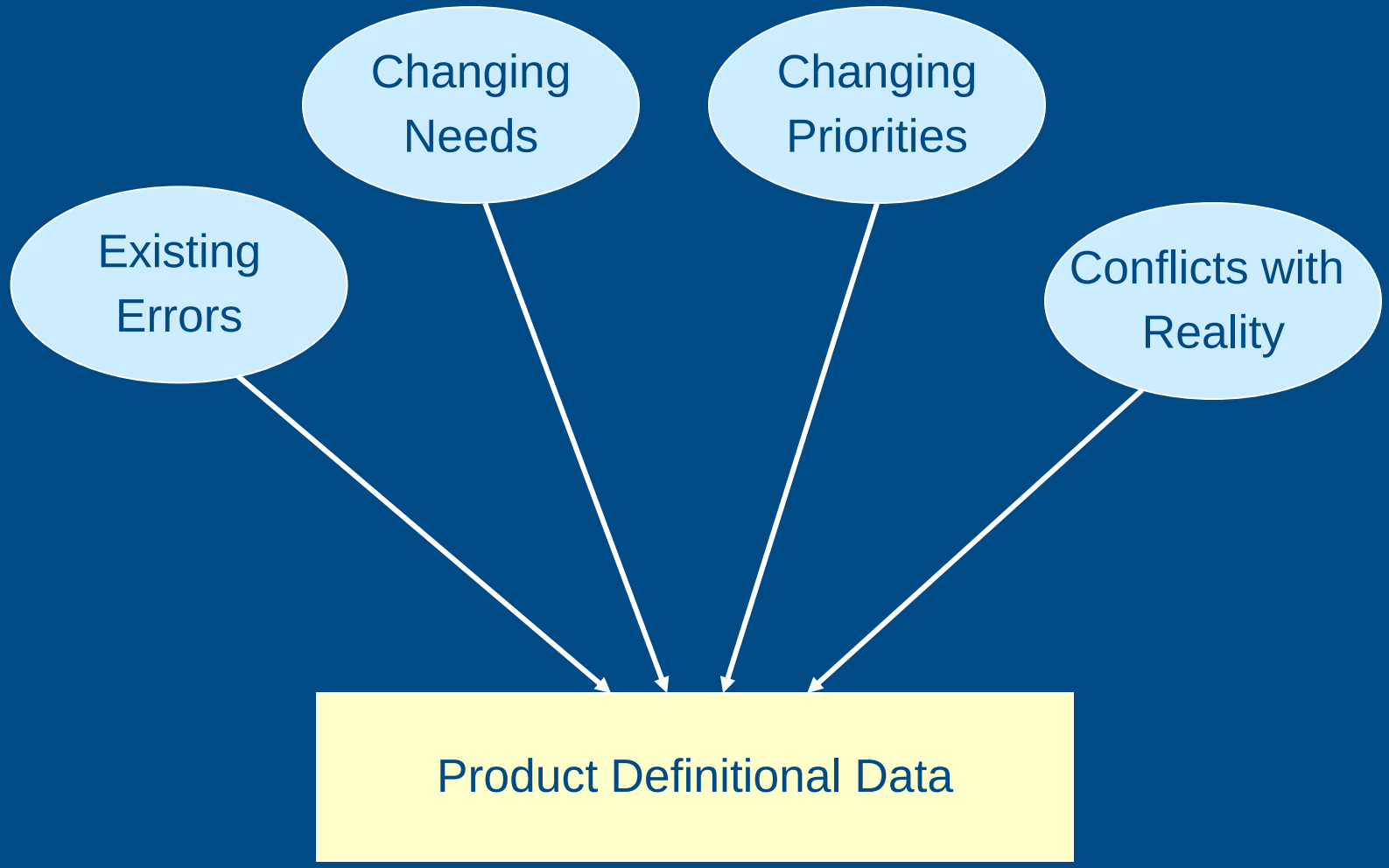
Requirements Management critically depends on the practice of **Configuration Management** (CM) and helps manage product scope

Problem Statement

Unmanaged and poorly managed changes to product requirements lead to mistakes, unnecessary features, and expensive rework.

- In your experience, how much scope creep occurs on a typical project? What was the effect?
- In your experience, what is the most common source of changes to requirements? Could the changes be prevented?
- What are the rate and volume with which Requirements Change Requests (RCRs) come in post-PRQ in your organization?

Sources of Requirements Change



103

Common Secondary Causes of Changes

- Requirements were not written well (or at all)
- Customer needs or input were not adequately determined when defining requirements
- Cost and impact of requirements change on product items and project activities were not understood
- Changes to requirements were not communicated to all stakeholders
- *No requirements baseline*, or a "baseline" was declared but left unmanaged and uncontrolled

No Baseline = No Requirements Management

What is a Baseline?

A formally reviewed and approved version of a work product that serves as the basis for future development and can be modified only through a defined, controlled process

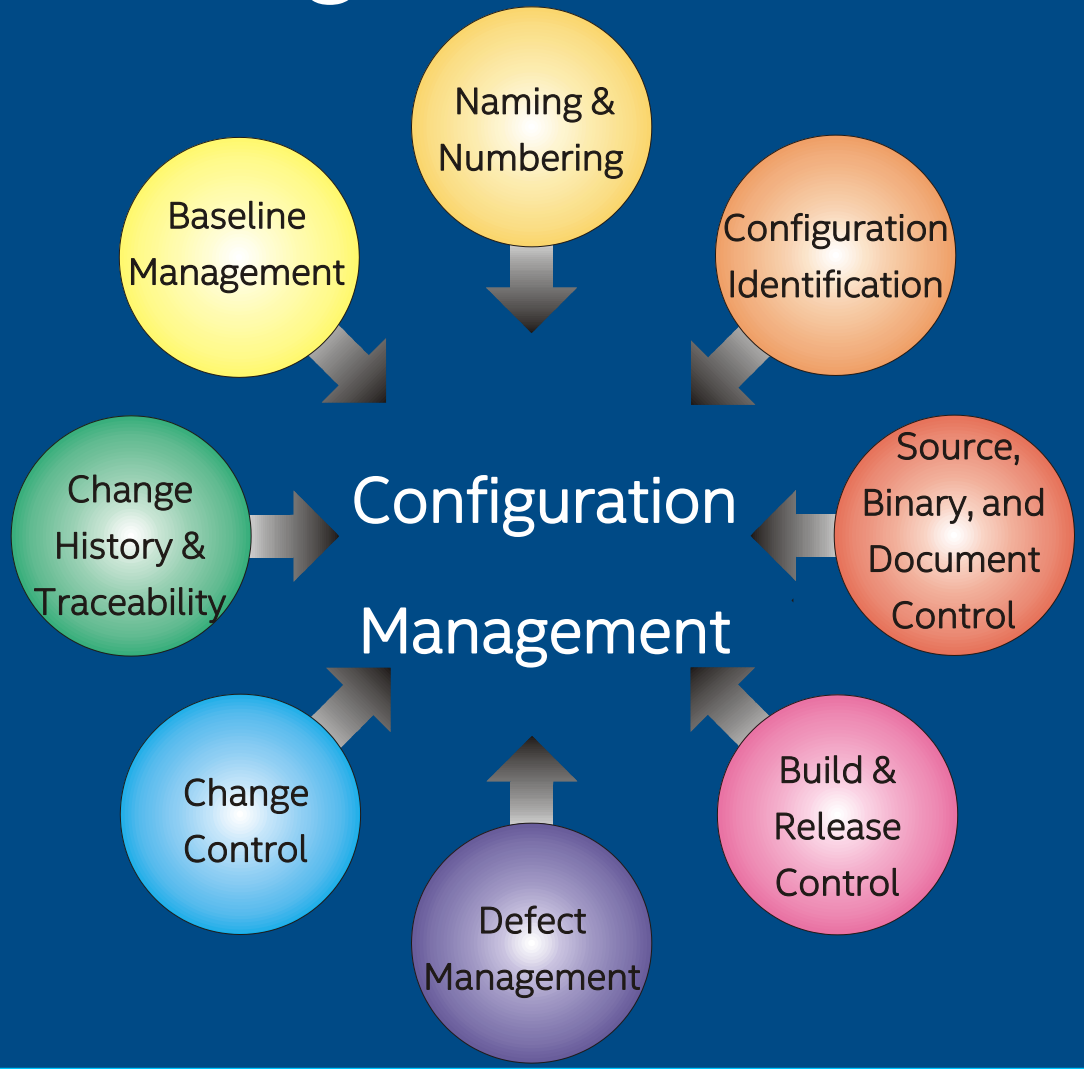
- Baselines can be established for all product definitional data
- Baselines MUST be established to define POR for a PLC milestone
- The Producer/Consumer Quality Exchange (PCQX) process is designed to coordinate steps leading to and including the formal review and approval step that establishes a baseline.
- The version of data that is baselined is established through a Configuration Management process
- *Agile programs still follow similar practices, but they're handled differently*

What is Configuration Management?

The process of identifying and defining the items in the system, controlling the change of these items throughout their lifecycle, recording and reporting the status of items and change requests, and verifying the completeness and correctness of items. (IEEE 828:2012)

The discipline of identifying the configuration of a system – hardware, software, or process - at discrete points in time with the purpose of systematically controlling changes to the configuration and maintaining the integrity and traceability of the configuration throughout the system lifecycle (Thayer and Thayer)

Configuration Management Activities

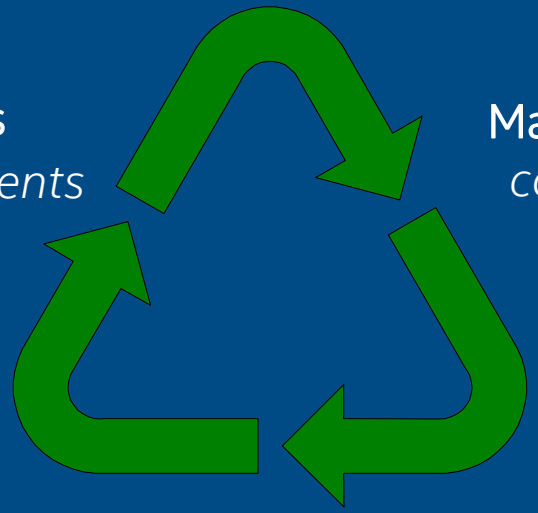


Configuration Management is a CRITICAL capability of Requirements Management.

No CM = No RM.

Configuration Management for Requirements

Engineering Functions
managing the requirements



Management Functions
controlling the process

Communication Functions
capturing and distributing information

Configuration Management for Requirements

Managing the Requirements

- Managing requirements defects and change requests
- Tracking and controlling revisions to requirements
- Managing and controlling the requirements baseline

Configuration Management for Requirements

Controlling the Process

- Policy enforcement
- Timing and sequence of procedures and tasks
- Assessment of data workstates at milestones
- Identification of roles, responsibilities, and owners
- Audits of procedural effectiveness and compliance

Configuration Management for Requirements

Capturing and Distributing Information

- Baseline composition and distribution
- Change history
- Platform and other requirements dependencies
- Notification of change and requests for change
- Requirements metrics

Requirements Traceability & Tools

Essential Knowledge

What is Traceability?

The ability to describe and follow the life of a requirement, in multiple directions, and understand its relationship to other product definitional data – including to related products in a product line.

Requirements can be traced to and from:

- Customer asks and value propositions
- Usage components, including scenarios and use cases
- Architecture, design, and implementation
- Test cases, documentation, change requests
- BOM data, IP, customer feedback

Why Trace Requirements?

- Enables verification that all necessary features are contained in the requirements
- Makes the relationship between customer asks, value propositions, product features, use cases, requirements, design, and other execution data visible
- Helps ensure full test coverage of product features
- Helps guarantee that no unnecessary or conflicting features are built
- Eases the requirements change process by:
  - ✓Identifying associated items to be changed
  - ✓Simplifying impact assessment

Traceability and Change Management

Traceability makes these questions easier to answer as part of the change management process:

- What are the potential positive and negative consequences for a change?
- Which customer asks are implicated by a change?
- What use cases, design documents, test cases, and other project items would be affected by the change?
- How much rework would need to be done, and where?
- Is the change justified?

Traceability also enables accurate tracking and assessment of change completion

Requirements Engineering Tools

Requirements Engineering Tools:

- Are necessary to enable traceability in the complex Intel context
- Coordinate change and version control activities
- Capture requirements attributes in a database
- Enable re-use of requirements
- Help control and distribute the requirements baseline(s)
- Assist with impact analysis (especially through traceability)

Requirements Engineering Tools

Common Tool Features

All standards-aligned and functional RM tools MUST have:

- Good to excellent functionality for requirements tracing, including across variants
- Configuration Management capabilities that enable versioning of sets of data
- The capability to create an immutable version of a collection of requirements (baseline)
- The ability to assemble different versions of baselines to create variants of a product across a product line.
- Interfaces that enable integration with other engineering tools

Support varies for some functions:

- Full integration of RM, PLM, and Agile capabilities (including SAFe/Agile@Scale)
- Modular and granular access control, including incorporation of “roomed” data
- Model-Based Systems Engineering

Requirements Engineering Tools

Harsh Realities

Requirements traceability is impossible without a tool or integrated set of tools *that possess the needed capabilities*

RM tools are essential when used correctly, but can require significant training and effort (RE is an engineering discipline!)

Great tool + poor process = certain failure

Inadequate or highly-tailored solutions invariably result in degradation of practice, sometimes to the point that RM is no longer practiced at all!

No tool is a panacea; tools make managing requirements *easier*, but not necessarily *easy*

Conclusion of Distance Learning Course

Sources for More Information (Part One)

Systems Thinking Made Simple: New Hope for Solving Wicked Problems, Derek Cabrera, Laura Cabrera, Independently published, 2016.

Systems Engineering: Coping with Complexity, Richard Stevens et al, Prentice Hall 1998

Systems Engineering: A 21st Century Systems Methodology, Derek Hitchins, Wiley 2007

Competitive Engineering, Tom Gilb, Elsevier 2005 *(free digital copy available to Intel personnel by request and Tom's courtesy – send email to Sarah Gregory.)*

Shu-Ha-Ri for RE? An Agile Approach to Requirements Engineering Practitioner Maturity, Sarah Gregory, IEEE Software, 12/2019. https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8938120

Acknowledgements

The Requirements Engineering curriculum at Intel Corporation was created by Erik Simmons (1999-2016), work that is still foundational to the courses.

RE@Intel has been housed – formally or informally – in various locations across the company. Present cross-Intel work is supported by courtesy of the Intel Internet of Things Group (IOTG).

Our work evolves steadily through lessons learned with hands-on engagement with Intel product and platform development teams.

We acknowledge with deep gratitude the work of the International Requirements Engineering community, especially the International IEEE RE Conference series, and the REFSQ conference community.


intel