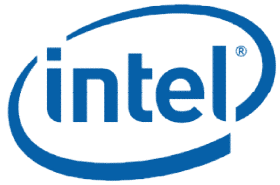


ICCGI Tutorial
Writing Higher Quality Software Requirements

John Terzakis

Intel Corporation

ICCGI Conference
September 20, 2010
Valencia, Spain

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.

Microsoft Trademark Notice:

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Non-Intel Trademark Notice:

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

Agenda

- Requirements, the Requirements Problem & Defects
- Natural Language & Its Issues
- Techniques for Writing Higher Quality Requirements
  - Common Requirements Syntax
  - Ambiguity Checklist
  - 10 Attributes of Well Written Requirements
  - Planguage

Objectives

Upon completing this tutorial, you should be able to:

- Define requirements and their purpose
- Identify common problems with natural language requirements
- Define and understand the ten attributes of a well written requirement
- Use checklists to write and review requirements
- Write higher quality functional and non-functional requirements using Planguage

Requirements, the Requirements Problem & Defects

What is a Requirement?

A **requirement** is a statement of:

1. What a system must do (a system function)
2. How well the system must do what it does (a system quality or performance level)
3. A known resource or design limitation (a constraint or budget)

More generally, a requirement is anything that drives a design choice

Functional vs. Non-Functional Requirements

There are many ways to classify requirements. The most common way is to divide them into **Functional** vs. **Non-functional*** requirements:

Functional Requirements

- What the product does
- Measured in "Yes/No" terms

Non-Functional Requirements

- The "ilities" (quality, reliability, availability, etc.)
- Performance
- Constraints, user interface, documentation, marketing, localization, legal, etc.
- Most are measured on some interval, but some are a simple "Yes/No"

* Quality & Performance Requirements

The Purpose of Requirements

Requirements help establish a **clear**, **common**, and **coherent** understanding of what the software 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 software is built

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

The Requirements Problem

Poor requirements accounted for **41-56% of errors discovered**, and 5 of the top 8 reasons for project failure (The CHAOS Report, 1995)

IBM and Bell Labs studies show that **80% of all product defects** are inserted at the requirements definition stage (Hooks and Farry, 2001)

Requirements errors consume from **28% to more than 40% of a typical project's budget** (Hooks and Farry, 2001)

122% average schedule *overrun*, 45% of delivered functions *never used* (Standish Group Report, 1995)

Poor requirements lead to requirements defects.

The Requirements Problem: An Example

Customer Requirement: “Build me a house”

Customer Vision

Developer Vision

Defects lead to costly rework, schedule overruns, decreased quality and lower customer satisfaction

Requirements Defects

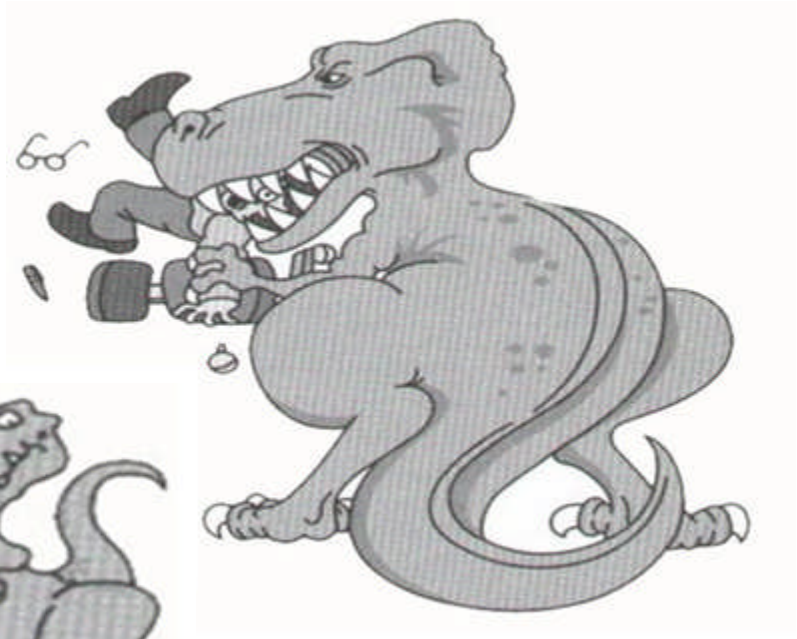
Requirements defects account for the vast majority of the total cost of all defects – often 70% or more (Leffingwell & Widrig, 2003)

- Requirements defects are often the most expensive defects because requirements form the basis of so many other work products (design specifications, code, test plans, etc.)
- Requirements defects account for up to 40% of many projects' total budget (Leffingwell & Widrig, 2003)

We commonly spend too much time developing the wrong thing!

A Bug's Life

(or why it is better correct defects while they are still young)



Production

Testing

Coding

Design

Inspection

Relative Cost to Correct a Defect

Phase	Relative Cost (avg.)
Inspection	1
Design & Coding	10x
Testing	25x
Production	100x or greater

Correcting defects earlier in the SW lifecycle pays huge dividends

If we are to do so, we must improve requirements quality

Natural Language & Its Issues

What is Natural Language?

Natural language is unconstrained, informal language as it is used in every day speech and writing

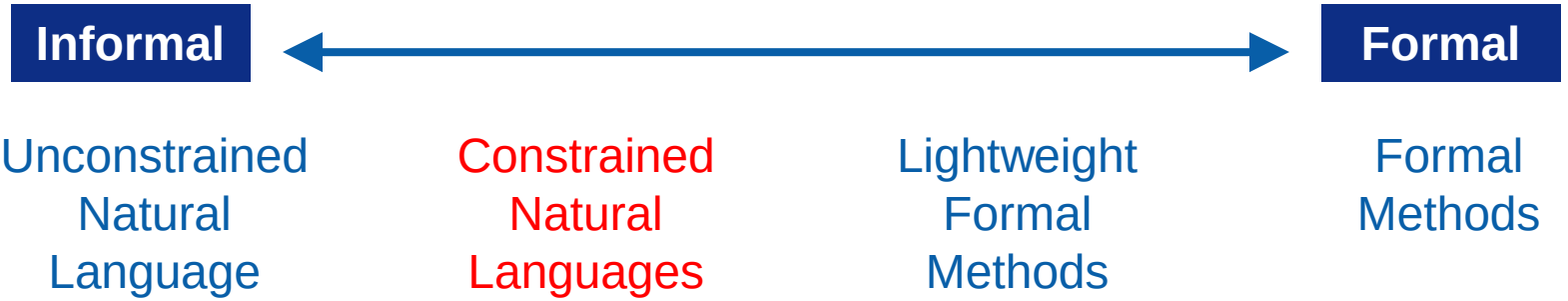
Natural language is the most common medium for expressing requirements in many industries; It is flexible, easy to use and requires no additional training.

Alternatives to natural language include various languages for formal specification, which are rooted in mathematical modeling and set theory.

While they remove ambiguity and allow for correctness proof, formal methods are used mostly within safety-critical or high-reliability applications because of their steep learning curves

A Spectrum of Formality

Natural language and formal methods are end points of a spectrum of formats for requirements

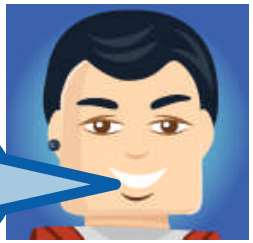


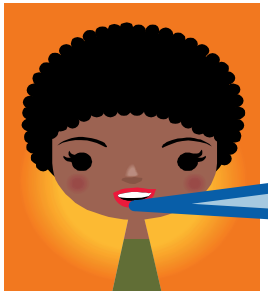
For most applications, constrained natural language is a good balance of reading ease, flexibility, and precision

Examples of Natural Language Requirements

It should be easy to install the software

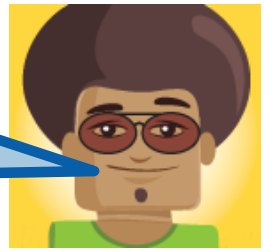
The help files must be available in multiple languages.





Make the web order entry SW user friendly

The account info must be updated



I need support for Windows® 7 or later

*3rd party names and brands may be claimed as the property of others

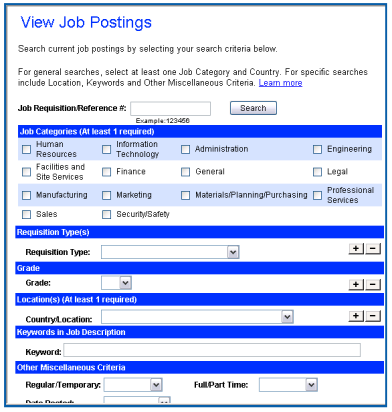
Exercise 1: Writing Natural Language Requirements

Instructions:

1. Form teams of 2 to 4 people
2. As a team, choose one of the two items below and write approximately five natural language requirements that it must satisfy. Use whiteboards or easels if available.
3. Be prepared to briefly share your results with the class.

Can Opener

Online job application software



Issues with Natural Language Requirements

While useful in everyday interactions, natural language is fertile ground for a number of issues relating to requirements as highlighted on the previous slide including:

- Weak words
- Unbounded lists
- Ambiguity

Natural language tends to produce requirements that lack a Clear, Common and Coherent understanding, leading to defects

Weak Words

Weak words are subjective or lack a common or precise meaning.

Examples include:

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

Weak Words: Examples

- The software must load quickly.
  - How quick is “quickly”? ½ second? 1 minute? 8 hours?
- It must be effortless to upgrade the software.
  - How can something be “effortless”? How can we test for it?
- The software must be reliable.
  - What does “reliable” mean? Does it mean “bug free”? Does it mean the software is up 100% of the time?

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

Unbounded Lists

An unbounded list is one that lacks a starting point, an end point, or both

Classic examples include:

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

Unbounded Lists: Examples

- The software must support at least 250 users.
  - How many is “at least” users? 251? 2500? More?

- The software must support Windows® versions including but not limited to Windows® XP.
  - Does this include Windows® XP SP1? SP2? SP3?
  - Does this include Windows® Vista (and its service packs)? Windows® 7?

- The software must work with DOCSIS® 3.0 or later
  - How can we test against a specification (3.5?, 4.0?) that doesn’t exist yet?

Unbounded lists are impossible to design for or to test against

*3rd party names and brands may be claimed as the property of others

Ambiguity

Use of natural language can lead to ambiguity caused by problems like:

- Vagueness
- Subjectivity
- Incompleteness
- Optionality
- Under-specification
- Under-reference
- Over-generalization
- Non-intelligibility
- Coordination ambiguity
- Passive voice
- Time-logic confusion
- Incomplete logic

Ambiguity Terms

- **Vagueness:** is caused by weak words without a precise meaning
- **Subjectivity:** is caused by weak words that rely on personal experience or opinion
- **Incompleteness:** comes from insufficient detail, use of *TBD*, and unbounded lists
- **Optionality:** is caused by use of *should*, *may*, *if possible*, *when appropriate*, etc.
- **Under-specification:** results from use of verbs such as *support*, *analyze*, or *respond,* or implicit collections of objects
- **Under-reference:** consists of incomplete or ambiguous references to other documents, standards, requirements, etc.

Ambiguity Terms

- **Over-generalization:** is caused by use of universal qualifiers such as *all* or *every*, and even unmodified nouns like *users*.
- **Non-intelligibility:** results from poor grammar, complex logic, “and/or” ambiguity, ambiguous negation or enumeration, and missing definitions
- **Coordination Ambiguity:** results from use of a conjunction between a modified noun and a pure noun (among other cases)
- **Passive Voice:** occurs when a requirement does not explicitly name an actor
- **Time-Logic Confusion:** exists when a logical condition is used in place of time-related language
- **Incomplete Logic:** refers to missing logical conditions, such as missing the *else* of an *if-then*

Ambiguity: Examples

- The system must support all current standards for video encoding before launch.
  - Vagueness: Support? Current when? How long before launch?

- When shipping information has been verified, shipping labels must be printed for each container in the order.
  - Passive voice: Verified by whom or what? Printed by whom or what? What part if any does the user play, and how much is automated?

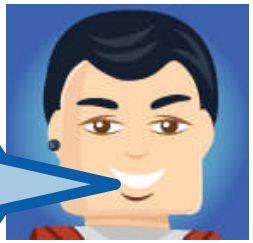
- If automatic calibration fails, then the system shall switch to manual calibration."
  - Incomplete Logic: Is manual calibration allowed if the automatic calibration is functional?

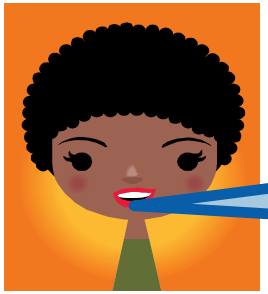
Remove ambiguity to improve understanding

Identifying Natural Language Issues

It should be easy to install the software

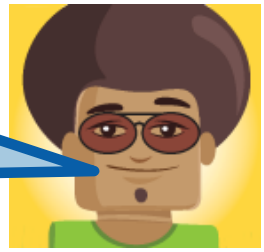
The help files must be available in multiple languages.





Make the web order entry SW user friendly

The account info must be updated



I need support for Windows® 7 or later

Do you see any issues with these requirements?

*3rd party names and brands may be claimed as the property of others

Issues Identified

- It should be easy to install the software
  - "Should" means it is optional. What does "easy" mean? It is subjective (reader dependent)
- The help files must be available in multiple languages.
  - How many is multiple? 3? 5? 75?
- Make the web order entry software user friendly
  - What is "user friendly"? Can we test for it?
- The account information must be updated
  - Who or what is updating the account information? What triggers the update? What account information is being updated?
- I need support for Windows® 7 or later
  - What does "support" mean? Can we "support" future versions of Windows®?

*3rd party names and brands may be claimed as the property of others

Exercise 2: Locate the Natural Language Issues

The usability objective of the AlphaBeta 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.

Techniques for Writing Higher Quality Requirements

Techniques for Writing Higher Quality Requirements

The following techniques will help overcome issues with natural language and lead to higher quality requirements:

- Use a common Requirements Syntax for all requirements
- Write and test requirements using an Ambiguity Checklist
- Write and test requirements against a list of Ten Attributes of Well Written Requirements
- Use Planguage, particularly to quantify non-functional requirements utilizing a Scale and Meter

Requirements Syntax

Requirements Syntax

[Trigger] Actor Action Object [Condition]

- Trigger: *What event causes the action to occur*
- Actor: W*ho or what is taking action*
- Action: *What is going to occur (use imperatives—shall & must)*
- Object: *Who or what is being acted upon*
- Condition: *What event can cause the action not to occur*

It is Clear who or what is taking action (Actor)

There is a Common understanding of what is occurring (Action) and to whom or what (Object)

Use of Imperatives

Shall – Used to dictate functional capabilities

Must – Establishes non-functional requirements and constraints

Should, **May** – Used only in cases where the requirement is optional

Should or May are rare in requirements – what is an "optional requirement", anyway?

Requirements Syntax Example

[Trigger] Actor Action Object [Condition]

Example:

When an Order is shipped, the system shall create an Invoice unless the Order Terms are “Prepaid”.

- Trigger: *When an Order is shipped*
- Actor: *the system*
- Action: *create*
- Object: *an Invoice*
- Condition: *unless the Order Terms are “Prepaid”*

Ambiguity Checklist

Ambiguity Checklist

Use the Ambiguity Checklist below to test for and remove ambiguity from your requirements:

- ✓Vagueness
- ✓Subjectivity
- ✓Incompleteness
- ✓Optionality
- ✓Under-specification
- ✓Under-reference
- ✓Over-generalization
- ✓Non-intelligibility
- ✓Coordination ambiguity
- ✓Passive voice
- ✓Time-logic confusion
- ✓Incomplete logic

Definitions were presented earlier
Examples are in the Backup slides

10 Attributes of Well Written Requirements

10 Attributes of Well Written Requirements

A Well Written 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 slides for a checklist

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.

Specifying Requirements using Planguage

What is Planguage?

- Planguage is an informal, but structured, keyword-driven planning language
- Planguage can be used to create all types of requirements
- The name Planguage is a combination of the words Planning and Language
- Planguage is an example of a Constrained Natural Language
- Planguage was developed by Tom Gilb

Planguage aids communication about complex ideas

Planguage

Planguage provides a rich specification of requirements that results in:

- Fewer omissions in requirements
- Reduced ambiguity and increased readability
- Early evidence of feasibility and testability
- Increased requirements reuse
- Effective priority management
- Better, easier decision making

This tutorial emphasizes Planguage use for requirements, but Planguage has many additional uses, including success criteria, roadmaps, and design documents

Basic Planguage Keywords

Tag: A unique, persistent identifier

Gist: A brief summary of the requirement or area addressed

Requirement: The text that details the requirement itself

Rationale: The reasoning that justifies the requirement

Priority: A statement of priority and claim on resources

Stakeholders: Parties materially affected by the requirement

Status: The status of the requirement (draft, reviewed, POR, etc.)

Owner: The person responsible for implementing the requirement

Author: The person that wrote the requirement

Continued…

Basic Planguage Keywords

Revision: A version number for the statement

Date: The date of the most recent revision

Assumptions: All assumptions or assertions that could cause problems if untrue now or later

Risks: Anything that could cause malfunction, delay, or other negative impacts on expected results

Defined: The definition of a term (better to use a glossary)

Fuzzy concepts requiring more details: **<***fuzzy concept***>**

A collection of objects: **{***item1, item2,* …**}**

The source for any statement: ←

Basic Planguage Keywords are useful for any requirement, and are sufficient for requirements measured as "present" or "absent"

A Simple Planguage Requirement

Tag: Invoice ← {C. Smith, 07/06/05}

Requirement: When an Order is Shipped, the system shall create an Invoice unless the Order Terms are “Prepaid”.

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

Priority: High. If not implemented, it will cause business process reengineering and reduce program ROI by $400K per year.

Stakeholders: Shipping, finance

Author, Revision, Date: Julie English, rev 1.0, 5 Oct 05

Planguage for Non-Functional Requirements

Ambition: A description of the goal of the requirement

Scale: The scale of measure used to quantify the statement

Meter: The process or device used to establish location on a Scale

Minimum: The minimum level required to avoid political, financial, or other type of failure

Target: The level at which good success can be claimed

Outstanding: A stretch goal if everything goes perfectly

Wish: A desirable level of achievement that may not be attainable through available means

Continued…

Planguage for Non-Functional Requirements

Past: An expression of previous results for comparison

Trend: An historical range or extrapolation of data

Record: The best known achievement

Notes on the keywords:

- Use the keywords that add value to your statement - no more, no less
- There are many more keywords to Planguage than presented here – See the backup slides for some additional examples
- Extend Planguage as needed with new keywords - but it's good to check to see whether there is already a keyword that will work

Scales & Meters

Non-functional requirements need to be quantified in order to be verifiable. Using a Scale and Meter provides that quantification.

- Scale: The scale of measure used to quantify the statement
- Meter: The process or device used to establish location on a scale

Scale	Definition	Example
Natural	A scale with obvious link to the measured quality	Time from power switch depressed to BIOS complete
Constructed	A scale built to directly measure a quality	User satisfaction where 5 = most satisfied and 1 = least satisfied
Proxy	A scale with an indirect measure of a quality	Estimated time to system failure

Quantifying Learnability Using Planguage

Tag: Learnable ←{C. Smith, 07/08/05}

Ambition: Make the system easy to learn ← VP marketing

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

Outstanding: No more than 3 minutes

Past: 11 minutes ← Recent site statistics

Defined: Novice: A person with less than 6 months experience with Web applications and no prior exposure to our Website.

Using Qualifiers

Qualifiers are expressed within square braces [ ] and may be used with any keyword

- They allow for conditions and events to be described, adding specificity to a requirement
- They most often contain data on *where*, *when*, etc.

Example

Past: [1st **quarter average, all orders, all regions, new customers only**] 11 minutes ← Recent site statistics

instead of

Past: 11 minutes ← Recent site statistics

Using Qualifiers

Qualifiers can be created within Scales by using specific language:

Scale: Average time required for **a stated class of user** to complete a 1-item order using only the online help system for assistance

Meter: Measurements obtained on 100 users during user interface testing

Minimum: [**Novice**] 7 minutes, [**Intermediate**] 3 minutes, [**Advanced**] < 1 minute

Placing qualifiers into a Scale forces all targets to be written in terms of those qualifiers

Exercise 3: Using Planguage to Rewrite a Requirement

Instructions:

1. Form teams of 3 to 5 people
2. Use Planguage to rewrite the requirement found on the next slide
3. A basic template has been provided for your use, but don't let its content or format limit your thinking…
4. Be ready to share your work with the class when done

Skill taught: Use the power of Planguage to improve an existing requirement

Exercise 3

Use Planguage to improve the following requirement:

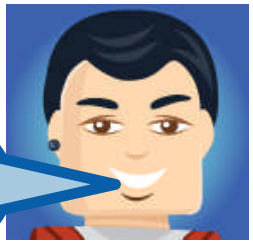
The second key requirement is that the acoustic noise generated by the PC be at levels similar to common consumer electronics equipment. Based on OEM feedback, this acoustic noise level while the PC is active (HDD active) needs to be in the range of 25-33dB. OEM1 shared the progress they have made in this area. They have moved from 38dB active in 1996 to 33dB active in 1997. Their goal for the platform is to maintain less than 33dB. OEM2's requirement is 25dB during active state.

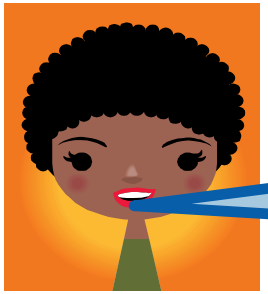
Back to the Beginning

Examples of Natural Language Requirements

It should be easy to install the software

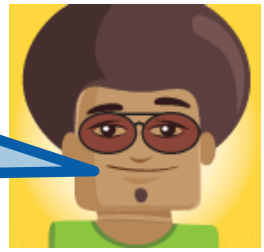
The help files must be available in multiple languages.





Make the web order entry SW user friendly

The account info must be updated



I need support for Windows® 7 or later

*3rd party names and brands may be claimed as the property of others

Back to Natural Language Requirements
Here's How to Rewrite Them

It should be easy to install the software

Tag	SW_Install_Time
Ambition	Make the software easy to install
Scale	Minutes measured with a stop watch
Meter	An **Experienced User** using only the installation manual.
Minimum	15 minutes or less
Target	10 minutes or less
Defined	**Experienced User**: A user who has installed the previous version of the software
Priority	High
Source	Customer visits at top 5 OEMs during June 2010
Auth/Date/Rev	John Terzakis, 3 Aug 2010, 1.0

Back to Natural Language Requirements
Here's How to Rewrite Them

The help files must be available in multiple languages

Tag	Help_File_Localization
Requirement	The help files must be translated from English into the following languages:
Minimum	French, Spanish, Dutch, German
Target	Minimum plus Greek
Outstanding	Target plus Korean and Japanese
Source	Requests from web site during customer survey
Stakeholders	Documentation team, Localization team
Priority	High
Rationale	Localization will increase market share in the indicated geographies
Auth/Date/Rev	John Terzakis, 3 Aug 2010, 1.0

Back to Natural Language Requirements
Here's How to Rewrite Them

Make the web order entry software user friendly

Tag	Usability
Ambition	Make the order entry software user friendly
Scale	minutes measured by the web software
Meter	A group of 50 **Novice Users** using only the online help files for assistance
Target	5 minutes or less
Defined	**Novice User**: A user who has not placed an order previously
Priority	Med
Source	Web statistics & survey from Q1 & Q2 2010
Rationale	Need to improve customer satisfaction based on survey
Auth/Date/Rev	John Terzakis, 3 Aug 2010, 1.0

Back to Natural Language Requirements
Here's How to Rewrite Them

The account information must be updated

Tag	Update_Account_Information
Requirement	When an order is approved, the purchasing software shall update the following account information: • Purchaser • Amount • Order date • Estimated ship date • Part number
Priority	High
Source	Accounting
Stakeholders	SW Team, QA Team, Accounting Team
Rationale	This information is needed for the invoice
Auth/Date/Rev	John Terzakis, 3 Aug 2010, 1.0

Back to Natural Language Requirements
Here's How to Rewrite Them

I need support Windows® 7 or later

Tag	Win7_Support
Requirement	The application software shall install and pass all functional validation testing under Microsoft® Windows® 7
Priority	High
Source	Accounting
Stakeholders	SW Team, QA Team, Sales Team
Rationale	Product revenue will be impacted without this requirement
Auth/Date/Rev	John Terzakis, 3 Aug 2010, 1.0

Wrap up

Tutorial Summary

In this tutorial we have:

- Defined requirements and the purpose of requirements
- Discussed the requirements problem, defects and the impact of defects on software
- Defined natural language and identified its issues
- Introduced techniques for writing higher quality requirements:
  - Use a common Requirements Syntax for all requirements
  - Write and test requirements using an Ambiguity Checklist
  - Write and test requirements against a list of Ten Attributes of Well Written Requirements
  - Use Planguage to write requirements

A Bug's Life

(with Higher Quality SW Requirements)

Inspection

Design

Coding

Testing

Production

Stop those bugs from developing into monsters!

Final Thoughts

Following these techniques, regardless of your development methodology, will produce higher quality requirements that result in fewer “downstream” defects.

The net result will be less rework, more stable code, faster Time To Market and higher customer satisfaction levels.

Contact Information

Thank You!

For more information, please contact:

John Terzakis

john.terzakis@intel.com

Backup

Examples of Ambiguity Issues with Natural Language

- **Vagueness**
  - "The system must support all current standards for video encoding before launch."
- **Subjectivity**
  - "A user must be able to easily and seamlessly transfer media between connected devices."
- **Incompleteness**
  - "The system must support at least 50 concurrent users."
- **Optionality**
  - "The system should include as many end-user help mechanisms as possible."
- **Under-specification**
  - "The software must support 802.11a, b, g, and other network protocols supported by competing applications."

Examples of Ambiguity Issues with Natural Language

- **Under-reference**
  - Users must be able to complete all previously-defined operations in under 5 minutes 80% of the time."
- **Over-generalization**
  - "All users must be able to delete all data they have entered."
- **Non-intelligibility**
  - "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."
- **Coordination Ambiguity**
  - "The system shall allow automated updates and deletions"
  - "The system must display categorized instructions and help documentation"

Examples of Ambiguity Issues with Natural Language

- **Passive Voice**
  - "When shipping information has been verified, shipping labels must be printed for each container in the order."
- **Time/Logic Confusion**
  - "If two orders are received from the same customer for the same part, the system shall follow the process described below."
- **Incomplete Logic**
  - "When automatic calibration fails, the system shall switch to manual calibration."

Examples of Scales and Meters

Tag: Environmental Noise
Scale: dBA at 1 meter
Meter: Lab measurements performed according to a <standard environmental test process>

Tag: Software Security
Scale: Time required to break into the system
Meter: An attempt by a team of experts to break into the system using commonly available tools

Tag: Software Maintainability
Scale: Average engineering time from report to closure of defects
Meter: Analysis of 30 consecutive defects reported and corrected during product development

Examples of Scales and Meters

Tag: 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

Tag: Revenue
Scale: Total sales in US$
Meter: Quarterly 10Q reporting to SEC

Tag: Market
Scale: Percentage of Total Available Market (TAM)
Meter: Quarterly market surveys

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

Possible Answer to Exercise 3

Noise: Noise levels similar to CE devices during use ←{*OEM1, OEM2*}

Stakeholders: OEMs, User Centered Design, Engineering

Priority: High. Failure to meet noise requirements will result in loss of design wins and significant revenue shortfall

Scale: dBA in Active State

Meter: "Acoustic Sound Pressure" test, in Environmental Test Handbook

Minimum: [*OEM1*] < 33dBA

Minimum: [*OEM2*] < 25dBA

Target: 25dBA

Trend: [1996 – 1997, *OEM1*] 38dBA → 33dBA

Defined: Active State: Device running under load, disk drive in use

Good Requirements Checklist

Complete:

- ✓ Formal review by domain experts and stakeholders indicates that all necessary material is included.
- ✓ Exceptional behavior is specified (the “else” of the requirement).
- ✓ No “TBDs” remain.
- ✓ The content is detailed enough to drive the current phase of the development process.
- ✓ The content is not arbitrarily or prematurely detailed.
- ✓ It is *economically safe* to proceed.

Correct:

- ✓ Stakeholder/SME review locates no errors.
- ✓ The requirements are consistent with all source materials.
- ✓ The requirements have been reviewed and approved by all appropriate parties.

Also see other attributes for related items.

Good Requirements Checklist

Concise

- ✓ Each requirement addresses a single issue.
- ✓ Rationale, examples, and other supporting data are separated from the requirement.
- ✓ The requirement is expressed using the simplest grammar and as few words as possible.

Feasible:

- ✓ All Requirements are known to be feasible through use in prior products, through analysis, or through prototyping.

Necessary:

- ✓ Each requirement can be traced to at least one of the following:
  - Market Segment Analysis or lateral benchmarking of similar products
  - A need expressed by the customer or end user
  - Planned implementation of a new usage model
  - Business strategy, roadmaps, or sustainability needs
- ✓ All stakeholders agree that each product requirement is necessary.

Good Requirements Checklist

Prioritized:

- ✓ Tradeoffs between requirements are clear.
- ✓ Multiple dimensions have been considered, such as cost, customer value, and development risk.
- ✓ All product stakeholders have provided input to the prioritization process.
- ✓ The requirements are realistically distributed among the priority levels.

Unambiguous:

- ✓ Each requirement is clear to the intended audience, possessing a single interpretation.
- ✓ Terms are defined where necessary and used consistently.
- ✓ The requirements are devoid of weak words (easy, fast, etc.) and unbounded lists (such as, including, …).
- ✓ Diagrams, algorithms, use cases, tables, or other devices are used to reduce ambiguity where appropriate.

Good Requirements Checklist

Verifiable:

- ✓ Each requirement is unambiguous.
- ✓ The implementation of each requirement can be clearly and effectively established via demonstration, inspection, or testing.
- ✓ Non-functional requirements (performance, reliability, etc.) are quantified using an appropriate scale of measure.

Consistent:

- ✓ Each requirement is represented only once in a specification and referenced where needed.
- ✓ Each requirement is internally consistent with other product requirements at its level.
- ✓ Each requirement is externally consistent with requirements at other levels (product, business, market, etc.).

Traceable:

- ✓ Each requirement is uniquely and persistently identified.
- ✓ Each requirement is written as concisely and simply as possible.
- ✓ Each requirement expresses only one function or idea.


intel®
Leap ahead™