UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Departamento de Informática
Universidad Técnica Federico Santa María

I Encuentro Latinoamericano de Ingeniería de Software
ELAIS 2017

Requirements Engineering at the 4th Industrial Revolution

Guilherme Horta Travassos

www.cos.ufrj.br/~ght

COPPE/UFRJ

CNPq

THE CLOVER
2030 ENGINEERING STRATEGY
AN ENGINE TO SURF THE WAVES FOR
CHILE'S DEVELOPMENT

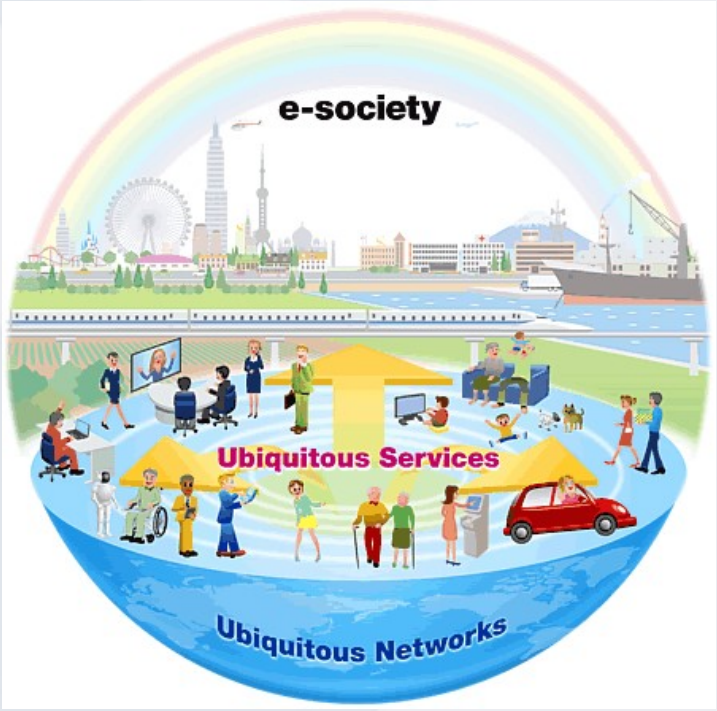
CORFO

State of the Practice

Society's Dependency on Software Systems Products

Computers everywhere demand software products that have made society highly dependent on software services and facilities.

Enormous economic damage and potential human suffering can occur when software systems do not behave as <u>required</u>!



Hardware advancements continue to outpace our ability to build software to tap hardware's potential

Software Quality

It is associated with the

"conformity to requirements"

that have been explicitly declared and specified, to development patterns clearly documented, and to implicit characteristics expected of all professionally developed software."

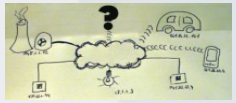
Software Development Perspectives

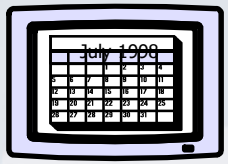
REQUIREMENTS

Loan-Arranger Requirements Specification – Jan. 8, 1999

Background

Banks generate income in many ways, often by borrowing money from their depositors at a low interest rate, and then lending that same money at a higher interest rate in the form of bank loans. However, property loans, such as mortgages, typically have terms of 15, 25 or even 30 years. For example, suppose that you purchase a $150,000 house with a $50,000 down payment and borrow a $100,000 mortgage from National Bank for thirty years at 5% interest. That means that National Bank gives you $100,000 to pay the balance on your house, and you pay National Bank back at a rate of 5% per year over a period of thirty years. You must pay back both principal and interest. That is, the initial principal, $100,000, is paid back in 360 installments (once a month for 30 years), with interest on the unpaid balance. In this case the monthly payment is $536.82. Although the income from interest on these loans is lucrative, the loans tie up money for a long time, preventing the banks from using their money for other transactions. Consequently, the banks often sell their loans to consolidating organizations such as Fannie Mae and Freddie Mac, taking less long-term profit in exchange for freeing the capital for use in other ways.





Tacit requirements

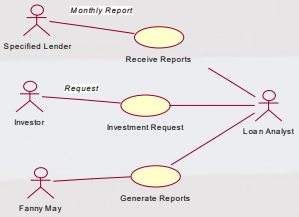
AD-HOC

Problem Domain

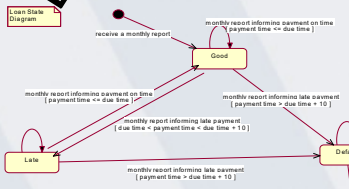
FORMAL

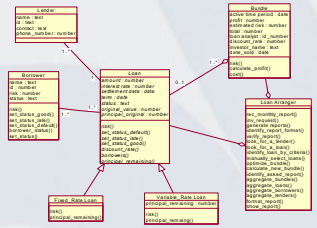
Scalene Triangle:

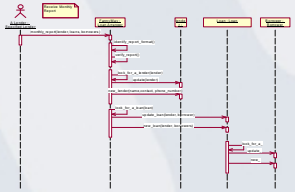
{<x,y,z>: (x != y) ^ (x != z) ^ (y != z)}



Solution Domain







TEST CASES			
CLASS	X	Y	Z
Scalene	3	4	5
Isosceles	5	5	8
Isosceles	3	4	3
Isosceles	4	7	7
Eqüiláteral	2	2	2
No-triangle	1	2	3
No-triangle	5	1	4
No-triangle	3	5	2

Computer Domain

SOURCE CODE

Travassos, G.H. (2014). Software Defects: Stay Away from Them. Do Inspections!. In: 9th International Conference on the Quality of Information and Communications Technology (QUATIC) . Guimarães, Portugal.
http://dx.doi.org/10.1109/QUATIC.2014.8

Software Development Perspectives

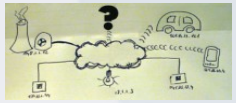
Requirements compose the basis of any well-engineered product!

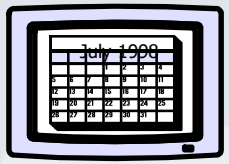
REQUIREMENTS

Loan-Arranger Requirements Specification

Background

Banks generate income in many ways, often by borrowing money from the... at a low interest rate, and then lending that same money at a high... form of bank loans. However, property loans, such as mortgages, typically have... 15, 25 or even 30 years. For example, suppose that you purchase a $150,000 house... a $50,000 down payment and borrow a $100,000 mortgage from National Bank for... thirty years at 5% interest. That means that National Bank gives you $100,000 to pay the... balance on your house, and you pay National Bank back at a rate of 5% per year over a... period of thirty years. You must pay back both principal and interest. That is, the initial... principal, $100,000, is paid back in 360 installments (once a month for 30 years), with... interest on the unpaid balance. In this case the monthly payment is $536.82. Although... the income from interest on these loans is lucrative, the loans tie up money for a long... time, preventing the banks from using their money for other transactions. Consequently,... the banks often sell their loans to consolidating organizations such as Fannie Mae and... Freddie Mac, taking less long-term profit in exchange for freeing the capital for use in... other ways.





Tacit requirements

AD-HOC

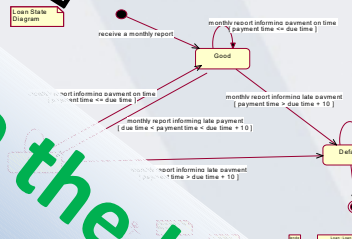
Problem Domain

FORMAL

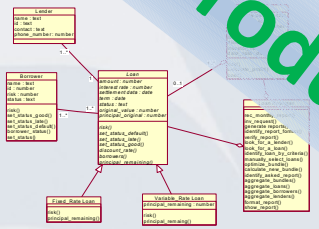
Scalene Triangle:

{<x,y,z>: (x != y) ^ (x != z) ^ (y != z)}

Solution Domain



TEST CASES			
CLASS	X	Y	Z
Scalene	3	4	5
Isosceles	5	5	8
Isosceles	3	4	3
Isosceles	4	7	7
Eqüiláteral	2	2	2
No-triangle	1	2	3
No-triangle	5	1	4
No-triangle	3	5	2



SOURCE CODE

Travassos, G.H. (2014). Software Defects: Stay Away from Them. Do Inspections!. In: 9th International Conference on the Quality of Information and Communications Technology (QUATIC) . Guimarães, Portugal.
http://dx.doi.org/10.1109/QUATIC.2014.8

Software Requirements Concepts

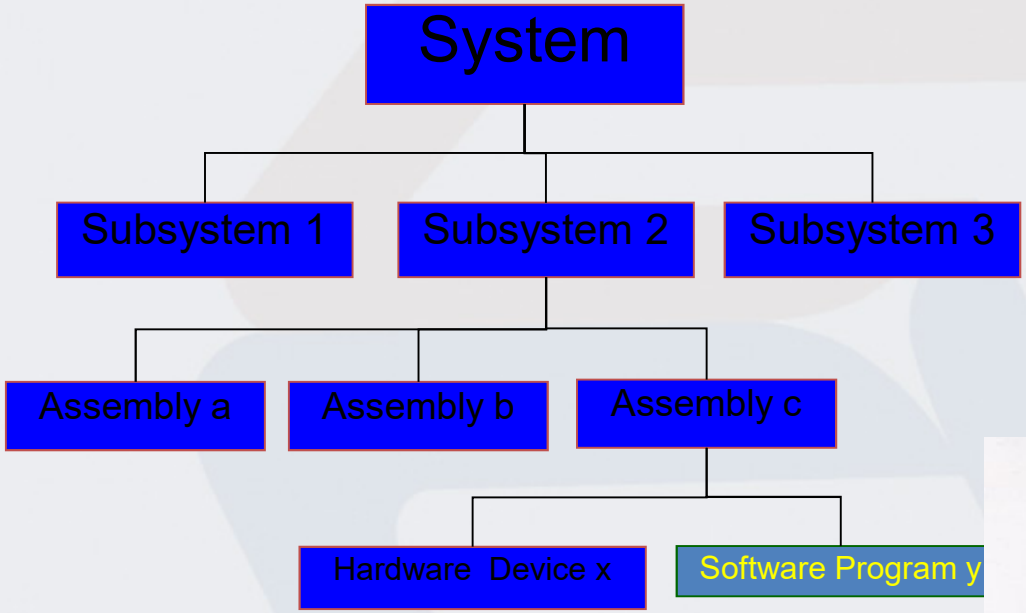
- A **requirement** is (IEEE Std. 610.12-1990):
  - (1) a condition or capability needed by a user to solve a problem or achieve an objective
  - (2) a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document.
  - (3) A documented representation of a condition or capability as in (1) or (2)
    - **Functional requirements:**
      - describe the interactions between the system and its environment
    - **Non-functional requirements:**
      - or constraints, describe the restrictions on the system that limit our technological choices for constructing a solution to the problem

Software Requirements Concepts

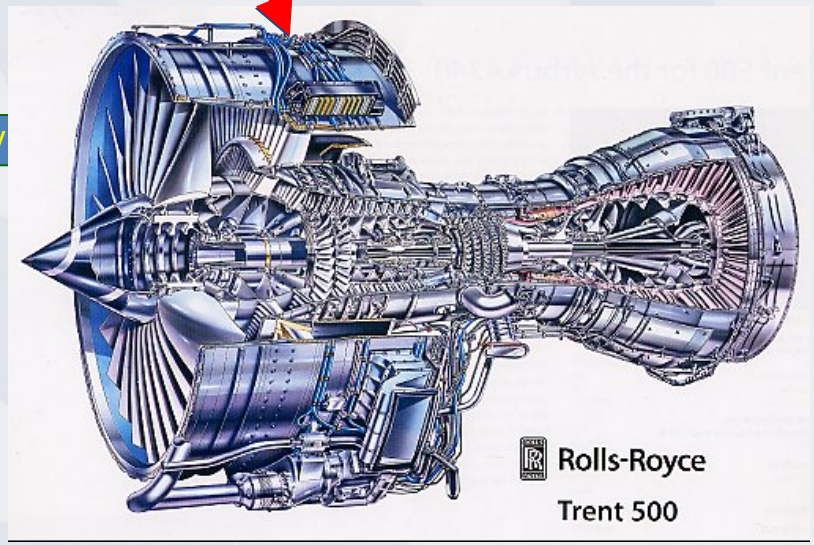
- The **Requirements Description Document** describes all the hardware and software requirements **(uses a understandable customer format)**

  - The **Software Requirements Specification** is the document specifying these requirements **(uses a format better suited for implementation)**

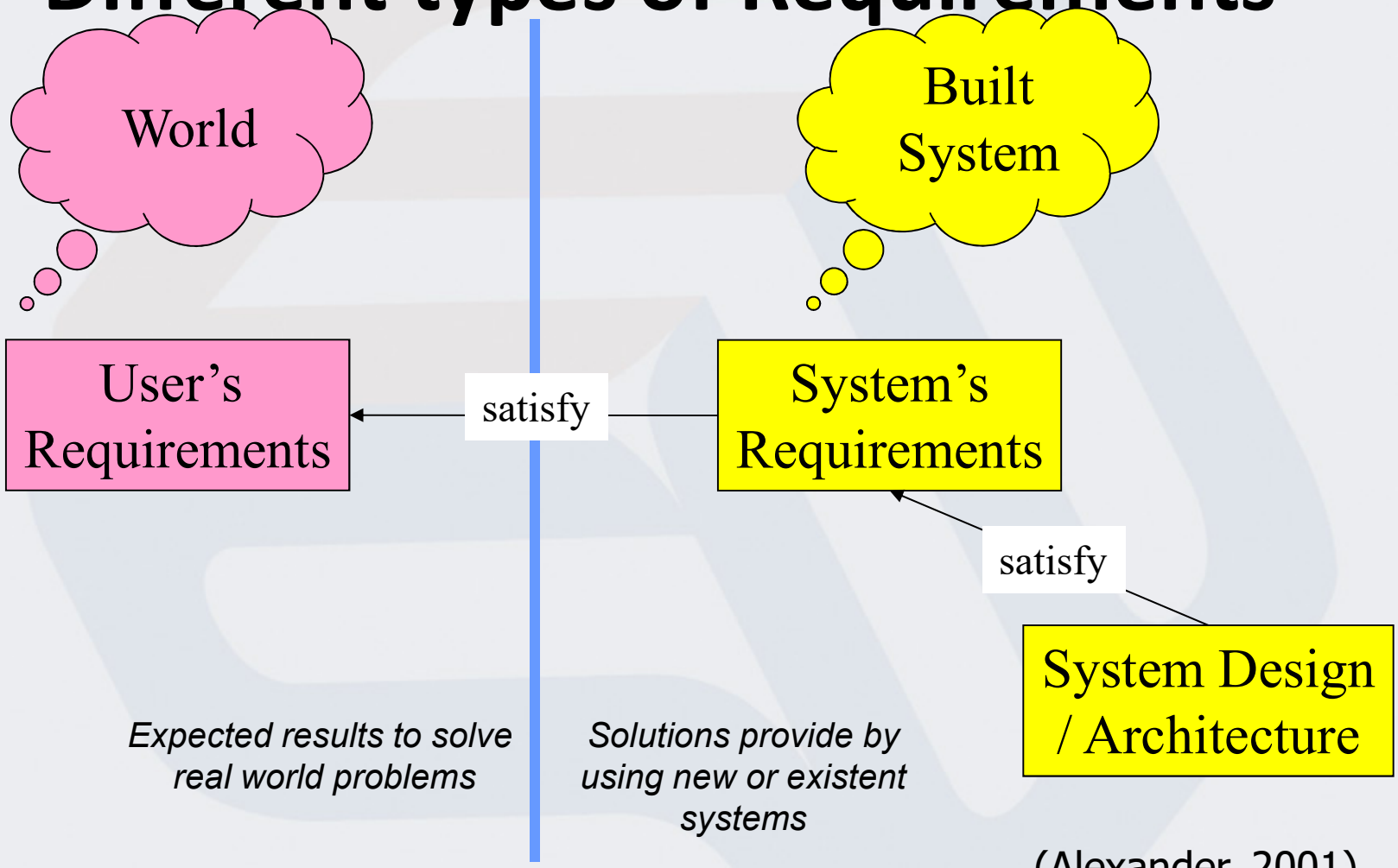
Requirements do not represent just software solutions!



Systems can be composed by subsystems of several types



Different types of Requirements

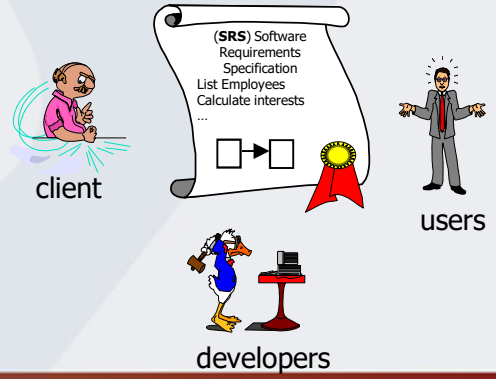


(Alexander, 2001)

… need to be worked in different ways

Importance of Software Requirements

- A (*good*) **Software Requirements Specification** is important because
  - *Establishes the basis for agreement between the customers and the suppliers on what the software product is to do.*
  - *Provides a basis for estimating costs and schedules.*
  - *Provides a baseline for validation and verification.*
  - *Reduces the development effort*
  - *Facilitates to transfer* the software product to new users or new equipment.
  - *Serves as a basis for enhancement.*



Requirements

We need QUALITY!

Therefore, we need

GOOD REQUIREMENTS!

Correct , Feasible, Necessary , Prioritized, Unambiguous, Verifiable

Good Requirements Specifications are…

Correct:

An SRS is correct if, and only if, every requirement stated therein is one that the software shall meet.

Unambiguous:

An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation.

Good Requirements Specifications are…

Complete:

An SRS is complete if, and only if, it includes the following elements:

a) All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces. In particular any external requirements imposed by a system specification should be acknowledged and treated.

b) Definition of the responses of the software to all realizable classes of input data in all realizable classes of situations. Note that it is important to specify the responses to both valid and invalid input values.

c) Full labels and references to all figures, tables, and diagrams in the SRS and definition of all terms and units of measure.

Good Requirements Specifications are…

Complete:

- No requirements or necessary information should be missing. Completeness is also a desired characteristic of an individual requirement. It is hard to spot missing requirements because they aren't there. Organize the requirements hierarchically in the SRS to help reviewers understand the structure of the functionality described, so it will be easier for them to tell if something is missing.

- If you focus on user tasks rather than on system functions during requirements elicitation, you are less likely both to overlook requirements and to include requirements that aren't really necessary. The use case method works well for this purpose. Graphical analysis models that represent different views of the requirements can also reveal incompleteness.

- If you know you are lacking certain information, use "TBD" ("to be determined") as a standard flag to highlight these gaps. Resolve all TBDs from a given set of the requirements before you proceed with construction of that part of the product.

Good Requirements Specifications are…

Consistent:

An SRS is internally consistent if, and only if, no subset of individual requirements described in it conflict.

– Consistent requirements do not conflict with other software requirements or with higher level (system or business) requirements. Disagreements among requirements must be resolved before development can proceed. You may not know which (if any) is correct until you do some research. Be careful when modifying the requirements, as inconsistencies can slip in undetected if you review only the specific change and not any related requirements.

Good Requirements Specifications are…

Ranked for importance and/or stability:

An SRS is ranked for importance and/or stability if each requirement in it has an identifier to indicate either the importance or stability of that particular requirement.

Verifiable:

An SRS is verifiable if, and only if, every requirement stated therein is verifiable.

Good Requirements Specifications are…

Modifiable:

An SRS is modifiable if, and only if, its structure and style are such that any changes to the requirements can be made easily, completely, and consistently while retaining the structure and style.

– You must be able to revise the Requirements Document when necessary and maintain a history of changes made to each requirement. This means that each requirement be uniquely labeled and expressed separately from other requirements so you can refer to it unambiguously. You can make a Requirements Description more modifiable by organizing it so that related requirements are grouped together, and by creating a table of contents, index, and cross-reference listing.

Good Requirements Specifications are…

Traceable:

An SRS is traceable if the origin of each of its requirements is clear and if it facilitates the referencing of each requirement in future development or enhancement documentation.

Backward traceability (i.e., to previous stages of development)
Forward traceability (i.e., to all documents spawned by the SRS)

– You should be able to link each software requirement to its source, which could be a higher-level system requirement, a use case, or a voice-of-the-customer statement. Also link each software requirement to the design elements, source code, and test cases that are constructed to implement and verify the requirement. Traceable requirements are uniquely labeled and are written in a structured, fine-grained way, as opposed to large, narrative paragraphs or bullet lists.

Quality of Requirements

- **Example #1:**

"The product shall provide status messages at regular intervals not less than every 60 seconds."

LOW QUALITY

This requirement is incomplete: what are the status messages and how are they supposed to be displayed to the user? The requirement contains several ambiguities. What part of "the product" are we talking about? Is the interval between status messages really supposed to be at least 60 seconds, so showing a new message every 10 years is okay? Perhaps the intent is to have no more than 60 seconds elapse between messages; would 1 millisecond be too short? The word "every" just confuses the issue. As a result of these problems, the requirement is not verifiable.

Quality of Requirements

- **Example #1:**

"1. Status Messages.

1.1. The Background Task Manager shall display status messages in a designated area of the user interface at intervals of 60 plus or minus 10 seconds.

1.2. If background task processing is progressing normally, the percentage of the background task processing that has been completed shall be displayed.

1.3. A message shall be displayed when the background task is completed.

1.4. An error message shall be displayed if the background task has stalled."

BETTER QUALITY

It is better to split it into multiple requirements because each will require separate test cases and because each should be separately traceable. If several requirements are strung together in a paragraph, it is easy to overlook one during construction or testing.

Quality of Requirements

- **Example #2:**

"The product shall switch between displaying and hiding non-printing characters instantaneously."

LOW QUALITY

Computers cannot do anything instantaneously, so this requirement is not feasible. It is incomplete because it does not state the conditions that trigger the state switch. Is the software making the change on its own under some conditions, or does the user take some action to stimulate the change? Also, what is the scope of the display change within the document: selected text, the entire document, or something else? There is an ambiguity problem, too. Are "non-printing" characters the same as hidden text, or are they attribute tags or control characters of some kind? As a result of these problems this requirement cannot be verified.

Quality of Requirements

- **Example #2:**

"The user shall be able to toggle between displaying and hiding all HTML markup tags in the document being edited with the activation of a specific triggering condition."

BETTER QUALITY

Now it is clear that the non-printing characters are HTML markup tags. This requirement does not constrain the design because it does not define the triggering condition. When the designer selects an appropriate triggering condition, you can write specific tests to verify correct operation of this toggle

Quality of Requirements

- **Example #3:**

"The HTML Parser shall produce an HTML markup error report which allows quick resolution of errors when used by HTML novices."

LOW QUALITY

The word "quick" is ambiguous. The lack of definition of what goes into the error report is a sign of incompleteness. It is hard to be sure how you would verify this requirement. Find someone who calls herself an HTML novice and see if she can resolve errors quickly enough using the report?

Quality of Requirements

- **Example #3:**

"The HTML Parser shall produce an error report that contains the line number and text of any HTML errors found in the parsed file and a description of each error found. If no errors are found, the error report shall not be produced."

BETTER QUALITY

Now we know what needs to go into the error report, but we've left it up to the designer to decide what the report should look like. We have also specified an exception condition: if there aren't any errors, don't generate a report.

Quality of Requirements

- **Example #4:**

"Charge numbers should be validated on-line against the master corporate charge number list, if possible."

LOW QUALITY

One can give up, what does "if possible" mean? If it's technically feasible? If the master charge number list can be accessed on line? Avoid imprecise words like "should." The customer either needs this functionality or he doesn't.

Quality of Requirements

- **Example #4:**

"The system shall validate the charge number entered against the on-line master corporate charge number list. If the charge number is not found on the list, an error message shall be displayed and the order shall not be accepted."

BETTER QUALITY

The customer either needs this functionality or he doesn't. It is possible to observe some requirements specifications in which subtle distinctions are drawn among keywords like "shall", "will", and "should" as a way to indicate priority. It is preferable to stick with "shall" as a clear statement of what is intended by the requirement and to explicitly specify the priorities

Requirements Specifications

We need always more QUALITY!

Therefore, we also need

GOOD
SOFTWARE REQUIREMENTS SPECIFICATIONS!

Correct, Unambiguous, Complete, Consistent,
Ranked for Importance and/or stability,
Verifiable, Modifiable, Traceable

Requirements Specifications

To acquire and warranty QUALITY,

we can follow

GUIDELINES AND TEMPLATES FOR WRITING (good) REQUIREMENTS SPECIFICATIONS!

Guidelines for Writing Quality Requirements

- **Keep sentences and paragraphs short.**
  - Use the active voice.
  - Use proper grammar, spelling, and punctuation.
  - Use terms consistently and define them in a glossary or data dictionary.

- **To see if a requirement statement is sufficiently well defined, read it from the developer's perspective.**
  - Mentally add the phrase, "call me when you're done" to the end of the requirement and see if that makes you nervous.
  - In other words, would you need additional clarification from the SRS author to understand the requirement well enough to design and implement it? If so, that requirement should be elaborated before proceeding with construction.

- **Requirement authors often struggle to find the right level of granularity. Avoid long narrative paragraphs that contain multiple requirements.**
  - A helpful granularity guideline is to write individually testable requirements.
  - If you can think of a small number of related tests to verify correct implementation of a requirement, it is probably written at the right level of detail.
  - If you envision many different kinds of tests, perhaps several requirements have been lumped together and should be separated.

Guidelines for Writing Quality Requirements

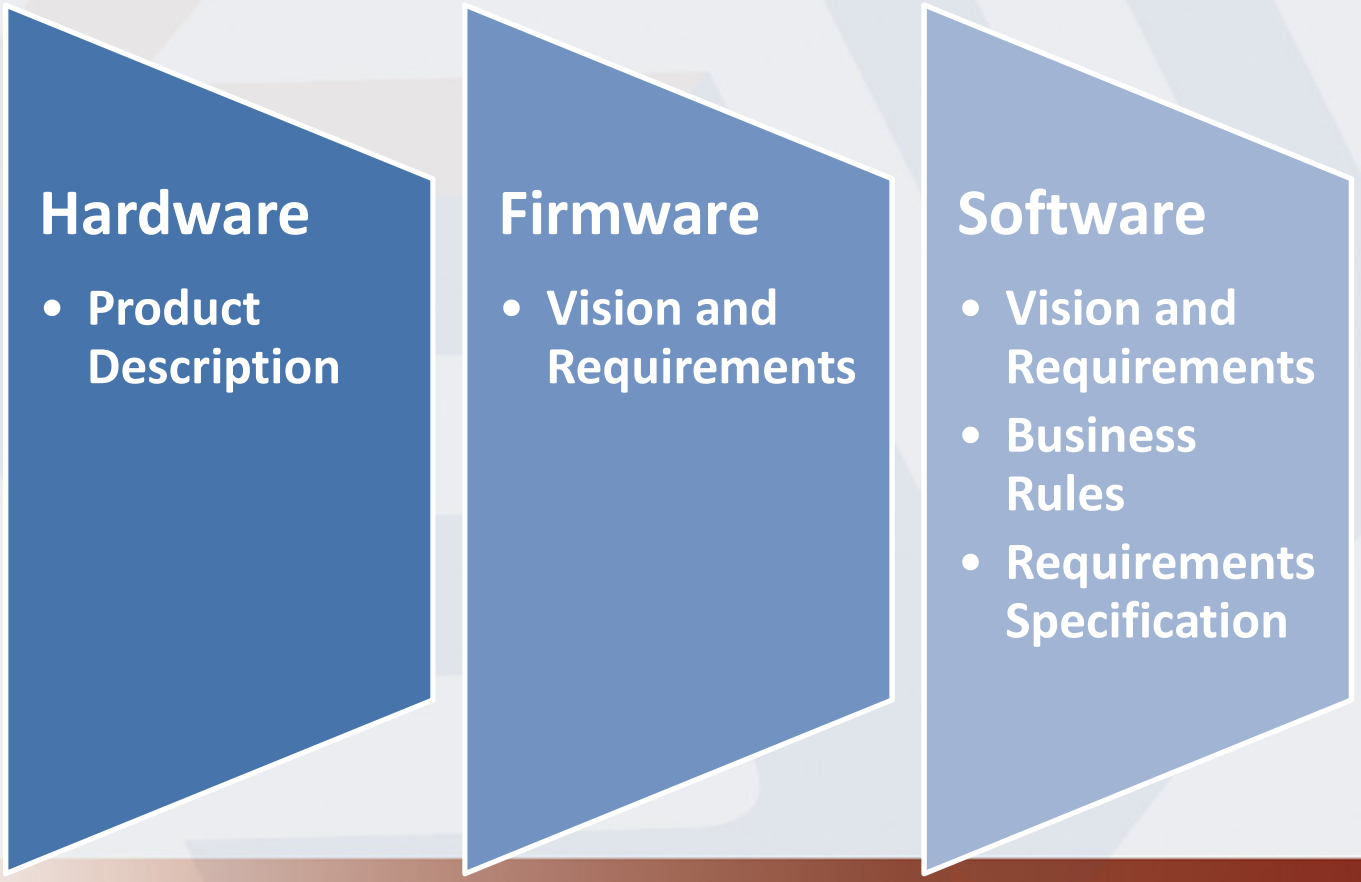
- **Watch out for multiple requirements that have been aggregated into a single statement.**
  - Conjunctions like "and" and "or" in a requirement suggest that several requirements have been combined.
  - Never use "and/or" in a requirement statement.

- **Write requirements at a consistent level of detail throughout the document.**
  - For example, "A valid color code shall be R for red" and "A valid color code shall be G for green" might be split out as separate requirements, while "The product shall respond to editing directives entered by voice" describes an entire subsystem, not a single functional requirement.

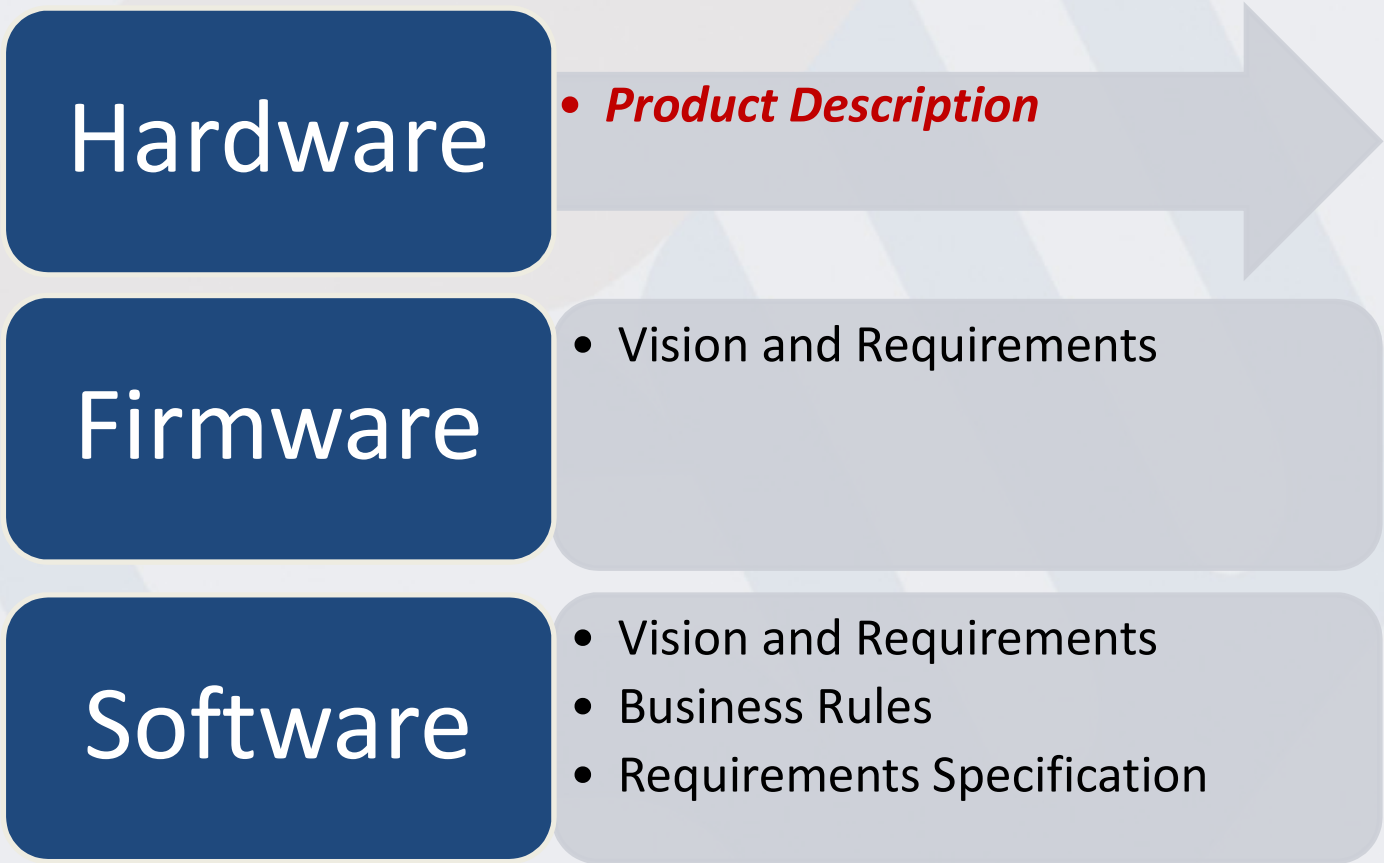
- **Avoid stating requirements redundantly in the Document.**
  - While including the same requirement in multiple places may make the document easier to read, it also makes maintenance of the document more difficult. The multiple instances of the requirement all have to be updated at the same time, lest an inconsistency creep in.

Requirements Documentation in Current Software Projects

- *Templates for 3 product dimensions*



Requirements Documentation

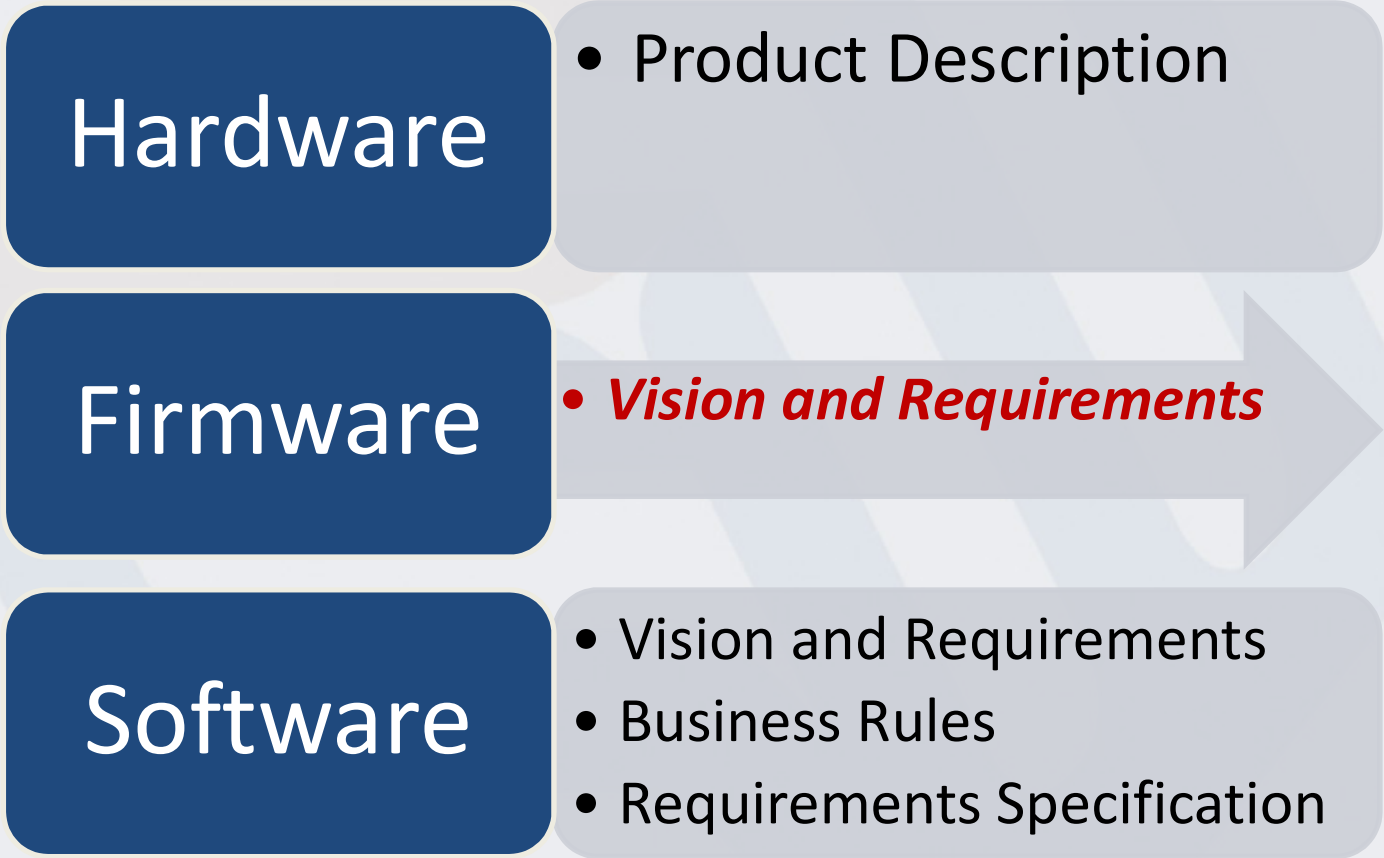


Product Description (hardware level)

Describes the characteristics for the hardware modules that will compose the product.

Acelerômetro	
Sensibilidade	10,2 mV/m/s2 (100 mV/g) ±5%.
Faixa de aceleração	Aceleração total com pico de 490 m/s2 (50 g) dentro da variação de frequência de 10 Hz a 15
Linearidade da Amplitude	±1% a 490 m/s2 (50 g) pico.
Piso do Ruído de Banda Larga (10 Hz a 15 kHz)	0,039 m/s2 (0,004 g) rms.
Sensibilidade de Tensão da Base	49 mm/s2/µstrain (0,005 g/µstrain)
Tensão de Entrada	-24 ± 0,5 Vcc.
Corrente de Polarização	2 mA nominal
Tensão de Polarização de Saída	-8,5 ± 0,5 Vcc.

Requirements documentation



Requirements Documentation (Firmware Level)

- *Vision and Requirements Template*
  - Basic Structure:

1. **Overview**
   - 1.1 Project Scope
   - 1.2 Scope not included in the project
   - 1.3 Involved in the project
   - 1.4 Glossary of Terms
2. **Firmware Requirements**
   - 2.1 Functional Requirements
   - 2.2 Non Functional Requirements
3. **References**

Vision and Requirements Template

2 – Firmware Requirements

This section must describe all firmware requirements using a level of detailing to allow that:

- Developers known what must be built.
- Testers can plan testing to verify whether the software fulfill the requirements .

Firmware requirements can appear in 2 types:

- Embedded functionalities
- Restrictions applied to functionalities

Vision and Requirements Template

2.1 - Functional Requirements

Functional requirements specify what actually the firmware must do. In this section, all embedded functionalities must be described and detailed.

Código	Descrição do Requisito Funcional	Parâmetros Configuráveis	Limites/Valores Default dos Parâmetros	Prioridade
RF001	O firmware deve garantir que o incremento das distancias percorridas através da avaliação do sinal de GPS somente aconteça em percursos acima de 150 metros.			
RF002	O firmware deve permitir a configuração de filtros GPS (Static Holder) quando o CHIP possuir a funcionalidade.			
RF003	O firmware deve permitir gravar como configuração o valor do hodometro (distancia percorrida). Este valor configurado deverá ser utilizado como marco para os incrementos das distancias percorrida	Valor do Hodometro em Km	Range = 0 a 9999999 Default = 0	
…	…..	…	……	

Vision and Requirements Template

2.2 - Non-Functional Requirements

They represent conditions and/or restrictions that must be taken into account when designing the solution for the firmware functionalities.

2.3 Requisitos Não Funcionais

Código	Descrição do Requisito Não Funcional	Prioridade
Requisitos de Desempenho e Robustez:		
RNF01	O firmware deve armazenar em memória até 60000 registros de variações de localização.	Média

Vision and Requirements Template

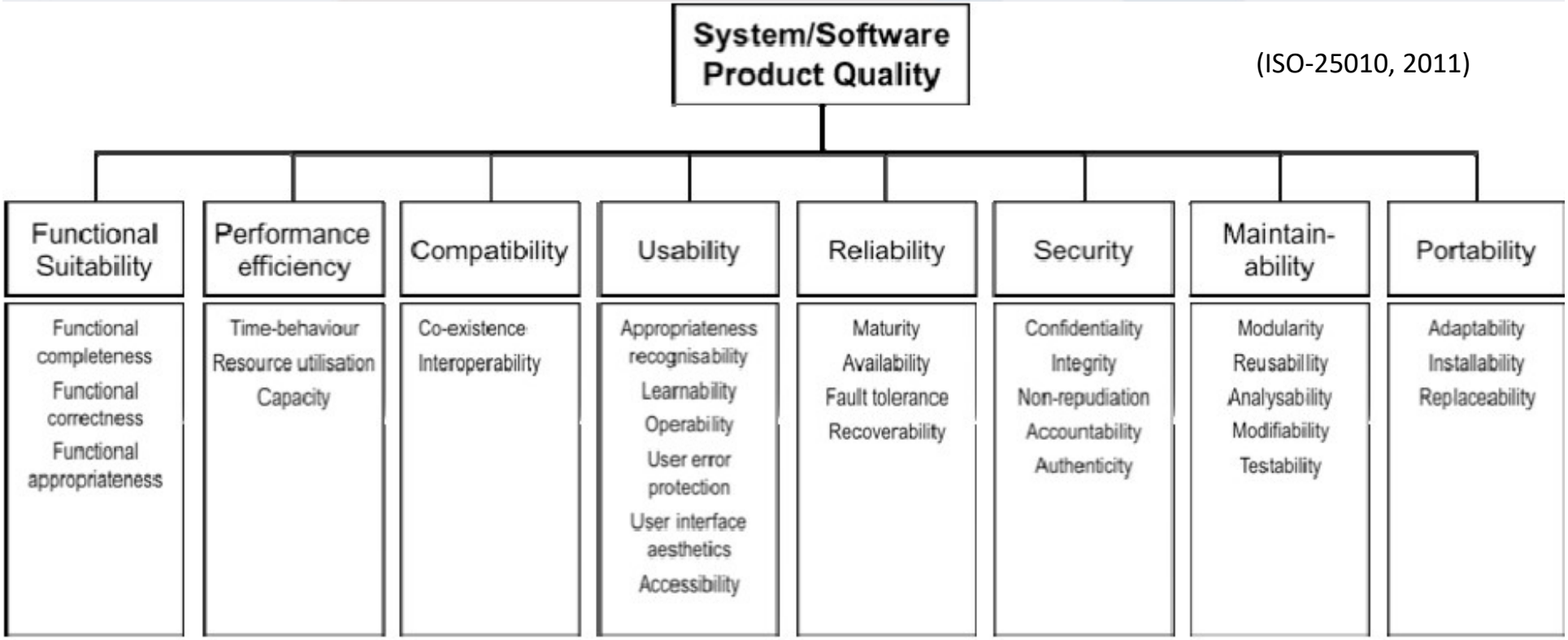
2.2 - Non-Functional Requirements

- Pre-defined Categories

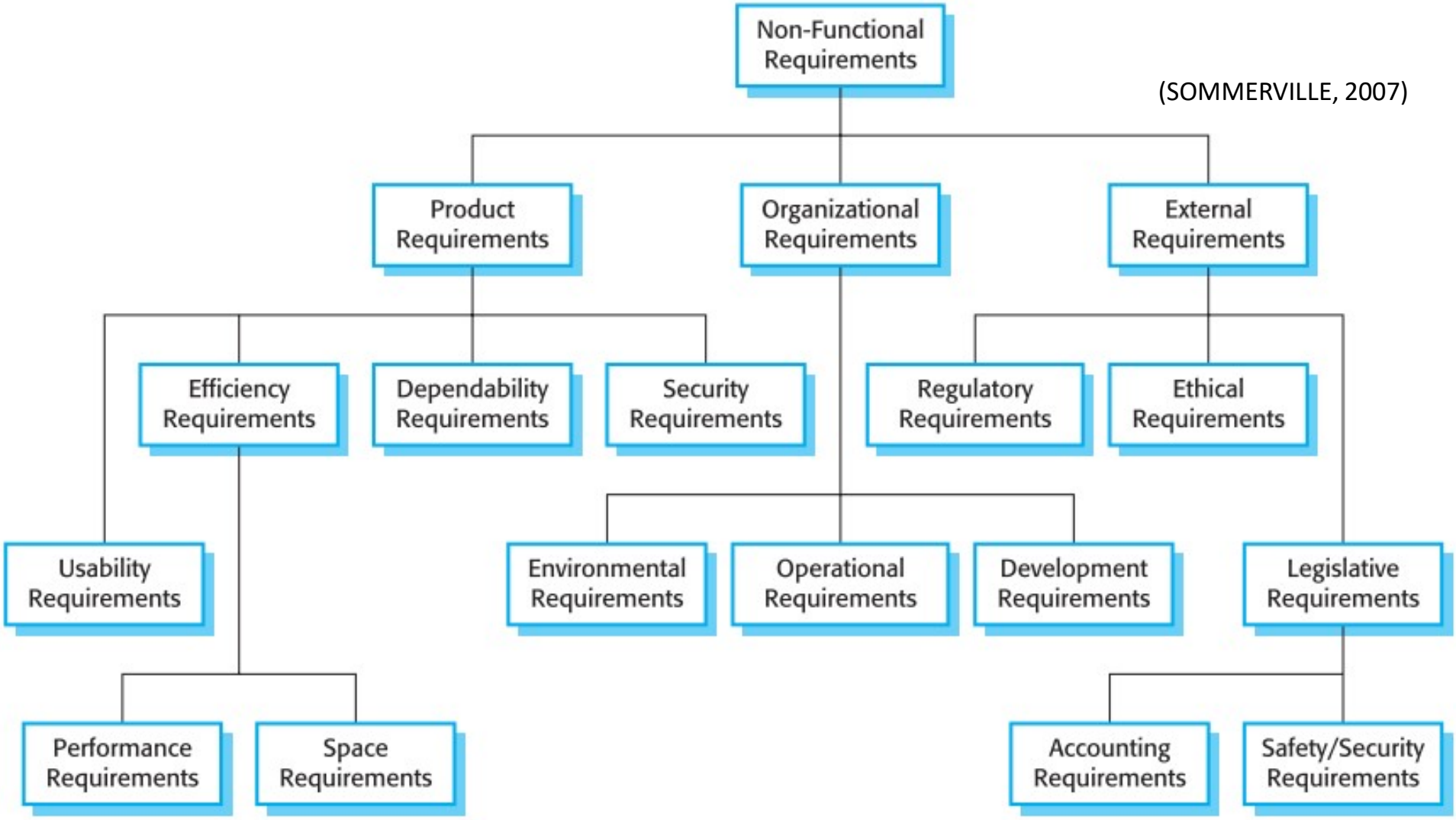
The t*emplate* just suggests these categories. Developers can add more whether is necessary!

Non-Functional Requirements Categories
Data Communication, Interface and Interoperability
Reliability
Performance and Robustness
Availability
Maintainability
Portability
Safety and Security
Usability
Technological Restrictions
Legal Restrictions

ISO-25010 – Product Quality



(ISO-25010, 2011)



(SOMMERVILLE, 2007)

Vision and Requirements Template

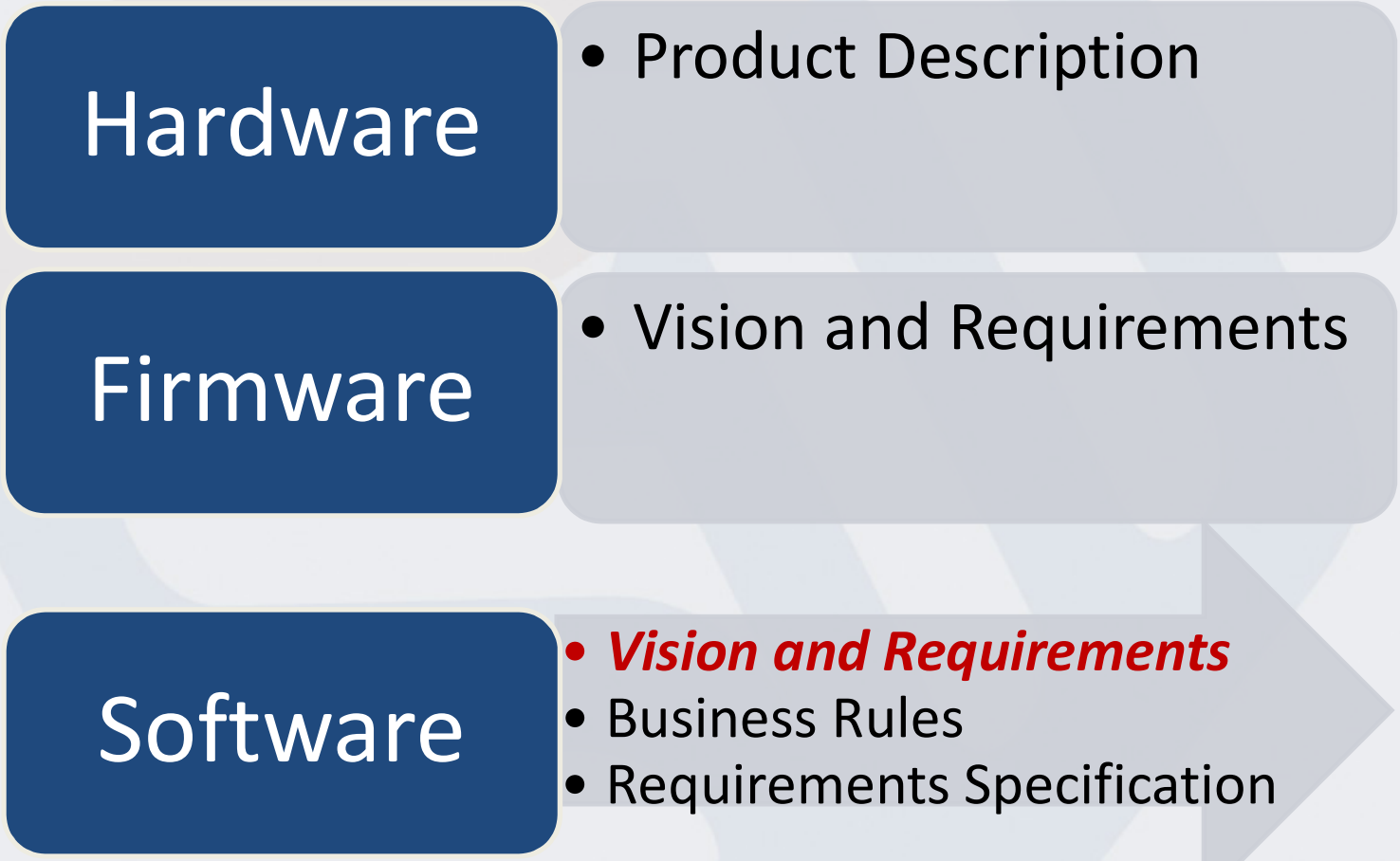
3 - References

This section contains references (traces) to all other documents needed to support the requirements description.

3 Referências

Código	Título do Documento	Versão	Onde pode ser obtido
REF01	Product Description IDP 780	1.4	Repositório svn da Maxtrack

Requirements Documentation



Requirements Documentation (Software Level)

- *Vision and Requirements Template* - SW
  - Basic Structure:

1. **Overview**
   - 1.1 Project Scope
   - 1.2 Scope not included in the project
   - 1.3 Involved in the project
   - 1.4 Glossary of Terms
2. **Software Requirements**
   - 2.1 Functional Requirements
   - 2.2 Non Functional Requirements
3. **References**

Vision and Requirements Template - SW

2.1 Requisitos Funcionais

Código	Descrição do Requisito Funcional	Prioridade
RF01	O software deve possibilitar o cadastro de veículos rastreados, devendo armazenar informações de placa e chassi do veículo e o identificador do rastreador do veículo.	Alta
RF02	O software deve possibilitar a visualização de localização dos veículos cadastrados.	Média
RF03	O software deve registrar desvio de rota dos veículos cadastrados.	Alta

2.2 Requisitos Não Funcionais

Código	Descrição do Requisito Não Funcional	Prioridade
Requisitos de Comunicação de Dados, Interface e Interoperabilidade:		
RNF01	O software deve possuir uma interface de comunicação com o rastreador IDP 780.	Alta

State of the Art

CONTEMPORARY SOFTWARE PROJECTS

- Experiences acquired due to intense collaboration with the Brazilian Industry through **COPPETEC FOUNDATION** (www.coppetec.coppe.ufrj.br)

- Pre-startups at *DELFOS* – Observatory at the Software Engineering Lab at COPPE/UFRJ
  - Currently three cells:
    - **QPS Model** (multi perspective Software Product Quality Evaluation)
    - **Parasite Watch** (IoT Based system to support the diagnosis of tropical parasite diseases)
    - **CATS** - Context-Awareness Software Testing Factory

Software Systems

System Software

Real-Time Software

Business Software

Embedded Software

Engineering and Scientific Software

Personal Computer Software

Artificial Intelligence Software

Mobile Apps

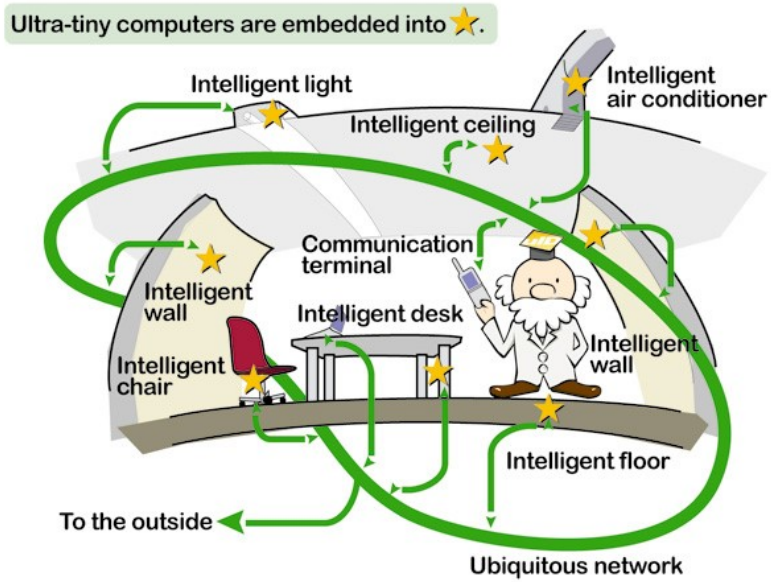
Systems of Systems

Ubiquitous Software Systems

Ubiquitous Software Systems

- Computational services or facilities are made available to people in such a way that the computer is no longer visible nor needed to be used as an essential tool to their access.
  - The services or facilities can materialize themselves at any time or place, transparently, through the use of common daily devices.

https://pbs.twimg.com/profile_images/378800000790349866/127857d65c47795d45f832f5dab323cb_400x400.jpeg



https://omarsbrain.files.wordpress.com/2010/08/ubicompex.jpg

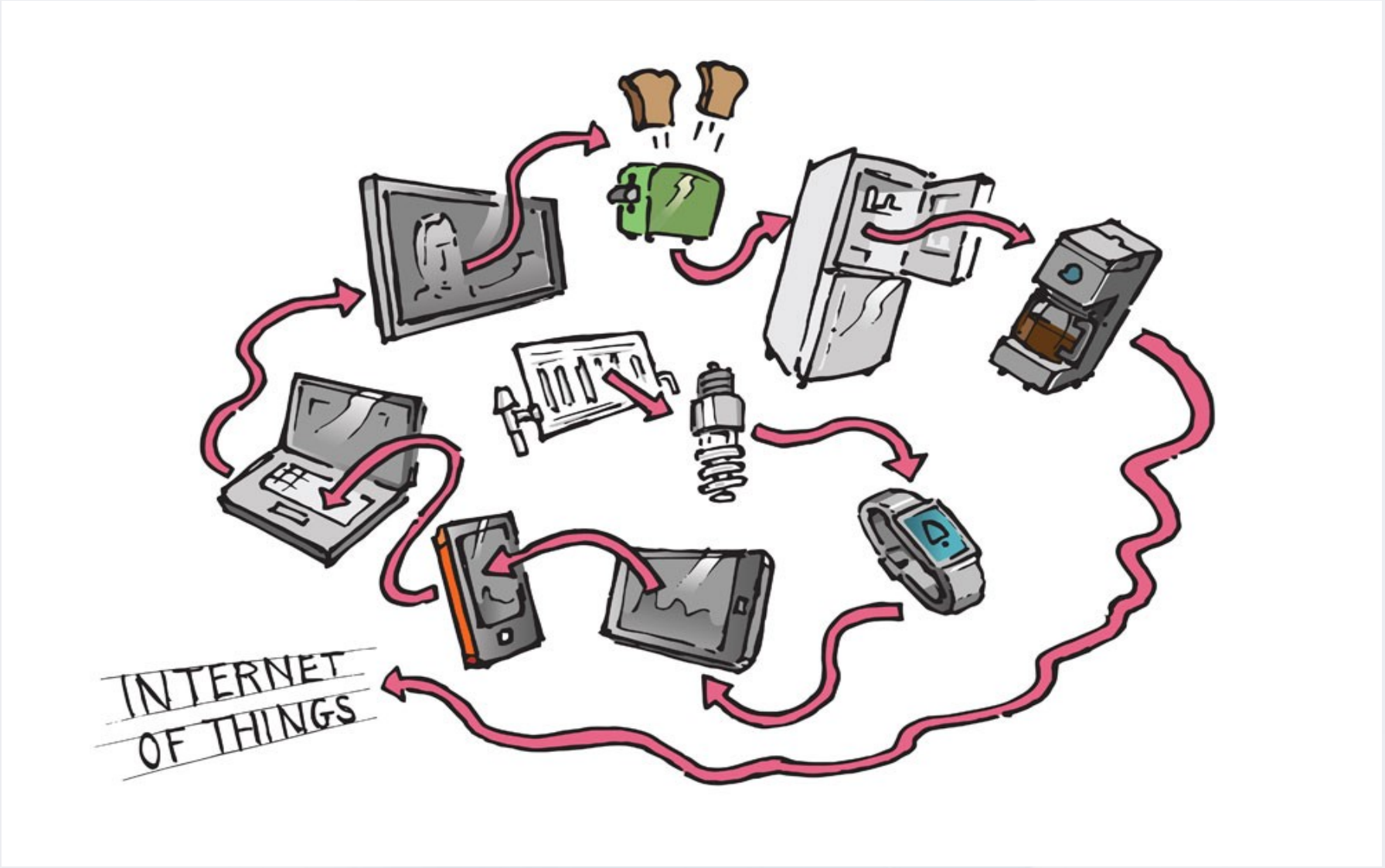
Ubiquitous Software Systems

- Computational services or facilities ...vailable to people in such a way that the co... visible nor needed to be used as an es...s.
  - The services or facilities can ...y time or place, transparently, throu... ...evices.

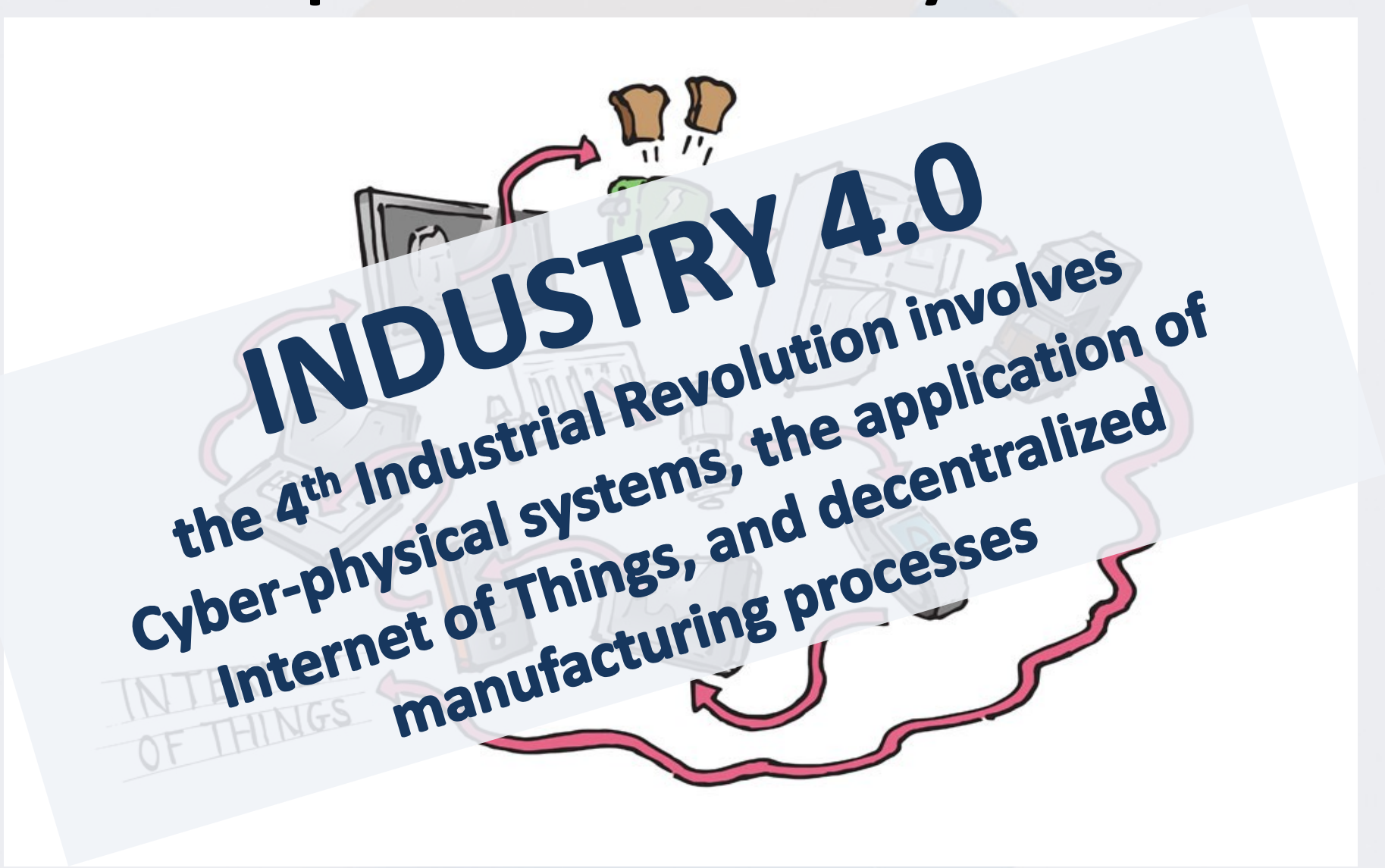
https://pbs.twimg.com/profile_images/378800...
127857d65c47795d45f832f5dab323cb_400x400.jp...

https://omarsbrain.files.wordpress.com/2010/08/ubi
compex.jpg

Ubiquitous Software Systems



Ubiquitous Software Systems

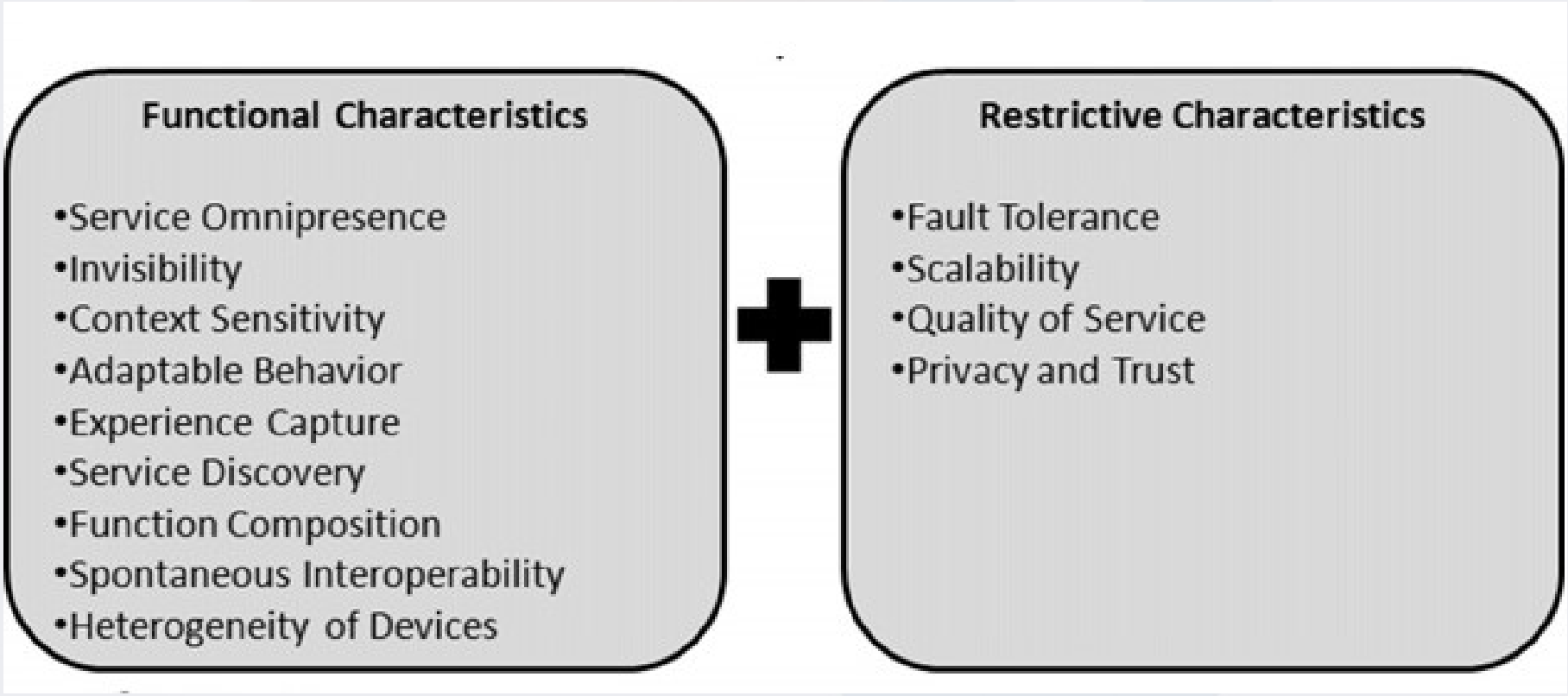


New Challenges in Software Development

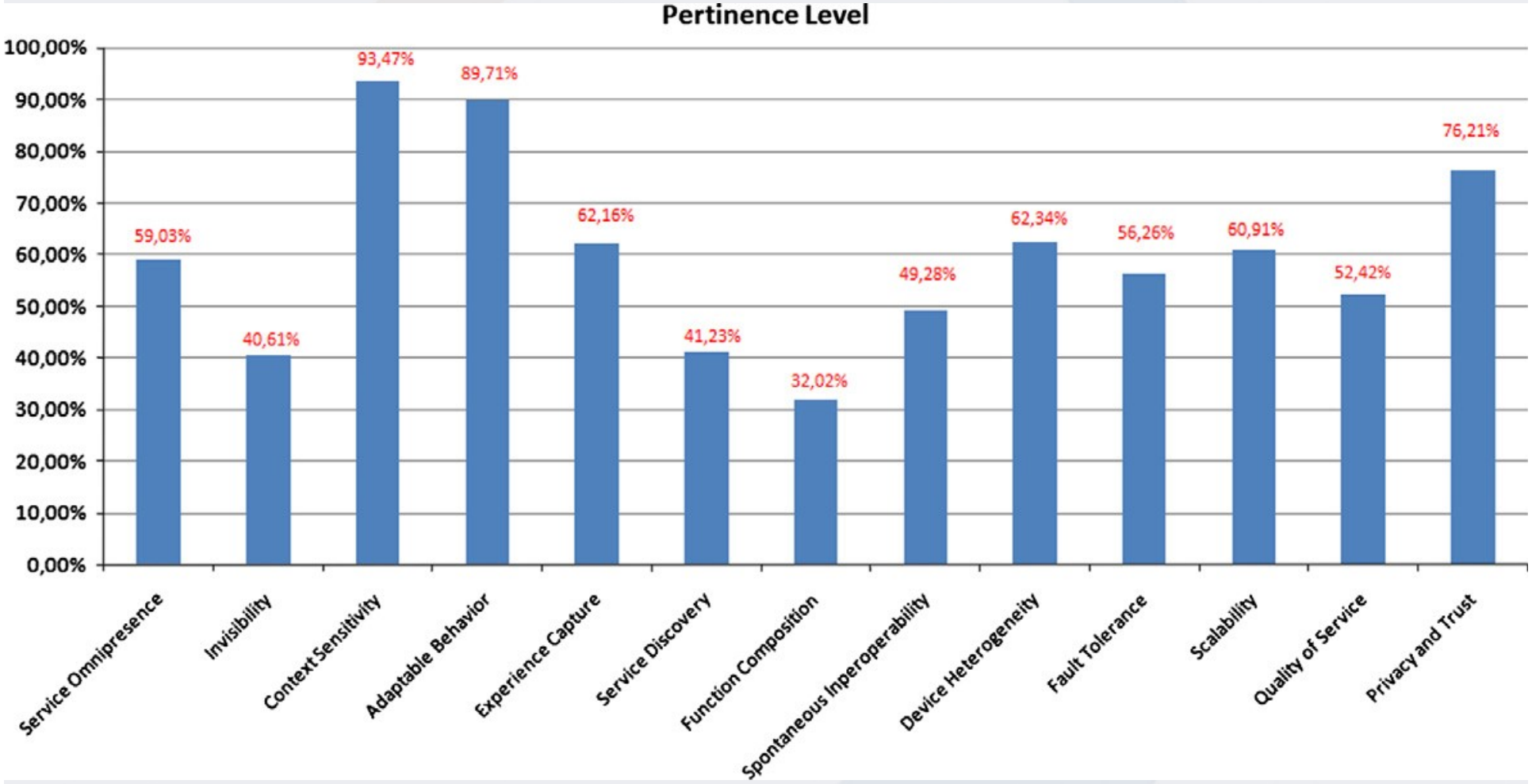
	Conventional Software	Ubiquitous Software
Device	Software is developed for an specific device: **Device-centric**.	Software adapts itself to the device used by the user: **Environment-centric**.
Software	User dependent to adapt to the environment.	As much as possible, software is invisible to the user and adapts itself to the environment. (*context-awareness, transparency, security, privacy, performance, …)*
Computerized Environment	Virtual	Physical environment rich on information. (*e-infrastructure,...*)

Ubiquitous Software Systems

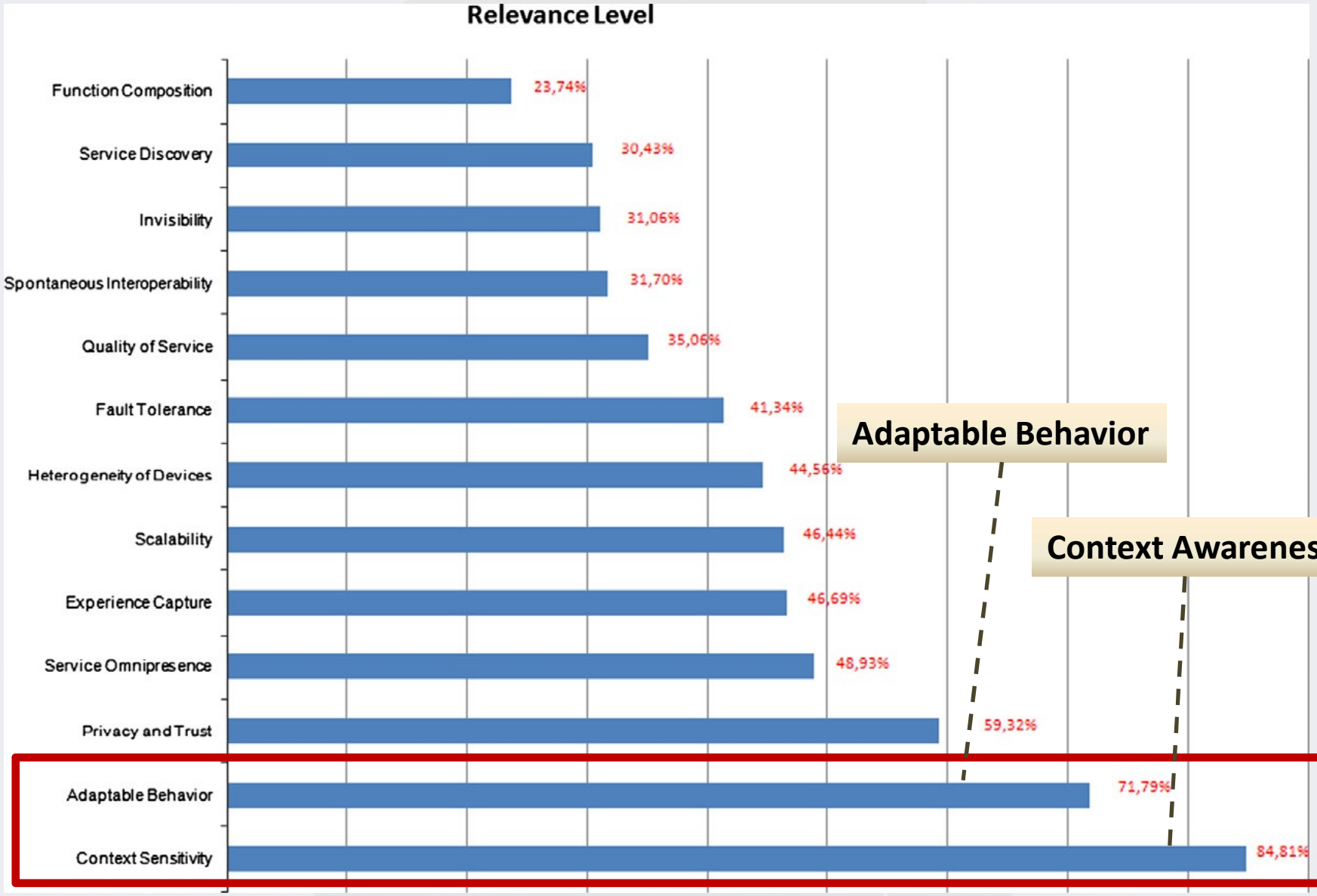
Further (novelty and challenging!) Requirements



Ubiquitous Software Systems Characteristics



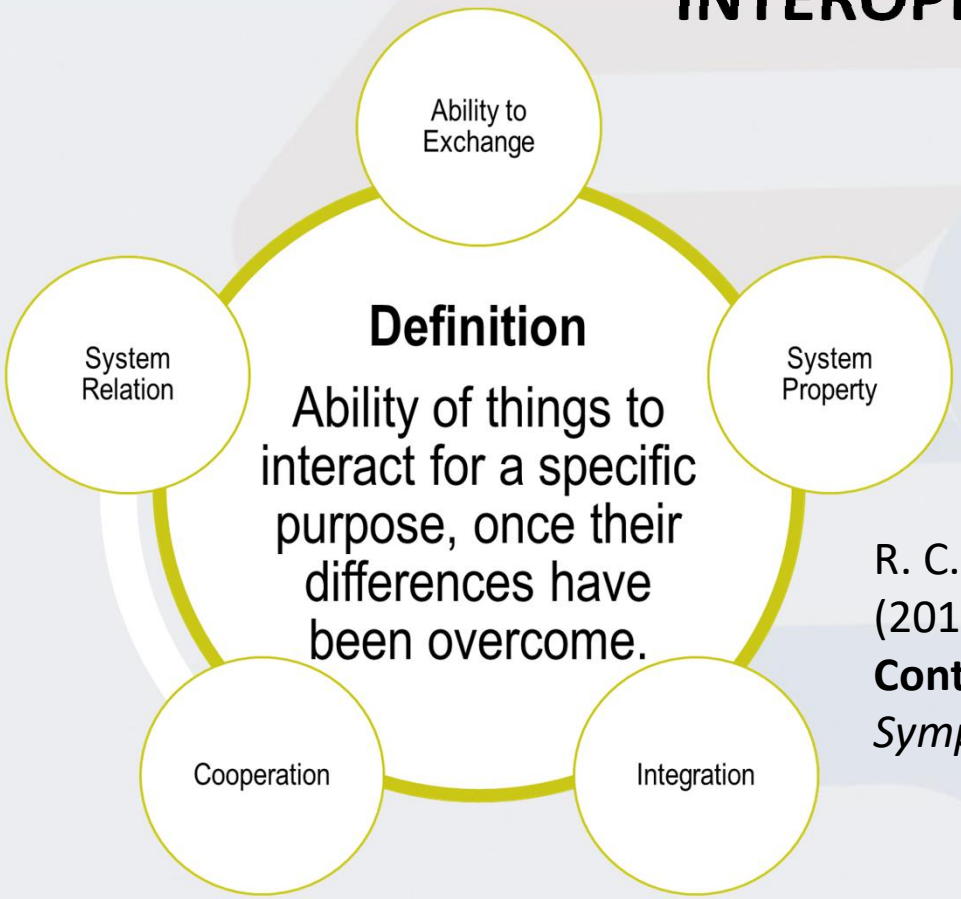
Spínola, R. O. ; Travassos, G. H. (2012) . Towards a framework to characterize ubiquitous software projects. Information and Software Technology v. 54, p. 759-785. http://dx.doi.org/10.1016/j.infsof.2012.01.009



Spínola, R. O. ; Travassos, G. H. (2012) . Towards a framework to characterize ubiquitous software projects. Information and Software Technology v. 54, p. 759-785. http://dx.doi.org/10.1016/j.infsof.2012.01.009

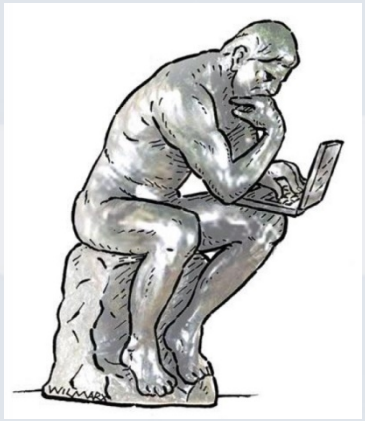
Ubiquitous Software Systems

Further (novelty and challenging!) Requirements: INTEROPERABILITY



R. C. Motta, K. M. Oliveira, and G. H. Travassos (2016). **Characterizing Interoperability in Context-aware Software Systems**," in *Brazilian Symposium on Computing System Engineering*.

IDEAS x REQUIREMENTS



*"**Innovation** incorporates both creation or discovery aspects, and diffusion or utilization aspects"*
(Deakins and Freel 2006)

Nascimento, L.M; Travassos, G.H, (2017). **Software Knowledge Registration Practices at Software Innovation Startups**. To appear at CBSOFT 2017 – SBES.

Ideas x Requirements x Software Development

- **Creation/Collection, Refinement, and selection of ideas.**

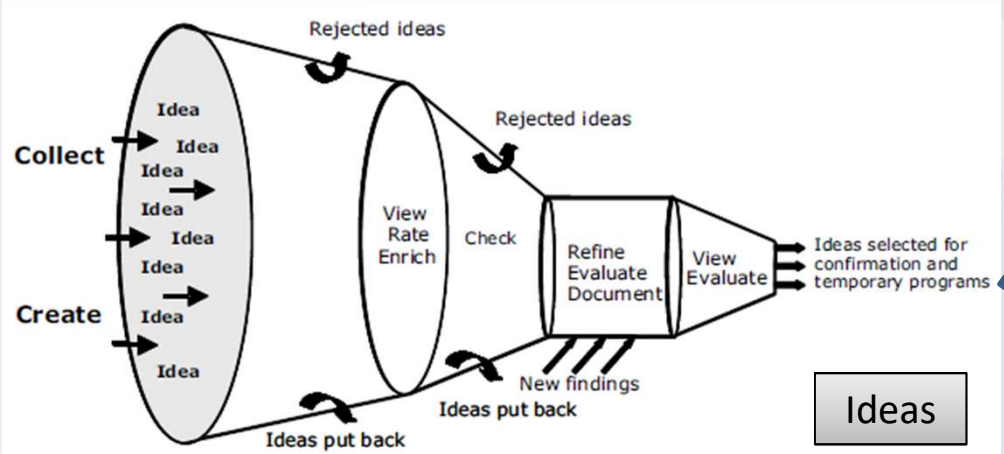
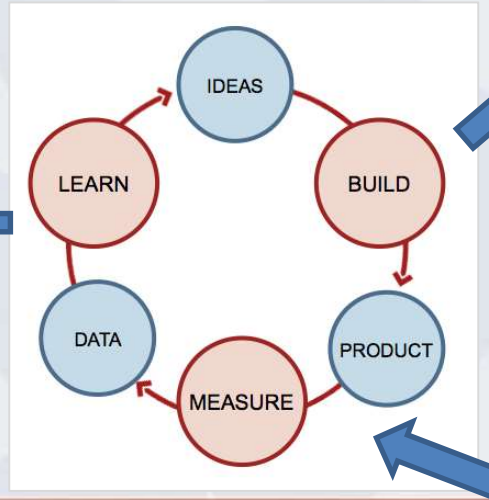


Figure 2-3: The idea tunnel (Deschamps et al. 1995)



The Build-Measure-Learn cycle

Engineering Activities

- Requirements
- Design
- Coding
- Test
- Implementation

CREATION of ideas

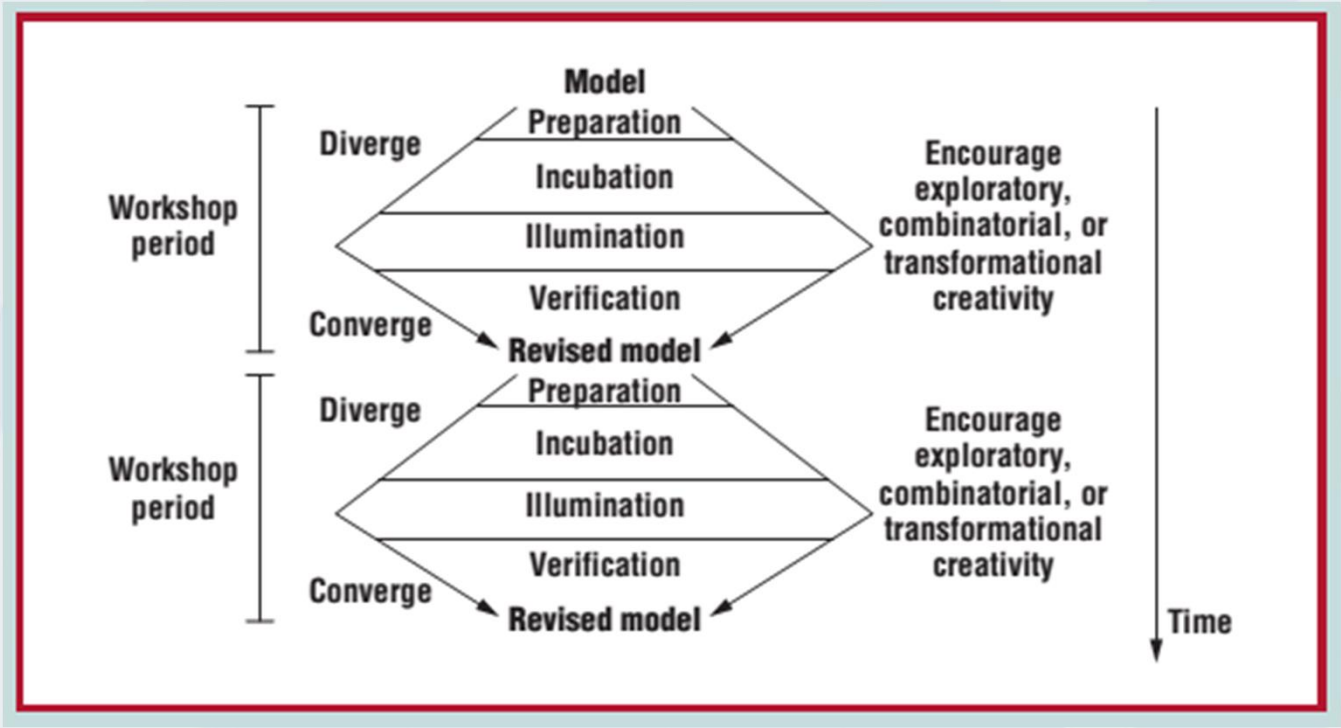
- **Practices, Methods and Techniques**
  - Creativity Support
    - Use of techniques to support creativity
    - Inclusion of Creativity Activities into the Requirements Process
    - Integration of Creativity and Requirements Tools
  - Group Interaction Techniques
  - Actual Software Analysis
  - Use of User-Centered Techniques

Creation of Ideas

- **Example of Practices, Methods and Techniques**
  - Brainstorming, Workshops
  - Role Play, Personas, One Day in life
  - Mind mapping
  - Gamming
  - Six hats
  - Prototyping
  - Observation
  - Combination of existent ideas/requirements

Creation of Ideas

- **Example of Practices, Methods and Techniques**

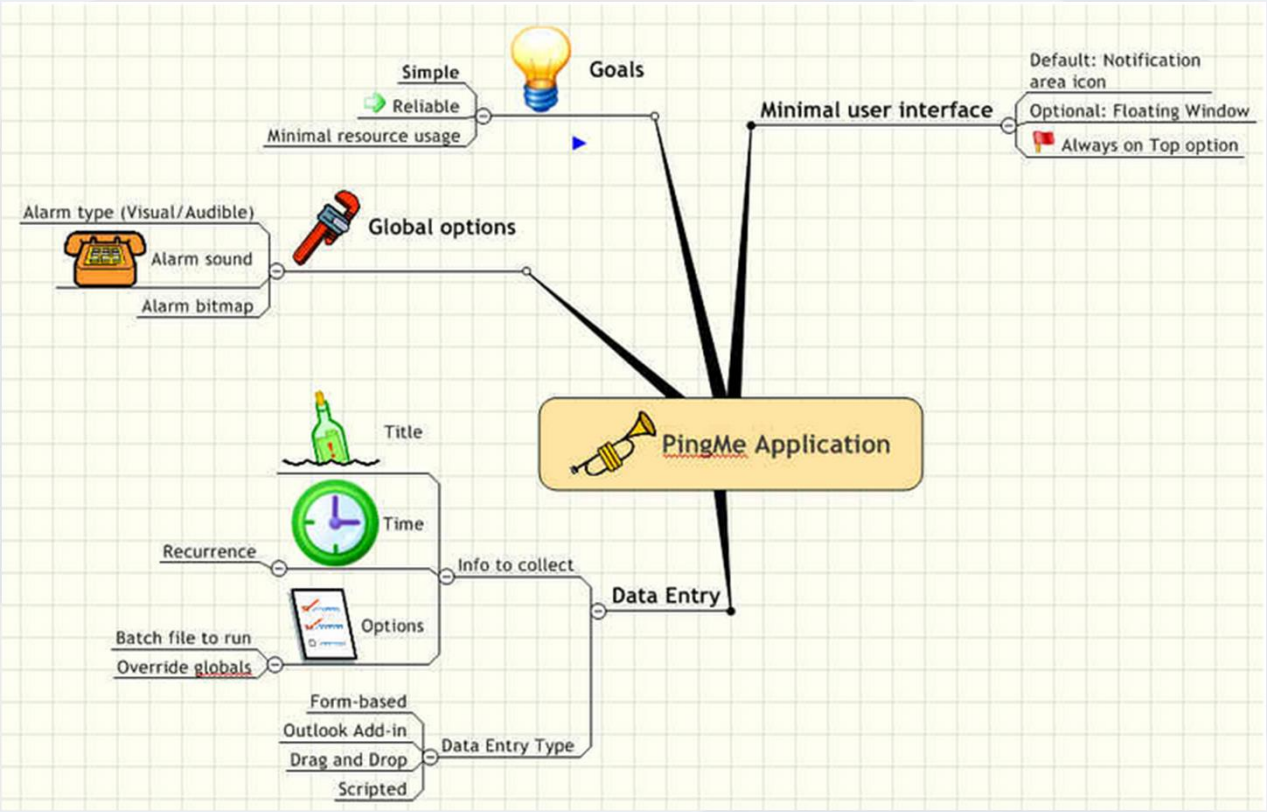


Creativity Workshop

. Maiden , S. Robertson and A. Gizikis , "**Provoking Creativity: Imagine What Your Requirements Could Be Like**" , *IEEE SoftÂware* , vol. 21 , no. 5 , pp.68 -75 , 2004

Creation of Ideas

- **Example of Practices, Methods and Techniques**



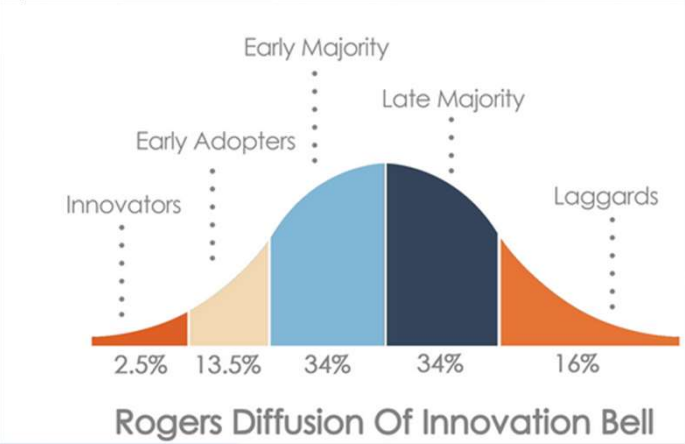
Mind mapping

Creation of Ideas

- Example: *Low Fidelity Prototype*

Collection of Ideas

- **Example of Practices, Methods and Techniques**
- Approaches using "User-driven development"
  - Integration of user representatives in the innovation development (creation of ideas, screen design, validation, priorization, and so on).
  - It can involve the identification of :
    - Lead-users (Innovators)
    - Early Adopters



Collection of Ideas

- **Example of Practices, Methods and Techniques**

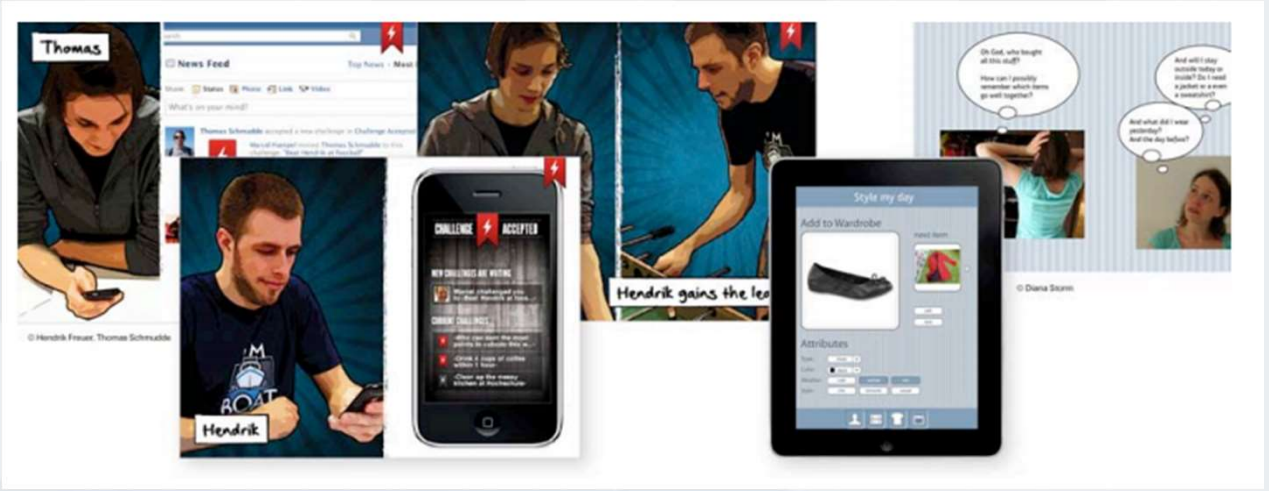
  - Interviews and Questionnaires
  - Face-to-face discussion groups (*Brainstorming, Focus Group, others*)
  - Text Mining in discussion forum, feedback systems and issue tracking
  - Related software analysis.
  - Ideas Competition (Internal/external)
  - Crowdsourcing
  - Use of Social Networking and Collaborative Tools

Validation of Ideas

- **Example of Practices, Methods and Techniques**
  - Interviews
  - Face-to-face discussion groups (meetings, *Brainstorming*, *Focus Group*)
  - Questionnaires
  - *Technology Probe*
  - Analysis of Reaction and perception (sampling the users)
    - Emotional reactions, values, perceptions, motivation

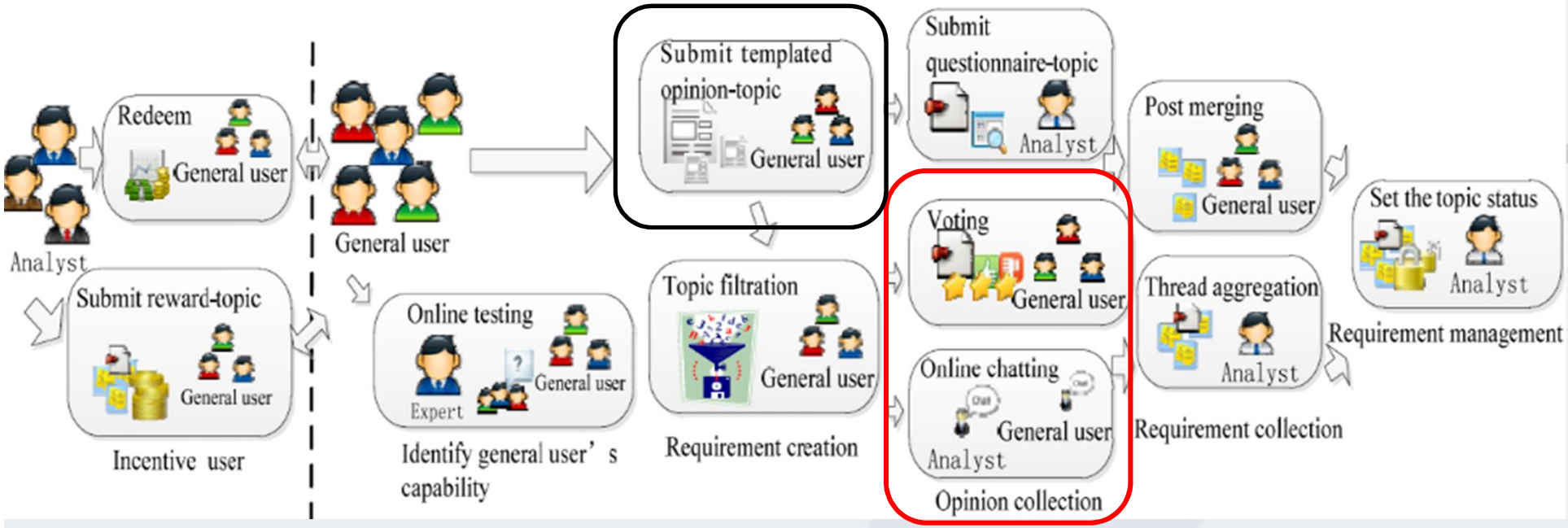
Validation of Ideas and Products

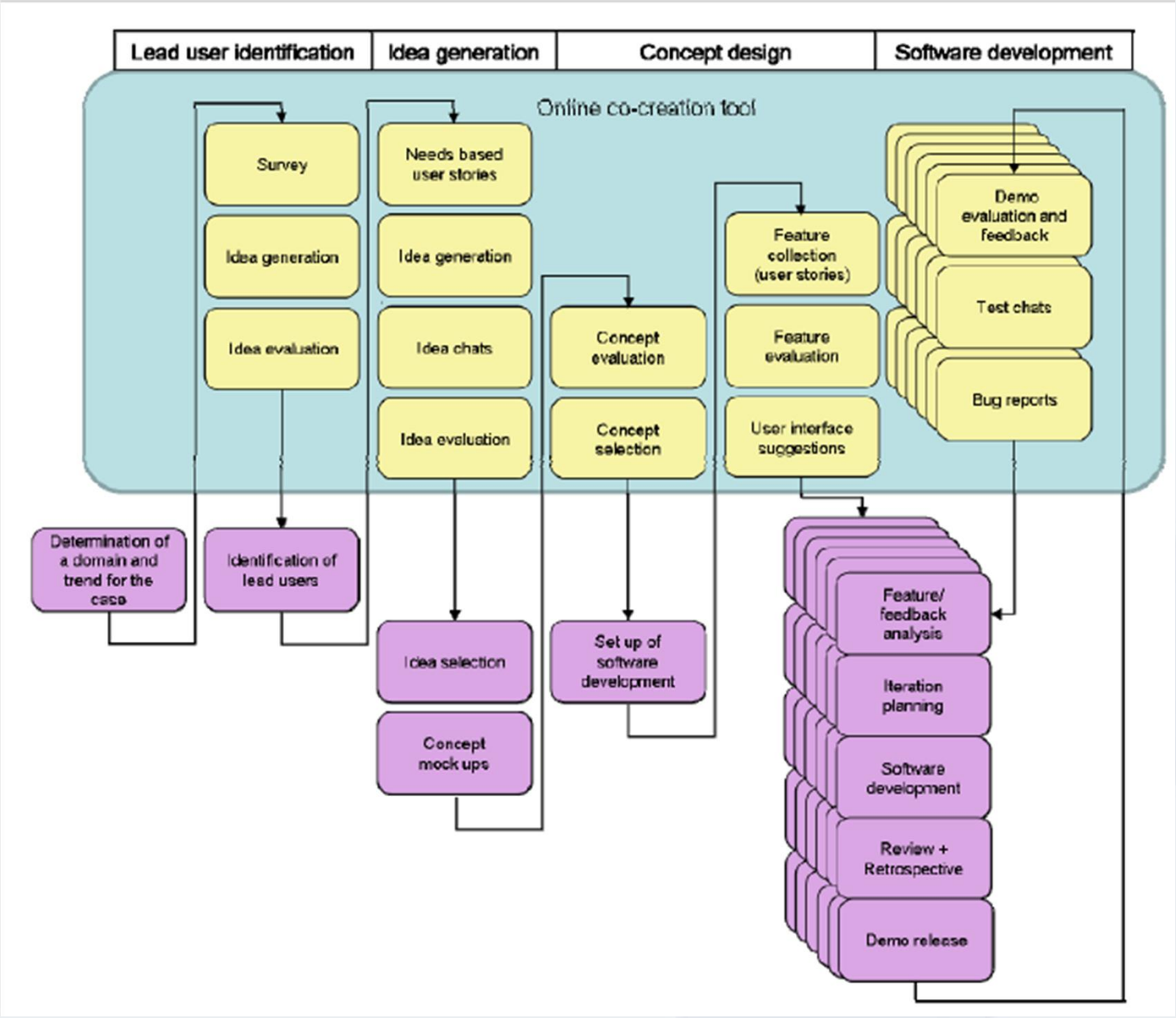
- **Ex. Practices, Methods and Techniques**
  - Prototypes
    - *Low-fidelity Prototype, Paper Prototype, Fake prototype (Wizard of Oz)*
  - Pilot study
  - *Demo testing*



Collection and Validation

- Example:
  - Social Networking and Collaborative Tools




Lead user identification
Idea generation
Concept design
Software development
Online co-creation tool
Survey
Idea generation
Idea evaluation
Needs based user stories
Idea generation
Idea chats
Idea evaluation
Concept evaluation
Concept selection
Feature collection (user stories)
Feature evaluation
User interface suggestions
Demo evaluation and feedback
Test chats
Bug reports
Determination of a domain and trend for the case
Identification of lead users
Idea selection
Concept mock ups
Set up of software development
Feature/ feedback analysis
Iteration planning
Software development
Review + Retrospective
Demo release

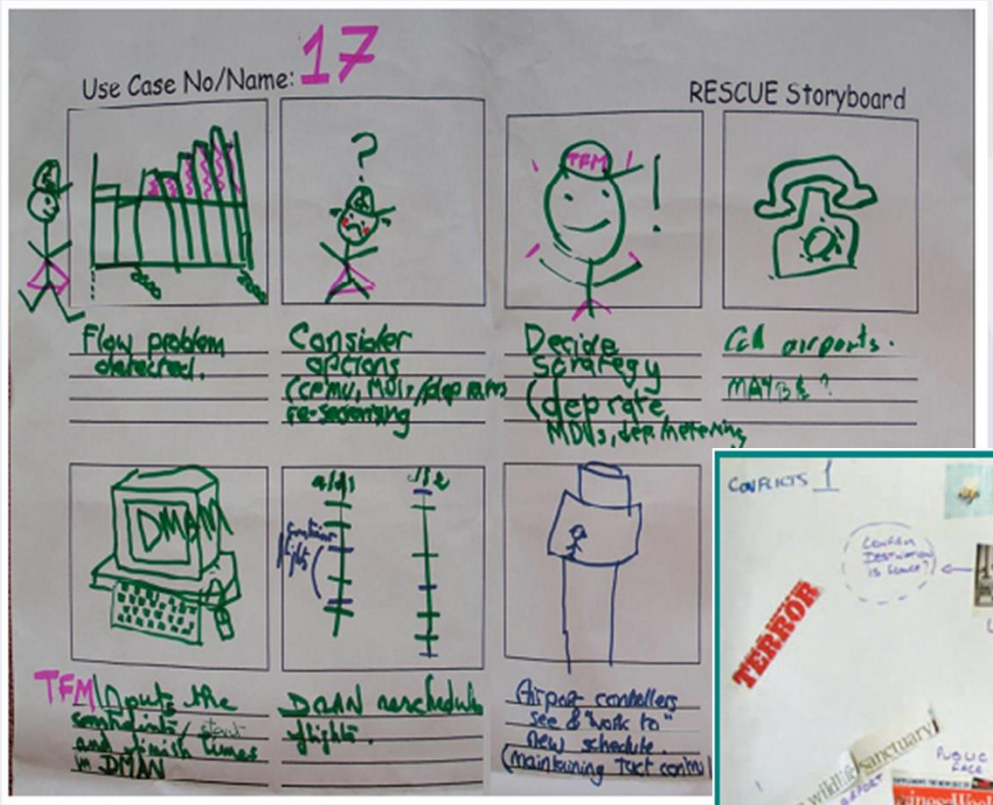
Priorization and Selection of Ideas

- **Example of Practices, Methods and Techniques**
  - Face-to-face discussion groups (meetings, *KJ technique*)
  - Voting mechanisms (for instance, by using social networks or other collaborative tools)
  - Definition of Decision Groups in the organization

Registering (documenting) Ideas

- Some examples of documentation reported in the technical literature (used by the organizations):
  - Non-structured free text
  - Drawings (*sketches*), images (reminders)
  - Scenario Representations
    - Scene descriptions ("Role Play")
    - Persona stories
    - Use case *précis*
    - Initial models and descriptions of use cases
    - Storyboards
  - Mind Maps
  - Video, audio
  - Paper prototypes

Registering (documenting) Ideas

- 

Requirements Engineering at the 4th Industrial Revolution

Practitioners consider the Requirements Engineering Practices “heavy” to support software development at this context.

- Favoring lesser time-to-delivery
- “*just-do-it*” approach– straight from the feature idea formulation to implementation .
- Focusing in them team´s communication ability, which uses to be small and multitask.
- Using not formal and opportunistic supporting practices.

Requirements Engineering at the 4th Industrial Revolution

- Observed Practices:
  - Not formal functional specifications.
  - Use of Tools based on tickets to support features lists.
  - Low precision features description, usually using user-stories.
  - Use of screen prototypes and mind-maps
  - Use of physical tools to register and share artifacts.

Requirements Engineering at the 4th Industrial Revolution

Reported Challenges:

Informal practices (seems to) **favor** lesser *time-to-delivery* in the initial phases of innovation software development.

However, in the case of product surveillance and after successive iterations, the **amount of accumulated tacit knowledge jeopardize productivity** in the software product evolution due to:

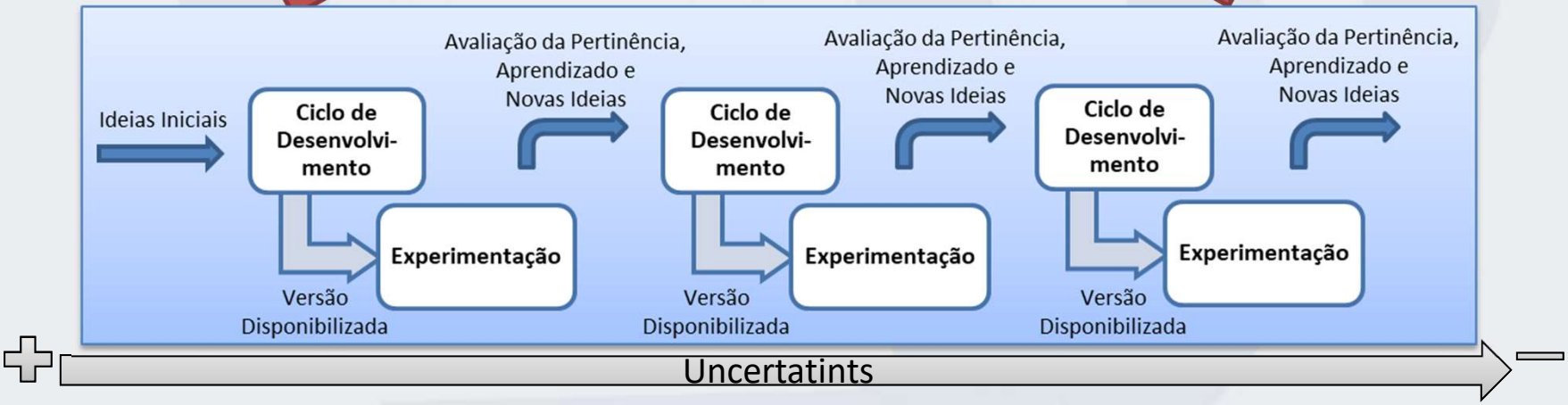
- **Continuous increasing of software complexity after each iteration;**
- **Increasing the number of versions to keep the software product in production;**
- **Increasing the technical debt**
- **Increase the need of fixing defects (rework);**
- **Risk of losing knowledge due to developers turnover**
- **Risk of forget software characteristics details**

Requirements Engineering at the 4th Industrial Revolution

Challenge: How to Balance?

The need of registering information of ideas and software requirements

Goal of lesser the *Time-to-delivery*



The 4th Industrial Revolution
The Brazilian Industry perspective

INDUSTRY 4.0 represents a new challenge for Brazilian industry!

- Adopting digital technologies is essential for competitiveness
- Low knowledge constitutes an obstacle to utilization
- High-tech industries are the ones that make greater use of digital technology
- Digitization is focused on improving processes
- High- and medium-high tech sectors make greater use of development-related technologies
- Computers, electronics and optical products stand out for making the most use of digital technologies across all production chain stages
- Reducing costs and increasing productivity are the most sought-after benefits
- High implementation cost is the main internal barrier
- Lack of skilled workers is the biggest challenge among external factors
- Government should focus on infrastructure and education

CNI Indicators. **Special Survey Industry 4.0**. National Confereration of Industry.
ISSN 2317-7330 • Year 17 • Number 2 • April 2016 . Brazil

Final Remarks

- Some Challenges on the 4th Industrial Revolution that we are working in collaboration with the industry
  - **Balancing QUALITY X EFFORT X SHORT TIME TO DELIVERY**
    - What is the impact of the lack of requirements engineering practices in the innovation software process development? How to observe such an impact?
    - How to know when to include more formal software requirements practices in the software development process?
    - How much detail is needed? (different observations depending on the development stage!
    - More rigor in the ideation process can bring benefits to the development process? If so, what is the cost?
  - **Integrating the Ideas (requirements?) in the development process.**
    - Practitioners do not like reworking! The models (representing the ideas) can be directly used to support the development? Evolved? If so, how?

Final Remarks

- Some Challenges on the 4th Industrial Revolution that we are working in collaboration with the industry
  - **Balancing QUALITY X EFFORT X SHORT TIME TO DELIVERY**
    - What is the impact of the lack of requirements engineering practices in the innovation software process development? How to observe such an impact?
    - How to know when to include more formal software requirements practices in the software development process?
    - How much detail is needed? Different observations depending on the development stage?
    - More rigor in the ideation process can bring benefits to the development process? If so, what is the cost?
  - **Integrating the Ideas (requirements?) in the development process.**
    - Practitioners do not like reworking! The models (representing the ideas) can be directly used to support the development? Evolved? If so, how?

The answers are easier and timely when academia and industry work collaboratively!

UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

Departamento de Informática
Universidad Técnica Federico Santa María

I Encuentro Latinoamericano de Ingeniería de Software ELAIS 2017

Gracias.

Guilherme Horta Travassos

www.cos.ufrj.br/~ght

CNPq

COPPE/UFRJ

THE CLOVER
2030 ENGINEERING STRATEGY
AN ENGINE TO SURF THE WAVES FOR
CHILE'S DEVELOPMENT

CORFO