Motivation

- What the Customer really needed -

Source: http://www.bowdoin.edu/~disrael/what-the-customer-really-needed/what-the-customer-really-needed.jpg
Requirements Engineering

- The requirements elicitation phase of embedded system development is about:
  - Getting all system functions together
  - Showing scope, usage, and constraints (performance, environment, regulation, threats, etc.) of the proposed system
  - Get a good understanding on effort and system architecture (risk reduction)

- Once all information are available and validated the requirements are translated into a requirements specification which is a technical document for further development (metrics and defined format on all requirements)

- Wrong (e.g. missing, contradicting) information will make us fail at a very cost intensive level → validation (analysis)
Requirements Elicitation

How do we get all these requirements?

- Involves technical staff working with customers or users to find out about the application domain (field technicians), the services that the system should provide and the system’s operational constraints.

- May involve end-users, our customers, managers, engineers involved in prior development and/or maintenance, domain experts, certification bodies, etc. These are called stakeholders.

- Also non-functional requirements can be discovered in a systematic way (QFD, FTA, RBD, PHA, ...)

A. Walsch, IN2244 WS2015/16
Challenges in Requirements Elicitation

- Stakeholders don’t know what they really want.
- Stakeholders express requirements in their own terminology – maybe not precise.
- Different stakeholders may have conflicting requirements.
- Political factors may influence the system requirements (e.g. disasters).
- The requirements change during the analysis process.
- Some requirements might be common sense and not explicitly mentioned.
Requirements Validation (Analysis)

- Validity
  Does the system provide the functions which the customer expects?

- Consistency
  Are there any requirements conflicts?

- Completeness
  Are all functions required by the customer included? Are more functions included?

- Realism
  Can the requirements be implemented given available budget and technology -> feasibility?

- Verifiability
  Can the requirements be shown to be implemented correctly (e.g. tested)?
Requirements

- There are basically two kinds of requirements from the customer:
  - Non-functional (quality)
  - Functional (operations – IO)
- In addition, we will look into tools that help to gather requirements for
  - Safety: hazard analysis, fault trees (FTA), risk assessment (quality)
  - Reliability: (failure mode and effect) FMEA, FTA
- There are more non-functional requirements which will not be covered.
Non-Functional Requirements

Non-Functional Requirement
Constraints on implementation – How should the system be?
Includes

- Global constraints that influence system as a whole (shock, vibration, temperature, cost…)
- Function performance (response time, repeatability, utilization, accuracy)
- The “-ilities” (reliability, availability, safety, security, maintainability, testability, …)
- Other quality (ease of configuration and installation, …)
Non-Functional Requirements Capture

Look at system as black-box and concentrate on a specific use case

- Look at real-time aspects: response time, sampling rate
- Data quality: accuracy, precision
- Refine functional requirements – make more specific and testable
- Look at comparable systems (prior art, competitors)
- Safety (new laws or regulation) and reliability
- Hardware constraints (memory, CPU, IO)
Non-Functional Requirements
- Textual Examples -

“Pressure samples shall be taken every 1s.”
“The response time for pressure measurement shall be less than 10ms.”
“Reliability: 1000 FIT”
“The measurement shall have an accuracy of 2%.”
“The measurement shall be repeatable with a precision not less than 0.5%.”
“The system shall meet the safety criteria according to [std.].”

Source:
wikipedia

A. Walsch, IN2244 WS2015/16
### Requirement Patterns

**Classification by Dwyer et. al.**

- Qualitative
- Real-time

**Category**

- Occurrence
- Order
- Duration
- Periodic
- Real-time Order
- Minimum Duration
- Maximum Duration
- Bounded Recurrence
- Bounded Response
- Bounded Invariance

**Pattern**

- Universality
- Existence
- Response
- Chain 1-2
- Chain 2-1
- Chain 1-2
- Chain 2-1
- Constrained Chain 1-2

### Pattern Table

<table>
<thead>
<tr>
<th>Category</th>
<th>Pattern Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration</td>
<td>Minimum Duration</td>
<td>Describes the minimum amount of time a state formula has to hold once it becomes true. (Ex.: Engine starter system &quot;The system has a minimum 'off' period of 120 seconds before it reenters the cranking mode.&quot;)</td>
</tr>
<tr>
<td></td>
<td>Maximum Duration</td>
<td>Captures that a state formula always holds for less than a specified amount of time. (Ex.: Engine starter system &quot;The system can only operate in engine cranking mode for no longer than 10 seconds at one time.&quot;)</td>
</tr>
<tr>
<td>Periodic</td>
<td>Bounded Recurrence</td>
<td>Denotes the amount of time in which a state formula has to hold at least once. (Ex.: ABS system &quot;The ABS controller checks for wheel skidding every 10 milliseconds.&quot;)</td>
</tr>
<tr>
<td>Real-time Order</td>
<td>Bounded Response</td>
<td>Restricts the maximum amount of time that passes after a state formula holds until another state formula becomes true. (Ex.: ABS system &quot;From direct client input, detection of and response to rapid deceleration must occur within 0.015 seconds.&quot;)</td>
</tr>
<tr>
<td></td>
<td>Bounded Invariance</td>
<td>Specifies the minimum amount of time a state formula must hold once another state formula is satisfied. (Ex.: Engine starter system &quot;If Error 502 is sent to the Driver Information System, then the braking system is inhibited for 10 seconds.&quot;)</td>
</tr>
</tbody>
</table>

Source: Konrad and Cheng, Real-time Specification Patterns

A. Walsch, IN2244 WS2015/16
## Tools

<table>
<thead>
<tr>
<th>Supplier</th>
<th>Fortiss</th>
<th>Graz University of Technology and Fondazione Bruno Kessler, Trento, Italy</th>
<th>Argosim</th>
<th>BTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tool Type (e.g. Commercial, TRL)</td>
<td>free, high TRL</td>
<td>open -LGPL, medium TRL (academic)</td>
<td>commercial</td>
<td>commercial</td>
</tr>
<tr>
<td>Tool Description</td>
<td>Tool suite from requirements engineering to deployment (auto test case generation, schedulability, etc.). Provides capability to enter text based requirements.</td>
<td>RATSY (Requirements Analysis Tool with Synthesis) is includes the capability to synthesize reactive systems from their temporal specification (input language). Furthermore, it includes a game-based approach for debugging specifications.</td>
<td>STIMULUS enables you to express natural language requirements using formalized language templates, state machines, and block diagrams (also with auto test case generation capability).</td>
<td>Starting with informal textual requirements, a formal and machine-readable specification is intuitively derived step-by-step. This machine-readable specification furthermore can be used for fully automated formal verification as recommended by automotive standard ISO 26262.</td>
</tr>
<tr>
<td>Input (e.g. natural language)</td>
<td>natural language using templates</td>
<td>PSL: <a href="http://www.mel.nist.gov/mSIDlibrary/doc/nistir6459.pdf">http://www.mel.nist.gov/mSIDlibrary/doc/nistir6459.pdf</a></td>
<td>natural language using templates</td>
<td>textual requirements, universal pattern</td>
</tr>
<tr>
<td>Functionality (e.g. formal analysis, simulation)</td>
<td>Structure, analyze and verify requirements. Formal validation of requirements (NuSMV model-checker and Yices SMT solver)</td>
<td>Property Simulation, Property Assurance, Property Realizability and Synthesis, and Property Debugging using Games</td>
<td>user extendable templates, graphical entry (state machines, blocks), detect errors in requirements by simulation, include environmental assumptions</td>
<td>formal specification used to allow for verification and automatic test case generation</td>
</tr>
</tbody>
</table>