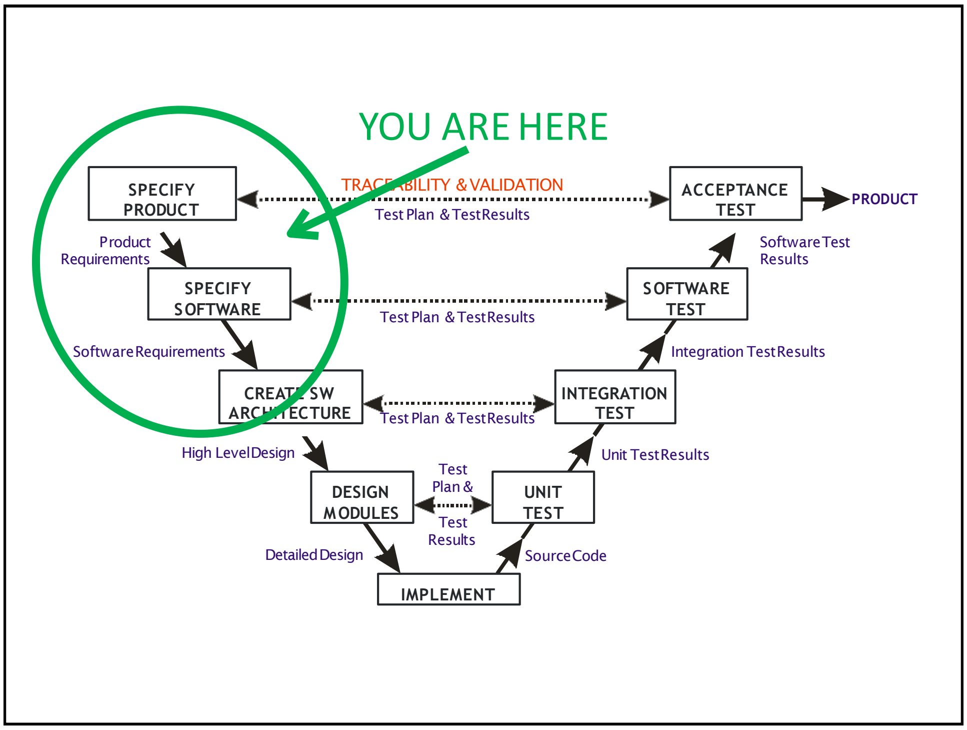
Lecture 3: CPS Requirements

Seyed-Hosein Attarzadeh-Niaki

Some Slides from Peter Marwedel and Philip Koopman

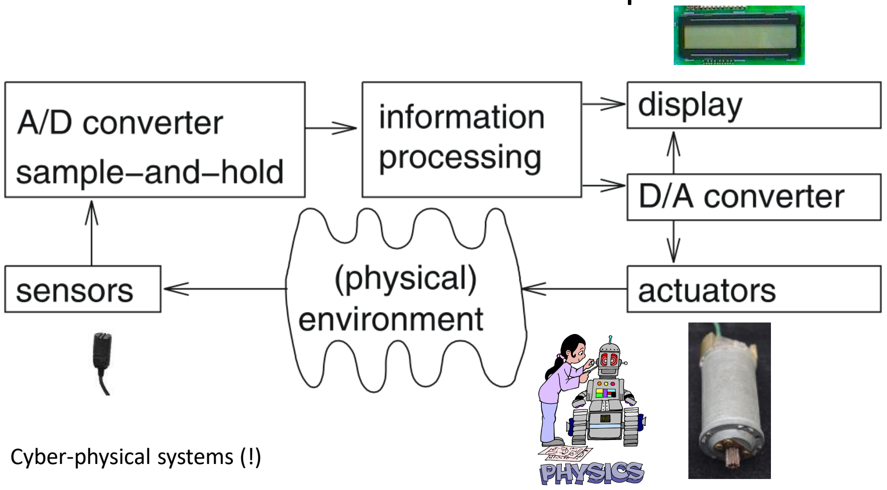
Review

- CPS design: hardware and software
- CPS software development process
- Model-based design

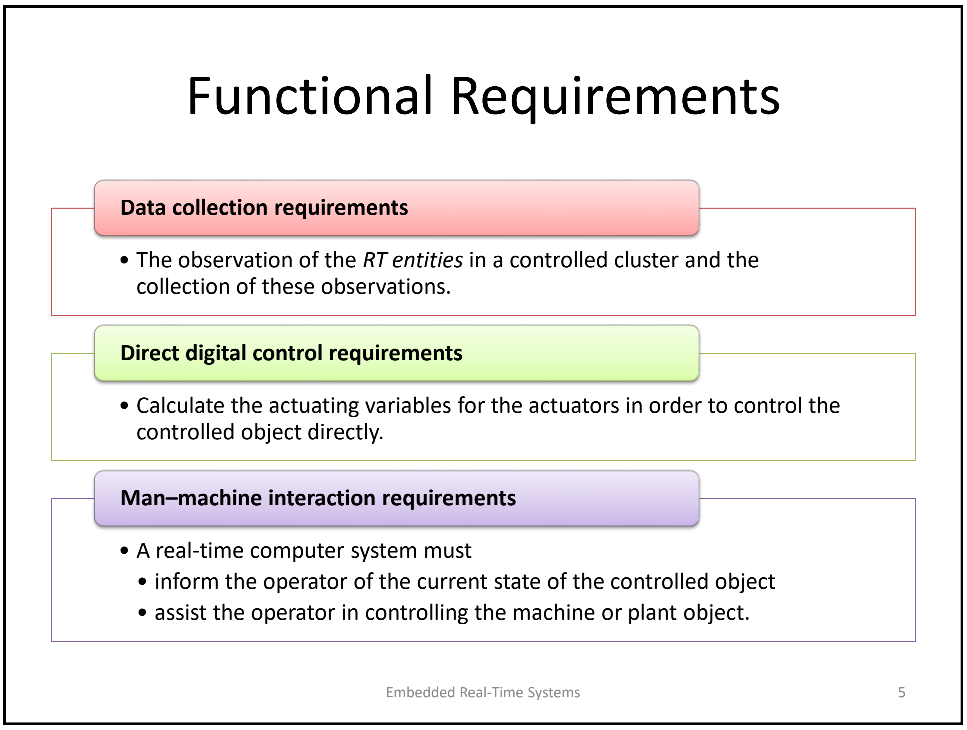


CPS & ES Requirements

- Functional and extra-functional requirements



Cyber-physical systems (!)



Functional Requirements

Data collection requirements

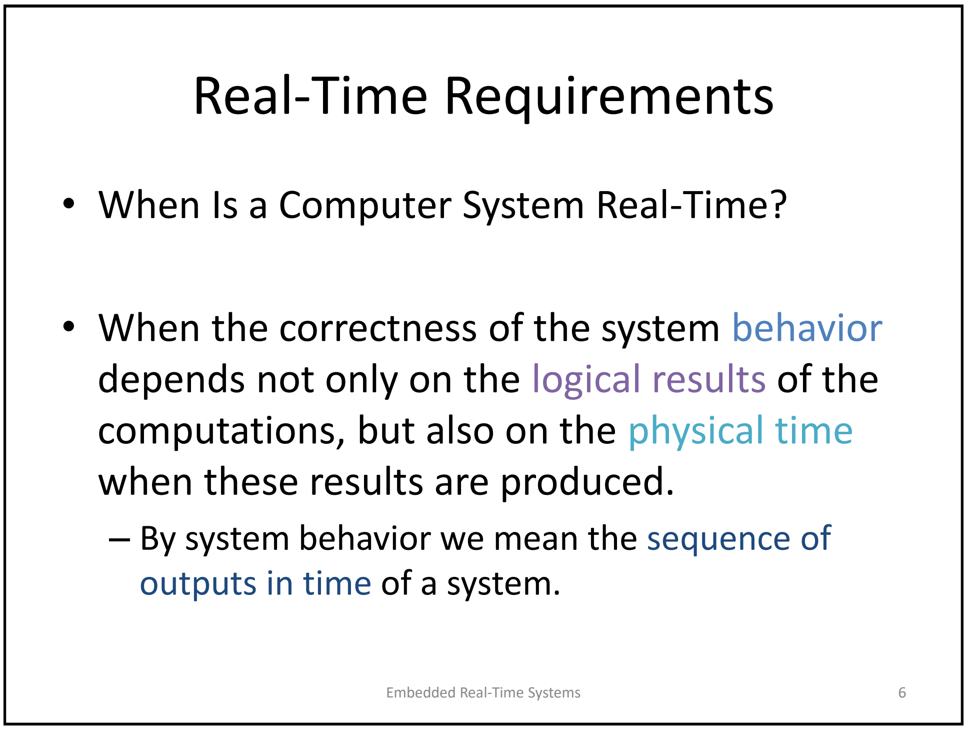
- The observation of the *RT entities* in a controlled cluster and the collection of these observations.

Direct digital control requirements

- Calculate the actuating variables for the actuators in order to control the controlled object directly.

Man–machine interaction requirements

- A real-time computer system must
  - inform the operator of the current state of the controlled object
  - assist the operator in controlling the machine or plant object.

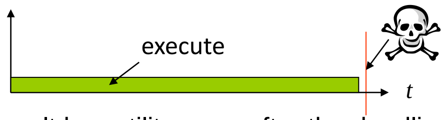


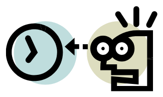
Real-Time Requirements

- When Is a Computer System Real-Time?
- When the correctness of the system behavior depends not only on the logical results of the computations, but also on the physical time when these results are produced.
  – By system behavior we mean the sequence of outputs in time of a system.

Real-Time Constraints

- CPS must meet **real-time constraints (deadlines)**
- A real-time system must react to stimuli from the controlled object (or the operator) within the time interval **dictated** by the environment.



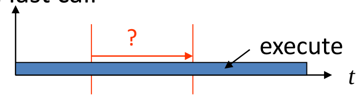


- If a result has utility even after the deadline has passed, the deadline is classified as soft;
- otherwise it is firm;
- If severe consequences could result if a firm deadline is missed, the deadline is called hard.

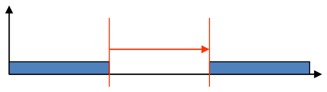
➢ A guaranteed system response has to be explained without statistical arguments [Kopetz, 1997].

Four Types of Timing Specs Required

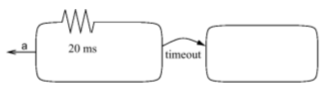
1. Measure elapsed time
   Check, how much time has elapsed since last call



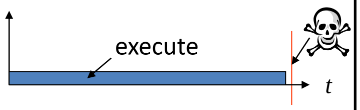
2. Means for delaying processes



3. Possibility to specify timeouts
   Stay in a certain state a maximum time.

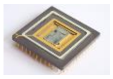


4. Methods for specifying deadlines
   Not available or in separate control file.

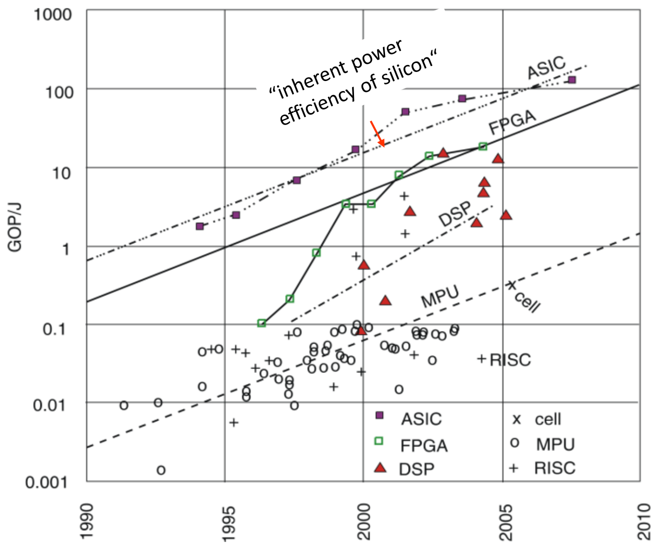


Efficiency Requirements

- CPS & ES must be **efficient**
  - Code-size efficient
    (especially for systems on a chip)
  - Run-time efficient
  - Weight efficient
  - Cost efficient
  - Energy efficient



Importance of Energy Efficiency

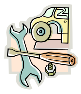


Efficient software design needed, otherwise, the price for software flexibility cannot be paid.

© Hugo De Man (IMEC) Philips, 2007

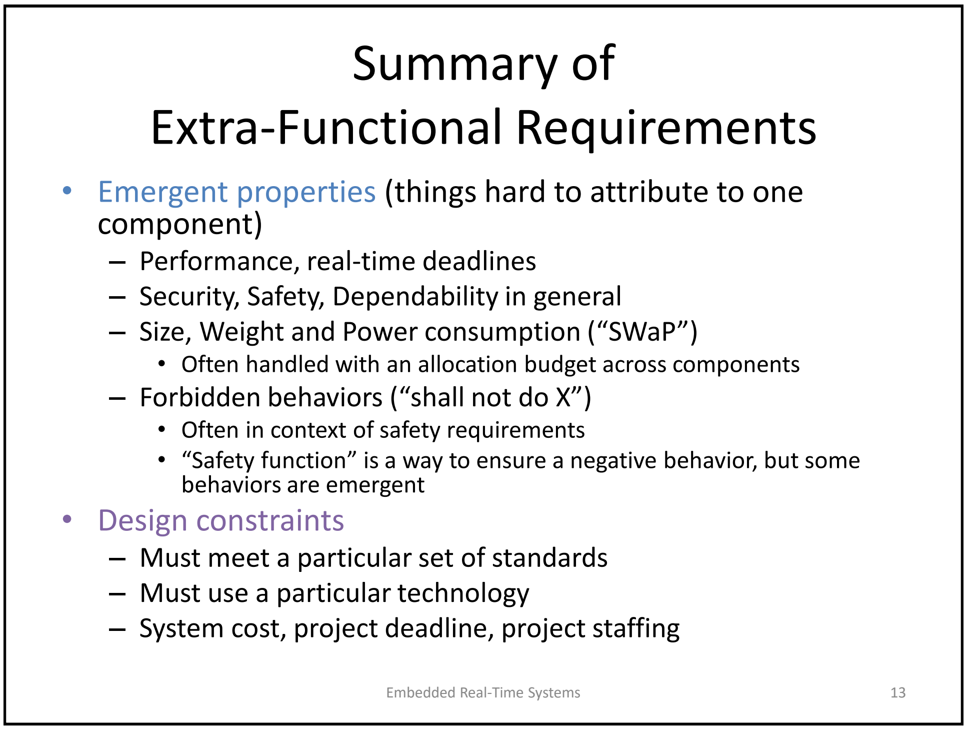
Dependability Requirements

- CPS/ES must be **dependable**, in the sense of
  - **Reliability** $R(t)$ **=** probability of system working correctly provided that is was working at t=0
  - **Maintainability** $M(d)$ **=** probability of system working correctly d time units after error occurred.
  - **Availability**: the fraction of time that the system works
  - **Safety**: no harm to be caused
  - **Security**: confidential and authentic communication



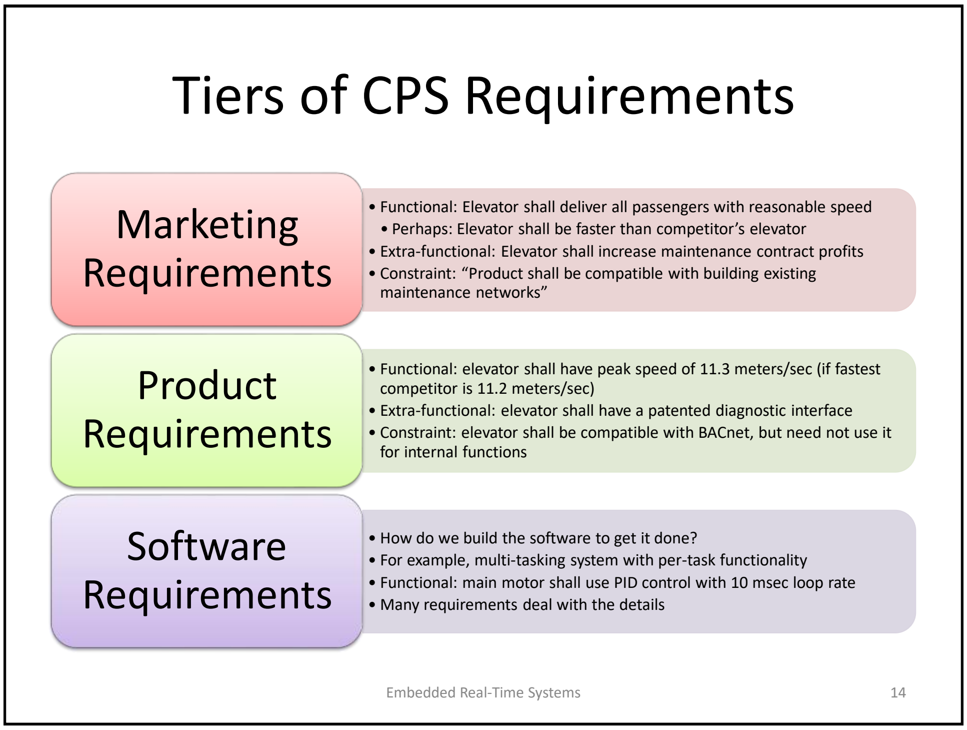
Security Requirements

- Defending against
  - Cyber crime („Annual U.S. Cybercrime Costs Estimated at $100 Billion; …[Wall Street Journal, 22.7.2013])
  - Cyber attacks (☞ Stuxnet)
  - Cyber terrorism
  - Cyber war
- Connectivity increases threats
  - entire production chains can be affected
  - local islands provide some encapsulation, but contradict idea of global connectedness
- Nowadays, we may have **cyber-physical attacks**.
  - Integration with physical domain complicates the security



Summary of Extra-Functional Requirements

- Emergent properties (things hard to attribute to one component)
  - Performance, real-time deadlines
  - Security, Safety, Dependability in general
  - Size, Weight and Power consumption (“SWaP”)
    - Often handled with an allocation budget across components
  - Forbidden behaviors (“shall not do X”)
    - Often in context of safety requirements
    - “Safety function” is a way to ensure a negative behavior, but some behaviors are emergent
- Design constraints
  - Must meet a particular set of standards
  - Must use a particular technology
  - System cost, project deadline, project staffing



Tiers of CPS Requirements

Marketing Requirements

- Functional: Elevator shall deliver all passengers with reasonable speed
  - Perhaps: Elevator shall be faster than competitor’s elevator
- Extra-functional: Elevator shall increase maintenance contract profits
- Constraint: “Product shall be compatible with building existing maintenance networks”

Product Requirements

- Functional: elevator shall have peak speed of 11.3 meters/sec (if fastest competitor is 11.2 meters/sec)
- Extra-functional: elevator shall have a patented diagnostic interface
- Constraint: elevator shall be compatible with BACnet, but need not use it for internal functions

Software Requirements

- How do we build the software to get it done?
- For example, multi-tasking system with per-task functionality
- Functional: main motor shall use PID control with 10 msec loop rate
- Many requirements deal with the details

Rules for Good CPS Software Requirements

Precise and minimally constrained

- Describes what system should do, not how it does it
- If possible has a numeric target instead of qualitative term
  - Has tolerance (e.g., 500 msec +/- 10%, "less than X")

Traceable & testable

- Each requirement has a unique label (e.g., "R-7.3")
- Each requirement cleanly traces to an acceptance test
- Requirement satisfaction has a feasible yes/no test

$$\begin{tikzcd}
U1 \arrow[r, leftrightarrow] & R1 \arrow[r, leftrightarrow] & T1 \\
U2 \arrow[r, leftrightarrow] & R2 \arrow[r, leftrightarrow] & T2 \\
U3 \arrow[r, leftrightarrow] & R3 \arrow[r, leftrightarrow] & T3 \\
U4 \arrow[r, leftrightarrow] & R4 \arrow[r, leftrightarrow] & T4 \\
U5 \arrow[r, leftrightarrow] & R5 \arrow[r, leftrightarrow] & T5
\end{tikzcd}$$

Supported within context of system

- Supported by rationale or commentary
- Uses consistent terminology
- Any conflicting requirements resolved or prioritized

Examples of Problematic Requirements

Untraceable (no label)	System shall shut down when E-STOP is activated.
Untestable	R-1.1: System shall never crash
Imprecise	R-1.7: The system provides quick feedback to the user.
No measurement tolerance	R-2.3: LED shall flash with a period of 500 msec
Overly complex	R-7.3: Pressing the red button shall activate Widget X, while pressing the blue button should cause LED Z to blink instead of LED Y illuminating steadily, which would be accomplished via the yellow button.
Describes implementation	R-8.3: Pressing button W shall cause two 16-bit integer values to be added, then …

Example of Challenging Requirements: Headlights On?

There was a highway department testing a new safety proposal. They asked motorists to turn on their headlights as they drove through a tunnel. However, shortly after exiting the tunnel the motorists encountered a scenic-view overlook. Many of them pulled off the road to look at the reflections of wildflowers in pristine mountain streams and snow-covered mountain peaks 50 miles away. When the motorists returned to their cars, they found that their car batteries were dead, because they had left their headlights on.

- So, the highway department decided to erect signs to get the drivers to turn off their head-lights.
- First they tried: TURN YOUR LIGHTS OFF. But someone said that not everyone would heed the request to turn their headlights on, and they couldn't turn their headlights off.
- So they tried: IF YOUR HEADLIGHTS ARE ON TURN THEM OFF. But someone objected that would be inappropriate if it were night time.
- So they tried: IF IT IS DAYTIME AND YOUR HEADLIGHTS ARE ON, THEN TURN THEM OFF. But someone objected that would be inappropriate if it were overcast and visibility was greatly reduced.
- So they tried: IF YOUR HEADLIGHTS ARE ON AND THEY ARE NOT REQUIRED FOR VISIBILITY, THEN TURN THEM OFF. But someone objected that many new cars are built so that their headlights are on whenever the motor is running, so they couldn't be turned off.
- So they tried: IF YOUR HEADLIGHTS ARE ON AND THEY ARE NOT REQUIRED FOR VISIBILITY, AND YOU CAN TURN THEM OFF, THEN TURN THEM OFF. But someone objected....
- They decided to stop trying to identify applicable states – just alert the drivers and let them take appropriate action. ARE YOUR LIGHTS ON?

• [Based on Gause and Weinberg, 1990]

Example Process for Embedded Requirements Creation

- **Elicitation**: Identify business/system requirements
  - Customer provides requirements in request for quote (RFQ)
  - Vendor may need to interview customer
  - engineering judgment ("guessing")
- **Create architecture**: allocate functions to subsystems
  - Class/Component diagram
  - And interfaces
- **Create scenarios/use cases**
  - High level flows through the system (flow chart)
  - "building-block" use cases catching snippets of functionality
- **Create detailed software requirements**
  - Behavioral requirements
  - Constraints

Requirements Approaches

Text document with list of requirements
- Works best if domain experts already know reqts.
- Over time, this can converge to better requirements.

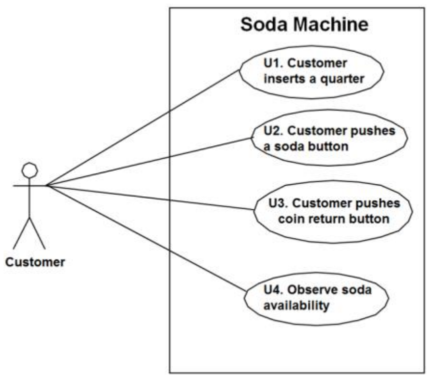
UML Use Cases
- Different activities performed by actors
- Requirements are scenarios attached to each use case

Agile User Stories
- Each story describes a system interaction

Prototyping
- Customers know it when they see it
- Sometimes a paper mock-up is enough

Functional decomposition
- Start with primary system functions
- Make more and more detailed lists of sub-functions (creates a "functional architecture")

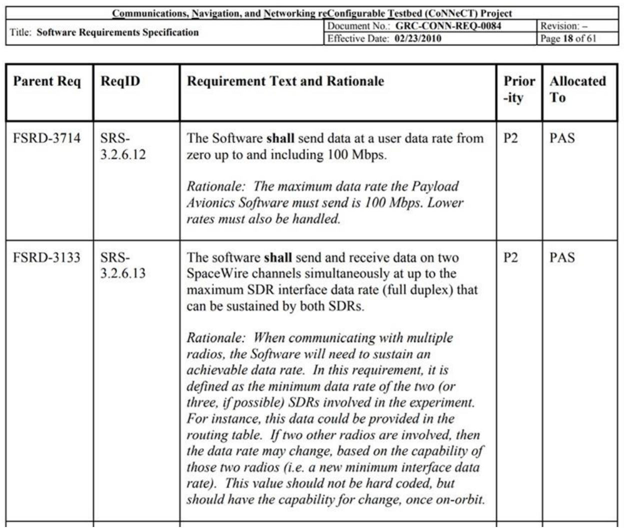
UML Use Case Diagram


Soda Machine
U1. Customer inserts a quarter
U2. Customer pushes a soda button
U3. Customer pushes coin return button
U4. Observe soda availability
Customer

Example Software Requirements

National Aeronautics and Space Administration

GRC-CONN-REQ-0084
EFFECTIVE DATE: 02/23/2010



Communications, Navigation, and Networking reConfigurable Testbed (CoNNeCT) Project		
Title: **Software Requirements Specification**	Document No.: **GRC-CONN-REQ-0084**	Revision: –
	Effective Date: **02/23/2010**	Page **18** of 61

Parent Req	ReqID	Requirement Text and Rationale	Prior-ity	Allocated To
FSRD-3714	SRS-3.2.6.12	The Software **shall** send data at a user data rate from zero up to and including 100 Mbps. *Rationale: The maximum data rate the Payload Avionics Software must send is 100 Mbps. Lower rates must also be handled.*	P2	PAS
FSRD-3133	SRS-3.2.6.13	The software **shall** send and receive data on two SpaceWire channels simultaneously at up to the maximum SDR interface data rate (full duplex) that can be sustained by both SDRs. *Rationale: When communicating with multiple radios, the Software will need to sustain an achievable data rate. In this requirement, it is defined as the minimum data rate of the two (or three, if possible) SDRs involved in the experiment. For instance, this data could be provided in the routing table. If two other radios are involved, then the data rate may change, based on the capability of those two radios (i.e. a new minimum interface data rate). This value should not be hard coded, but should have the capability for change, once on-orbit.*	P2	PAS

https://goo.gl/qct5tL

IEEE Standard 830-1998

- "IEEE Recommended Practice for Software Requirements Specifications"
  - "SRS" = Software requirements specification
  - In the embedded world, it should be "*System Requirements Specification*"!
- Areas addressed
  - Functionality (what does it do)
  - External interfaces (this is really architecture, but is important to have it in SRS)
  - Performance (speed, real-time issues)
  - Attributes (non-functional requirements such as maintainability, reliability)
  - Design constraints (applicable standards, policies, etc.)
- Outline

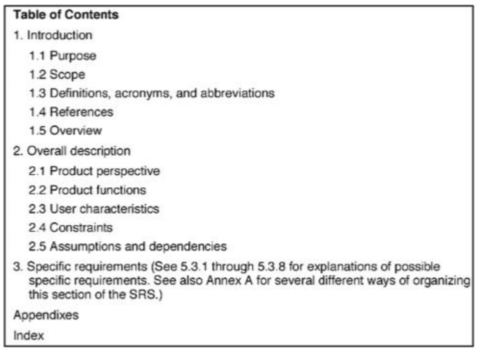


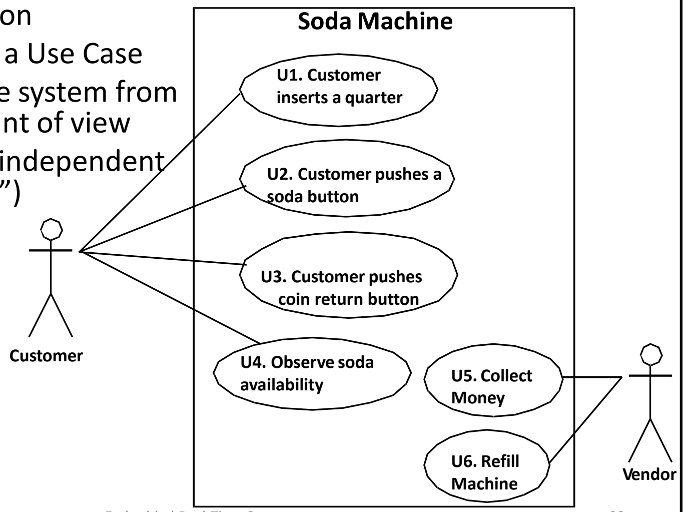
Table of Contents

Example System: Soda Vending Machine

- High Level Requirements: Make it work like a real vending machine
- Simplification:
  - Sodas cost some number of quarters
  - All other coins are rejected (invisible to your control system)
- Assume a Distributed System per given diagram
  - Processor for each button, coin return controller, vending controller
  - You get the message dictionary and most of the requirements specification (the "Architecture")

UML Use Cases for Requirements Development

- Actor is a person
- Actor initiates a Use Case
- Represents the system from the actor's point of view
- Use cases are independent ("transactions")


Soda Machine
U1. Customer inserts a quarter
U2. Customer pushes a soda button
U3. Customer pushes coin return button
U4. Observe soda availability
U5. Collect Money
U6. Refill Machine
Customer
Vendor

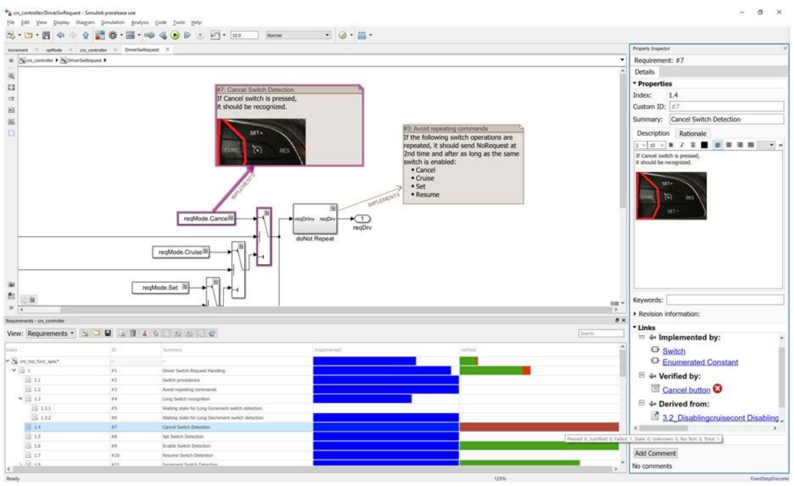
System-Level Text Requirements

- Goal: implement a soda vending machine
  - R1. Pushing a button **shall** vend a soda of the type corresponding to that button.
  - R2. The machine **shall** permanently retain exactly SODACOST coins for each can of soda vended.
  - R3. Coin return **shall** return all deposited coins since the last vend cycle.
  - R4. The machine **shall** return all deposited money in excess of SODACOST coins before a vend cycle.
  - R5. The machine **shall** flash the light for a selected item while vending is in progress to indicate acceptance of a selection to the buyer.
  - R6. The machine **shall** illuminate the light for any out-of-stock item
- Assume a Fully Distributed System
  - Processor for each button, coin return controller, vending controller

Traceability: UML & Text Requirements

	Text Requirements					
Use Cases	R1	R2	R3	R4	R5	R6
U1. Customer inserts a quarter				X		
U2. Customer pushes a soda button	X				X	
U3. Customer pushes coin return button			X			
U4. Observe soda availability						X
U5. Collect money		X				
U6. Refill machine		X				X

Simulink Requirements


crs_controller/DriverSwRequest - Simulink prerelease use
File Edit View Display Diagram Simulation Analysis Code Tools Help
Increment opMode crs_controller DriverSwRequest
IMPLEMENTS
reqMode.Cancel
reqMode.Cruise
reqMode.Set
reqDrInv reqDrv
doNot Repeat
reqDrv
SET+
CANC
RES
SET-
View: Requirements
Ready
125%
FixedStepDiscrete

#7: Cancel Switch Detection
If Cancel switch is pressed,
it should be recognized.

#3: Avoid repeating commands
If the following switch operations are repeated, it should send NoRequest at 2nd time and after as long as the same switch is enabled:
- Cancel
- Cruise
- Set
- Resume

Requirements - crs_controller

Index	ID	Summary
crs_req_func_spec*	--	--
1	#1	Driver Switch Request Handling
1.1	#2	Switch precedence
1.2	#3	Avoid repeating commands
1.3	#4	Long Switch recognition
1.3.1	#5	Waiting state for Long Increment switch detection
1.3.2	#6	Waiting state for Long Decrement switch detection
1.4	#7	Cancel Switch Detection
1.5	#8	Set Switch Detection
1.6	#9	Enable Switch Detection
1.7	#10	Resume Switch Detection
1.8	#11	Increment Switch Detection

Passed: 0, Justified: 0, Failed: 1, Stale: 0, Unknown: 0, No Test: 0, Total: 1

Property Inspector
Requirement: #7
Details
Properties
Index: 1.4
Custom ID: #7
Summary: Cancel Switch Detection
Description | Rationale
If Cancel switch is pressed,
it should be recognized.
Keywords:
Revision information:
Links
Implemented by:
Switch
Enumerated Constant
Verified by:
Cancel button
Derived from:
3.2_Disablingcruisecont Disabling
Add Comment
No comments

Best Practices for Requirements

- Six C-terms for Good Requirements
  - Clear, Concise, Correct, Coherent, Complete, and Confirmable
- Also
  - Deal with *extra-functional* issues
  - Relate requirements to design flow
    - E.g., associate with user stories or use cases
- Requirements pitfalls
  - Avoid unnecessary details and implementation
  - If it's missing from requirements, it won't get done
  - If it's not testable, you won't know if it got done

Next Lecture

- Models of computation
  - Actor-based modeling
- Continuous Dynamics
  - Causal systems
  - Memoryless systems
  - Linear time-invariant systems
- Read chapter 2 of LeeSeshia
- Review your knowledge from the "Signals & Systems" course