



# Applying Systems Engineering to embedded software development and experiences introducing formal modelling in such a process.

16 Nov 2006

[www.OpenLicenseSociety.org](http://www.OpenLicenseSociety.org)

[www.melexis.com](http://www.melexis.com)

Unifying and systematic system development methodologies with trustworthy embedded components

[Eric.Verhulst@OpenLicenseSociety.org](mailto:Eric.Verhulst@OpenLicenseSociety.org)  
[erv@melexis.com](mailto:erv@melexis.com)

16.11.2006

[www.OpenLicenseSociety.org](http://www.OpenLicenseSociety.org)

1

## Content

- Target of Open License Society
- The wider context:
  - Unified Systems Engineering Process
  - Consequences:
    - Need for trustworthy components
    - Standardisation (/= overspecification)
- OpenComRTOS project
  - Goal
  - Why formal methods
- Results so far
  - Trustworthy and better component
  - Some metrics



# From very small to very large

- **Targets:**
  - Deeply embedded distributed systems
  - But linked with rest of the world
  - Heavy constraints: real-time, memory, ...
- **Examples**
  - **Automotive:**
    - 50-100 nodes / car
    - SoC: e.g. Tire pressure sensor: A/D, uC, 2KB, wireless, ...
    - Connected wirelessly with manufacturer
  - **EU security GRID**
    - Millions of sensors, satellites, real-time analysis, storage, ...
    - Widely distributed
    - Fully linked, incl. with back-office on-line databases,...

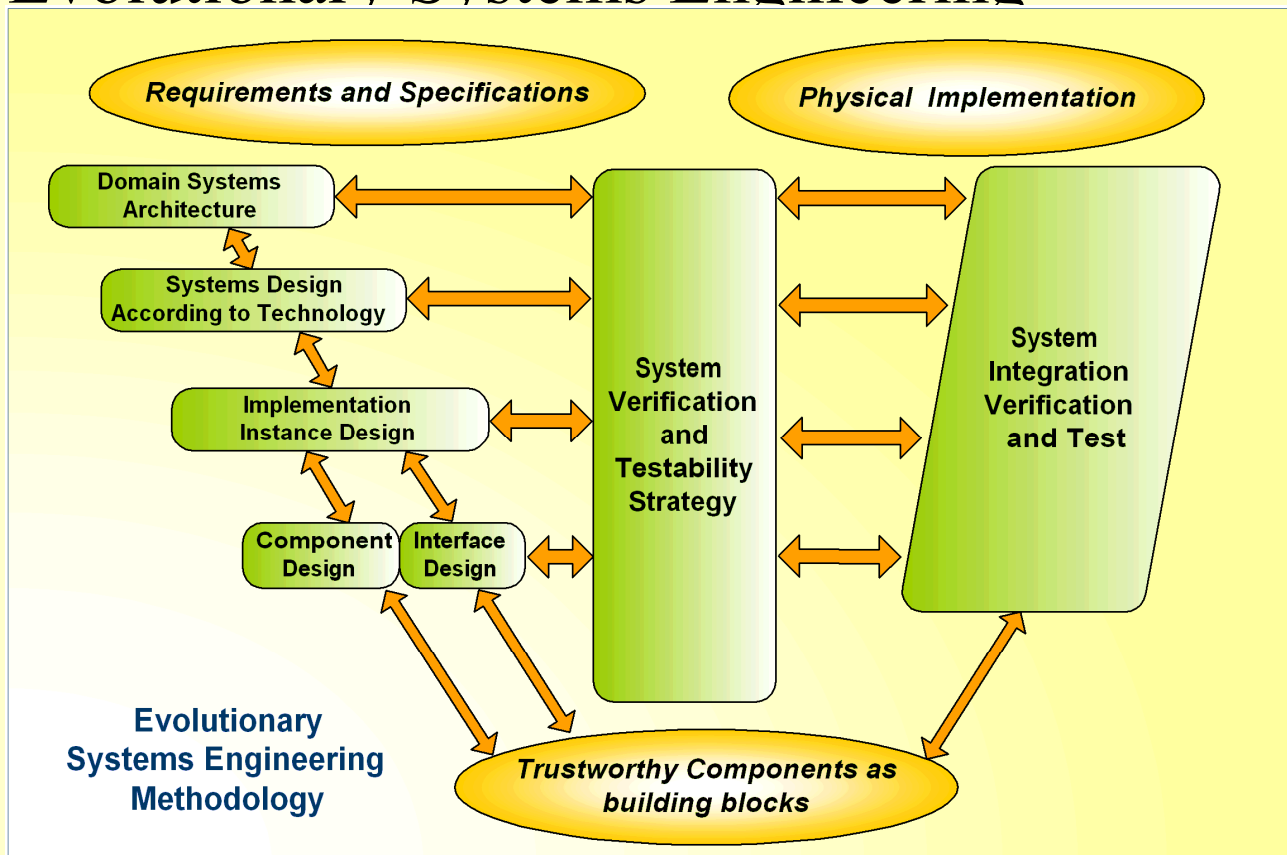
## The Wall of Complexity challenge(1)

- **Murphy's law:**
  - things going wrong is a matter of probability
  - and the odds are getting worse
- **Focus domain: embedded systems**
- **Products become systems:**
  - ,Smart' by using 10's of processors
  - Distributed operation
  - Connected to other systems (incl. wireless)
  - Human in the loop
  - Requirements:
    - High quality, high reliability
    - High level of safety, fault-tolerance
    - Secure operation
  - And as well:
    - Cost-efficient (life-cycle cost)
    - Competitive
    - Upgradeable

# The Wall of Complexity challenge(2)

- The solution is NOT to link the myriad of existing processors, software tools, etc.
- Because they are semantically too different
- Because we have too many of them
- But none supports scalability requirement
- But few support graceful degradation
- Time to apply occam's razor:
  - Back to basics
  - Get rid of unnecessary complexity and historical ballast
  - More engineering, less crafting, more scalability and real re-use
- => ,Trustworthy Embedded Components'
  - Reliability, correctness
  - Safety, fault-tolerance
  - Security
  - Formally developed and validated software & IP
  - Open License: source code + validation & design data

## Evolutionary Systems Engineering

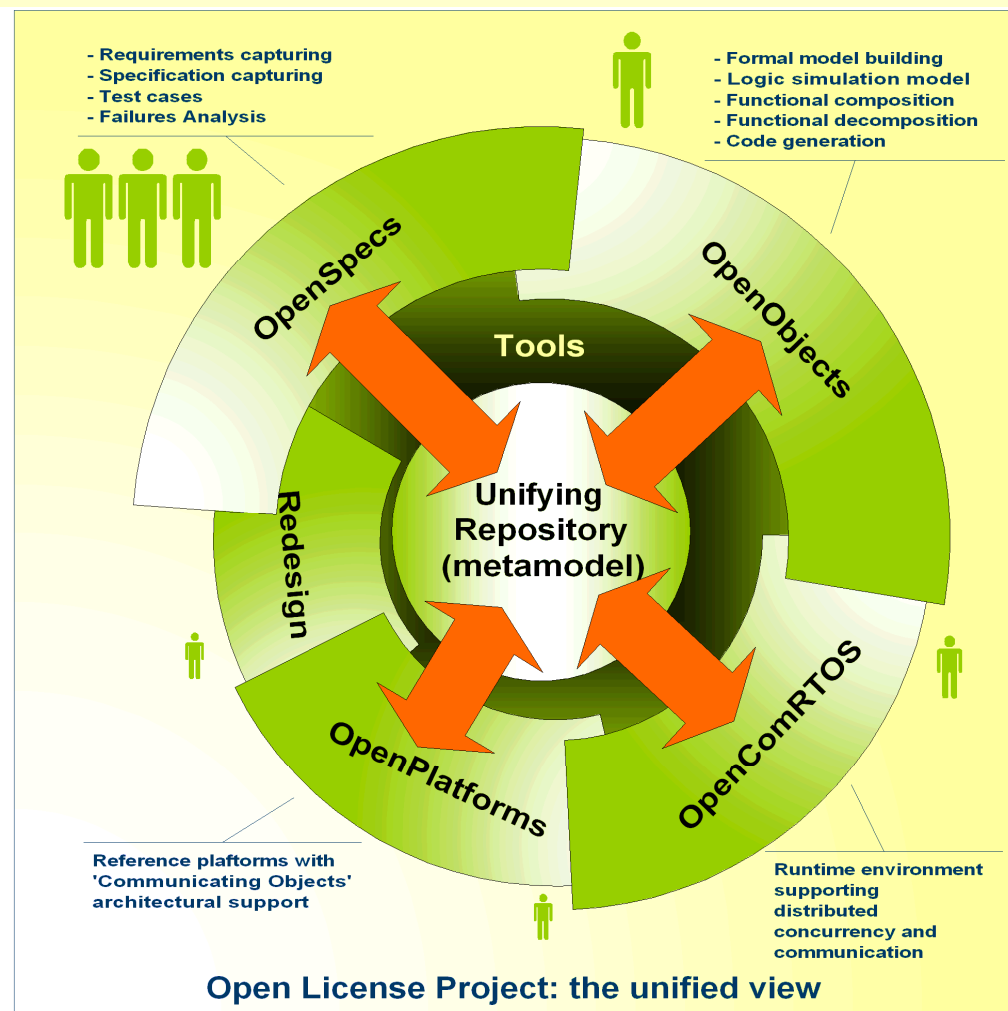


# Evolutionary: what does it mean?

- Waterfall and V-method:
  - less applicable to dynamic markets because the feedback loops are now much shorter
- Big bang approach does not work
- Evo = pragmatic blend between, extreme programming` and static project planning
  - Longer term planning as a backbone
  - Weekly milestones, weekly deadlines
  - Limits scope and increases control
  - More in line with human capacity
- Most tools on market still assume top-down approach only or start with design

# Requirements for Evo (ideally)

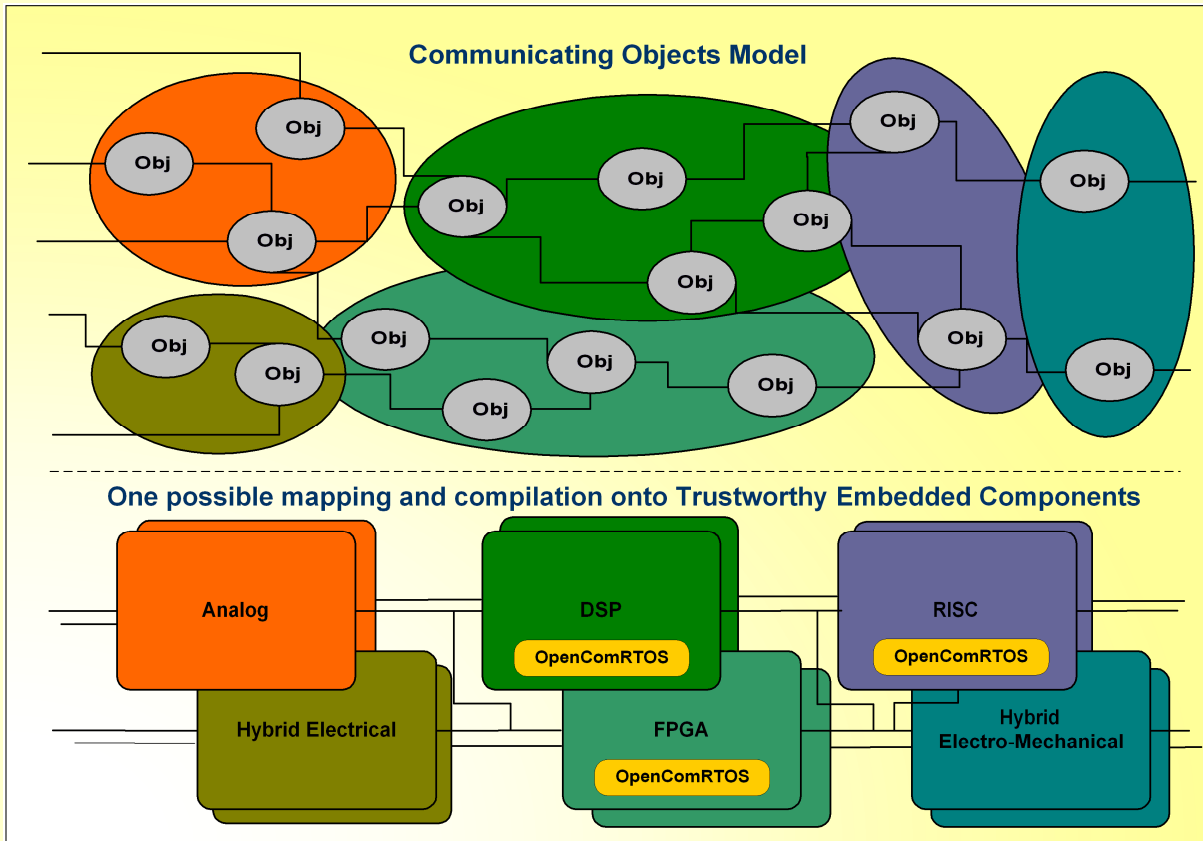
- All supporting tools must be integrated
  - Requires common ,semantics`
  - All activities from design to software to hardware must be compatible (for all properties)
  - Instant notification of any changes to all activities
- Scale of development must be consistent:
  - Modular architecture
  - Information hiding and isolation between modules
  - Formal basis
- Selected paradigm:
  - ,Interacting Entities`: universal model
  - CSP, CommUnity as formal foundation
  - Concurrent programming model



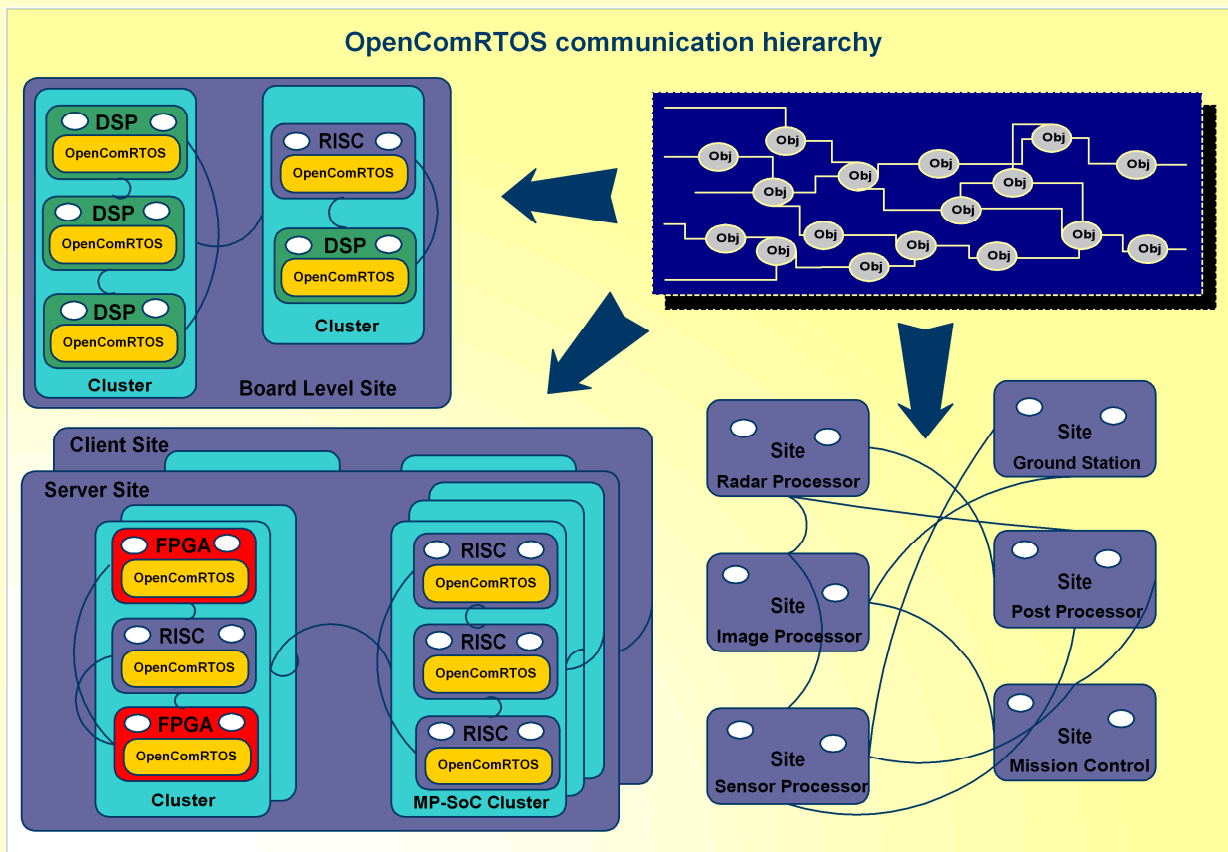
## Applicable now?

- ,Looking ahead` unifying solution
- Software not to be seen as ,add-on` but is growing component of any product
- Hardware-software co-design is more than using SystemC, it`s a process paradigm
- Essence: better ,formalised thinking`
- Benefits: higher quality, less life-cycle costs, higher added-value and margins
- Hence, can be applied incrementally
- Better specifications: easier to develop in a decentralised way
- A goal to be aimed for incrementally

# Unifying paradigm (1): Interacting Entities



# Unifying paradigm (2): Scalable Communication



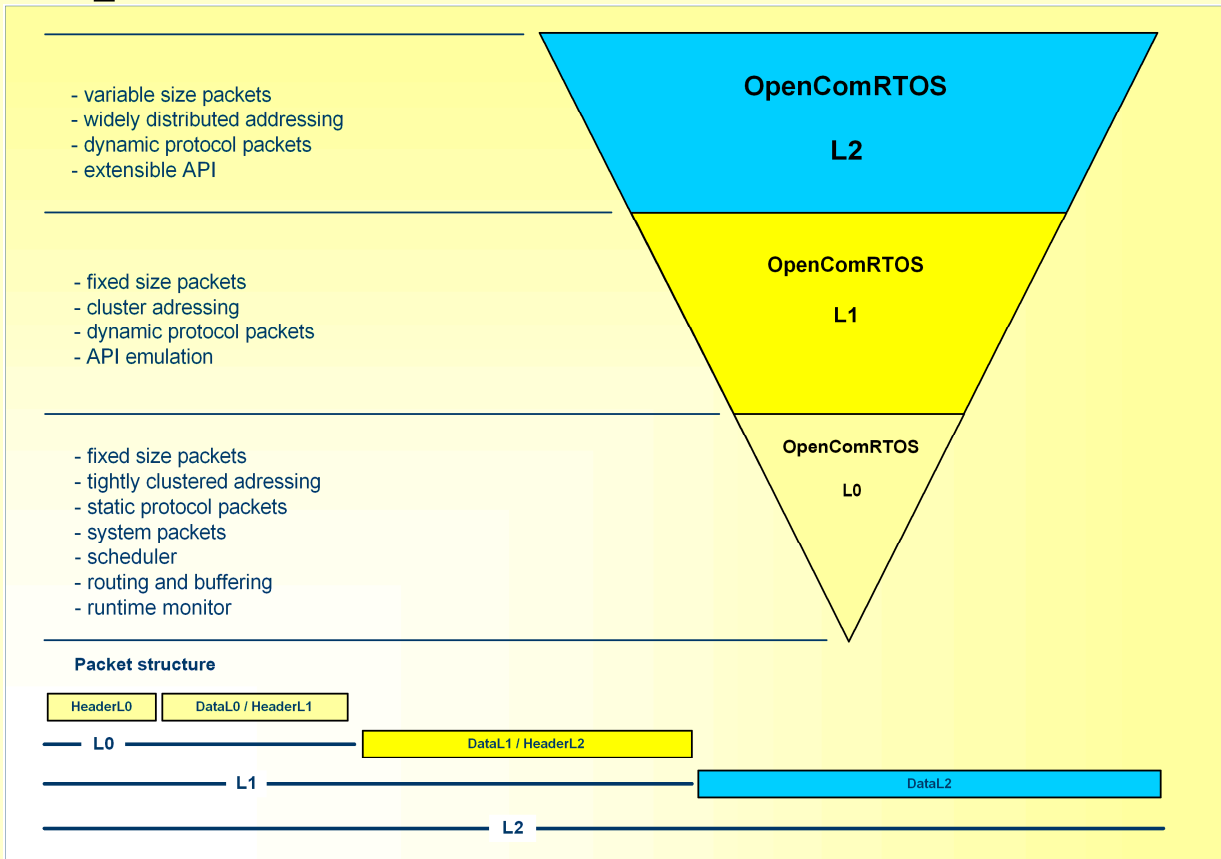
# Autosar: a well needed step

- No real unifying architecture for cars
  - Serious safety issues
  - Serious maintenance issues
  - Supply chain lock-in
  - Interconnectivity issues for software
- Autosar:
  - Top-down approach
  - Main goal: interchangeable software components
  - + better specifications
- Autosar consequences:
  - How to guarantee timing properties? (LIN+CAN+TTP+FlexRay+...)
  - Autosar defines 40 communication schemes ☹
  - Don't we need a new architecture?
    - Scalable, more margin ?
    - Unified ?
    - Trade-offs: unit cost vs. long term costs
    - Counter-example: SpaceWire (LVDS IEEE1355) -> AutoWire ?

# OpenSpecs

- Requirements and Specifications capturing
  - Redeveloped as web-based tool (team-aware) under Plone
  - Improved approach:
    - Systems engineering project with different ,views`
      - Requirements & Specifications
      - Entities and Interactions
      - Attributes, dependencies, ...
      - Project planning and workpackages
      - Normal cases, Test cases, Fault cases
    - Automatic document generation
  - Conceptually OK and tested in various domains
  - Project to add formalised analysis of natural language statements

# OpenComRTOS



## Why do we need formal techniques?

- How precise is the engineer's brain?
- How precise is the management's brain?
- How precise can we define requirements?
- How precise can we define specifications?
- How precise can we « write » software?
- How precisely do we know all dependencies?
- How sure can we be of the end-result?

# Can we trust our mind ?

- How many « F » do you count ?

**FINISHED FILES ARE THE RESULT OF YEARS OF SCIENTIFIC STUDY COMBINED WITH THE EXPERIENCE OF YEARS**

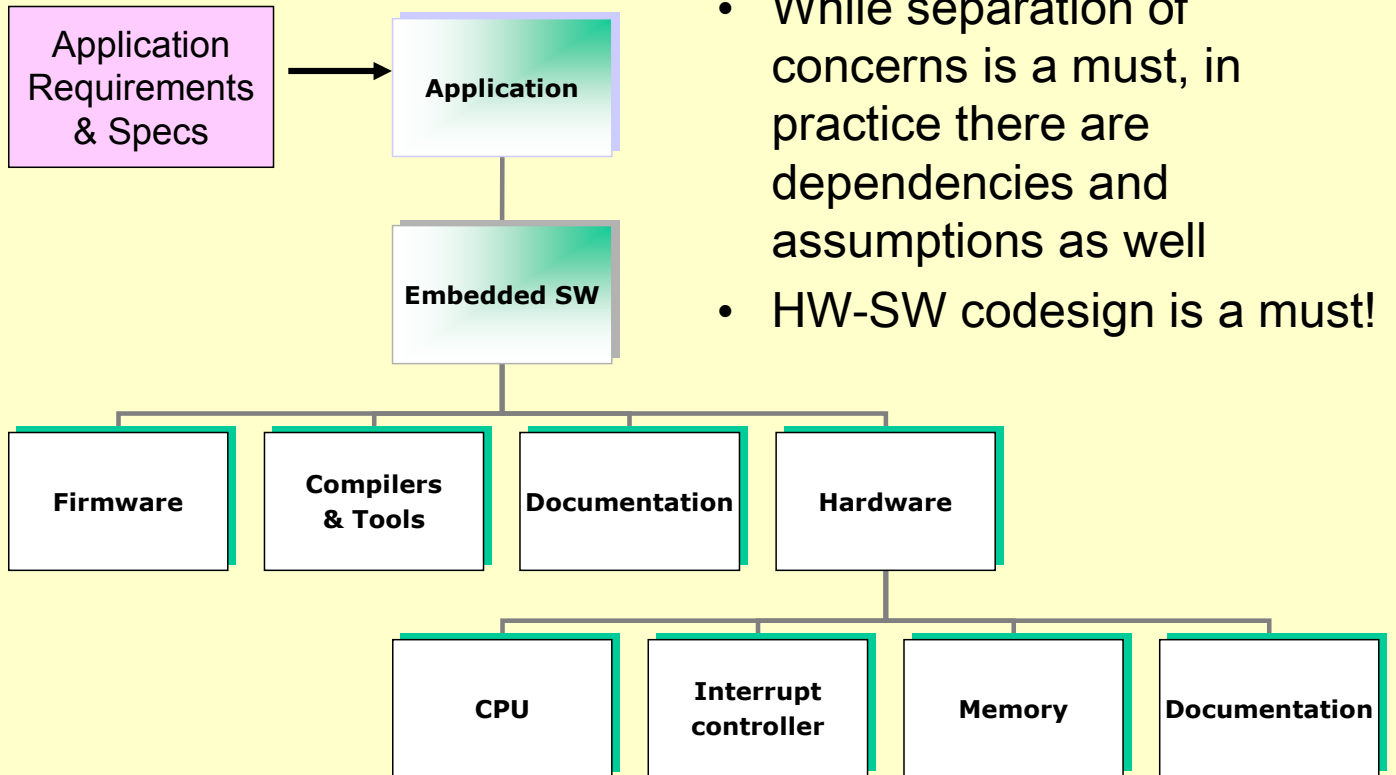
## Melexis Quality Policy

**Zero defects through  
Prevention**

**Prevention through  
Risk Management**

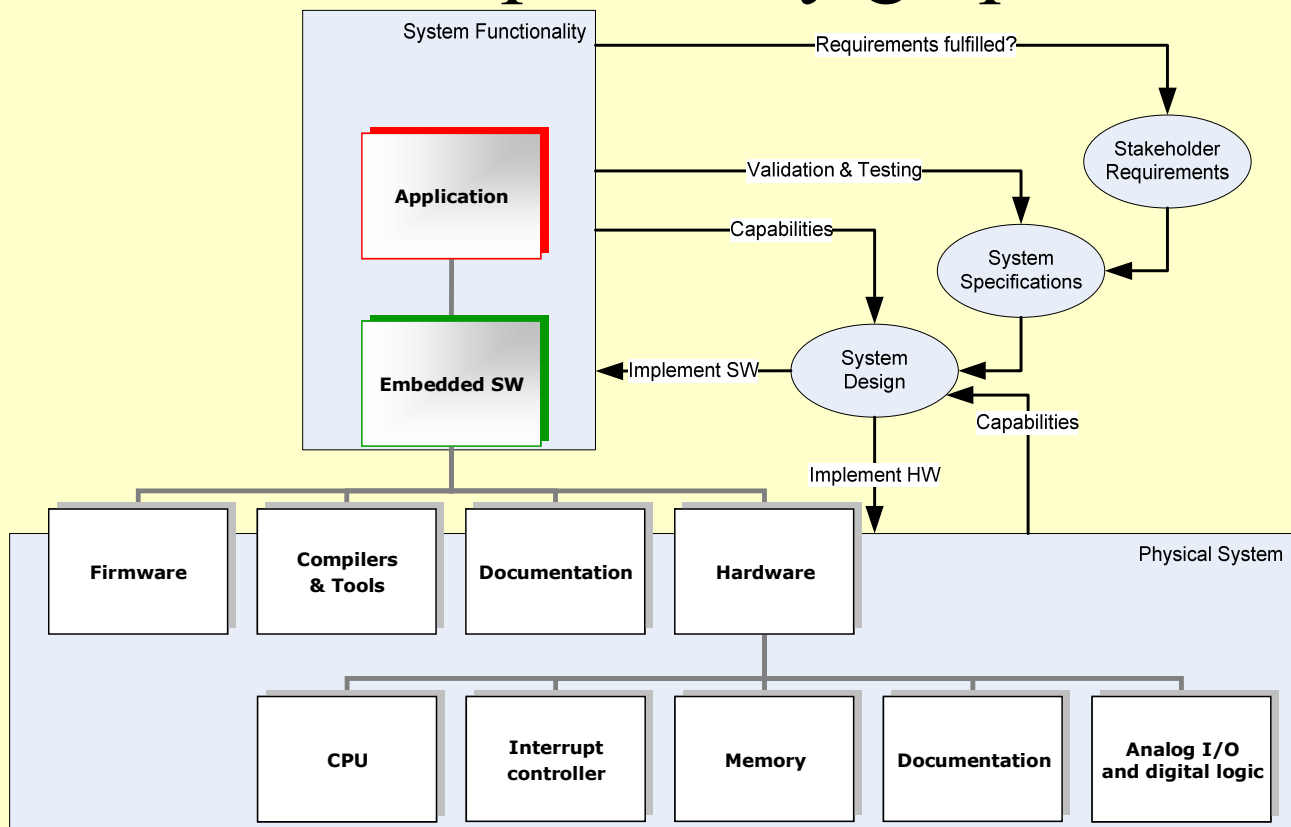
Risk Management through  
Better Concepts and Processes

# Embedded SW dependency tree

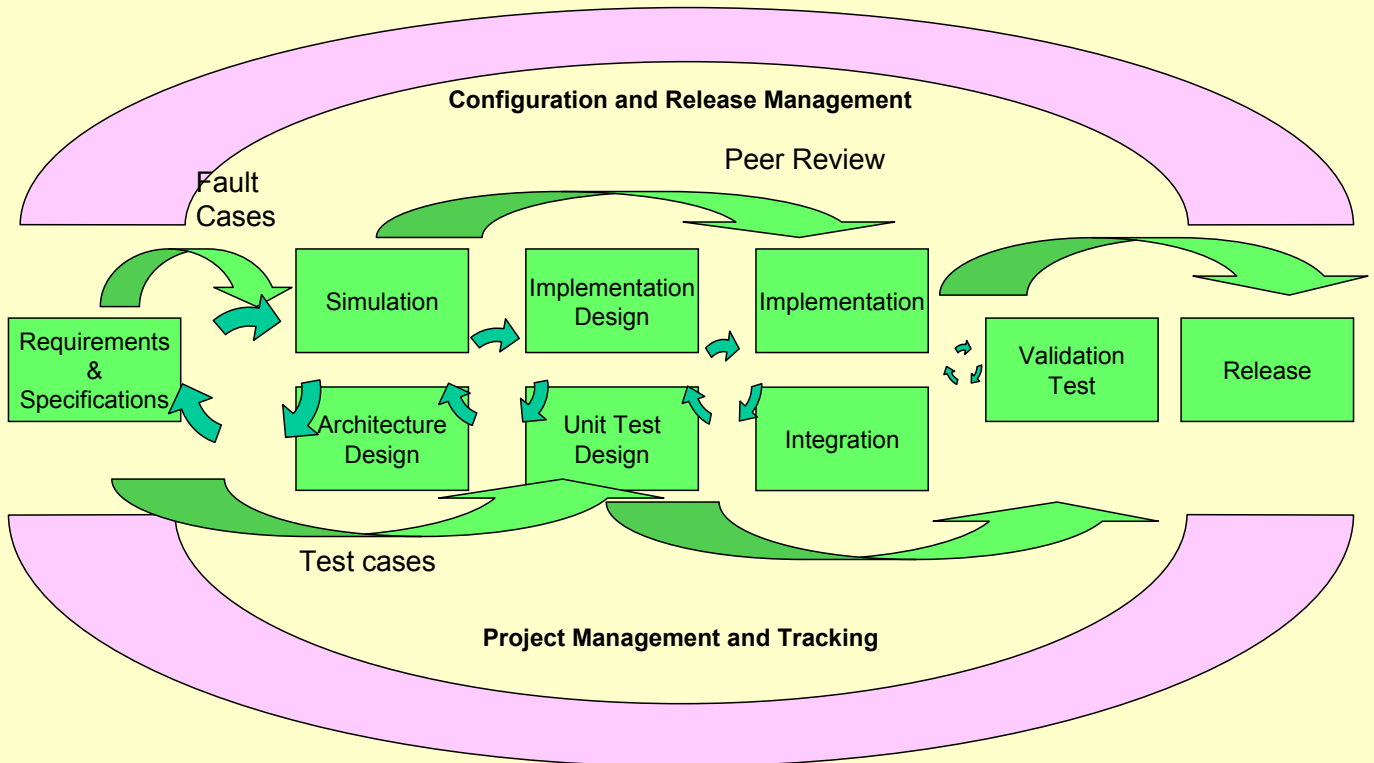


- While separation of concerns is a must, in practice there are dependencies and assumptions as well
- HW-SW codesign is a must!

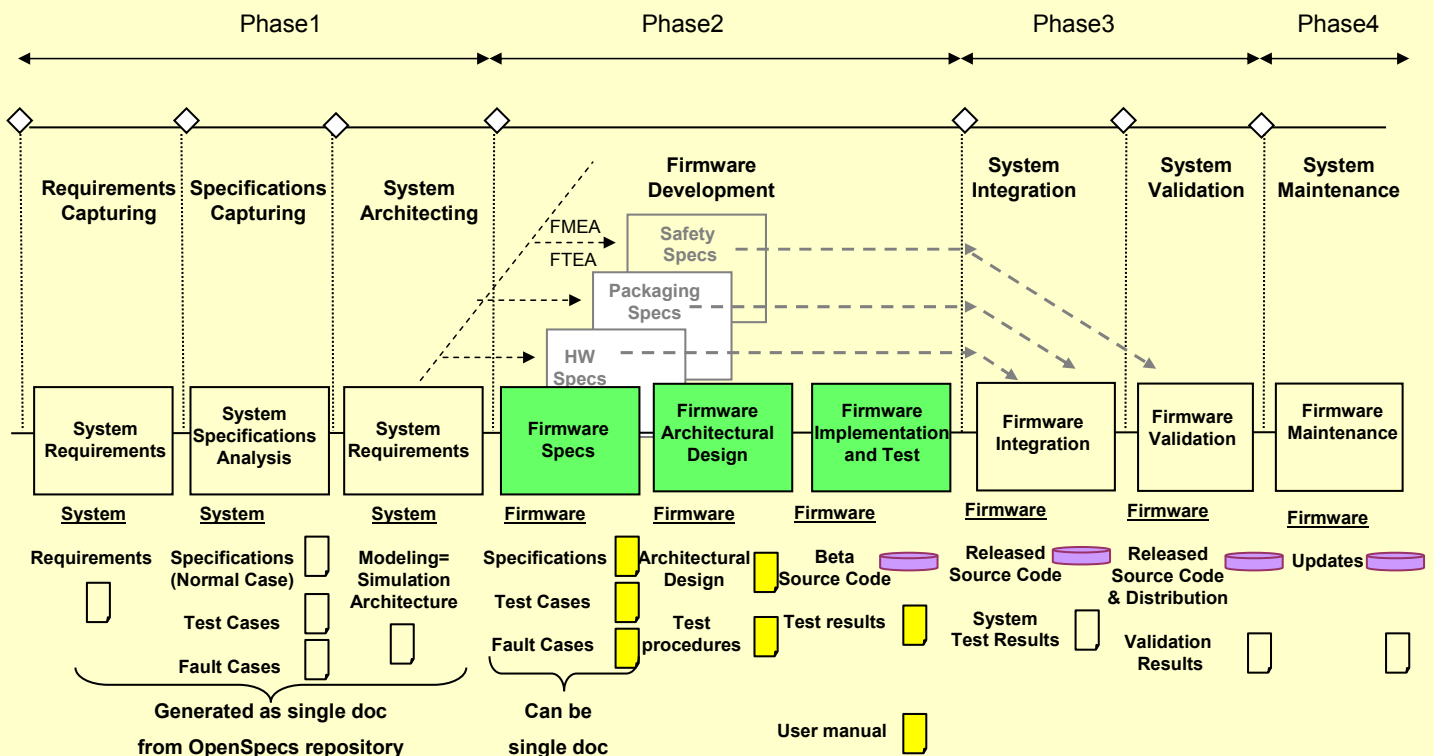
# SE Process dependency graph



# (Software) Engineering Process Flow



# Process Flow Deliverables



# Formal modeling for developing OpenComRTOS

- **Funded R&D project (IWT, Flanders)**
  - Lancelot Research: management, commercialisation
  - Open License Society: technology development
  - University Gent (INTEC, Prof. Boute): formal modeling
  - Melexis: co-sponsor and first user (16bit uC)
- **GUI tools:**
  - graphical modeling/development environment
- **Goal:**
  - Develop Trustworthy distributed RTOS
    - Follow OLS SE methodology
    - Formal verification & analysis: formal modelling
  - Scalable distributed RTOS
  - Verify benefits and issues of using Formal Modeling

## Formal modeling tools

- **Default mathematical approach:**
  - Correctness by proof
    - Labor and time intensive
    - Needs specialists
    - (Human) Error prone process
  - Tools needed
    - State space is exponentially large
    - Issues always in « hidden corners »
    - Allow incremental process
  - Requirements:
    - Support state machines
    - Support concurrency and communication
    - Low notational barrier

# Can we trust our mind ?

- How many « F » did you find ?

**FINISHED FILES ARE THE RESULT OF YEARS OF SCIENTIFIC STUDY COMBINED WITH THE EXPERIENCE OF YEARS**

Did you see the similarity with source code (debugging) ?

## Formal modeling tools: selected options

- Investigated:
  - SPIN, B, CSP (FDR), TLA+/TLC
- Outcome of process:
  - SPIN OK, initially preferred, good documentation, wide user base, but very C-like style
  - CSP: hard notation, FDR not readily available
  - B: waiting for Event B, incremental approach and compositionality very good
  - TLA+/TLC
    - Based on Temporal Logic
    - Mathematical notation, but standard
    - Works for any domain (SW, HW, ...)

# Benefits of TLA+/TLC

- TLA+/TLC home page on <http://research.microsoft.com/users/lamport/tla/tla.html>
- Initial models reflect “programming style”
  - That’s the way the mind works (after being conditioned ...)
  - > 28 successive models from 2 pages to 25 pages
    - Initially very abstract, neglecting details
    - All successive models were correct, why ?
      - Iterative, incremental process!
      - Takes 15 minutes from one model to the next
    - Interplay between software architects and formal modeling engineer
      - Architectural model polluted by programming concepts
      - Abstraction from TLA helped to find these issues
      - Result: much cleaner, safer and performant architecture
- TLA models do not prove software is correct (! ?)
  - TLC proves that Formal **Models** are correct

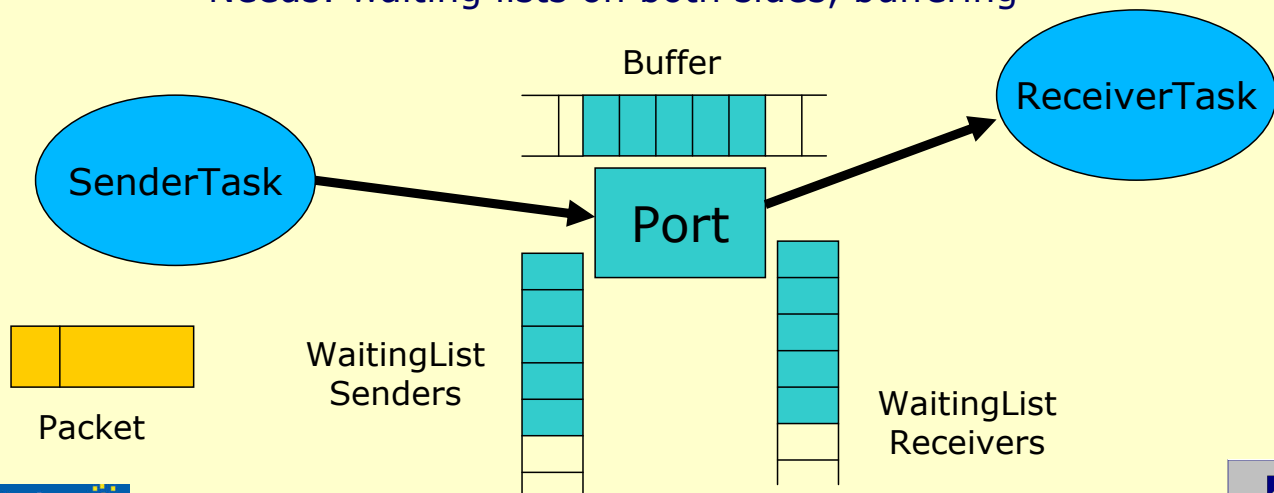
# Issues with TLA+/TLC

- Needs a few months to get the right modeling style (especially concurrency)
- TLC declares critical section over all actions
  - In RTOS must be minimal
  - Requires good know-how of target processor
  - Why can't FM not give the minimum critical sections?
- State Space is exponential
  - Millions of states for small application test model
  - Might need hours to check
  - Tracing illegal states not always trivial

# Example

- OpenComRTOS-L0

- Tasks (= function with its own workspace and context)
- Ports: for exchanging L0\_Packets
  - SendPacket\_(N)W(T)(async)
  - ReceivePacket\_(N)W(T)(async)
  - Needs: waiting lists on both sides, buffering



# Formally modeled

```

TypeInvariant == /\ ppool \in [Adr -> Packet \union {NoData}]
                /\ PQ \in [ FIFO : [Port -> Seq(Adr)],
                           WL  : [Port -> Seq(Adr)]]

                /\ chan \in [ val: [HLink -> Packet \union {NoData}],
                           stt: [HLink -> {"free", "busy"}]]

                /\ TxQ \in [TxChan -> Seq(Packet)]
                \*      /\ tstate \in [UTask -> {"running", "ready", "wait4anS", "wait4anR"}]
    
```

67  $TypeInvariant \triangleq \wedge ppool \in [Adr \rightarrow Packet \cup \{NoData\}]$

69  $\wedge PQ \in [FIFO : [Port \rightarrow Seq(Adr)],$   
 70  $WL : [Port \rightarrow Seq(Adr)]]$

$\wedge chan \in [val: [HLink \rightarrow Packet \cup \{NoData\}], stt: [HLink \rightarrow \{"free", "busy"\}]]$

75  $\wedge TxQ \in [TxChan \rightarrow Seq(Packet)]$

77  $\wedge tstate \in [UTask \rightarrow \{"running", "ready", "wait4anS", "wait4anR"\}]$

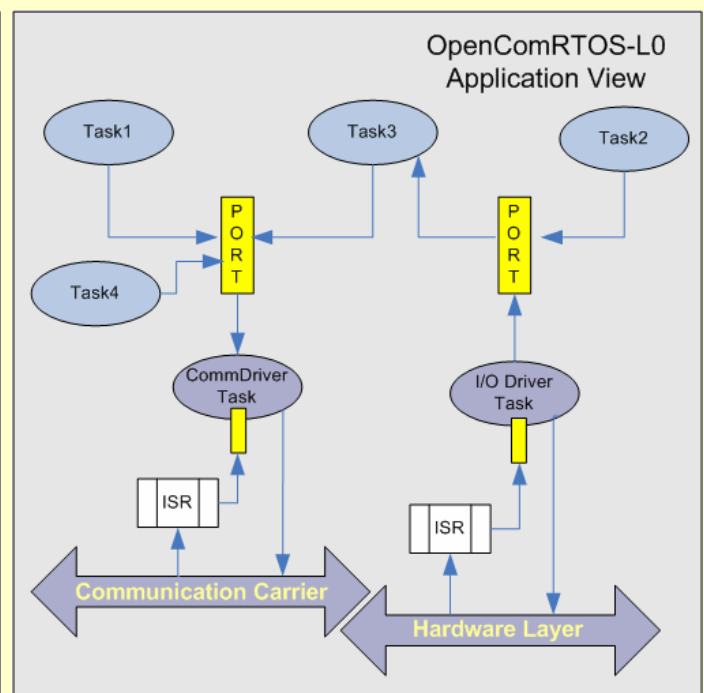
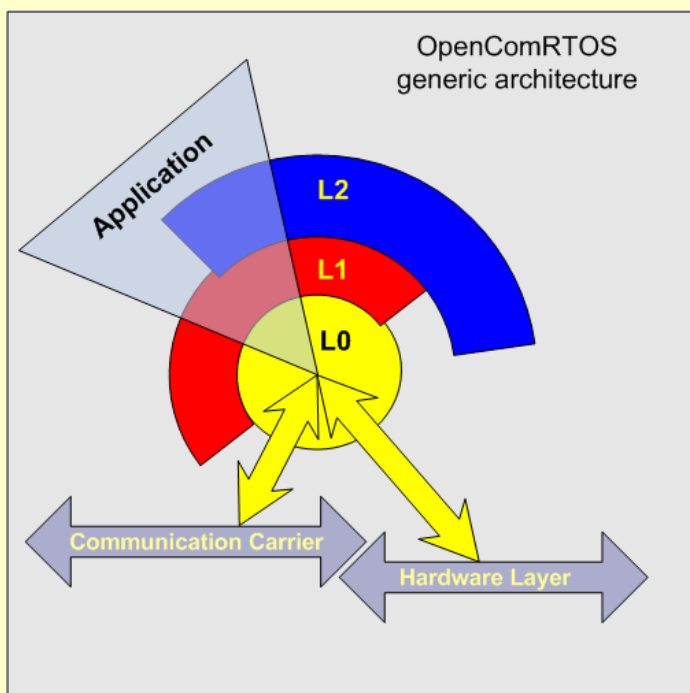
# What was wrong?

FIFO : [Port -> Seq(Adr) ] ,

WL : [Port -> Seq(Adr) ] ]

- Both (abstract) models are the same
- Natural language is imprecise, semantics are context driven
- We forget the hidden assumptions
  - FIFO := buffering of data
  - WaitingList := waiting, descheduled
  - But: both « buffer » and « wait »
  - Result: either FIFO, either WaitingList
  - « Trick »: all entries are pointers (Addr) to Packets
- Benefits:
  - Infinite buffering until no more memory (for Packets)
  - Overflow-free buffering

# Generic Open-Comm-RTOS



# Other example (L1)

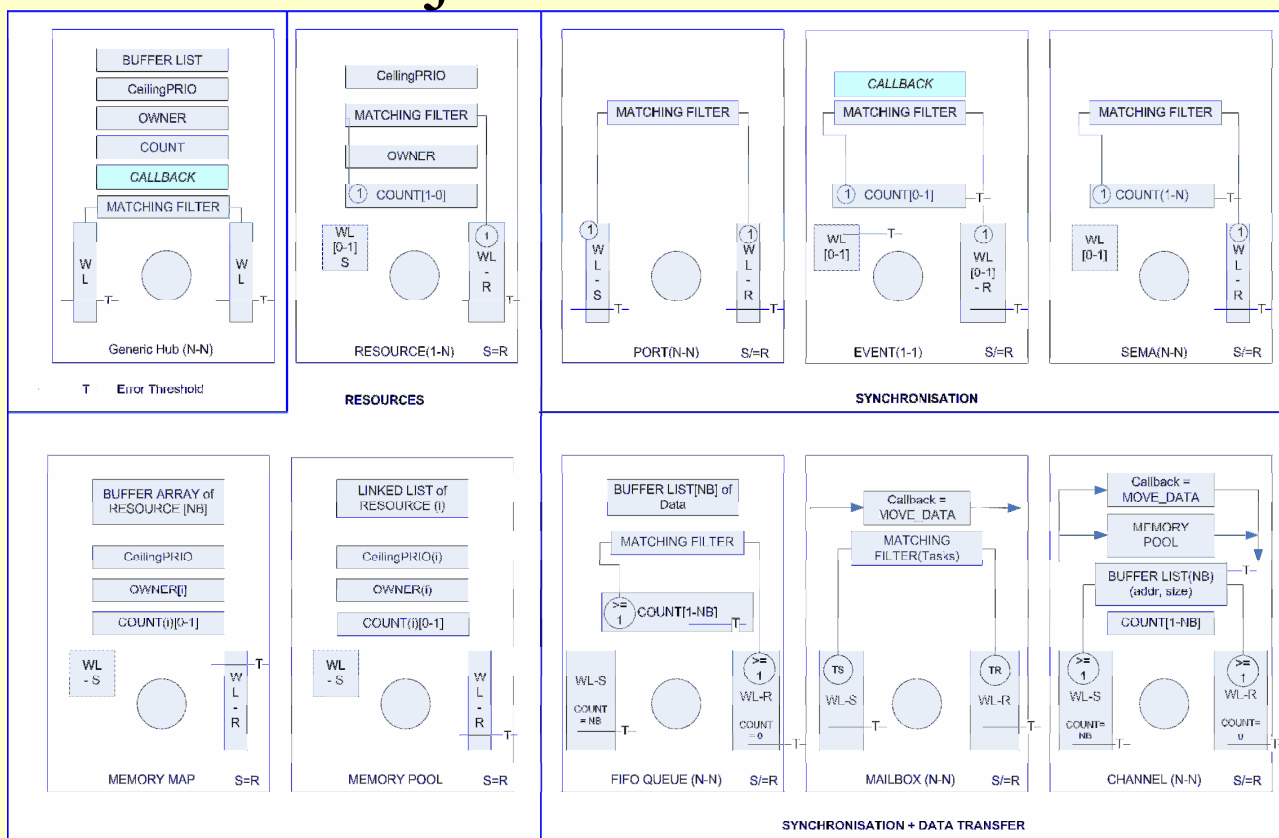
- OpenComRTOS layer L1

- Traditional RTOS support:
  - Preemptive scheduling with priority inheritance support
- Services with rich and diverse semantics:
  - Events, semaphores, FIFO-queues, mailboxes, channels, pipes, mutex, resources, memory pools, ...
  - « distributed semantics »
- 100's of RTOS with such support
- 15 years of experience, 3 generations of RTOS design
- L1 layer can be any API

- What did we find?

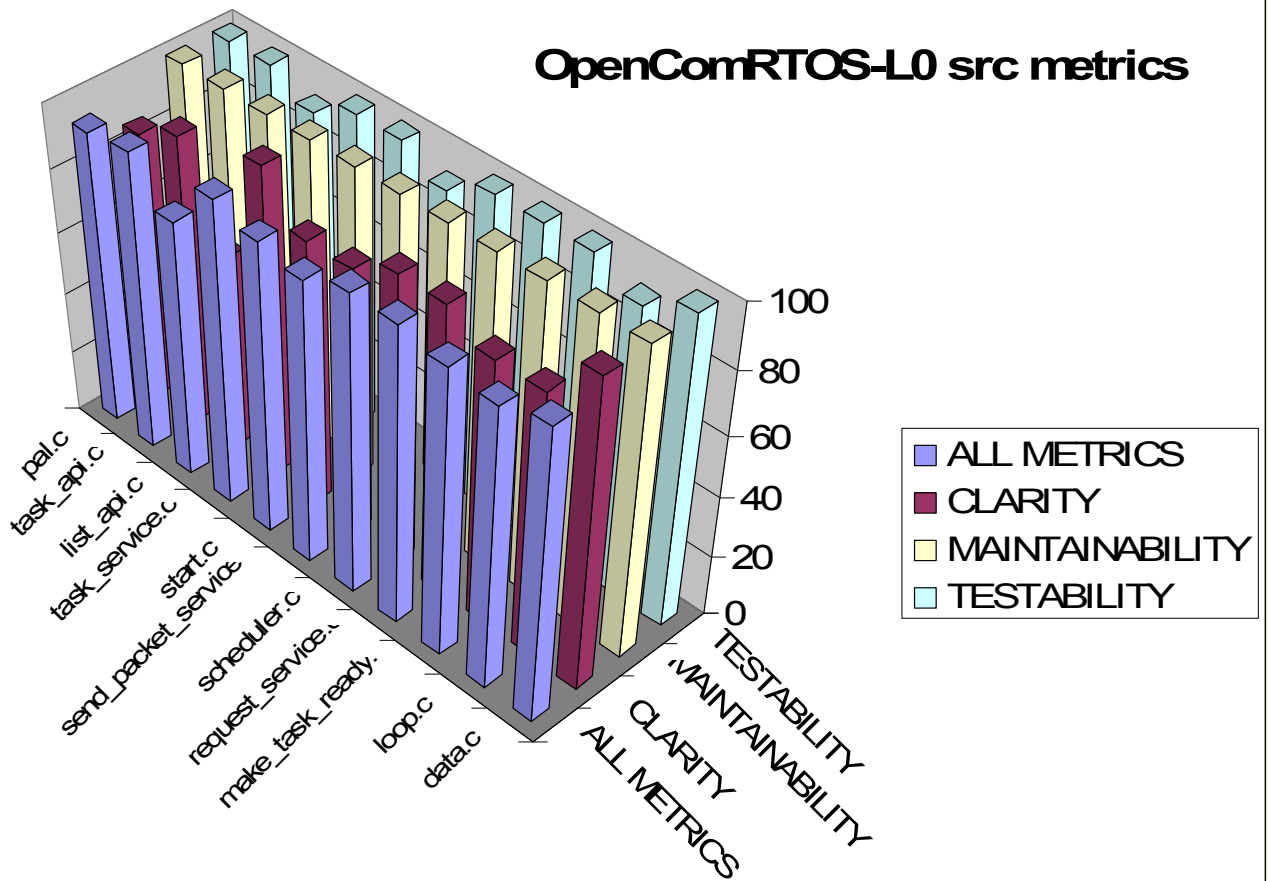
- Scheduling algorithm can be improved to reduce worst-case rescheduling latency and blocking time
- All RTOS objects are variations of the same generic « hub » object. Result: less but faster code
  - 2 KBytes vs. 50 KBytes before

## All RTOS objects: variation on a theme





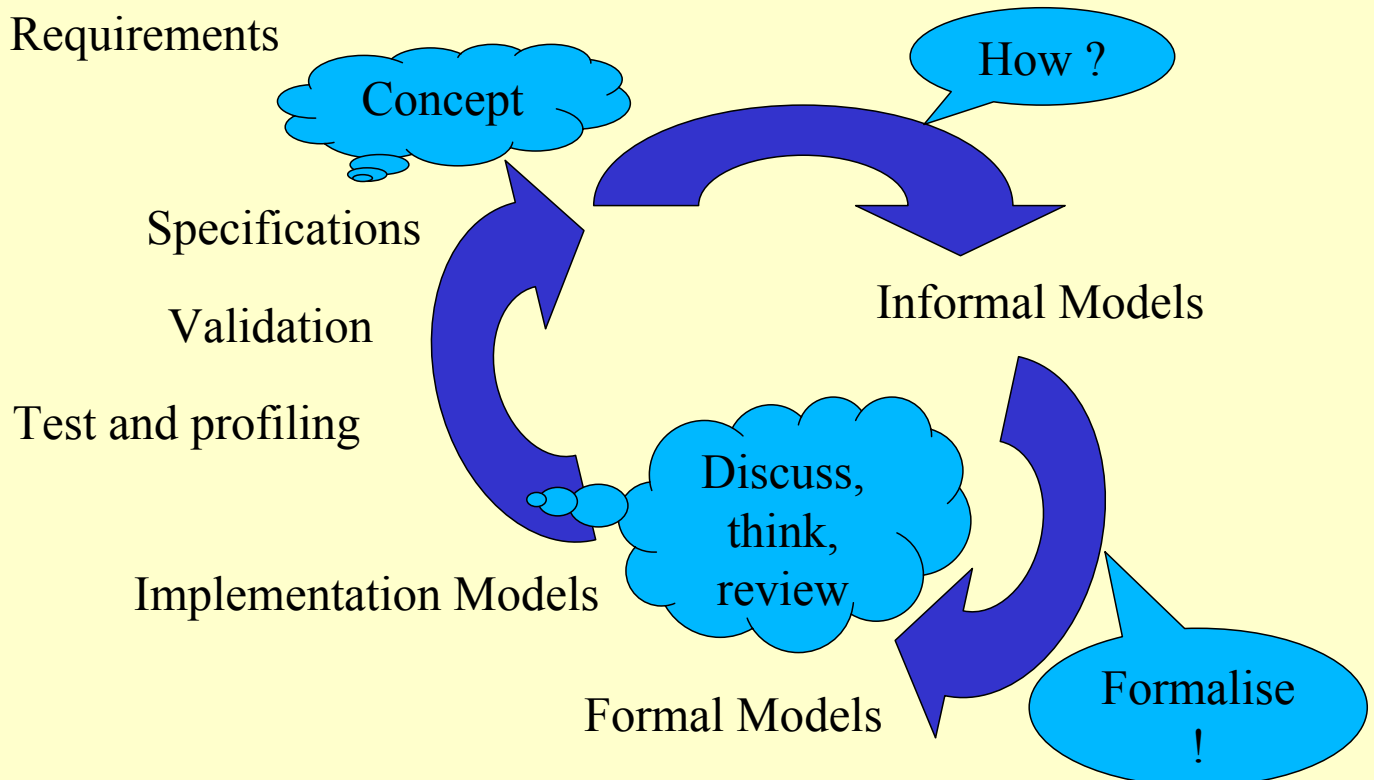
## OpenComRTOS-L0 src metrics



Output from LDRA

## How it really works: teamwork

Requirements



# Summary

- Open License Society's approach is about ,formalised thinking`
- The essence is the SE process
  - not the tools, but they help a lot
- The benefits are "things being done better"
  
- Contact:  
eric.verhulst@OpenLicenseSociety.org  
erv@melexis.com