System design techniques

- Design methodologies.
- Requirements and specification.

Design methodologies

- Process for creating a system.
- Many systems are complex:
  - large specifications;
  - multiple designers;
  - interface to manufacturing.
- Proper processes improve:
  - quality;
  - cost of design and manufacture.

Product metrics

- Time-to-market:
  - beat competitors to market;
  - meet marketing window (back-to-school).
- Design cost.
- Manufacturing cost.
- Quality.

Spacecraft design errors

- Mars Climate Observer lost on Mars in September 1999.
- Requirements problem:
  - Requirements did not specify units.
  - Lockheed Martin used English; JPL wanted metric.
- Not caught by manual inspections.

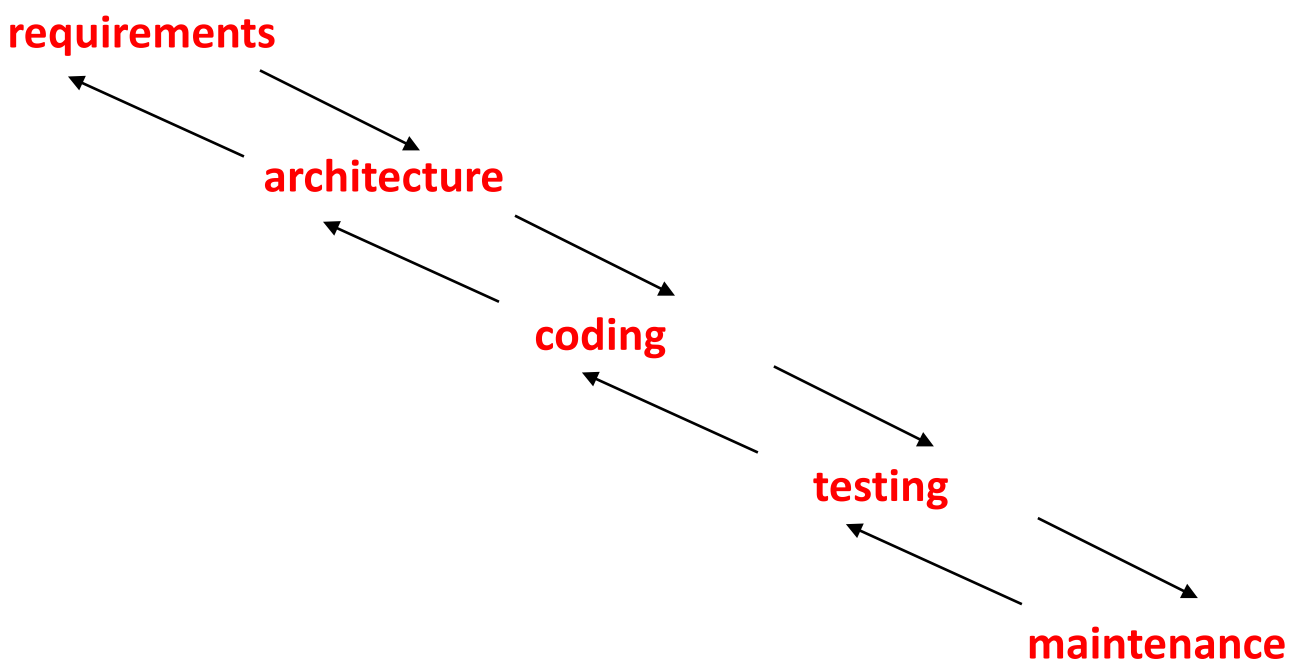
- New Horizons experienced 81-minute comm blackout while approaching Jupiter in 2015.
- LightSail malfunctioned in orbit due to a file overflow bug.
  - Bug occurred only after about 40 hours of operation, ground tests did not run long enough.

Design flow

- Design flow: sequence of steps in a design methodology.
- May be partially or fully automated.
  - Use tools to transform, verify design.
- Design flow is one component of methodology. Methodology also includes management organization, etc.

Waterfall model

- Early model for software development:



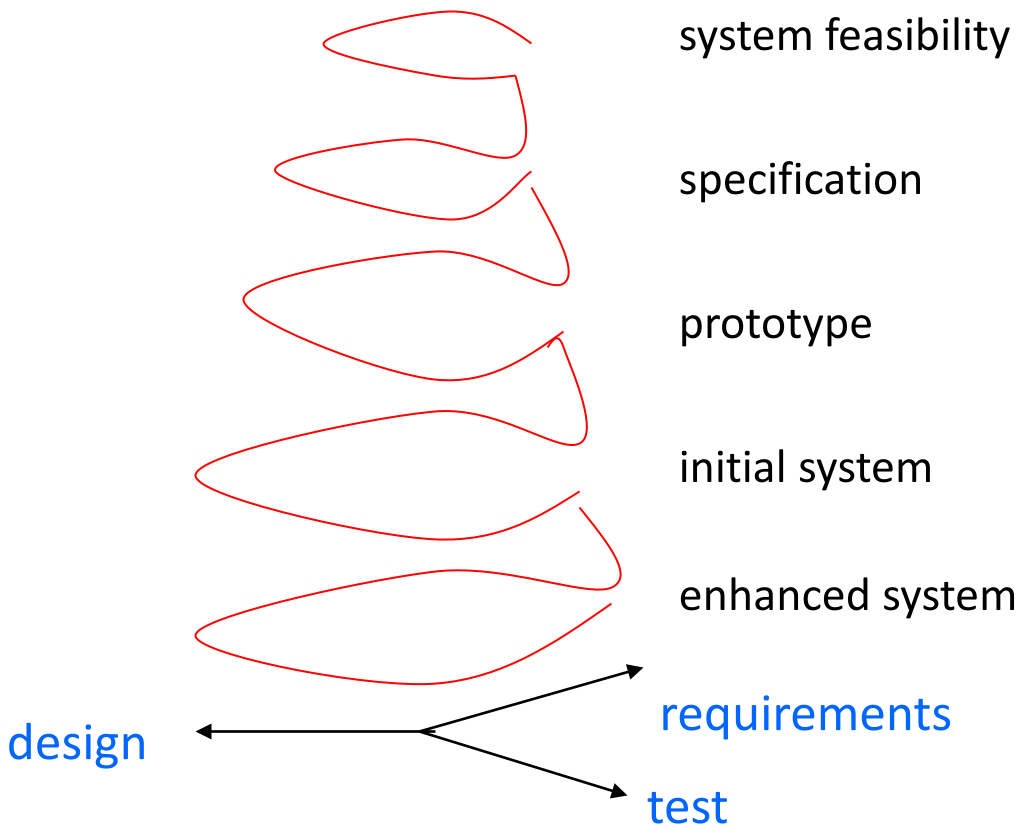
Waterfall model steps

- Requirements: determine basic characteristics.
- Architecture: decompose into basic modules.
- Coding: implement and integrate.
- Testing: exercise and uncover bugs.
- Maintenance: deploy, fix bugs, upgrade.

Waterfall model critique

- Only local feedback---may need iterations between coding and requirements, for example.
- Doesn’t integrate top-down and bottom-up design.
- Assumes hardware is given.

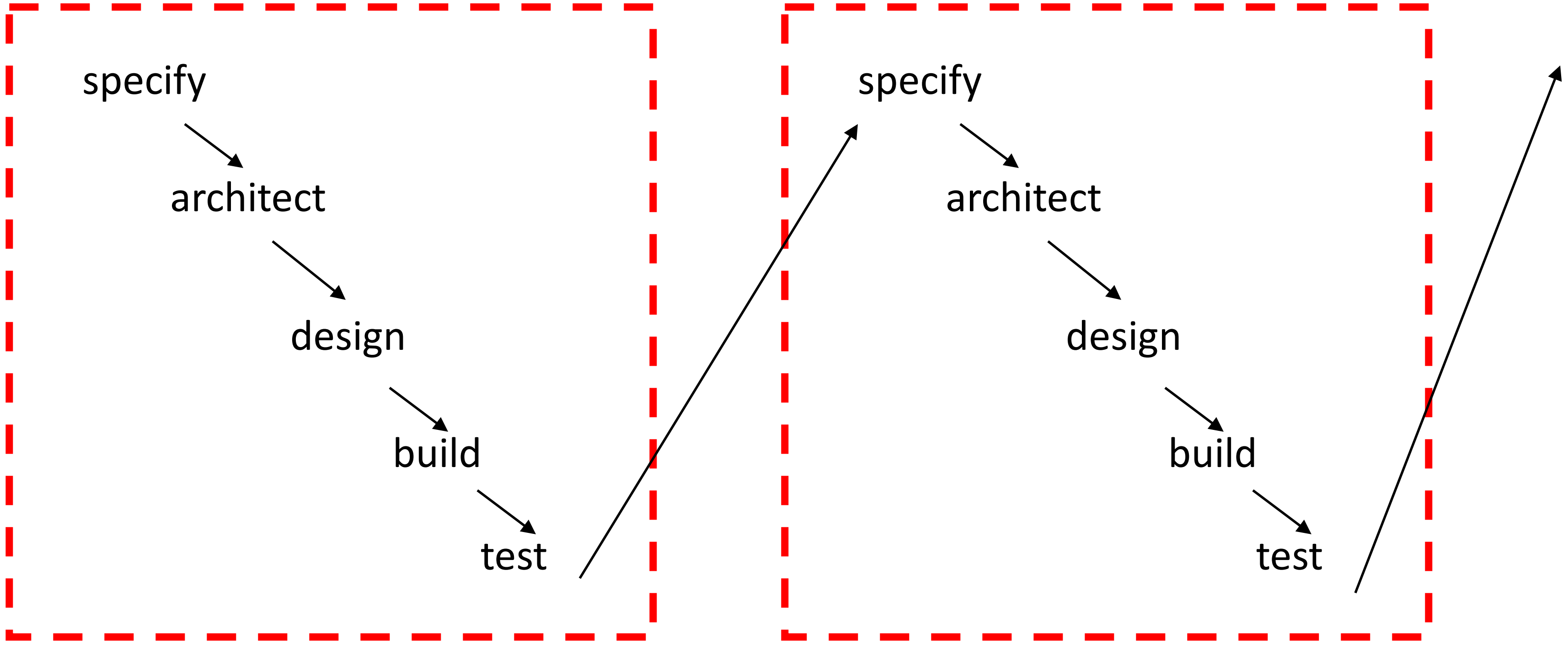
Spiral model



Spiral model critique

- Successive refinement of system.
  - Start with mock-ups, move through simple systems to full-scale systems.
- Provides bottom-up feedback from previous stages.
- Working through stages may take too much time.

Successive refinement model



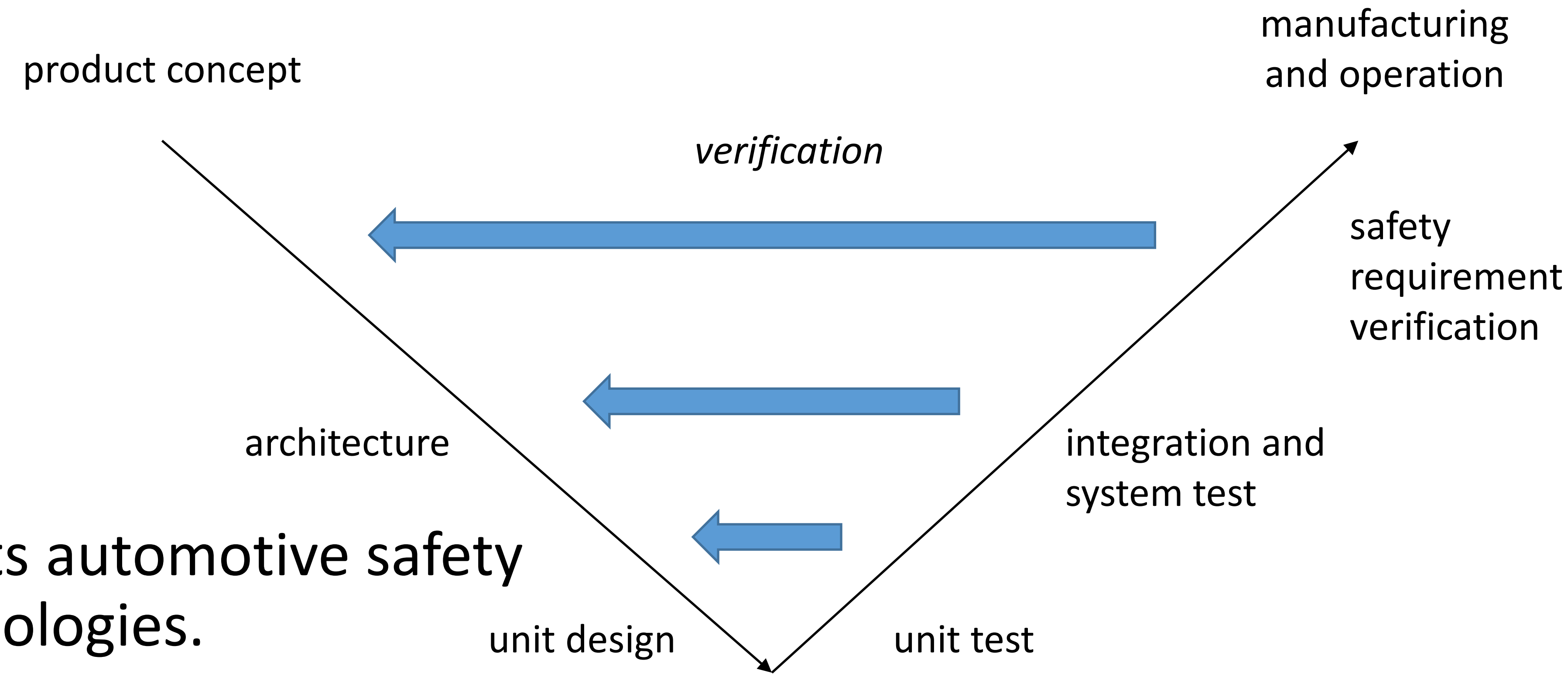
initial system

refined system

Agile software design

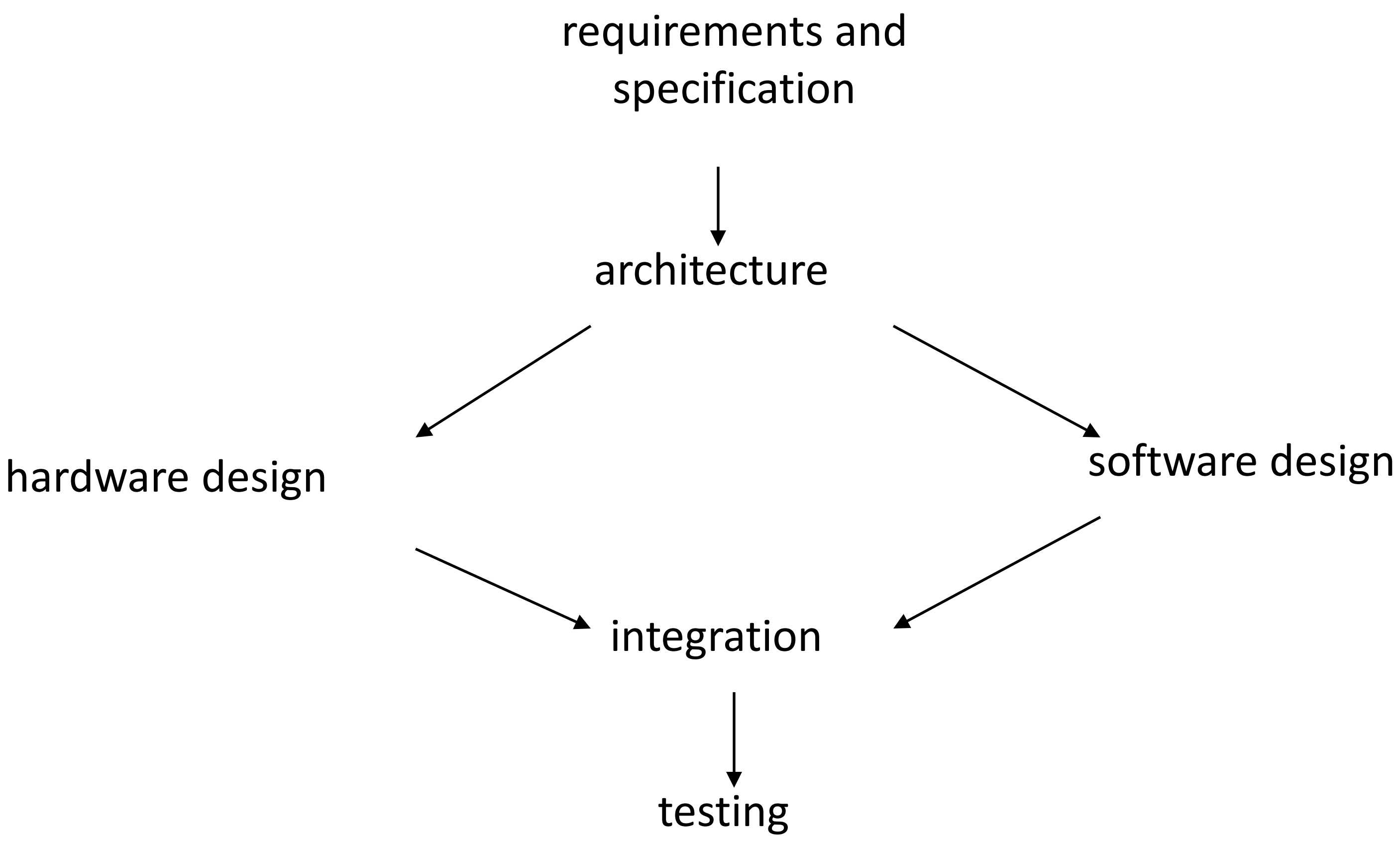
- Minimize time, cost of deployment.
  - Reduced or eliminated requirements process.
  - Reduced documentation.
  - Simplified testing.
- Often used in consumer-oriented software.
- Does not provide documentation, testing required in many real-time embedded system applications.

V model



- Supports automotive safety methodologies.
- Top-down design followed by bottom-up verification.

Hardware/software design flow



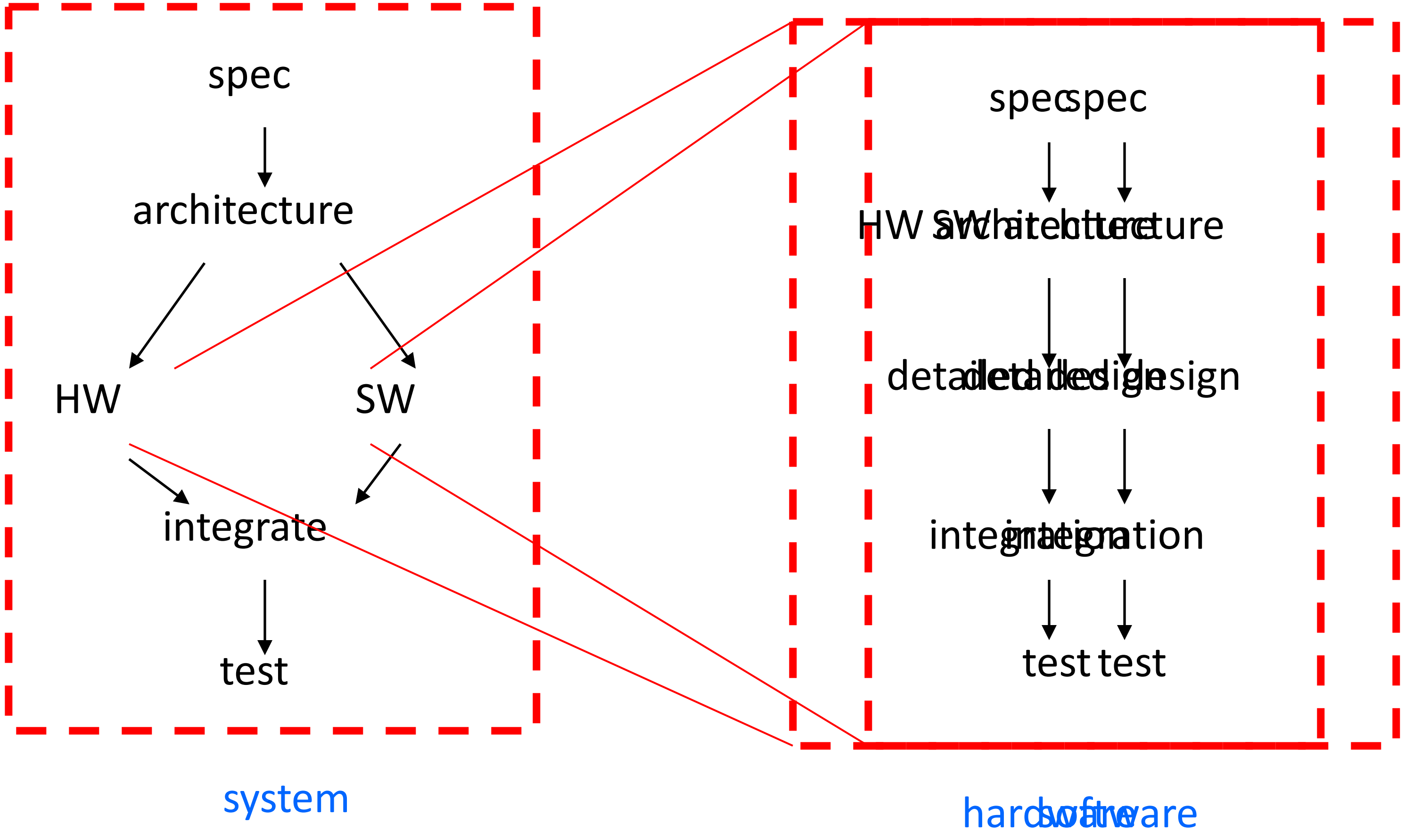
Co-design methodology

- Must architect hardware and software together:
  - provide sufficient resources;
  - avoid software bottlenecks.
- Can build pieces somewhat independently, but integration is major step.
- Also requires bottom-up feedback.

Hierarchical design flow

- Embedded systems must be designed across multiple levels of abstraction:
  - system architecture;
  - hardware and software systems;
  - hardware and software components.
- Often need design flows within design flows.

Hierarchical HW/SW flow



Concurrent engineering

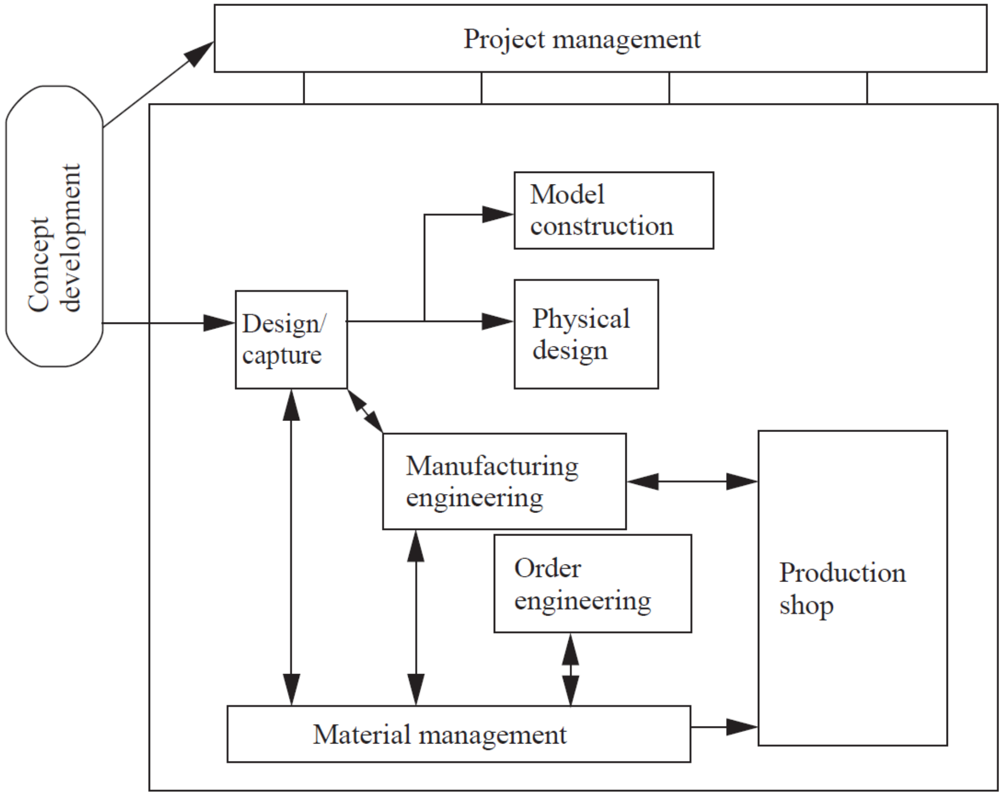
- Large projects use many people from multiple disciplines.
- Work on several tasks at once to reduce design time.
- Feedback between tasks helps improve quality, reduce number of later design problems.

Concurrent engineering techniques

- Cross-functional teams.
- Concurrent product realization.
- Incremental information sharing.
- Integrated product management.
- Supplier involvement.
- Customer focus.

AT&T PBX concurrent engineering

- Benchmark against competitors.
- Identify breakthrough improvements.
- Characterize current process.
- Create new process.
- Verify new process.
- Implement.
- Measure and improve.



Requirements analysis

- Requirements: a description of a desired system characteristic.
- Specification: complete set of system requirements.
- Requirements phase links customers with designers.

Types of requirements

- Functional: input/output relationships.
- Non-functional:
  - timing;
  - power consumption;
  - manufacturing cost;
  - physical size;
  - time-to-market;
  - reliability.

Eliciting requirements

- Customer interviews.
- Comparison with competitors.
- Sales feedback.
- Mock-ups, prototypes.
- Next-bench syndrome (HP): design a product for someone like you.

CRC cards

- Well-known method for analyzing a system and developing an architecture.
- CRC:
  - classes;
  - responsibilities of each class;
  - collaborators are other classes that work with a class.
- Team-oriented methodology.

CRC card format

Class name:
Superclasses:
Subclasses:
Responsibilities: Collaborators:

front

Class name:
Class's function:
Attributes:

back

CRC methodology

- Develop an initial list of classes.
  - Simple description is OK.
  - Team members should discuss their choices.
- Write initial responsibilities/collaborators.
  - Helps to define the classes.
- Create some usage scenarios.
  - Major uses of system and classes.

CRC methodology, cont'd.

- Walk through scenarios.
  - See what works and doesn't work.
- Refine the classes, responsibilities, and collaborators.
- Add class relatoinships:
  - superclass, subclass.

CRC cards for elevator

- Real-world classes:
  - elevator car, passenger, floor control, car control, car sensor.
- Architectural classes: car state, floor control reader, car control reader, car control sender, scheduler.

Elevator responsibilities and collaborators

class	responsibilities	collaborators
Elevator car*	Move up and down	Car control, car sensor, car control sender
Car control*	Transmits car requests	Passenger, floor control reader
Car state	Reads current position of car	Scheduler, car sensor

Good specification

- Correct.
- Unambiguous.
- Complete.
- Verifiable: is each requirement satisfied in the final system?
- Consistent: requirements do not contradict each other.

Good specification, cont’d.

- Modifiable: can update requirements easily.
- Traceable:
  - know why each requirement exists;
  - go from source documents to requirement;
  - go from requirement to implementation;
  - back from implementation to requirement.

Validating the specification

- Validate spec to ensure correctness, completeness, consistency, etIc.
- Specification bugs caught late in the design process are expensive to fix.
- Validation techniques:
  - Prototyping and user testing.
  - Use cases.
  - Formal methods.

System modeling

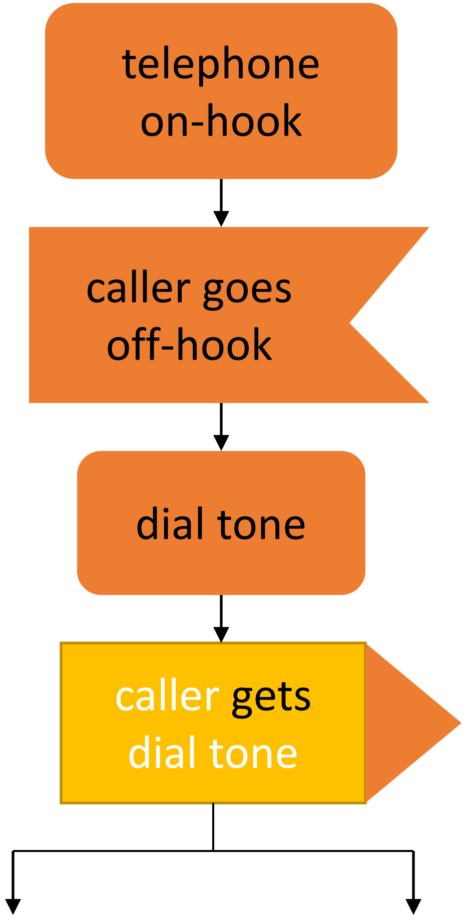
- Capture functional and non-functional properties:
  - verify correctness of spec;
  - compare spec to implementation.
- Many specification styles:
  - control-oriented vs. data-oriented;
  - textual vs. graphical.
- UML is one modeling/design language.

Model-based design

- Targets cyber-physical system = computing system + physical plant.
  - Unified view of physical plant and embedded computing system (cyber plant).
- Methodology:
  - Capture system in domain-specific modeling language (DSML).
  - Validate using simulation, formal methods.
  - Compile into implementation components:
    - Software.
    - Computing platform.
    - Physical plant components.

SDL

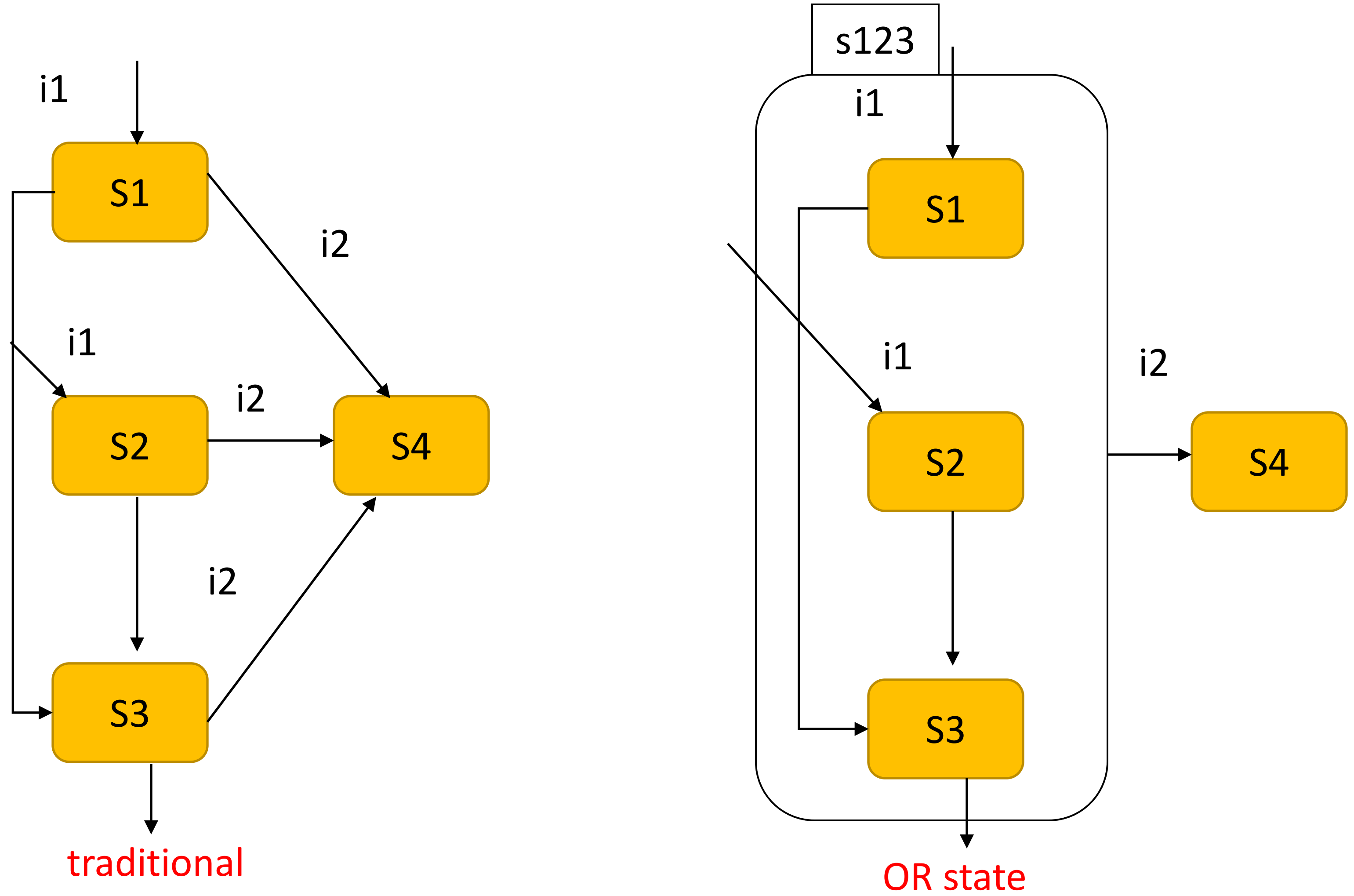
- Used in telecommunications protocol design.
- Event-oriented state machine model.



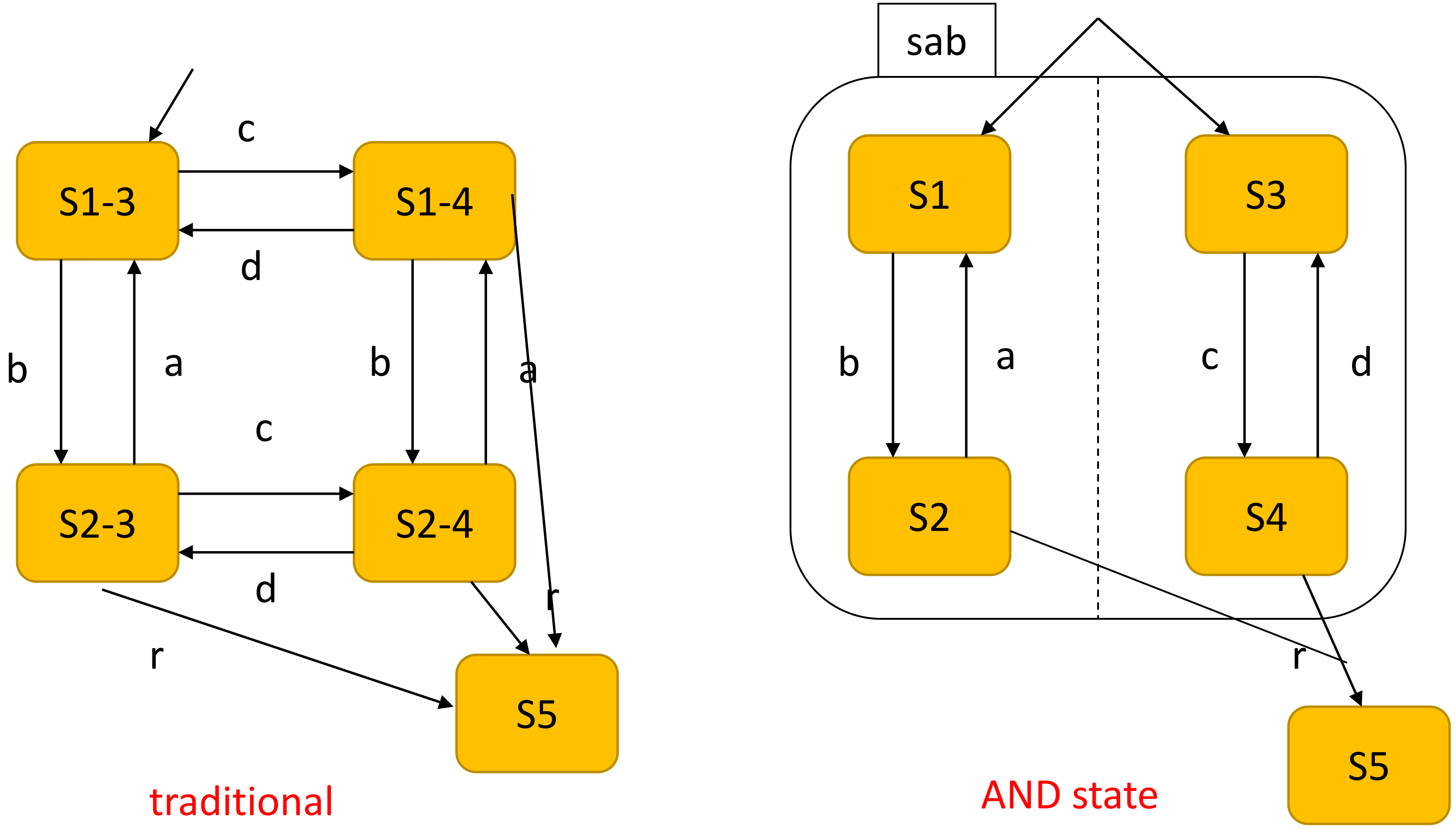
Statecharts

- Ancestor of UML state diagrams.
- Provided composite states:
  - OR states;
  - AND states.
- Composite states reduce the size of the state transition graph.

Statechart OR state



Statechart AND state



AND-OR tables

- Alternate way of specifying complex conditions:

  cond1 or (cond2 and !cond3)

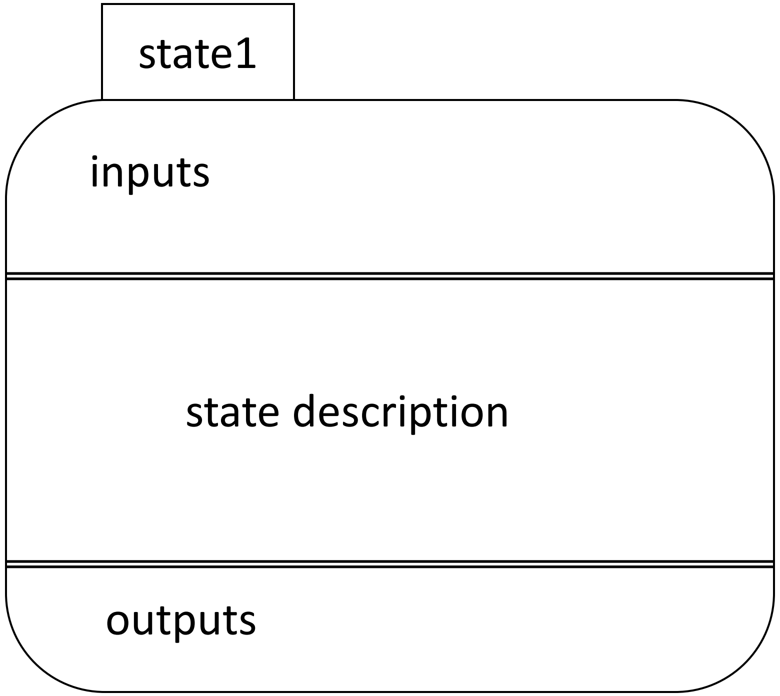
	OR	
cond1	T	-
cond2	-	T
cond3	-	F
AND		

TCAS II specification

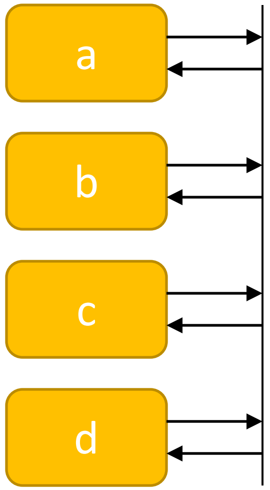
- TCAS II: aircraft collision avoidance system.
- Monitors aircraft and air traffic info.
- Provides audio warnings and directives to avoid collisions.
- Leveson et al used RMSL language to capture the TCAS specification.

RMSL

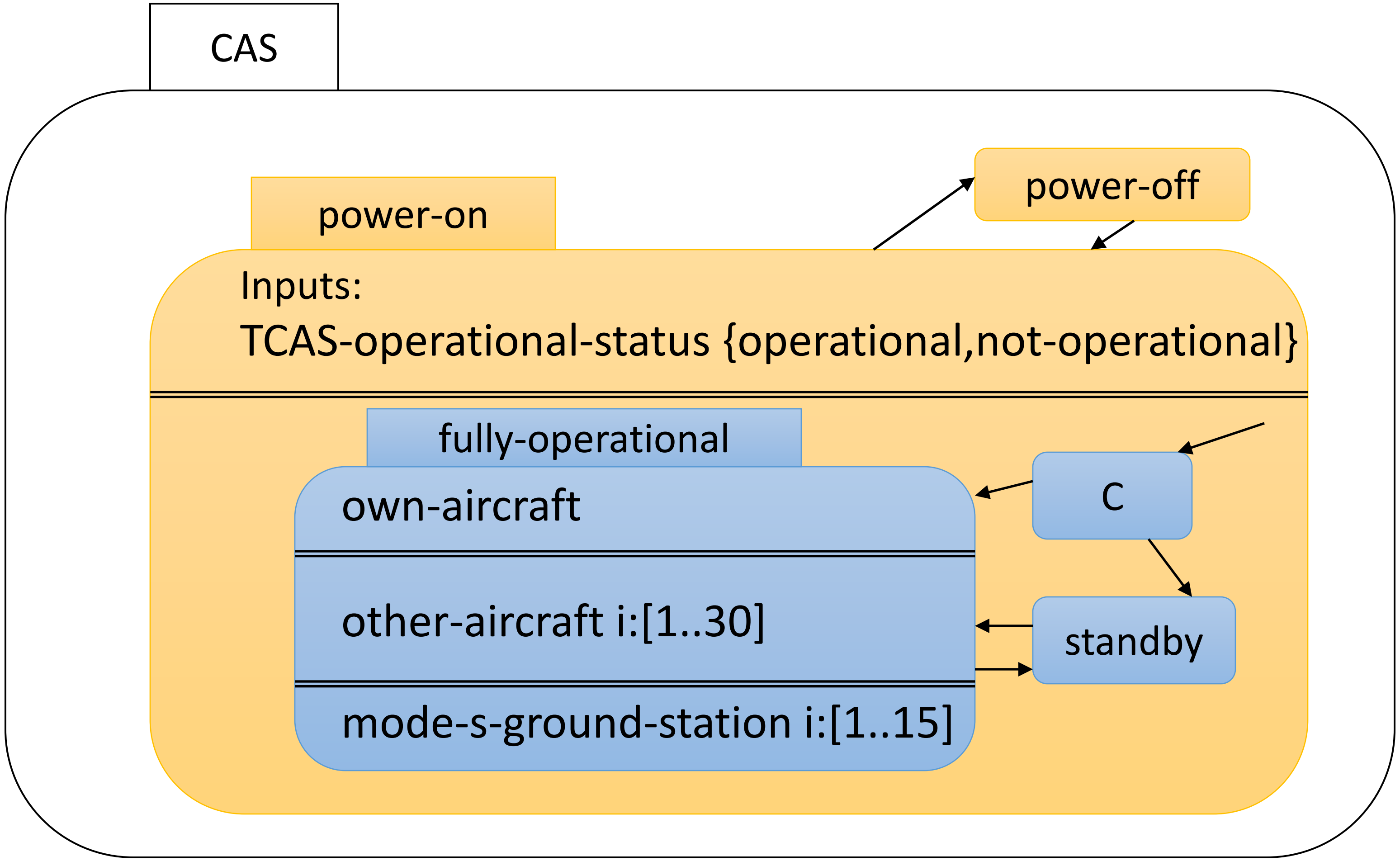
- State description:



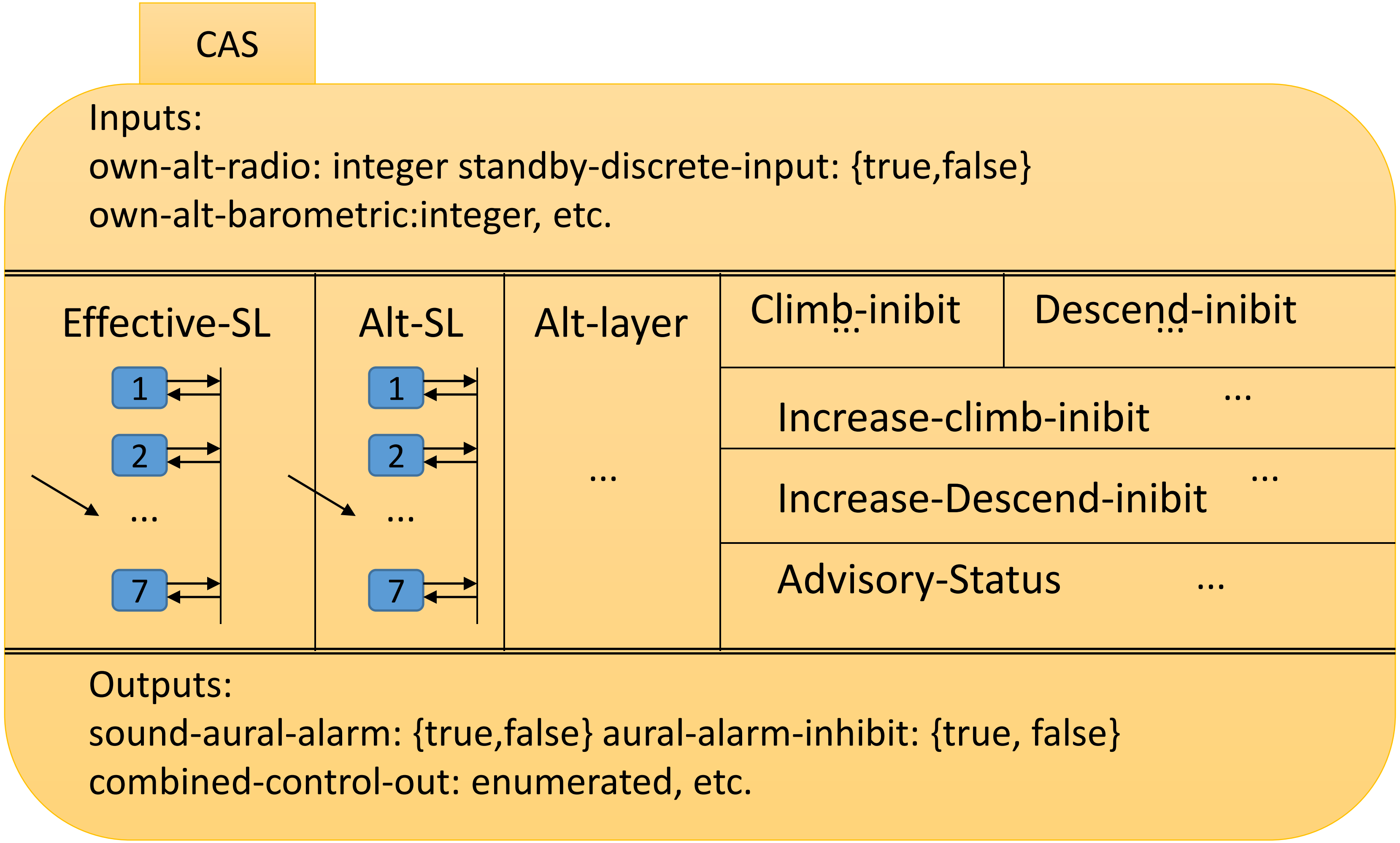
- Transition bus for transitions between many states:



TCAS top-level description



Own-Aircraft AND state



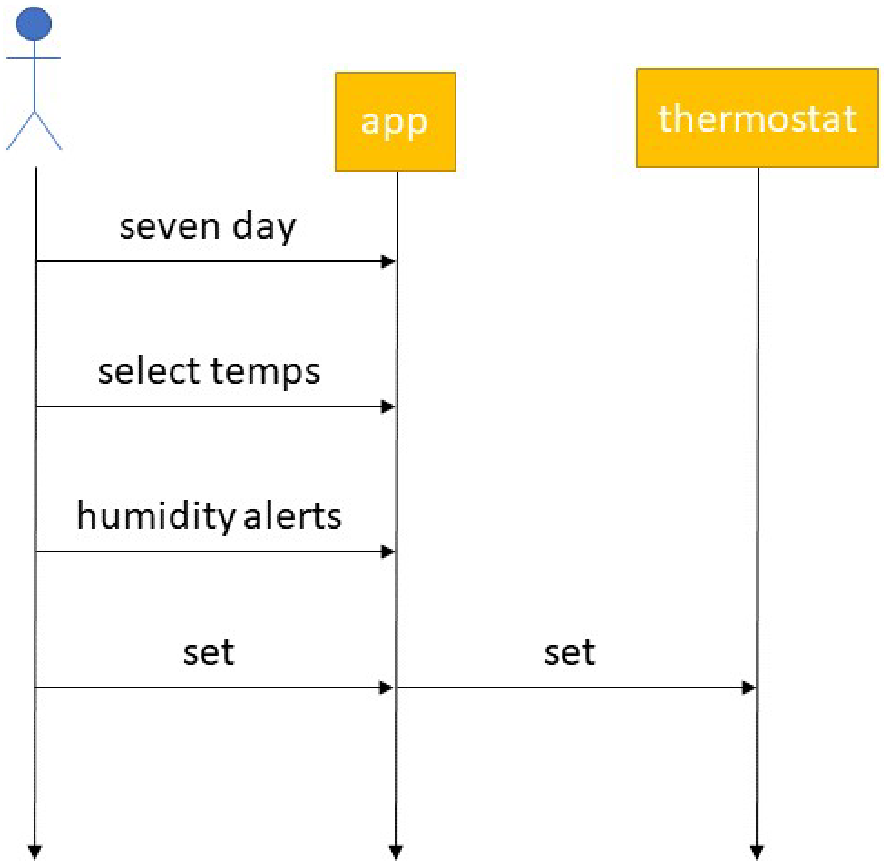
SysML

- Systems Modeling Language is based on UML.
- Broad-spectrum systems engineering diagram.

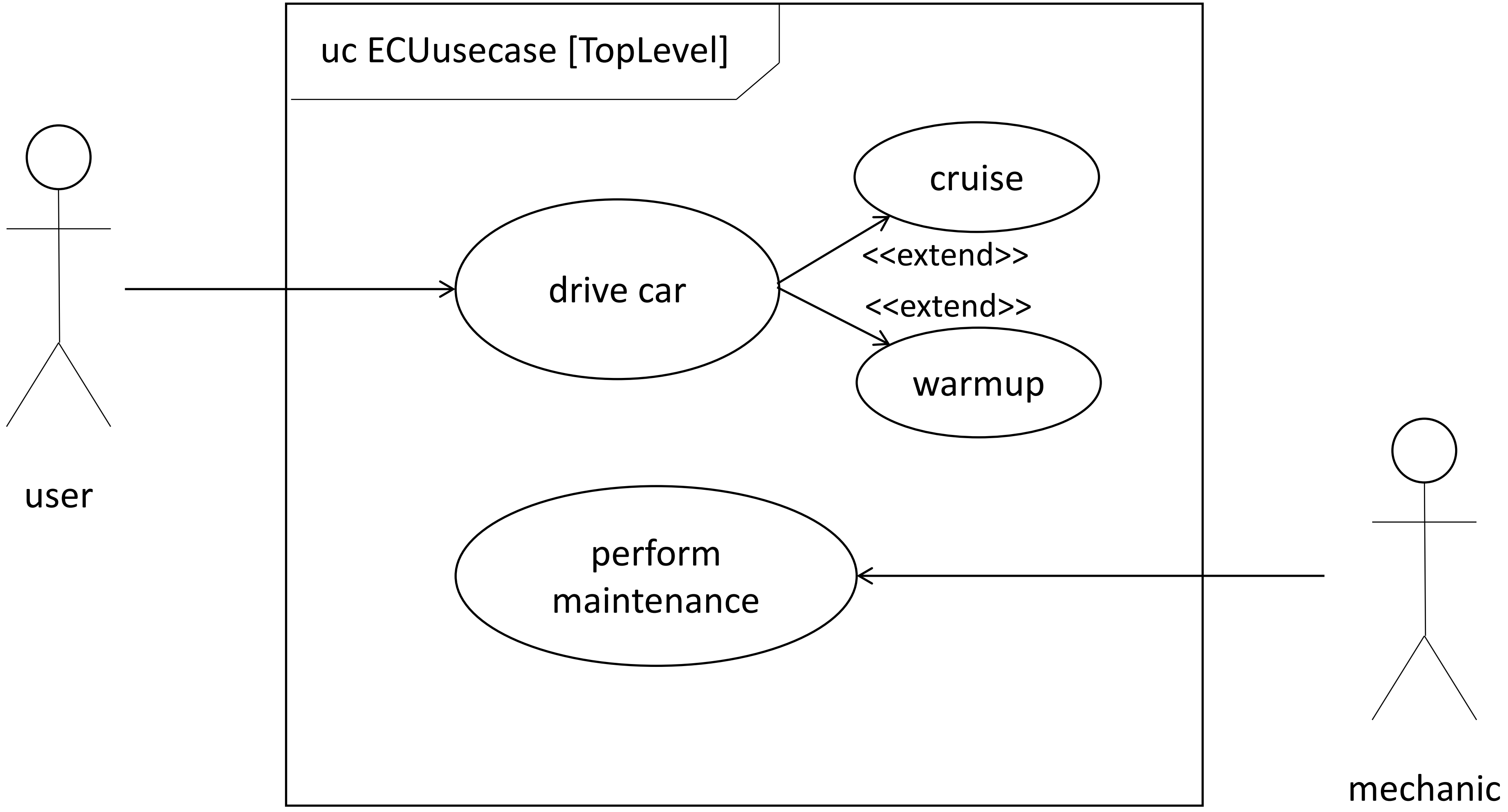
- Nine types of diagrams:
  - Sequence (sd).
  - State machine (stm).
  - Package (pkg).
  - Use case (uc).
  - Block definition (bdd).
  - Internal block (ibd).
  - Activity (act).
  - Requirement (req).
  - Parametric (par).

Use case

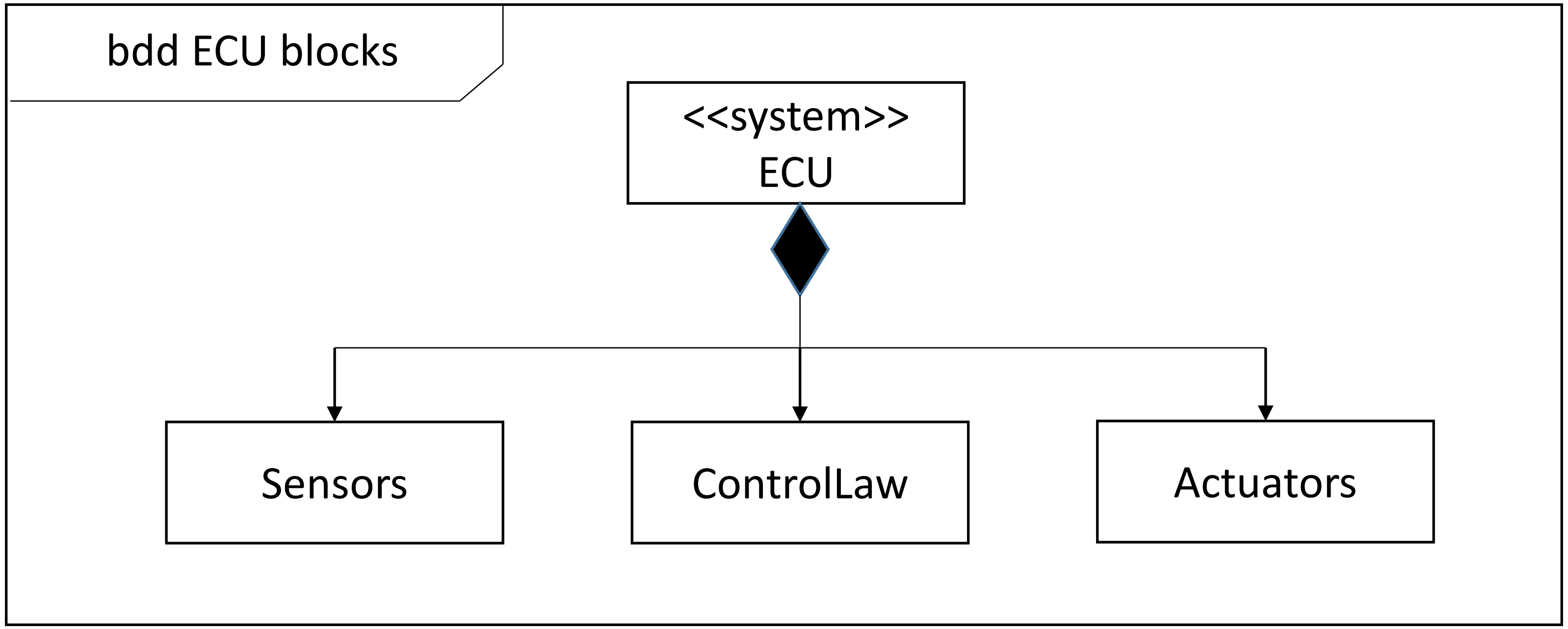
- Defines example of user interaction with system.
- Collection of use cases helps to define the system.



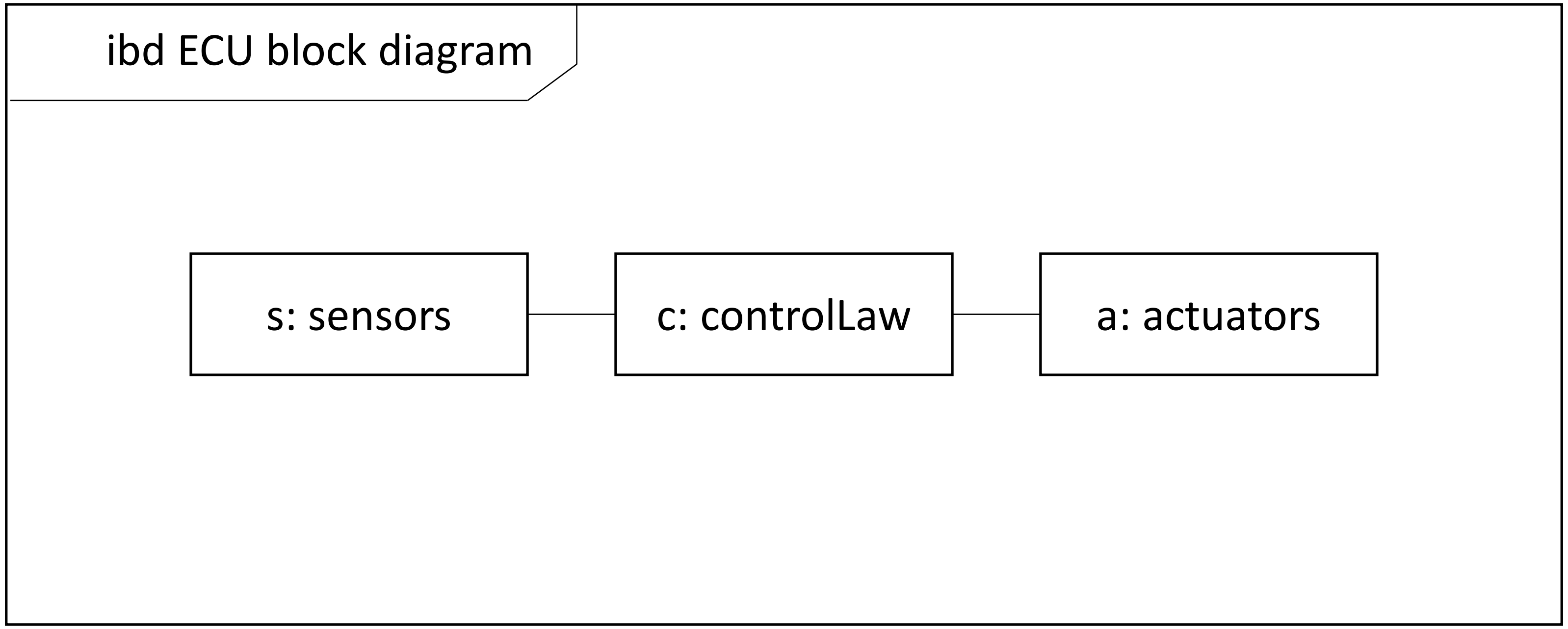
SysML use case diagram



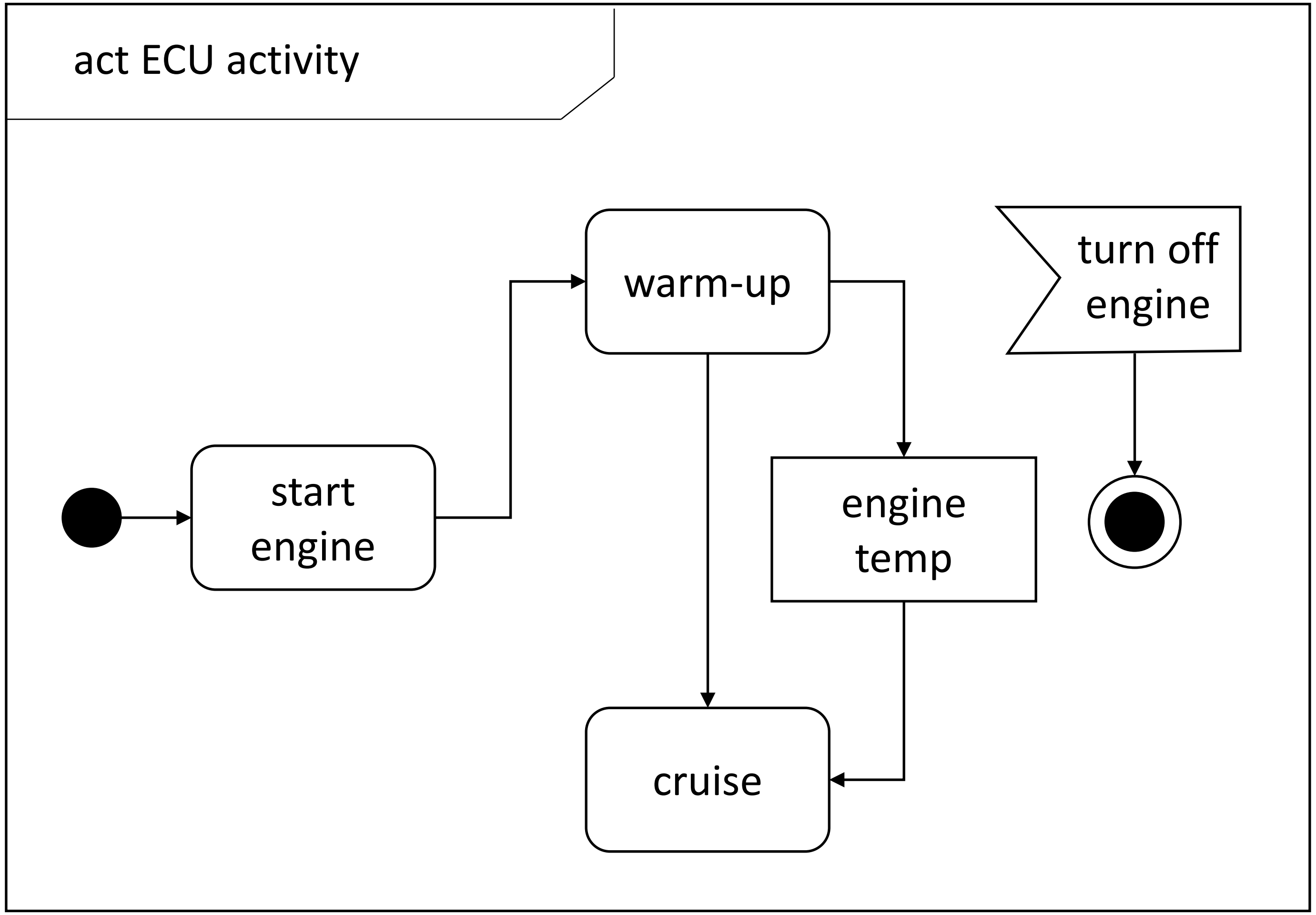
SysML block definition diagram



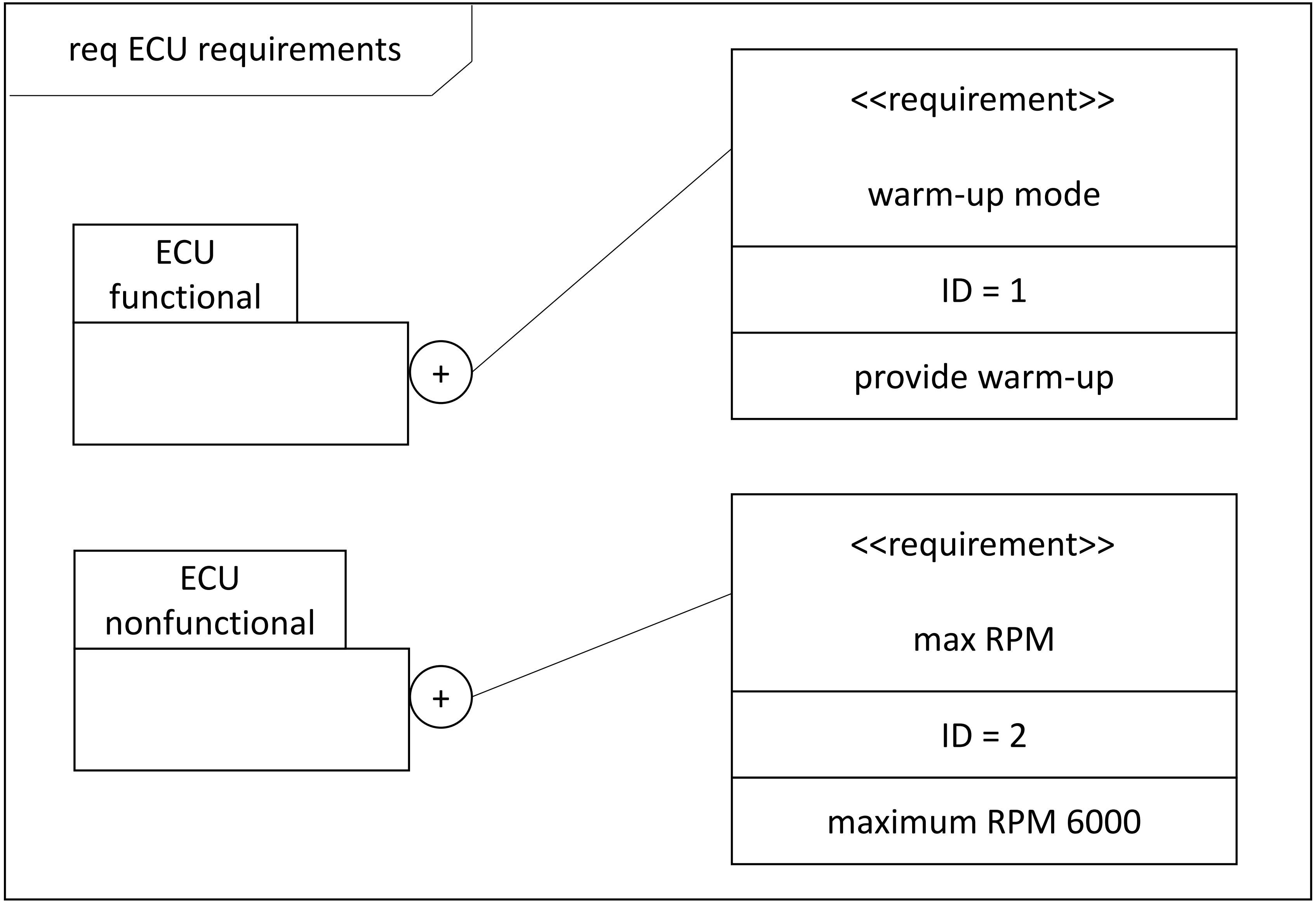
SysML internal block diagram



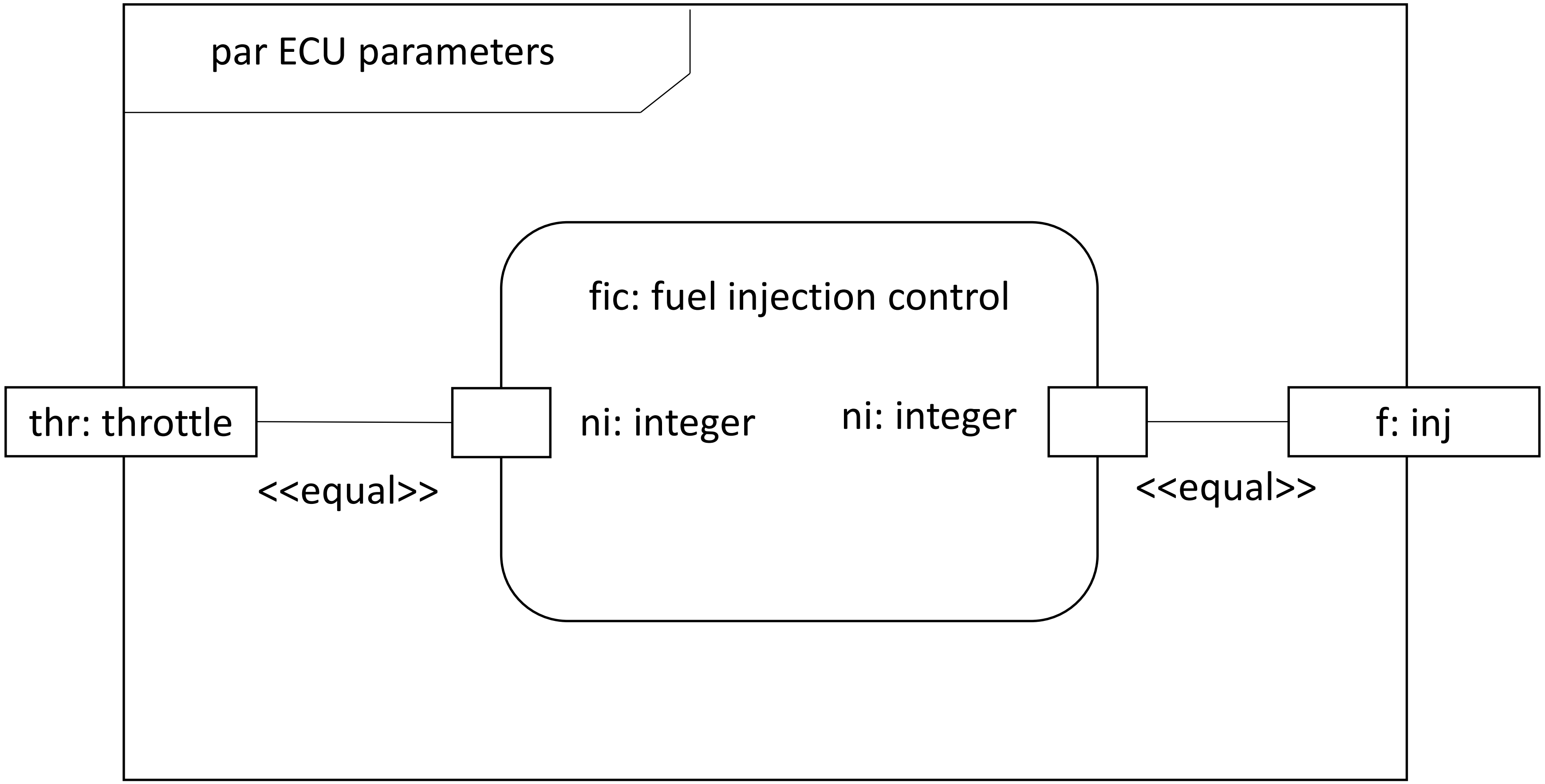
SysML activity diagram



SysML requirement diagram



SysML parametric diagram



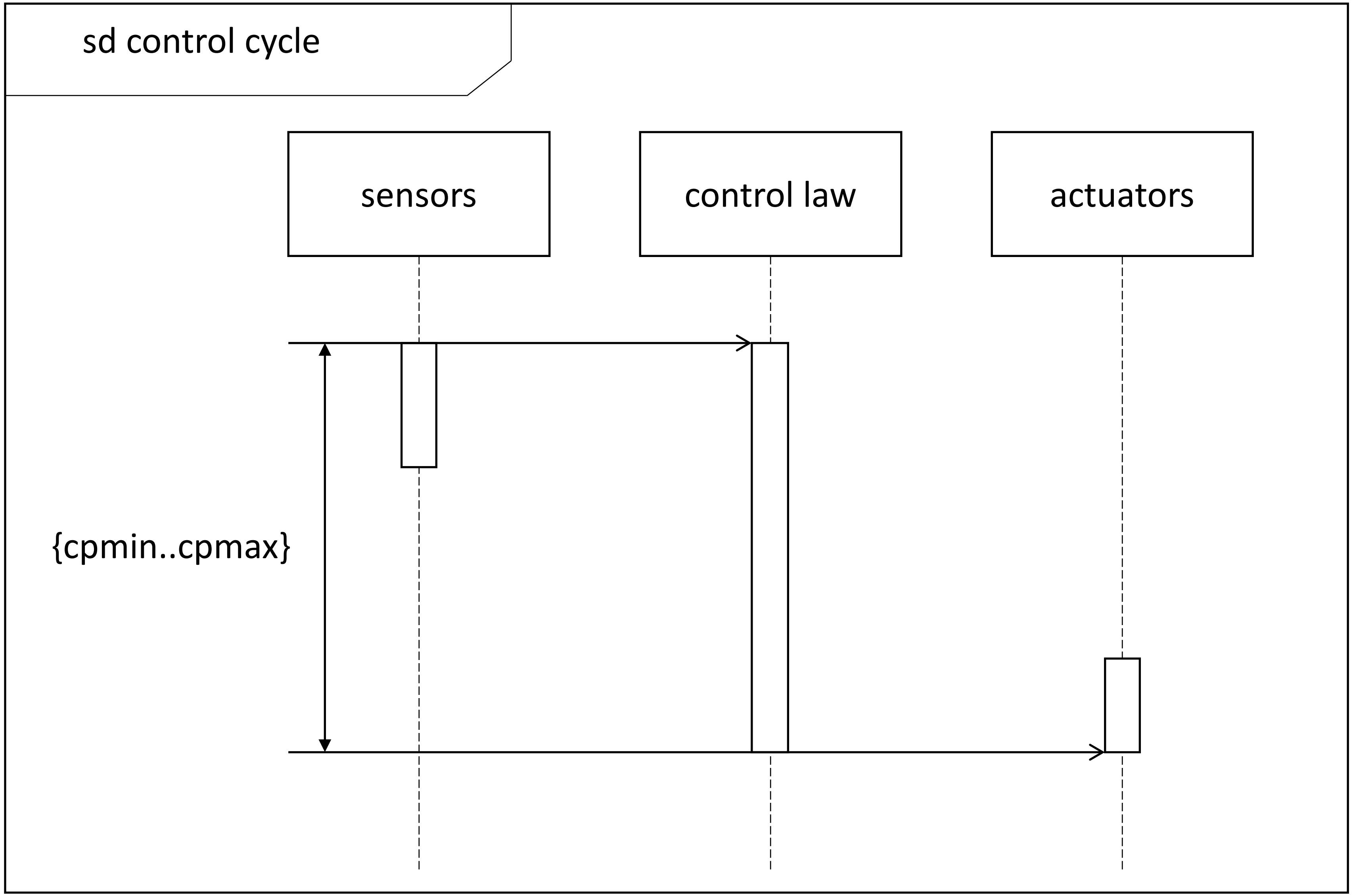
MARTE

- Targets model-based design of real-time embedded computing systems.
- Non-Functional Properties Modeling (NFP) module captures non-functional properties.
- Time Modeling (Time) model captures chronometric and sequential time characteristics.

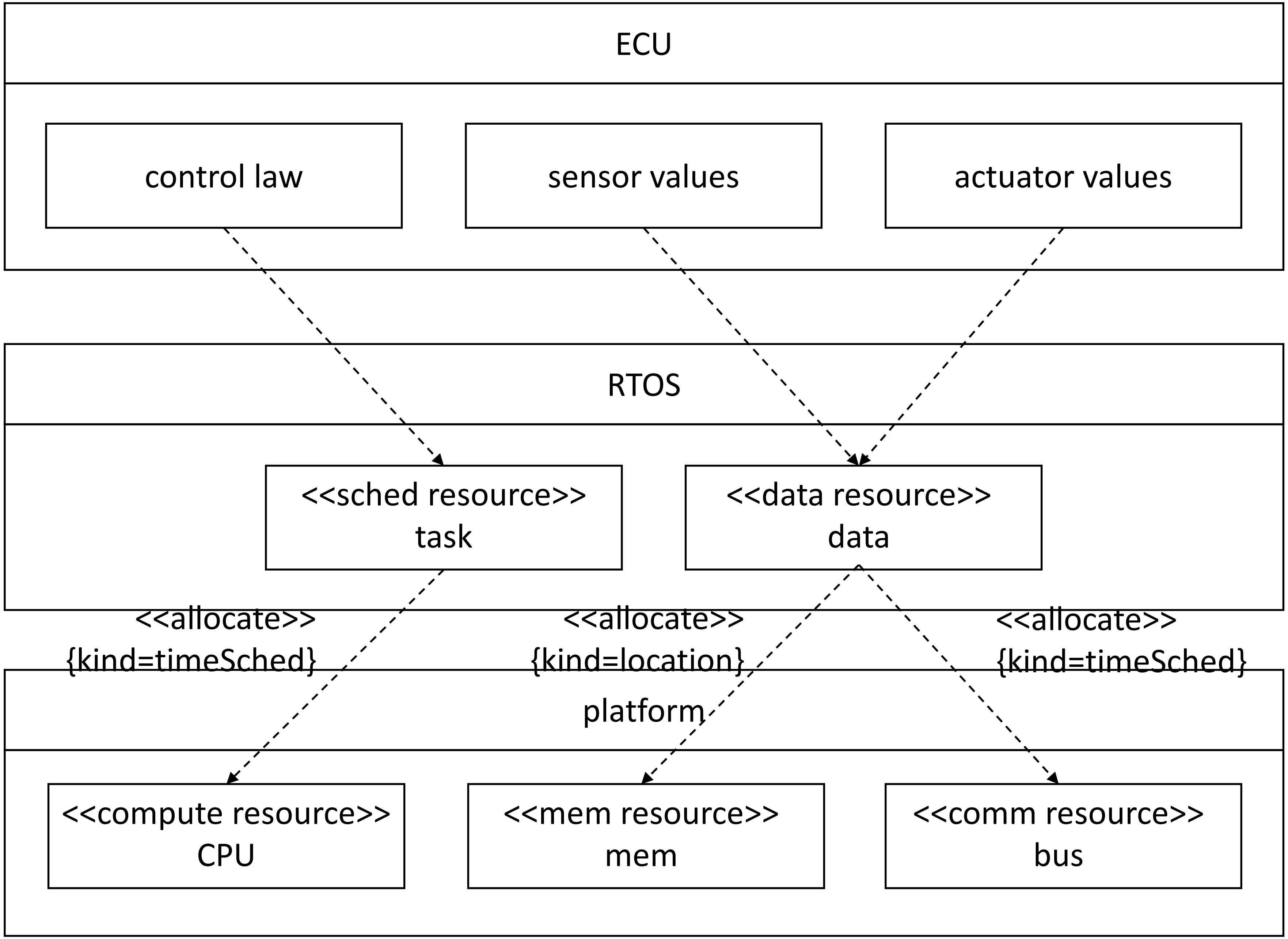
MARTE, cont'd.

- High-Level Application Modeling (HLAM) profile describes quantitative and qualitative features.
- Detailed Resource Modeling (DRM) profile provides for software, hardware resource modeling (SRM, HRM).
- Generic Quantitative Analysis Modeling (GQAM) describes schedulability, performance analysis.
- Performance Analysis Modeling (PAM) profile describes best-effort, soft real-time systems.

Sequence diagram with MARTE timing constraint



Object diagram with MARTE allocation annotations



System design techniques

- System analysis and architecture design.
  - Design patterns.
  - Transaction-level modeling.
- Dependability, safety, security:
  - Quality assurance.
  - Design reviews.
  - Safety-oriented methodologies.
  - Security.

Design pattern

- A solution to a recurring engineering problem.
  - Best practice.
- Design pattern may consider:
  - Computing platform.
  - Operation allocation.
  - Schedules for computation, data movement.
- Example---networked control system:
  - Computing platform architecture.
  - Allocation of data on bus.
  - Bus scheduling pattern.

Transaction-level modeling.

- Between architecture and register-transfer levels.
- Transaction – communication between concurrent entities.
  - May be between software threads, hardware units.
- Several types of transaction models:
  - Untimed models provide partial ordering but no hard timing.
  - Bus-accurate models provide estimates of bus timing.
  - Cycle-accurate models are fully synchronous to clock cycle.
- SystemC supports transaction-level modeling.

Dependability, safety, and security

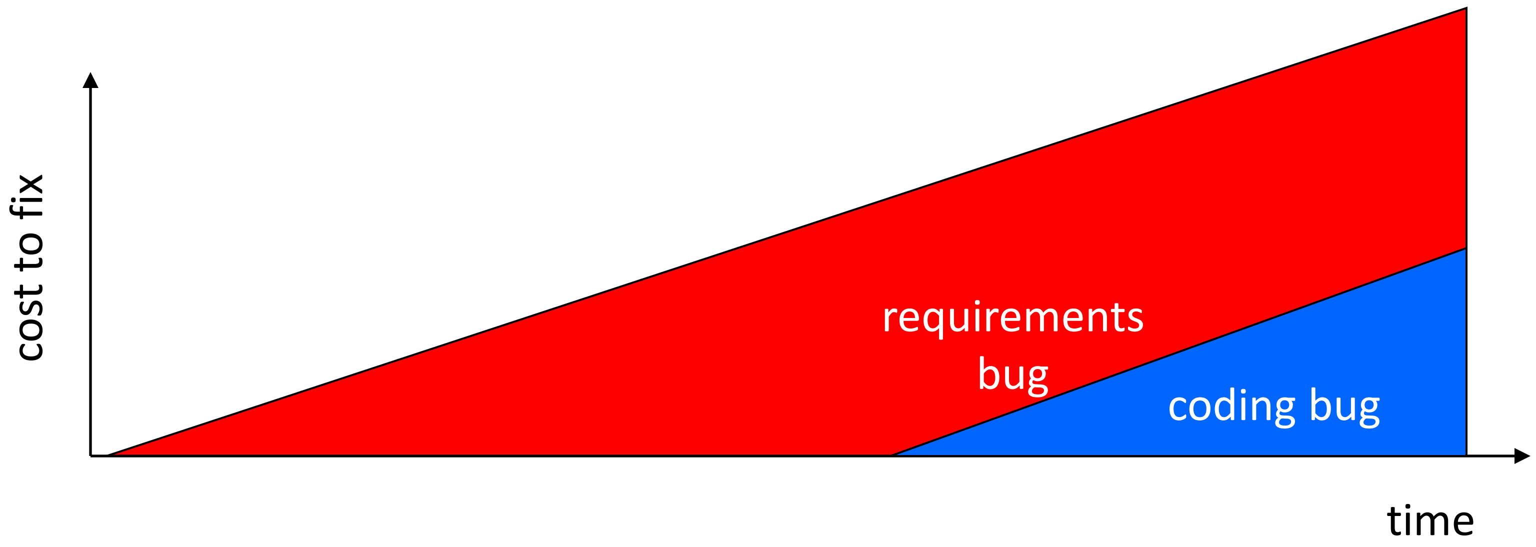
- Dependability: length of time for which a system can operate without defects.
- Safety and security can be quantified using dependability concepts.

Quality assurance

- Quality judged by how well product satisfies its intended function.
  - May be measured in different ways for different kinds of products.
- Quality assurance (QA) makes sure that all stages of the design process help to deliver a quality product.

Verification

- Verification and testing are important throughout the design flow.
- Early bugs are more expensive to fix:



Verifying requirements and specification

- Requirements:
  - prototypes;
  - prototyping languages;
  - pre-existing systems.
- Specifications:
  - usage scenarios;
  - formal techniques.

ISO 9000

- Developed by International Standards organization.
- Applies to a broad range industries.
- Concentrates on process.
- Validation based on extensive documentation of organization’s process.

CMU Capability Maturity Model

- Five levels of organizational maturity:
  - Initial: poorly organized process, depends on individuals.
  - Repeatable: basic tracking mechanisms.
  - Defined: processes documented and standardized.
  - Managed: makes detailed measurements.
  - Optimizing: measurements used for improvement.

Design review

- Uses meetings to catch design flaws.
  - Simple, low-cost.
  - Proven by experiments to be effective.
- Use other people in the project/company to help spot design problems.

Design review players

- Designers: present design to rest of team, make changes.
- Review leader: coordinates process.
- Review scribe: takes notes of meetings.
- Review audience: looks for bugs.

Before the design review

- Design team prepares documents used to describe the design.
- Leader recruits audience, coordinates meetings, distributes handouts, etc.
- Audience members familiarize themselves with the documents before they go to the meeting.

Design review meeting

- Leader keeps meeting moving; scribe takes notes.
- Designers present the design:
  - use handouts;
  - explain what is going on;
  - go through details.

Design review audience

- Look for any problems:
  - Is the design consistent with the specification?
  - Is the interface correct?
  - How well is the component's internal architecture designed?
  - Did they use good design/coding practices?
  - Is the testing strategy adequate?

Follow-up

- Designers make suggested changes.
  - Document changes.
- Leader checks on results of changes, may distribute to audience for further review or additional reviews.

Therac-25 Medical Imager (Leveson and Turner)

- Six known accidents: radiation overdoses leading to death and serious injury.
- Radiation gun controlled by PDP-11.
- Four major software components:
  - stored data;
  - scheduler;
  - set of tasks;
  - interrupt services.

Therac-25 tasks

- Treatment monitor controlled and monitored setup and delivery of treatment in eight phases.
- Servo task controlled radiation gun.
- Housekeeper task took care of status interlocks and limit checks.

Therac-25 treatment monitor task

- Treat was main monitor task.
  - Eight subroutines.
  - Treat rescheduled itself after every subroutine.

Therac-25 software timing race

- Timing-dependent use of mode and energy:
  - if keyboard handler sets completion behavior before operator changes mode/energy data, Datent task will not detect the change, but Hand task will.

Therac-25 software timing errors

- Changes to parameters made by operator may show on screen but not be sensed by Datent task.
- One accident caused by entering mode/energy, changing mode/energy, returning to command line in 8 seconds.
- Skilled operators typed faster, more likely to exercise bug.

Leveson and Turner observations on Therac-25

- Performed limited safety analysis: guessed at error probabilities, *etc*.
- Did not use mechanical backups to check machine operation.
- Used overly complex programs written in unreliable styles.

Security and safety-critical systems

- Security problems threaten safety.
- The air gap myth---today's systems are vulnerable to software attacks from many sources.
- Attack surface: set of program locations and use cases in which system can be attacked.

Stuxnet

- Series of attacks on Iranian nuclear processing facilities.
- Facilities were not directly connected to Internet---*air gap.*
  - Attack code was carried by maintenance workers using USB devices infected on outside machines.

Safety-oriented methodologies

- Availability: probability of a system's correct operation over time.
  - Often modeled as an exponential distribution.
  - $R(t) = e^{-\lambda t}$, failures per unit time λ.
- Safety analysis phases:
  - Hazard analysis: types of safety-related problems that may occur.
  - Risk assessment: effects of hazards, severity or likelihood of injury, etc.
  - Risk mitigation: design modifications to improve system's ability to handle identified hazards.

Safety-oriented standards

- ISO 26262: Functional safety management for automotive electrics and electronics (EE).
- DO-178C: safety-related procedures for avionics software.
  - Failure software level: A catastrophic, B hazardous, C major, D minor, E no effect.
- Several SAE standards for automotive software.
- MISRA coding standards for C and C++.
- CERT C standard for C coding.