System Design

José Costa

Software for Embedded Systems

Departamento de Engenharia Informática (DEI)
Instituto Superior Técnico

2015-11-02

Outline

- System Design Techniques
- Design methodologies
- Requirements Analysis
- Specification
- System Analysis and Architecture Design

Design Methodology

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
  - e.g., engineers' salaries, computers used
- Manufacturing cost
- Quality
  - correcteness, reliability and usability must be explicitly address from the beginning of the design job

Case-study: 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

TÉCNICO LISBOA

Design flow

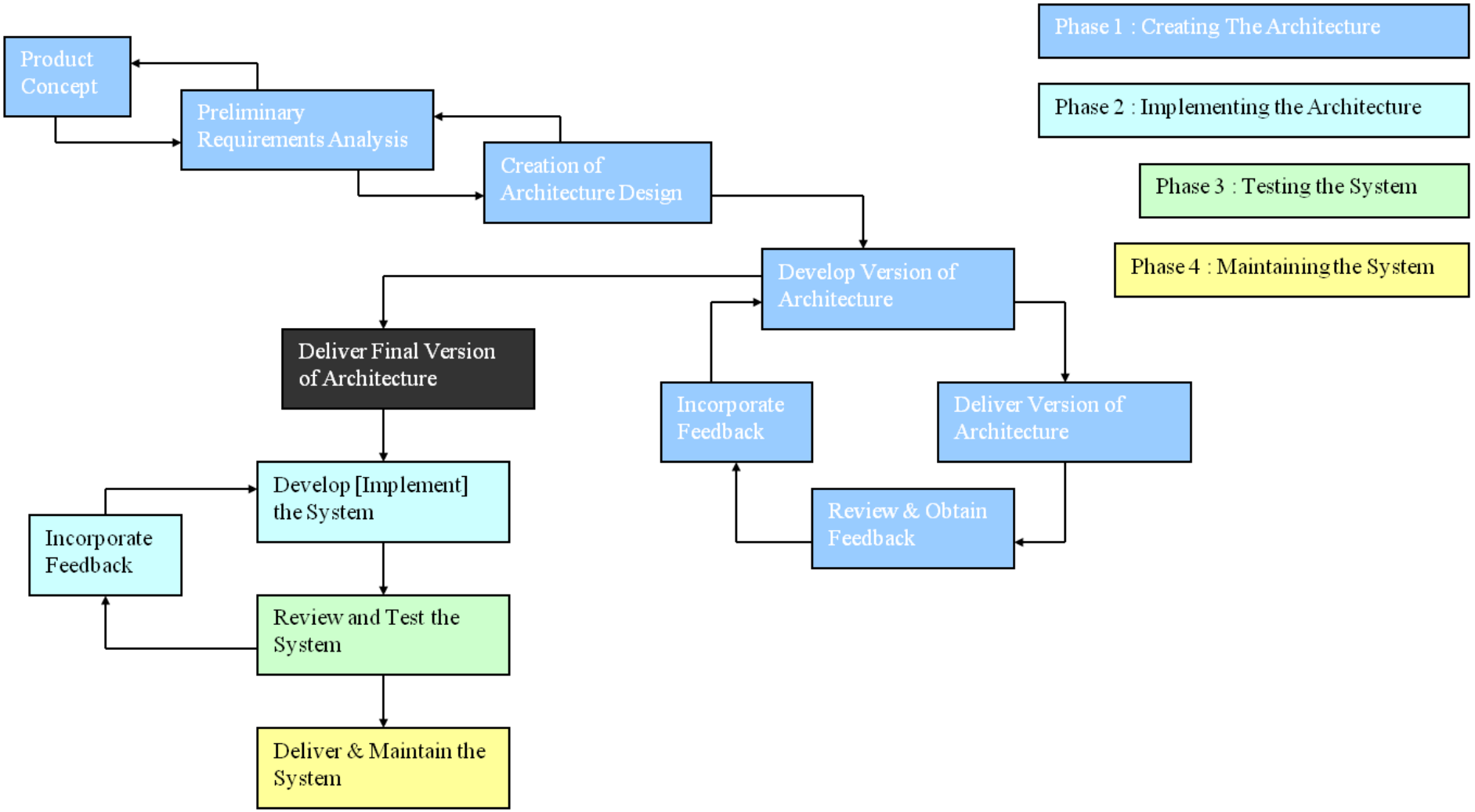
Sequence of steps in a design methodology.

- May be partially or fully automated
  - use tools to transform, verify design
- Design flow is one component of methodology
- Methodology also includes management organization
  - e.g., change management, marketing

Design Flows

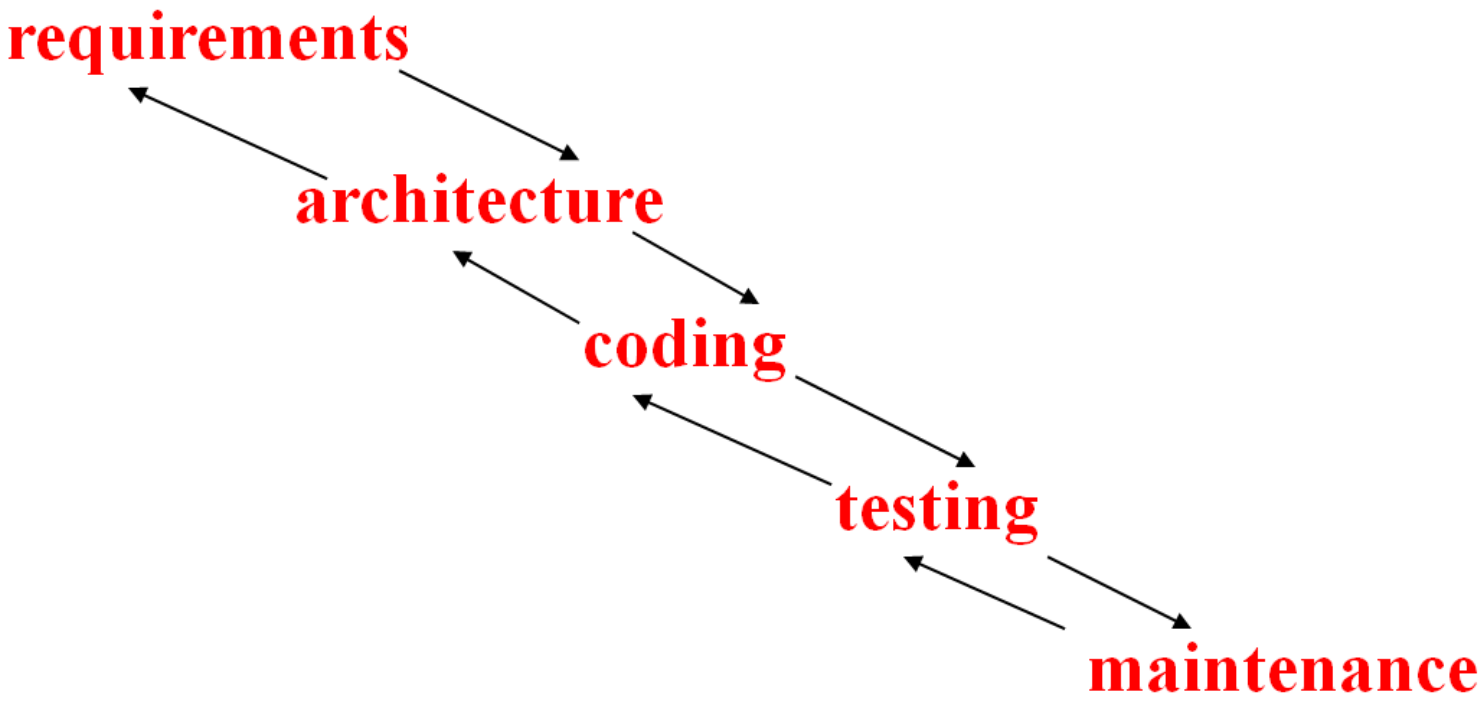
- Embedded Systems Design and Development Lifecycle Model (Noergaard)
- Waterfall Model
- Spiral Model
- Successive Refinement Model
- Concurrent Engineering

Embedded Systems Design and Development Lifecycle Model



Waterfall Model

Early model for software development.



Waterfall Model Steps

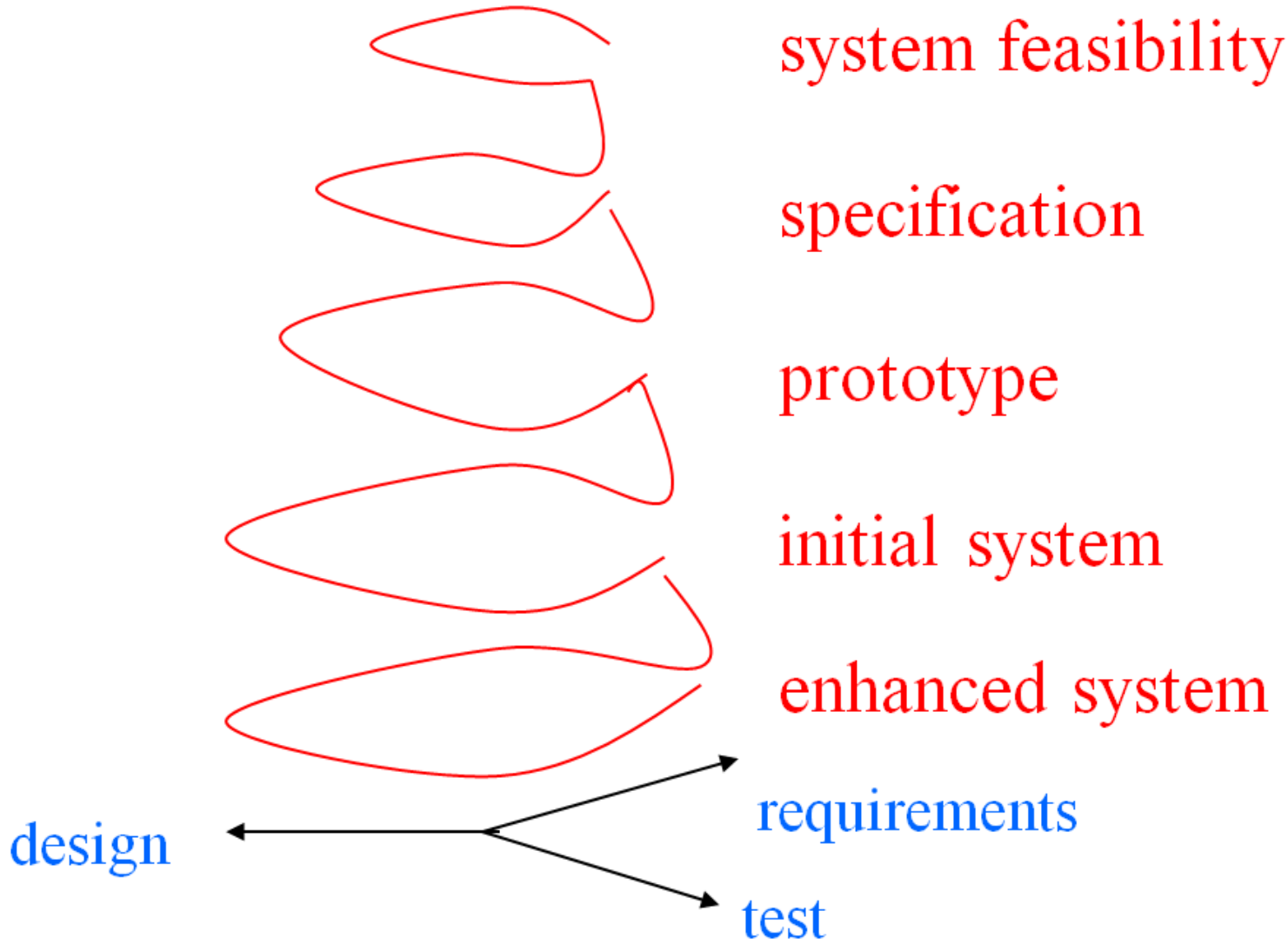
- Requirements
  - determine basic characteristics
- Architecture
  - decompose into basic modules
- Coding
  - implement and integrate
- Testing
  - exercise and uncover bugs
- Maintenance
  - deploy, fix bugs, upgrade

Waterfall Model Critique

- Only local feedback
  - may need iterations between coding and requirements
- Doesn't integrate top-down and bottom-up design
- Assumes hardware is given

Unrealistic design process!

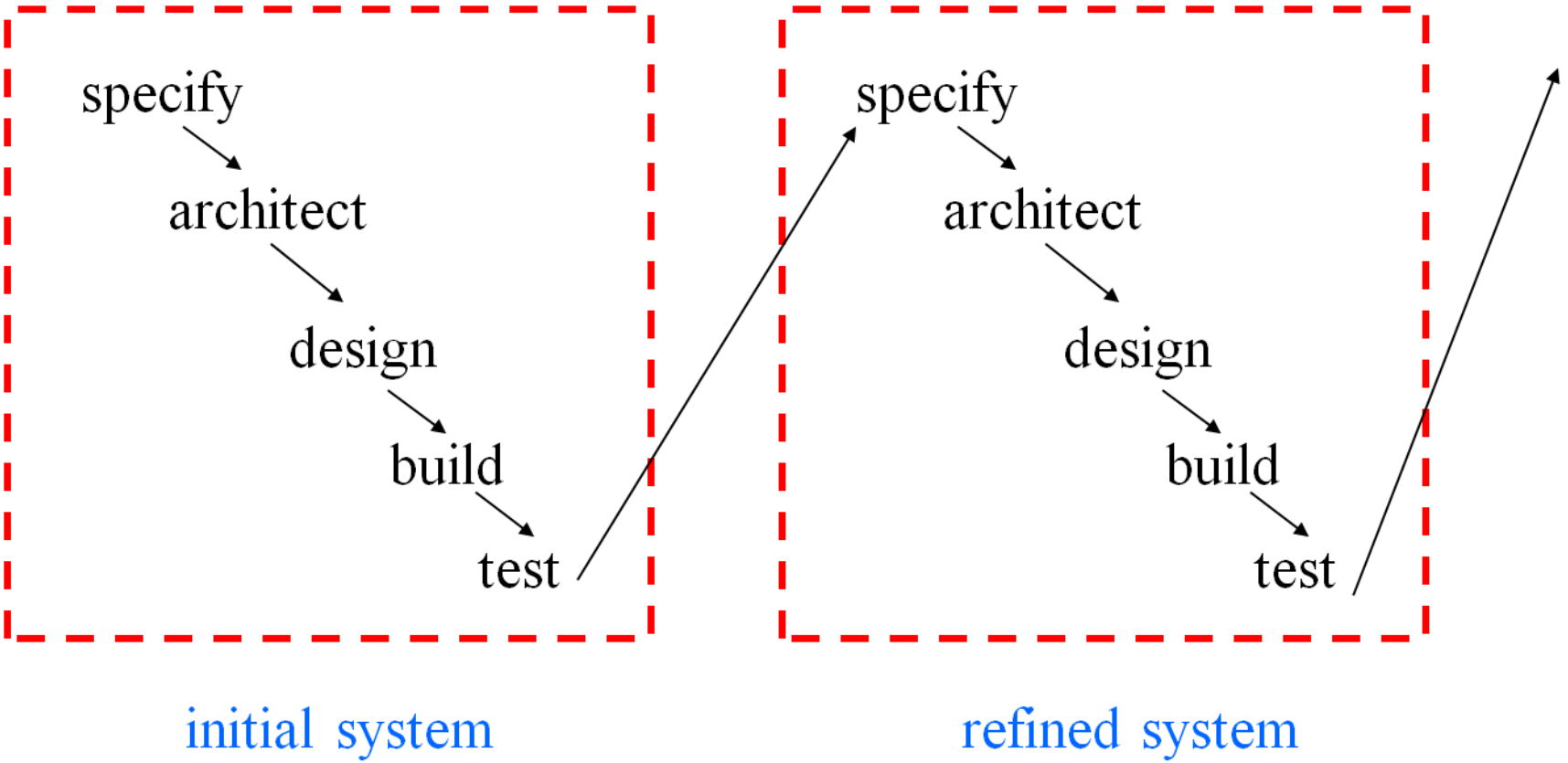
Spiral Model



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

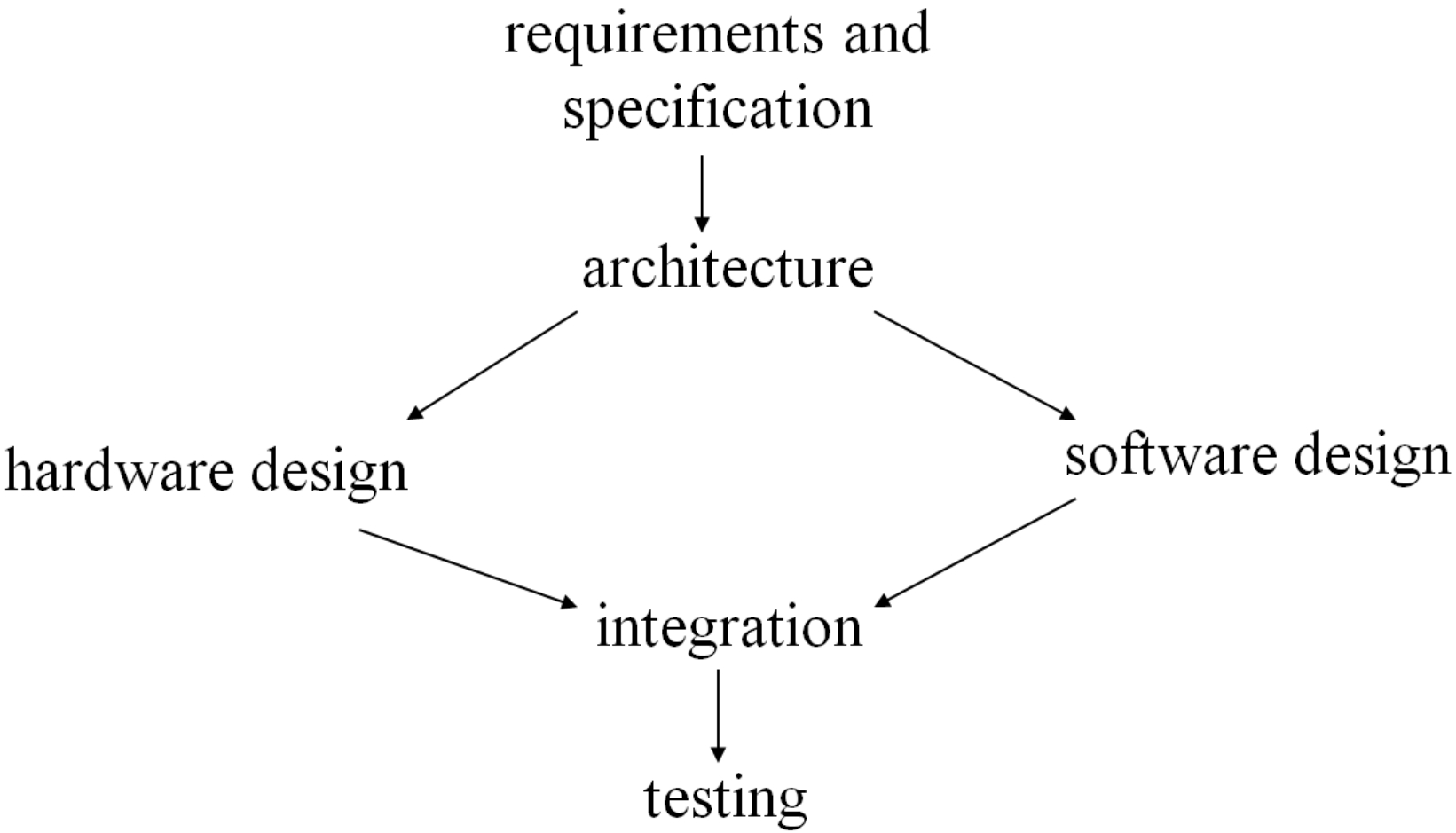
Successive Refinement Model



Successive Refinement Model

- System is built many times
- Makes sense when relatively unfamiliar with application domain
- Various iterations may only be partial

Hardware/Software Design Flow



Co-Design Methodology

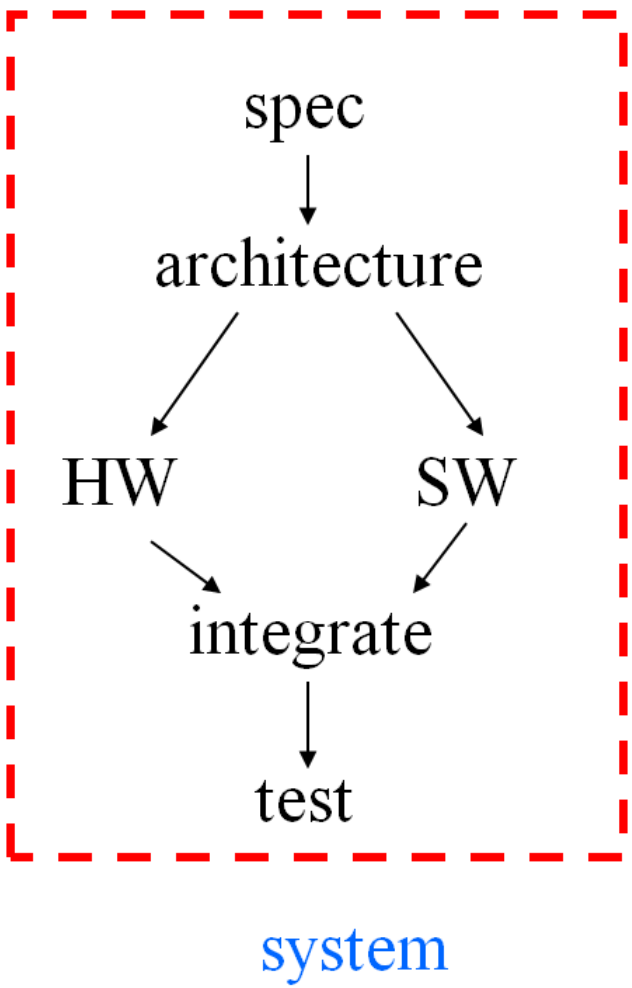
- Must architect hardware and software together
  - provide sufficient resources
  - avoid software bottlenecks
- Can build pieces somewhat independently
  - but integration is major step
- Also requires bottom-up feedback

Hierarchical Design Flow

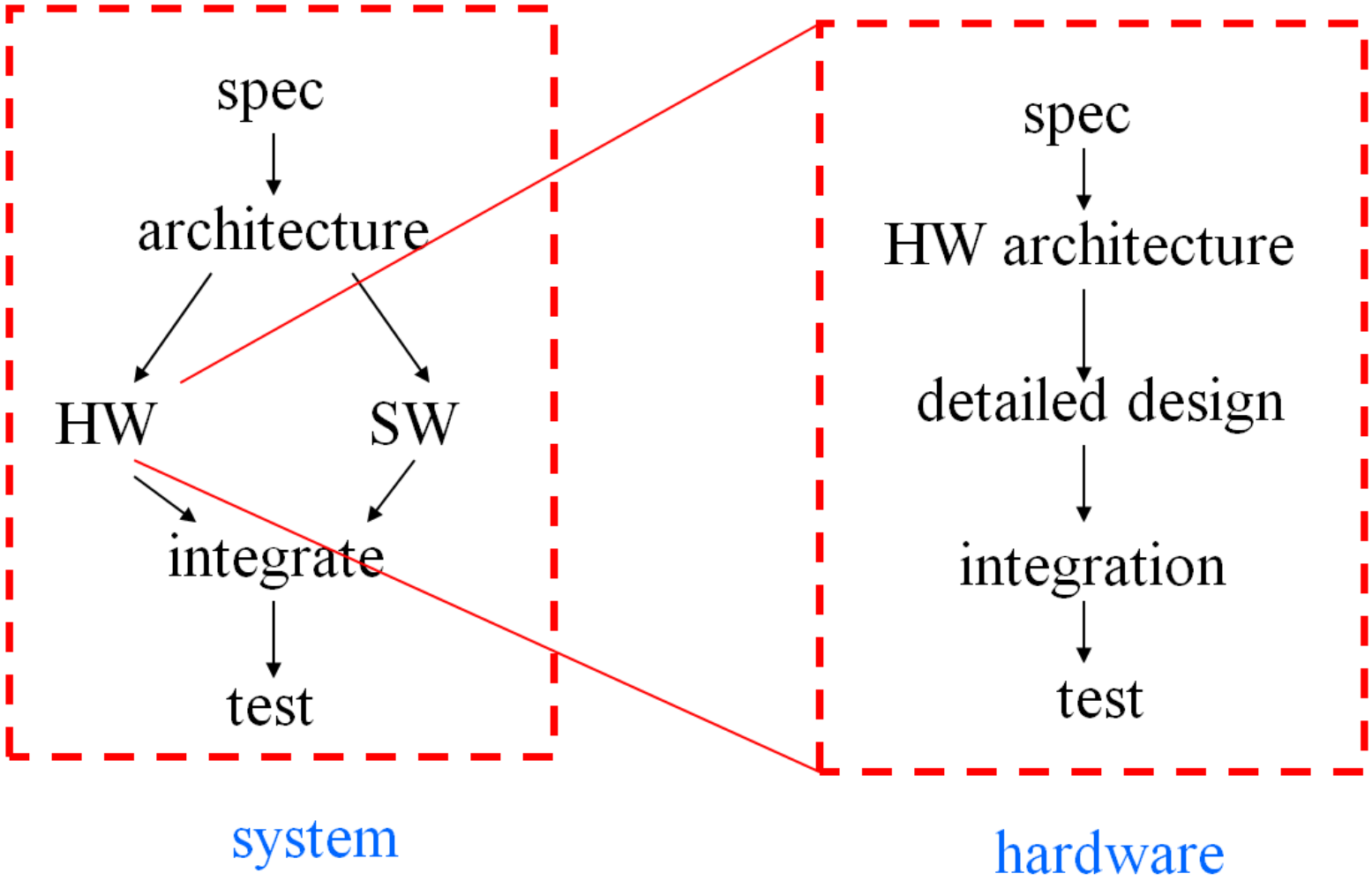
- Embedded systems must be designed across multiple levels of abstraction:
  - system architecture
  - hardware and software systems
  - hardware and software components

- Often need design flows within design flows

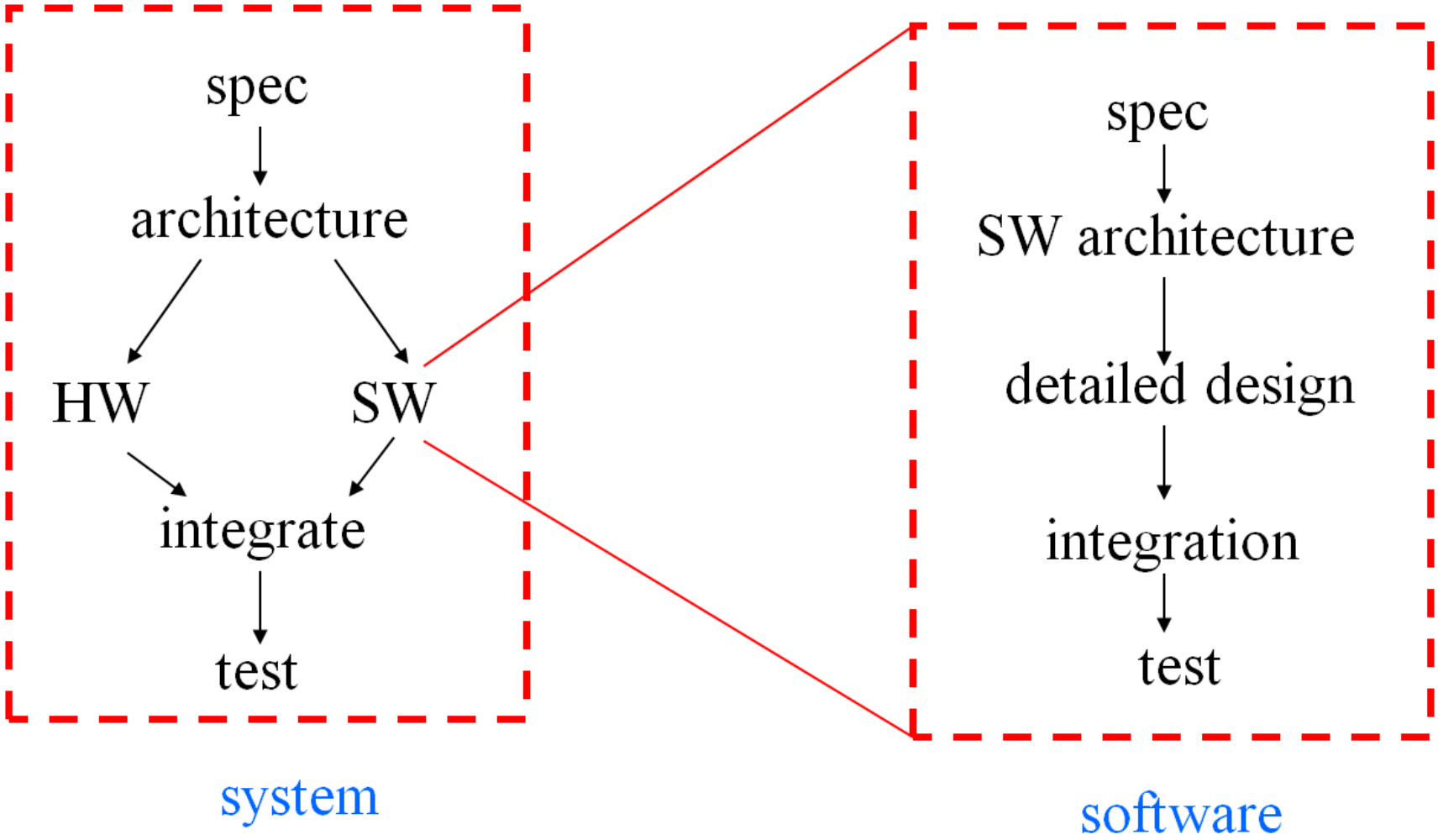
Hierarchical HW/SW Flow



Hierarchical HW/SW Flow



Hierarchical HW/SW Flow



Concurrent Engineering

- Large projects use many people from multiple disciplines
- It's easy to loose track of the complete design flow

Concurrent Engineering

- 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
  - e.g., manufacturing, hardware and software design, marketing
- Concurrent product realization
  - e.g., designing various subsystems simultaneously
- Incremental information sharing
  - share information as soon as is available
- Integrated product management
  - someone is responsible for the entire project
- Supplier involvement
  - makes best use of suppliers' capabilities
- Continual customer focus
  - ensures that product best meets customers' needs

Case-study: AT&T PBX Concurrent Engineering (1/2)

- Benchmark against competitors
  - took 30% longer than competition to introduce new product
  - decided to shoot for a 40% redution in design time
- Identify breakthrough improvements
  - identified factors that would influence their effort
  - better communication between design and manufacturing
  - better organization of design labs and manufacturing
  - better manager support
- Characterize current process
  - several root cause of delays were identified

Case-study: AT&T PBX Concurrent Engineering (2/2)

- Create new process
  - creation of a model for the new development process
- Verify new process
  - process was tested with pilot product development project
- Implement
  - the new methodology was rolled out across the product lines
- Measure and improve
  - development time had been reduced from 18-30 months to 11 months

Requirements Analysis

Requirements

Informal description of what customer wants.

Specification

Precise description of what design team should deliver.

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

Good Requirements (1/2)

- Correct
  - avoid over-requiring
- Unambiguous
  - clear and with only one plain language interpretation
- Complete
  - all requirements should be included
- Verifiable
  - is each requirement satisfied in the final system?

Good Requirements (2/2)

- Consistent
  - requirements do not contradict each other
- Modifiable
  - can update requirements easily
- Traceable
  - know why each requirement exists
  - go from source documents to requirements
  - go from requirement to implementation
  - back from implementation to requirement

Setting Requirements

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

Specifications

- 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 specification/design language

Control Oriented Specification Languages

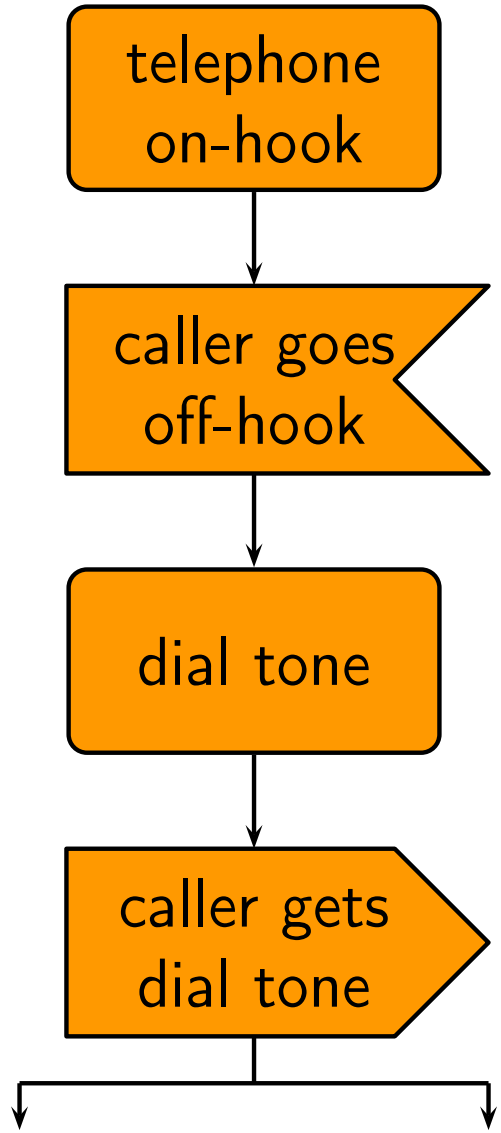
We need languages/techniques to specify the structure of a state-based specification

Control Oriented Specification Languages

- SDL
- Statecharts
- AND-OR Tables

Specification and Description Language

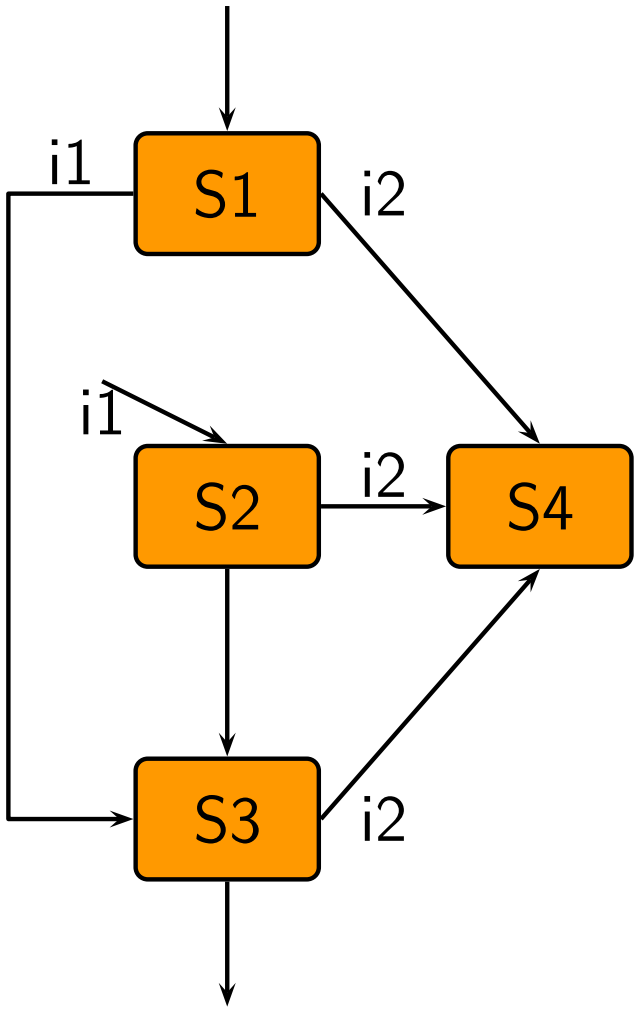
- Used in telecommunications protocol design
- Event-oriented state machine model



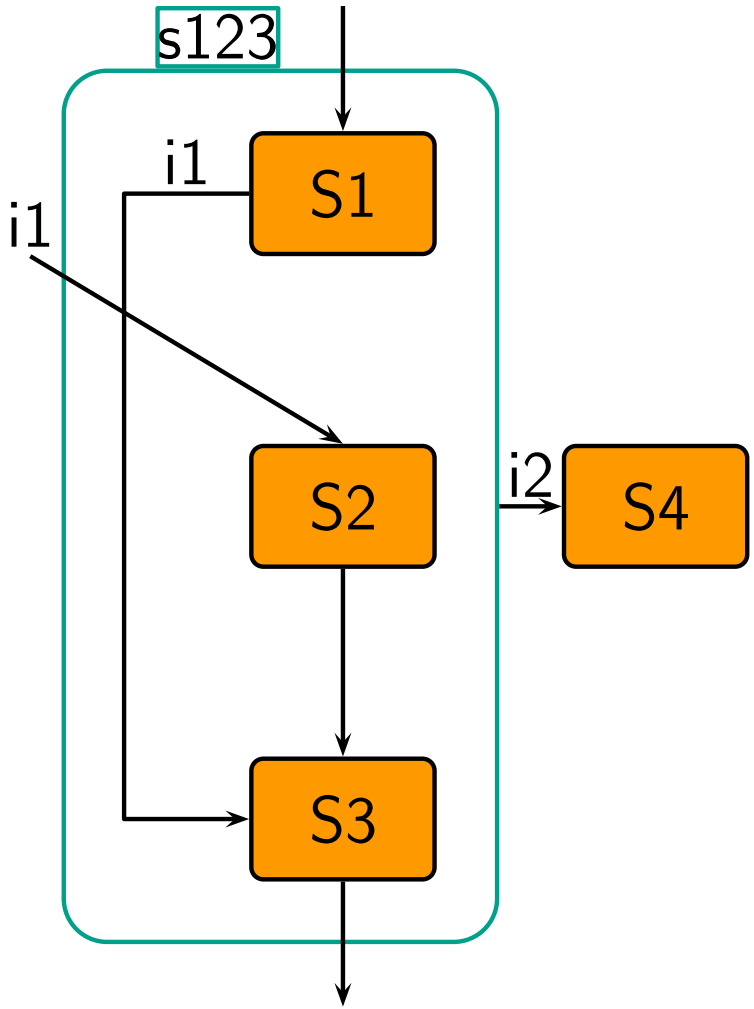
Statecharts

- Ancestor of UML state diagrams
- Event driven
- Provides composite states:
  - OR states
  - AND states
- Composite states reduce the size of the state transition graph

Statechart OR State

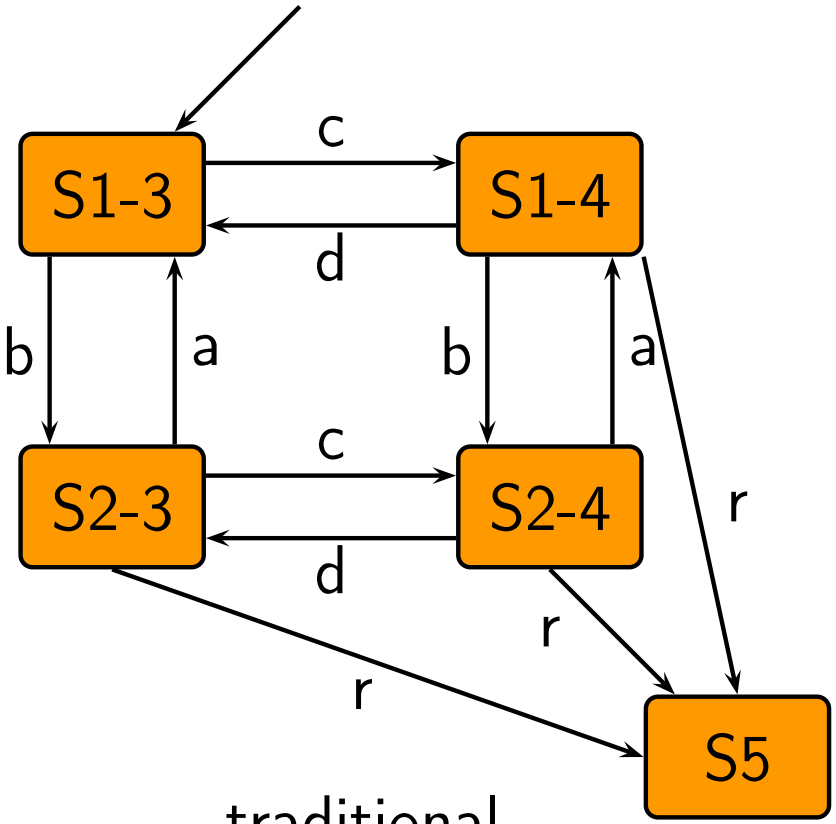


traditional

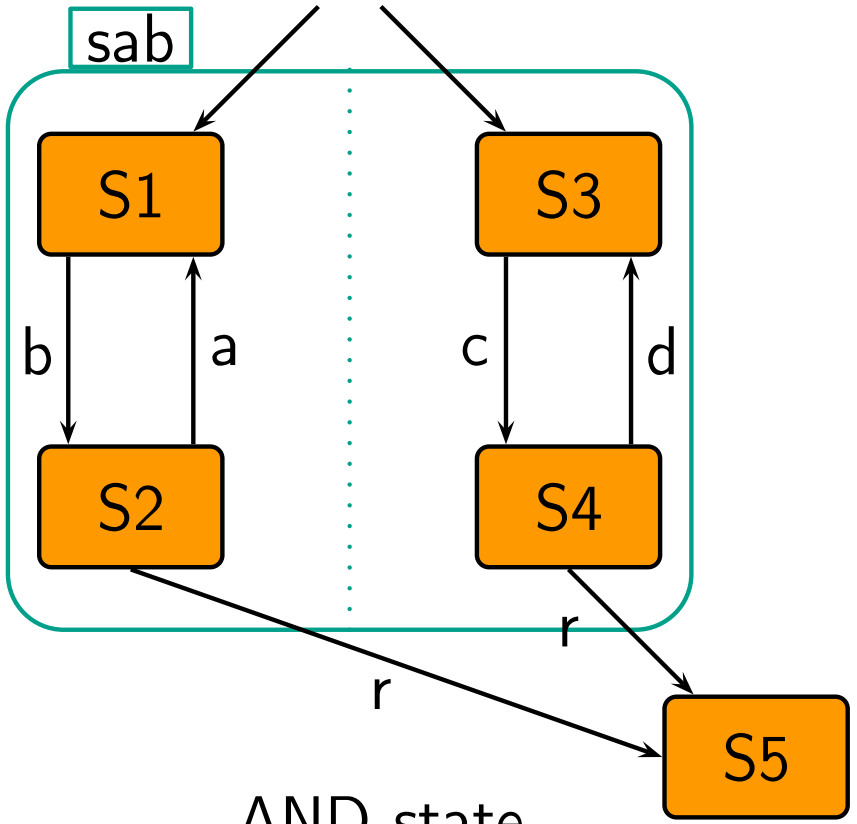


OR state

Statechart AND State



traditional



AND state

AND-OR Tables

- Alternate way of specifying complex conditions

Example (cond1 or (cond2 and !cond3))

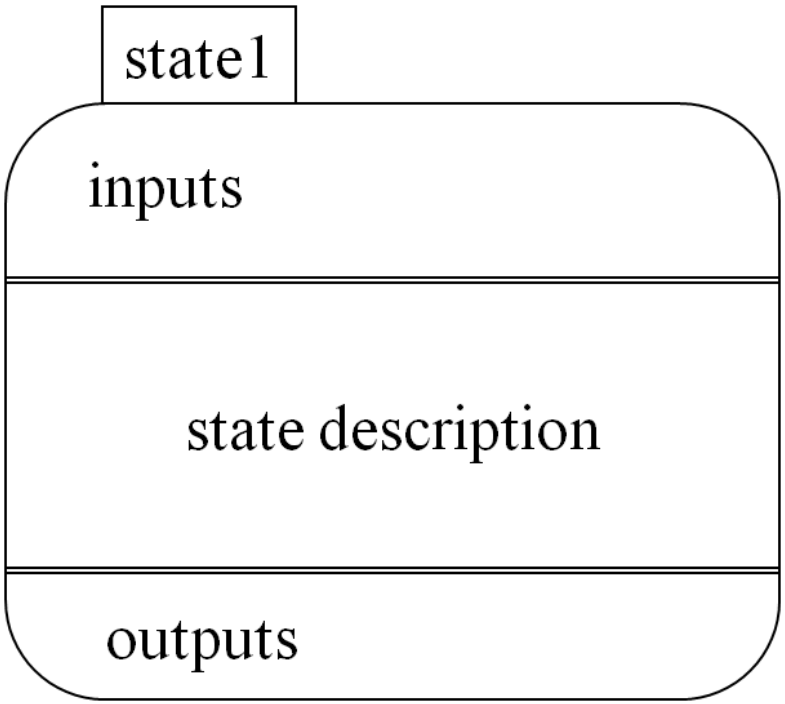
		OR	
AND	cond1	T	-
	cond2	-	T
	cond3	-	F

Case-study: TCAS II Specification

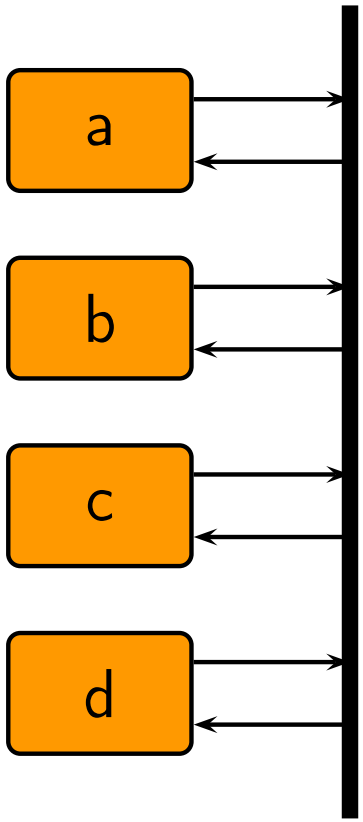
- TCAS II: aircraft collision avoidance system
- Monitors aircraft and air traffic info
- Provides audio warnings and directives to avoid collisions
- Leveson et al used Requirements State Machine Language (RMSL) to capture the TCAS specification

Requirements State Machine Language

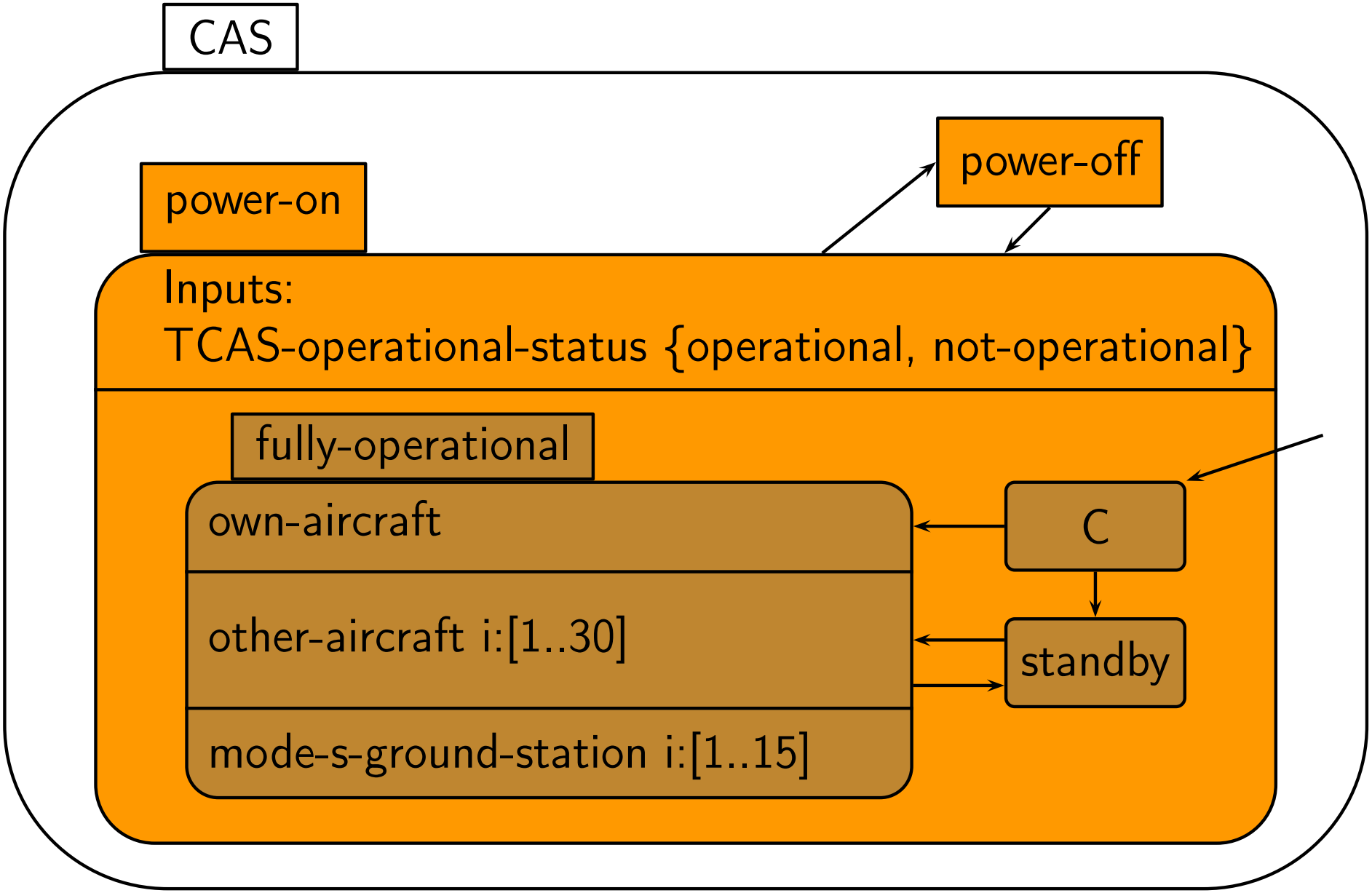
- State description:



- Transition bus for transitions between many states:



TCAS Top-Level Description



How do we turn a specification into a an architecture design?

CRC Cards

- Well-known method for analyzing a system and developing an architecture
- Team-oriented methodology

CRC Cards

CRC

- Classes
  - define the logical groupings of data and functionality
- Responsibilities of each class
  - describe what the classes do
- Collaborators
  - other classes that work with a class

CRC Card Format

Class name:
Superclasses:
Subclasses:
Responsabilities:
Collaborators:

front

Class name:
Class's functions:
Attributes:

front

CRC Methodology (1/2)

- 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 (2/2)

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

Case-study: 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	responsabilities	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

Outline

- System Design Techniques
- Design methodologies
- Requirements Analysis
- Specification
- System Analysis and Architecture Design

References

- Computers as Components: Principles of Embedded Computing System Design. Marylin Wolf. Morgan Kaufman. Ch 9.1-4
- Embedded Systems Architecture, Tammy Noergaard. Elsevier, 2005.

Next Class

- Quality assurance