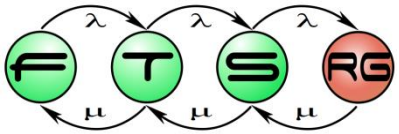
Verifying specifications

Istvan Majzik, Zoltan Micskei

Budapest University of Technology and Economics
Fault Tolerant Systems Research Group



Main topics of the course

- Overview (1)
  - V&V techniques, Critical systems
- **Static techniques (2)**
  - **Verifying specifications**
  - Verifying source code
- Dynamic techniques: Testing (7)
  - Developer testing, Test design techniques
  - Testing process and levels, Test generation, Automation
- System-level verification (3)
  - Verifying architecture, Dependability analysis
  - Runtime verification

Static techniques

WHAT: Documents, code or other artefact

HOW: Without execution

USING: Manual examination (reviews) OR automated analysis (static analyses)

Motivation

Incomplete or inconsistent specification is a major source of failures!

The 60-70% of IT project failures can be traced back to insufficient requirement analysis – Meta Group (2003)

"Significantly more defects were found per page at the earlier phases of the software life cycle. " [inspection of 203 documents]
An analysis of defect densities found during software inspections (JSS, DOI: 10.1016/0164-1212(92)90089-3)

78% (149 from 192) of faults were due to incomplete specifications from the faults uncovered during testing the Voyager and Galileo probes

Requirement and specification

Requirement

- Vision, request, expectation from
  - Users
  - Stakeholders (authority, management, operator…)
- Basis for validation

Specification

- Request transformed for designer and developers
- Result of analysis (abstraction, structuring)
- Basis for verification

Types of specifications

Level

- System Requirements
- System Architecture
- Software Requirements
- Software Architecture
- Software Module
- …

Language

- Natural language text
- Semi formal
  - UML, SysML models
  - Controlled language
- Formal
  - B, Z…
  - logics

RECAP: REQUIREMENTS

Learning outcomes

- Explain the properties and good practices of textual requirements (K2)

Definition of a requirement

"A condition or capability needed by a user to solve a problem or achieve an objective" (IEEE)

"A condition or capability that must be met or possessed by a system, system component, product, or service to satisfy an agreement, standard, specification, or other formally imposed documents" (IEEE)

Properties of good requirements

- **Identifiable + Unique** (unique IDs)
- **Consistent** (no contradiction)
- **Unambiguous** (one interpretation)
- **Verifiable** (e.g. testable to decide if met)

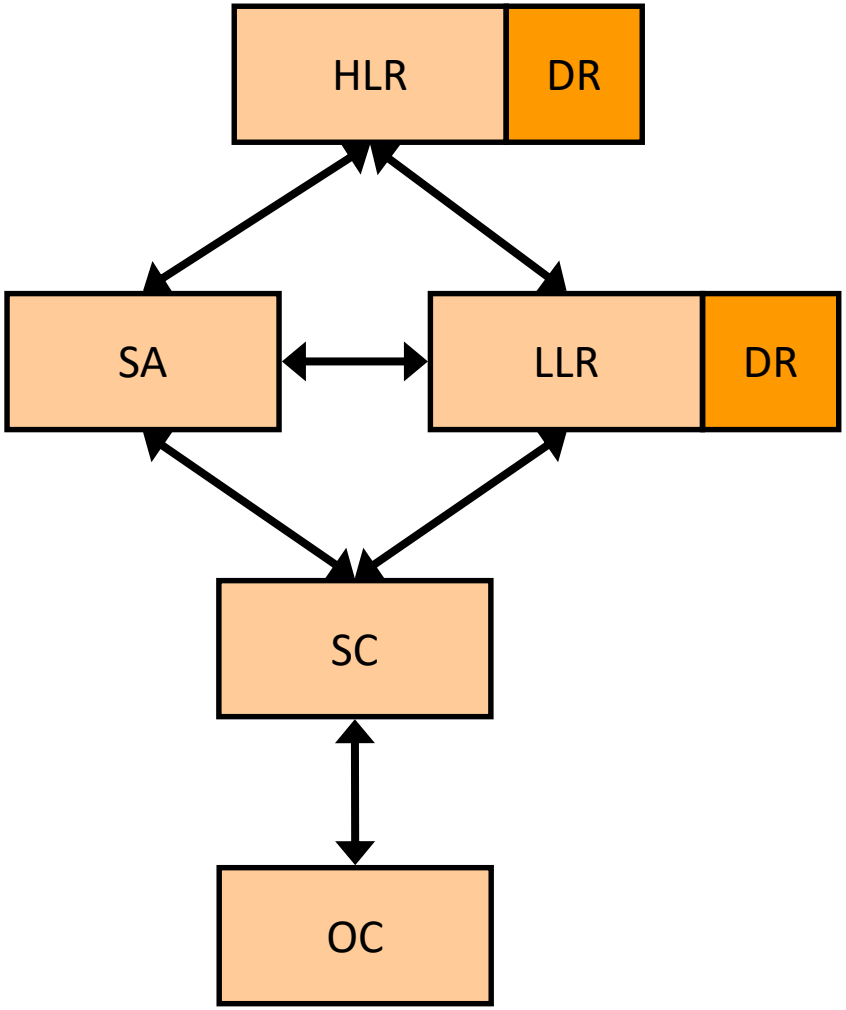
Captured with special statements and vocabulary

Good practices for writing textual requirements

a short description (stand-alone sentence / paragraph) of the problem and not the solution

- English phrasing:
  - Pattern: Subject Auxiliary Verb Object Conditions
  - E.g.: The system shall monitor the room's temperature when turned on.
- Use of auxiliaries (see RFC 2119)
  - Positive: SHALL / MUST > SHOULD > MAY
  - Negative: MUST NOT > SHOULD NOT
  - They specify priorities!

The Certification Perspective: High-level vs Low-Level

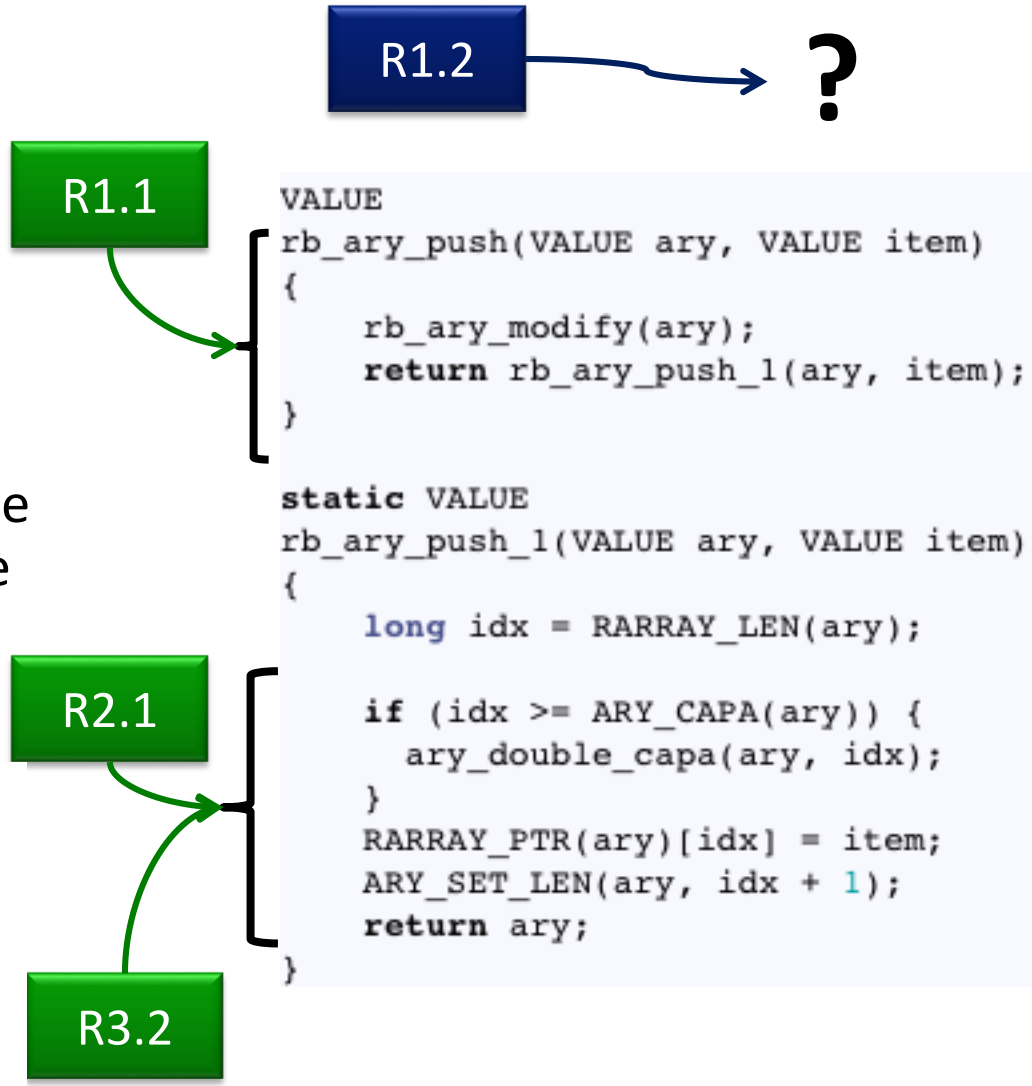


Concepts from DO-178C standard

- High Level Requirements (HLR)
  - customer-oriented
  - black-box view of the software,
  - captured in a natural language (e.g. using shall statements)
- Derived Requirements (DR)
  - Capture design decisions
- Low Level Requirements (LLR)
  - SC can be implemented without further information
- Software Architecture (SA)
  - Interfaces, information flow of SW components
- Source Code (SC)
- Executable Object Code (EOC)

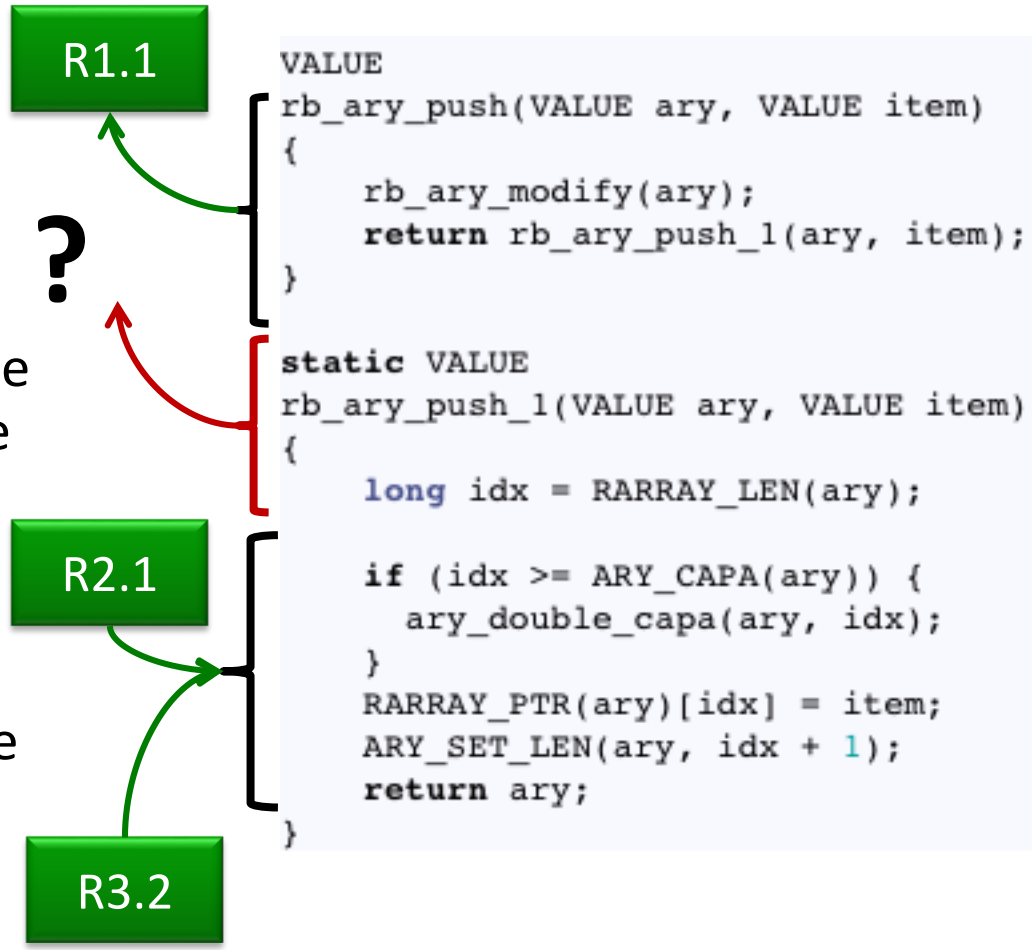
The Concept of Traceability

- Traceability is a core **certification concept**
  - For safety-critical systems
  - See safety standards (DO-178C, ISO 26262, EN 50126)
- **Forward traceability**:
  - From each requirement to the corresponding lines of source code (and object code)
  - Show responsibility



The Concept of Traceability

- Traceability is a core **certification concept**
  - For safety-critical systems
  - See safety standards (DO-178C, ISO 26262, EN 50126)
- **Forward traceability**:
  - From each requirement to the corresponding lines of source code (and object code)
  - Show responsibility
- **Backward traceability**:
  - From any lines of source code to one ore more corresponding requirements
  - No extra functionality



Anti-patterns

1. The system should be safe
2. The system shall use Fast Fourier Transformation to calculate signal value.
3. The system shall continue normal operation soon after a failure.
4. Sensor data shall be logged by a timestamp
5. Unauthorized personnel could not access the system

Too general / high-level

Describes a solution (and not only the problem)

Imprecise (how to verify „soon”?)

Passive should be avoided!

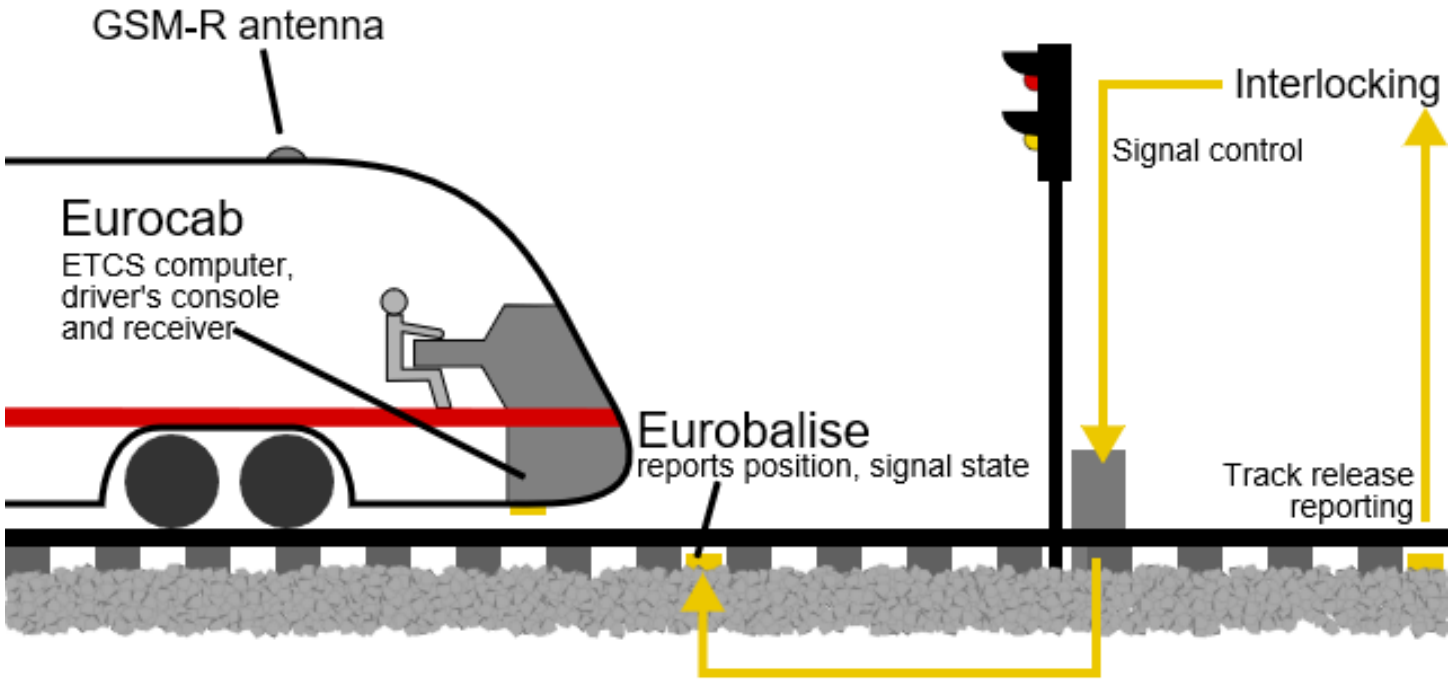
Use specific auxiliaries!

How to identify missing or inconsistent requirements?

Example requirements: ETCS

- European Rail Traffic Management System (ERTMS)
  - European Train Control System (ETCS) + GSM-R

http://www.era.europa.eu/Core-Activities/ERTMS/Pages/Set-of-specifications-3.aspx



Source: https://en.wikipedia.org/wiki/European_Train_Control_System

Example requirements: ETCS

3.4.1 Balise Configurations – Balise Group Definition

3.4.1.1 A balise group shall consist of between one and eight balises.

3.4.1.2 In every balise shall at least be stored:

a) The internal number (from 1 to 8) of the balise

b) The number of balises inside the group

c) The balise group identity.

3.4.3.2 A balise may contain directional information, i.e. valid either for nominal or for reverse direction, or may contain information valid for both directions. In level 1, this information can be of the following type (please refer to section 3.8.5):

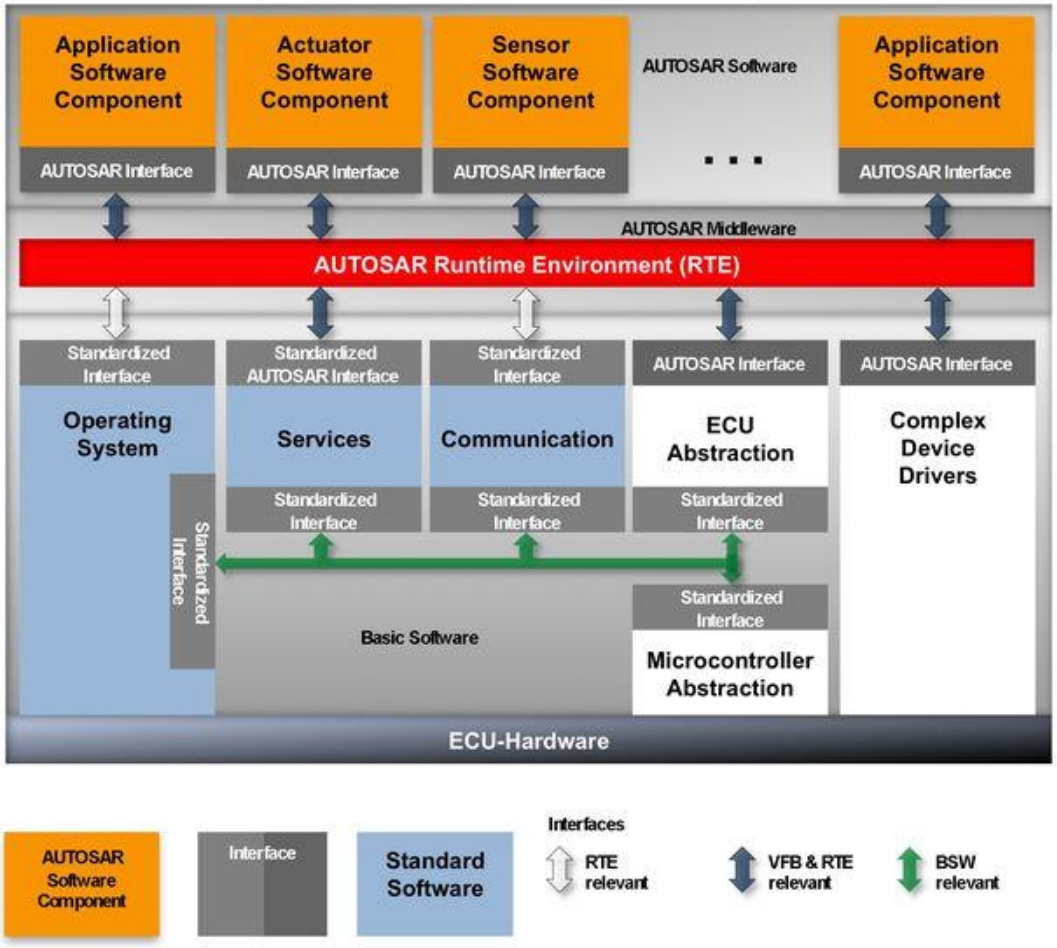
a) Non-infill

b) Intentionally deleted

c) Infill.

Example requirements: AUTOSAR

AUTomotive Open System Architecture



https://www.autosar.org/specifications/

Example requirements: AUTOSAR

3.1 [RS_PO_00001] AUTOSAR shall support the transferability of software.

⌜

Type:	Valid
Description:	AUTOSAR shall enable OEMs and suppliers to transfer software across the vehicle network and to reuse software.
Rationale:	Transferring software across the vehicle network allows overall system scaling and optimization. Redevelopment of software is expensive and error prone.
Use Case:	Application software is reusable across different product lines and OEMs. Scaling and optimizing of vehicle networks by transferring application software. Basic software is reusable across different ECUs and domains.
Dependencies:	RS_PO_00003, RS_PO_00004, RS_PO_00007, RS_PO_00008
Supporting Material:	--

High-level requirement

3 Requirements Tracing

The following table references the requirements specified in **[RS_ProjectObjectives]** and links to the fulfilments of these.

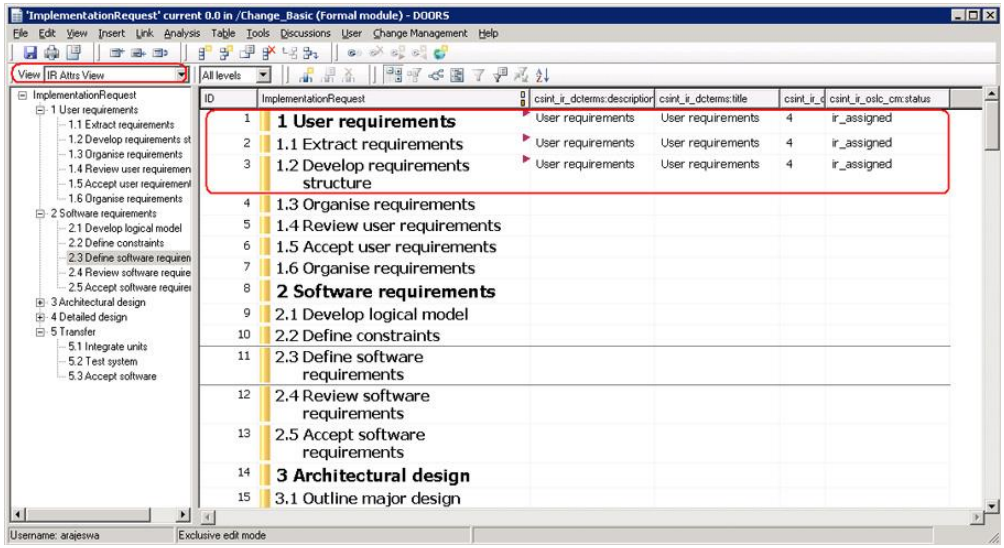
Requirement	Description	Satisfied by
RS_PO_00001	AUTOSAR shall support the transferability of software.	RS_Main_00060, RS_Main_00100, RS_Main_00130, RS_Main_00140, RS_Main_00150, RS_Main_00270, RS_Main_00310, RS_Main_00400, RS_Main_00410, RS_Main_00440, RS_Main_00450, RS_Main_00460, RS_Main_00480

Traceability

[SWS_EcuM_03022]⌈The SHUTDOWN phase handles the controlled shutdown of basic software modules and finally results in the selected shutdown target OFF or RESET.⌋(SRS_ModeMgm_09072)

Low-level requirement

Requirement management tools



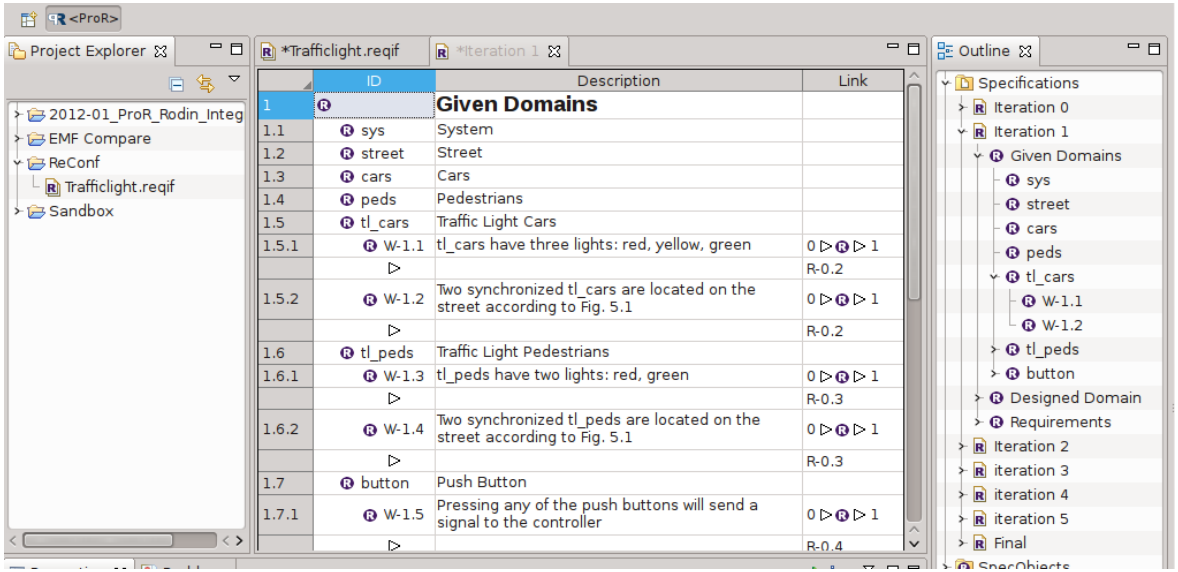
IBM Rational DOORS Next Generation

https://www.youtube.com/watch?v=qYK7_g4Fy44

ProR

ReqIF
Requirements Interchange Format

https://www.youtube.com/watch?v=YC_NrseqWcc



Agile requirements: User stories

"As a <type of user>, I want <some goal> so that <some reason>."

- (Many different templates)
- Index card format
- “Just-in-time requirements”
- Connected to acceptance tests (→BDD)

REVIEW PROCESS

Based on ISTQB Foundation Level Syllabus

Learning outcomes

- Recall the different types of review processes (K1)

Levels of formality in review

Informal review

- No formal process
- Peer or technical lead reviewing

Walkthrough

- Meeting led by author
- May be quite informal

Technical review

- Documented process
- Review meeting with experts
- Pre-meeting preparations for reviewers

Inspection

- Formal process
- Led by a trained moderator

Source: ISTQB CTFL

Activities of a formal review

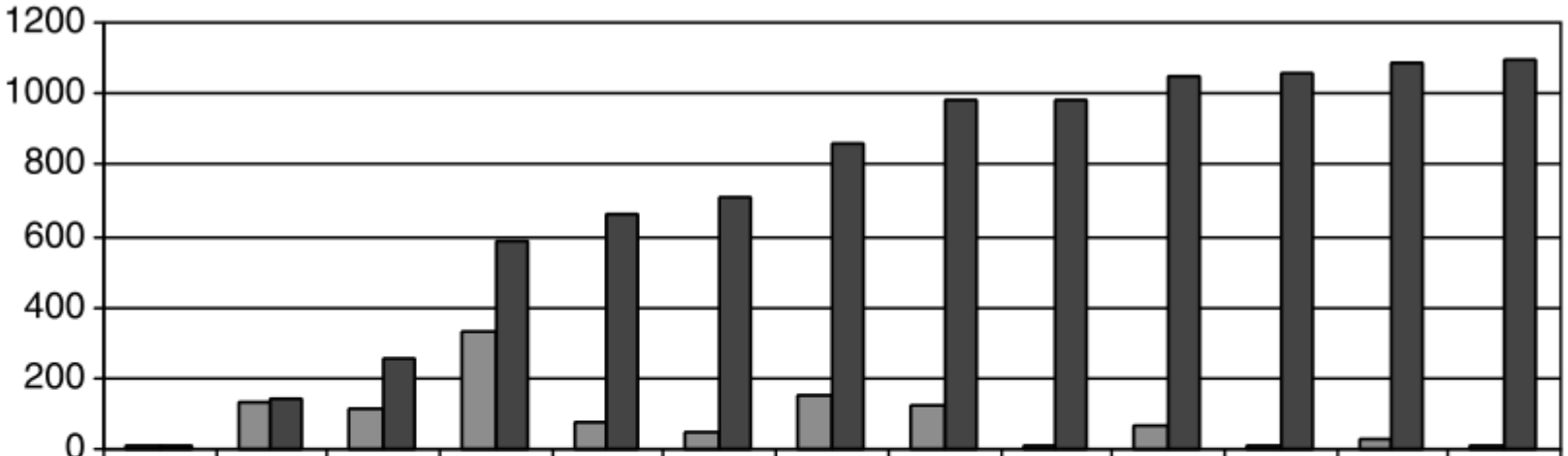
Planning	• Defining review criteria • Allocating roles
Kick-off	• Distributing documents • Explaining objectives
Individual preparation	• Reviewing artefacts • Noting potential defects, questions and comments
Review meeting	• Discussing and logging results • Noting defects, making decisions
Rework	• Fixing defects • Recording updated status
Follow-up	• Checking fixes • Checking on exit criteria

Source: ISTQB CTFL

Recommendations for reviews

- Thorough review is time consuming
  - Usually 5-10 pages / hour
  - Can be 1 page / hour
- Increasing the number of pages to review can greatly reduce the defects found
  - Practical limits: meeting is 2 hours, max 40 pages

Data on safety-critical projects



	fs	fs rev	des	des rev	code	code rev	ut des	ut run	int des	int run	sys des	sys run	engine
correction	10	133	109	332	78	46	149	123	7	64	7	24	13
cumulative	10	143	252	584	662	708	857	980	987	1051	1058	1082	1095

Fig. 2 Corrections found at each phase and cumulative totals

fs – functional specification
fs rev – fs review
des – design
des rev – review
ut des – unit test design
ut run – ut execution
int – integration test
sys – system test

Source: The Economics of Unit Testing, ESE 11: 5–31, 2006

REVIEW CRITERIA

Learning outcomes

- List typical review criteria for requirements and specifications (K1)
- Perform review of requirements and specifications (K3)

Typical review criteria

Completeness	• Functions • References
Consistency	• Internal and external • Traceability
Implement-ability	• Resources • Usability, Maintainability • Risks: budget, technical, environmental
Verifiability	• Specific • Unambiguous • Measurable

Criteria from IEEE Std 830-1998

Correct

- Every requirement stated therein is one that the software shall meet
- Consistent with external sources (e.g. standards)

Unambiguous

- Every requirement has only one interpretation
- Formal or semi-formal specification languages can help

Complete

- For every (valid, invalid) input there is specifies behavior
- TBD only possible resolution

Consistent

- No internal contradiction, terminology

Ranked for importance and/or stability

- Necessity of requirements

Verifiable

- Can be checked whether the requirement is met

Modifiable

- Not redundant, structured

Traceable

- Source is clear, effect can be referenced

Criteria from IEEE Std 29148-2011

Necessary

- If it is removed or deleted, a deficiency will exist, which cannot be fulfilled by other capabilities

Implementation Free

- Avoids placing unnecessary constraints on the design

Unambiguous

- It can be interpreted in only one way; is simple and easy to understand

Consistent

- Is free of conflicts with other requirements

Complete

- Needs no further amplification (measurable and sufficiently describes the capability)

Singular

- Includes only one requirement with no use of conjunctions

Feasible

- Technically achievable, fits within system constraints (cost, schedule, regulatory…)

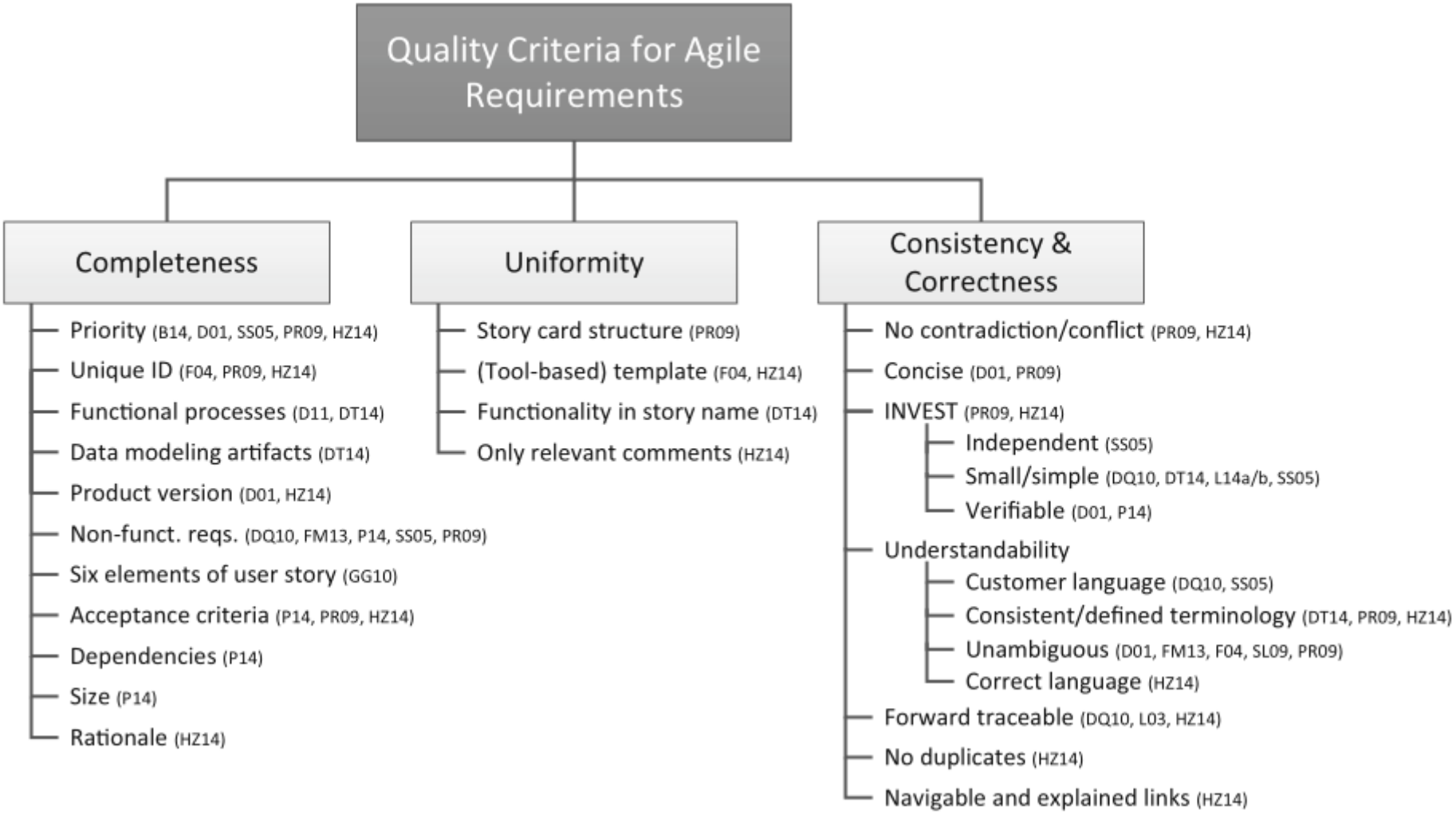
Traceable

- Upwards traceable to the stakeholder statements; downwards traceable to other documents

Verifiable

- Has the means to prove that the system satisfies the specified requirement

Quality criteria for agile requirements



Source: Heck, P. & Zaidman, A. A systematic literature review on quality criteria for agile requirements specifications. Software Qual J (2016). DOI: [10.1007/s11219-016-9336-4](https://doi.org/10.1007/s11219-016-9336-4)

EXERCISE Specification review

CALCULATOR

Követelményspecifikáció

Jelen dokumentum célja, hogy a CALCULATOR alkalmazással kapcsolatos követelményeket és tervezési szempontokat összefoglalja.

Az alkalmazás bemutatása

A Calculator alkalmazás célja, hogy egy egyszerű számológépet megvalósítson, mely képes nemnegatív egész számokkal alapműveletek elvégzésére.

Az alkalmazás felhasználói felülete

A rendszernek egyfajta felülete van, ezt használja az alkalmazás összes felhasználója.

Ezen a felületen keresztül a felhasználók a következő funkciókat érhetik el:

- Számológép be- és kikapcsolása.
- Számrendszerváltás: bekapcsolt állapot esetén a számológép bármikor átállítható, hogy a számokat kettes vagy tízes számrendszerben jeleníti meg.
- Alapműveletek elvégzése egész számokkal.

Az alkalmazás részletes követelményei

Funkcionális követelmények

Az alkalmazásnak a következő funkcionális követelményeket kell teljesítenie.

Azonosító	Név	Prioritás	Leírás
REQ_1	Be- és kikapcsolás	Magas	A számológépet bármikor be és ki kell tudni kapcsolni. Kikapcsoláskor nem kell semmilyen állapotot megőriznie, bekapcsolás után mindig az alapállapotból kell indulnia.
REQ_2	Számrendszerek	Közepes	A számokat meg kell tudnia jelenítenie tízes és kettes számrendszerben.
REQ_3	Alapműveletek elvégzése	Magas	A számológépnek a következő alapműveleteket kell tudnia elvégeznie: összeadás, kivonás, szorzás.
REQ_3	32 bites számok kezelése	Magas	A rendszernek 0 és 2^{32}-1 közötti számokat kell tudnia kezelnie.

Nem-funkcionális követelmények

A számológép felületének az adott számítás komplexitásával arányos időn belül választ kell adnia, a felület nem „fagyhat le".

Az alkalmazással szemben nincsenek speciális egyéb nem-funkcionális követelmények.

Read and review the example specification

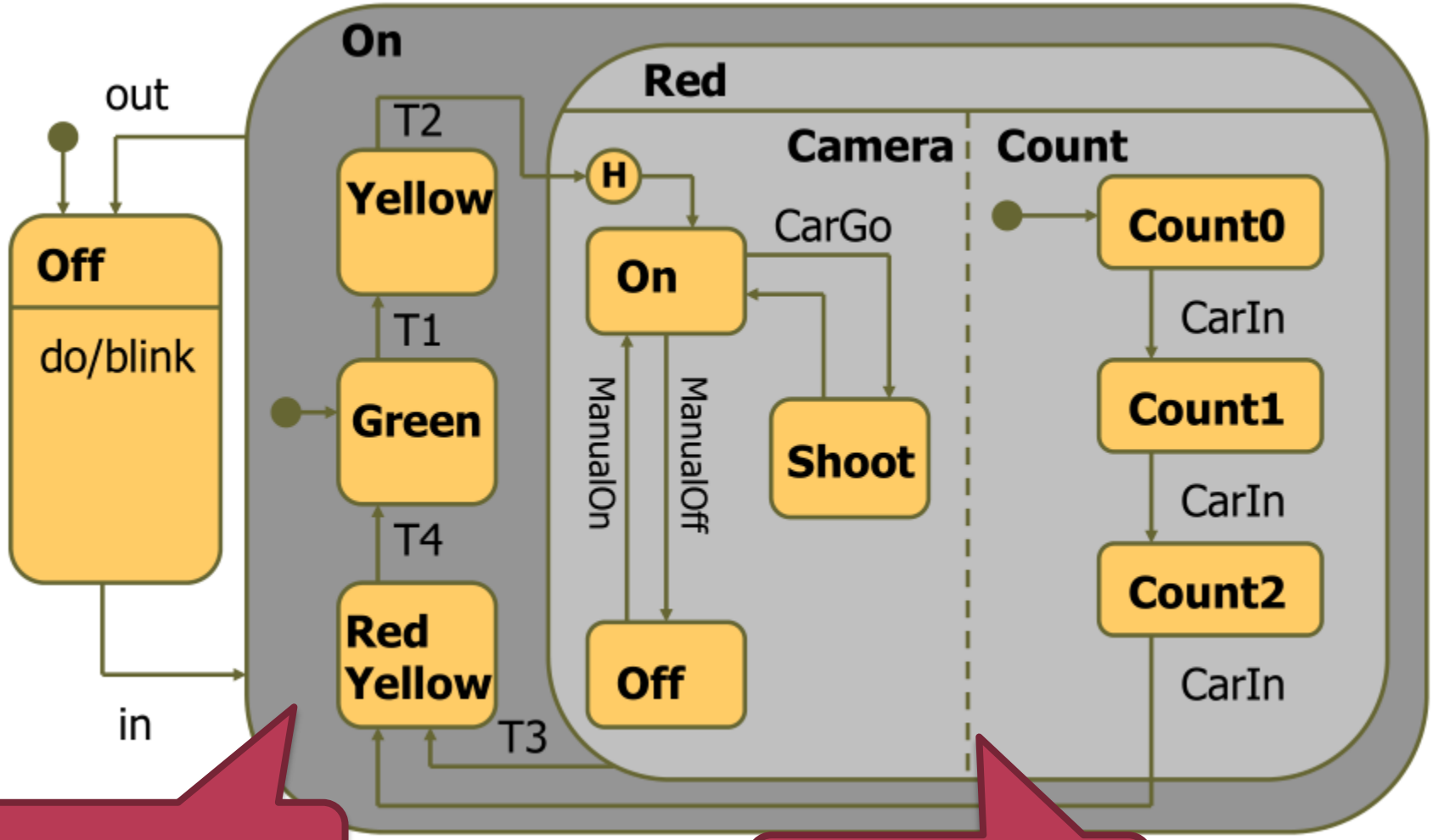
Note possible defects and comments

VERIFYING STATE MACHINES

Learning outcomes

- Perform checking of UML state machines for completeness and unambiguousness (K3)

Recap: UML 2 State Machines



Recap: UML 2 State Machines

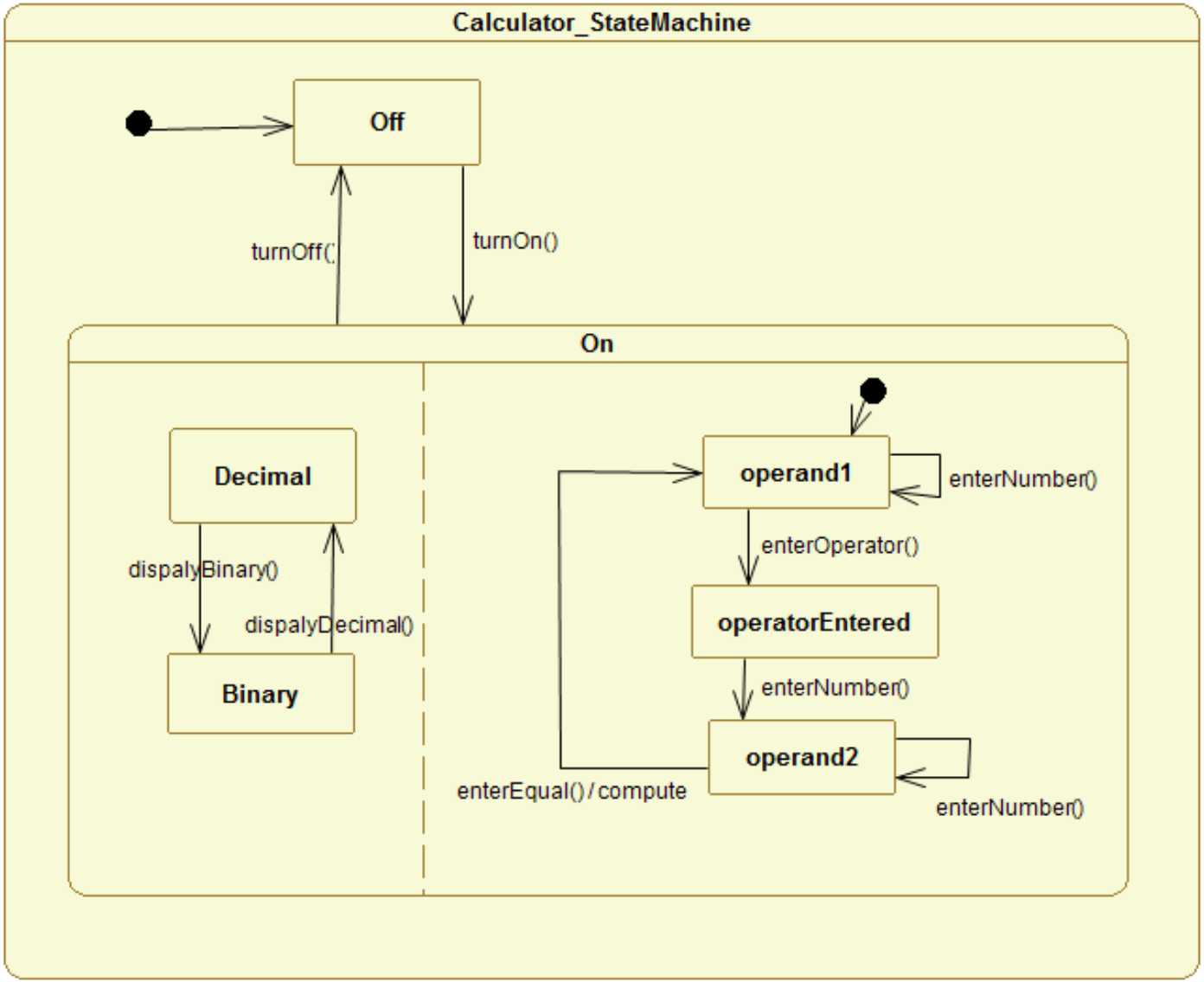
- Challenges for understanding
  - Hierarchical states -> state configuration
  - Conflicting transitions -> priorities, non-determinism
  - Concurrent regions -> concurrent transitions
  - Evaluation of guards
- For more information
  - Formal methods course (VIMIMA07)
  - UML 2.5 specification (OMG)
  - G. Pinter: Model based program synthesis and runtime error detection for dependable embedded systems, PhD dissertation, BME, 2007

Typical criteria for state machines

- Completeness:
  - For each event
  - in each state configuration
  - the behavior is specified (transition or self-transition)

- Unambiguous:
  - for a given event
  - in a given state configuration
  - there is only one enabled transition

EXERCISE State machine review



1. Review the state machine

2. Check completeness

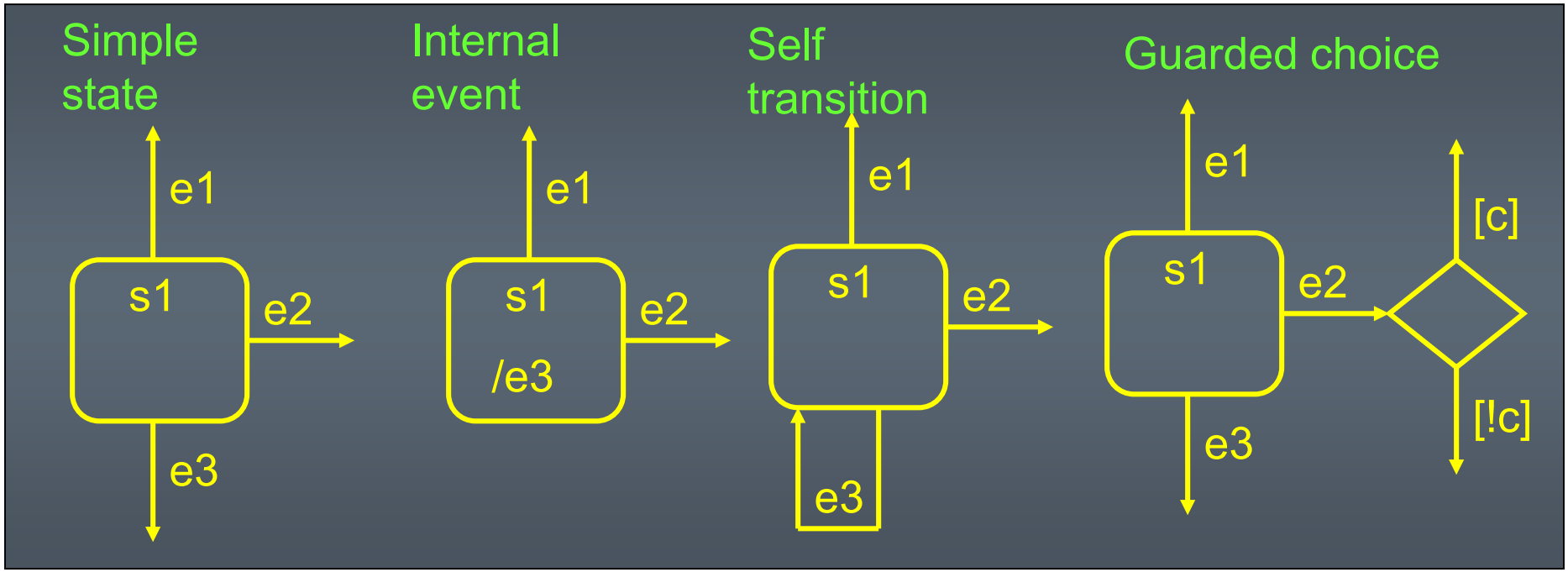
Detailed criteria for UML state machines

- Completeness
- Unambiguousness
- Initial pseudo-states
- Hiding transitions
- Reachability
- Timeout

Source: Zs. Pap. Checking Safety Criteria under UML. PhD dissertation, BME, 2006.

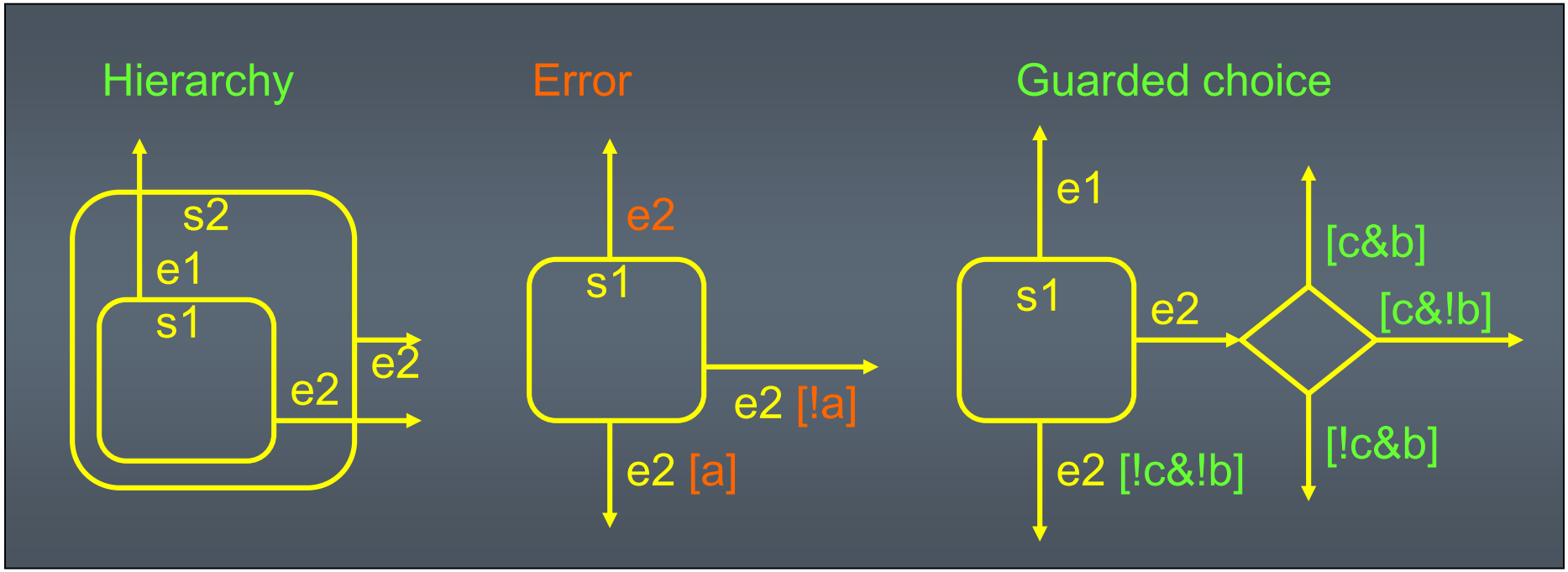
UML State Machines: Completeness

In every state configuration, for every event, for all possible evaluation of guards there is a defined transition.



UML State Machines: Unambiguousness I.

For all state configuration and for all event, for all possible evaluations of guards, for a given hierarchy level there can be only one enabled transition any time.

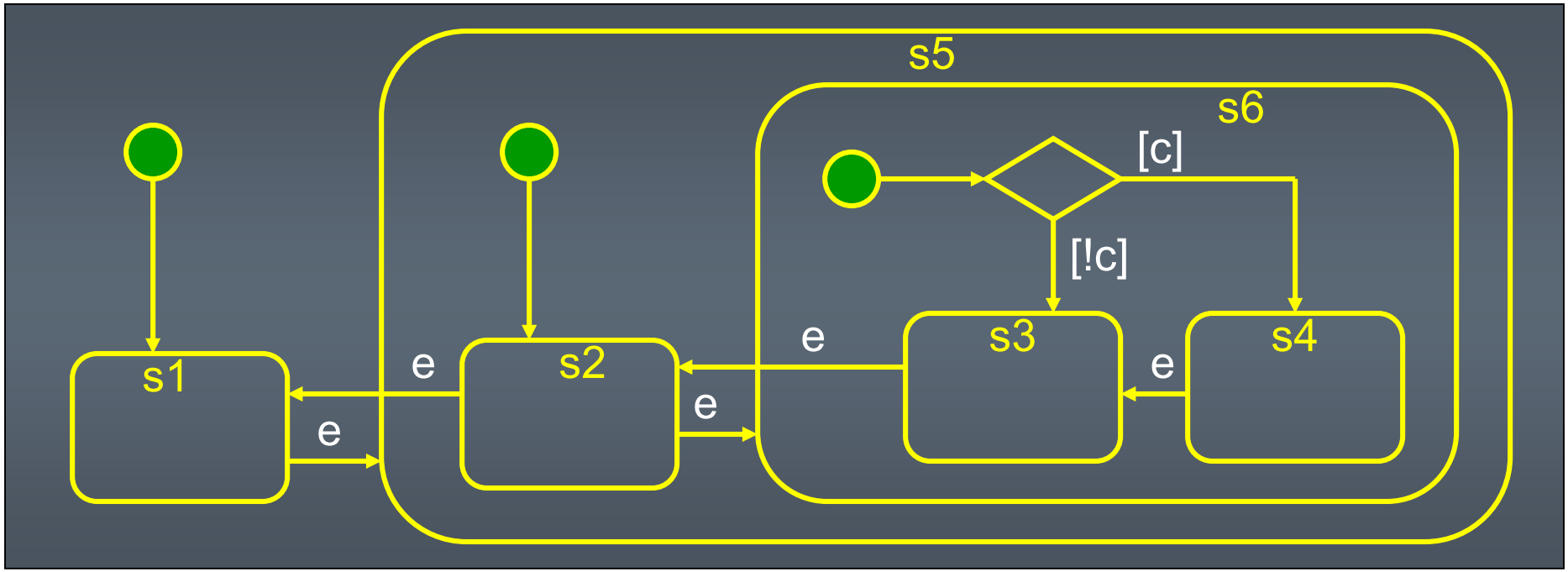


UML State Machines: Unambiguousness II.

In concurrent regions for a given event there should be only in one of the regions an action be defined.

UML State Machines: Initial pseudo-state

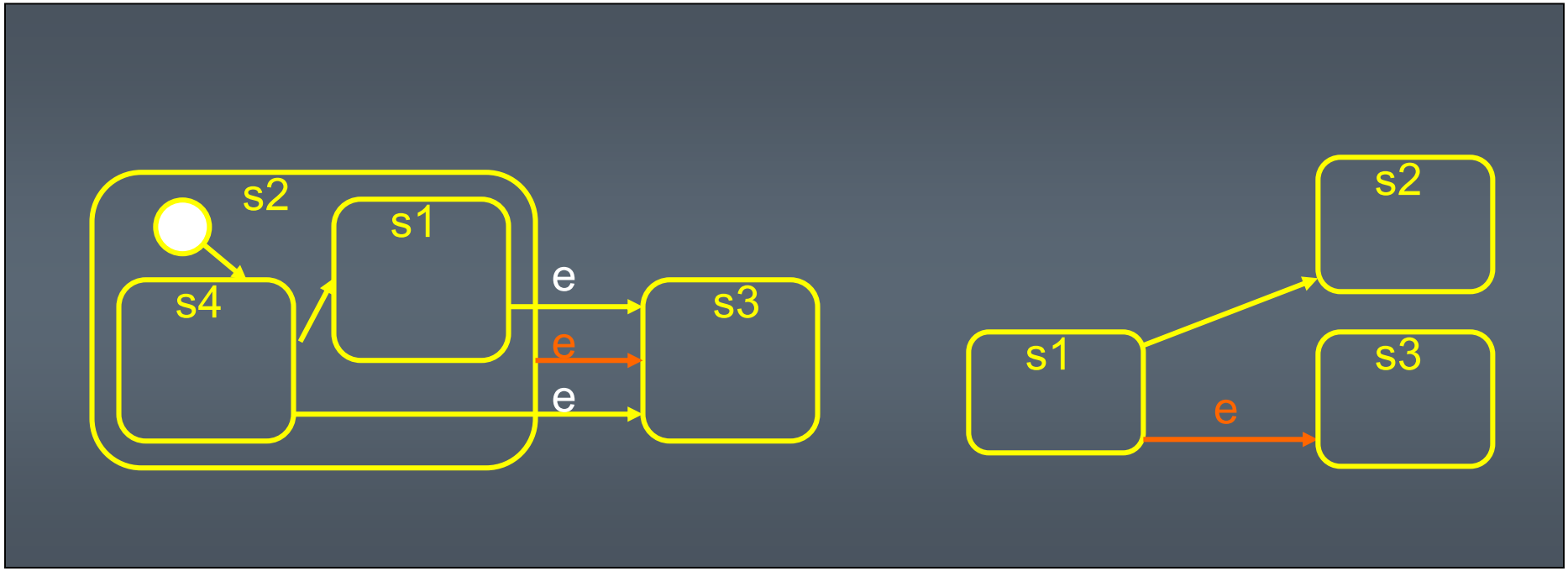
In every region (including the top-level region) there should be an initial pseudo-state.



UML State Machines: Hiding transitions

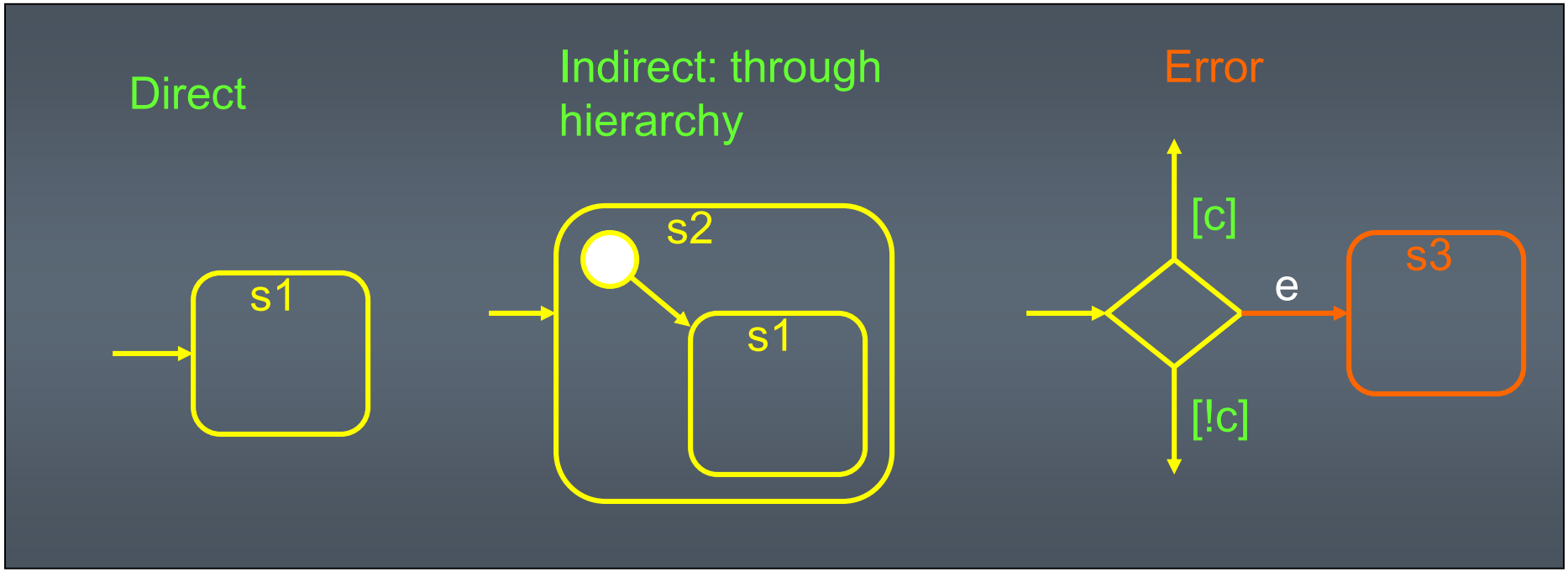
Transitions should not be hided due to

- hierarchies,
- other transitions without triggers



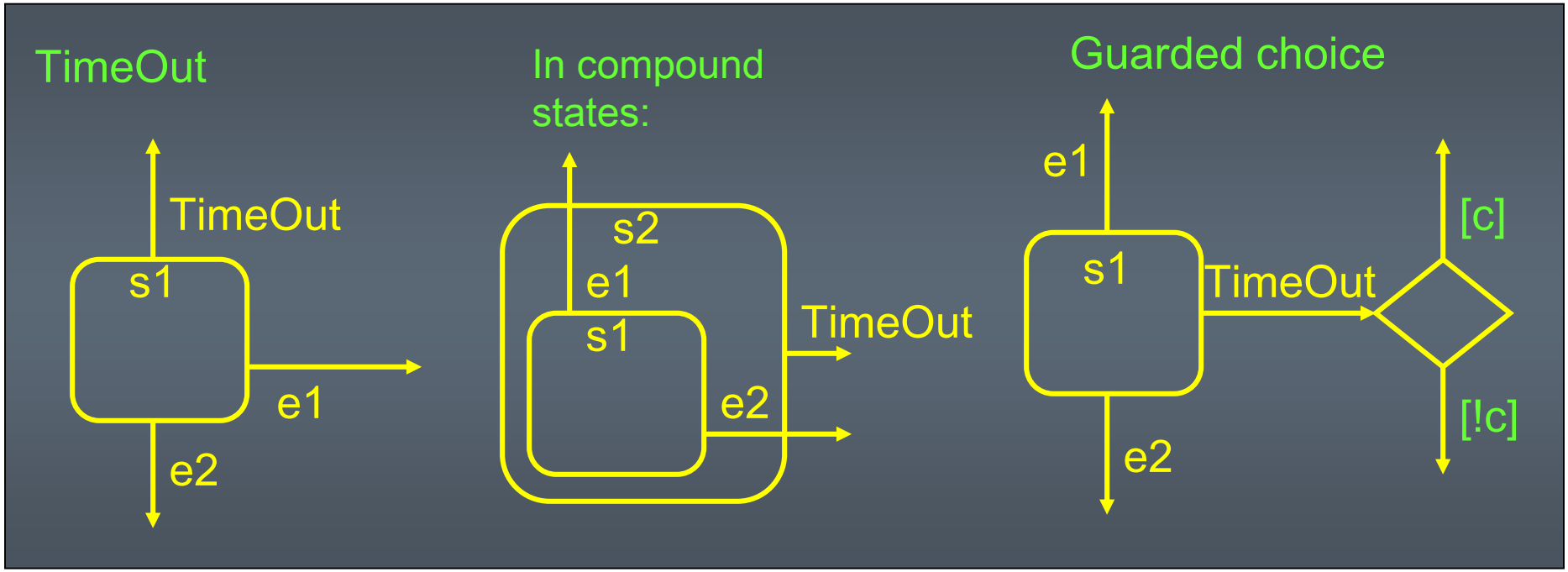
UML State Machines: Reachability

Every state should be reachable either directly or indirectly.

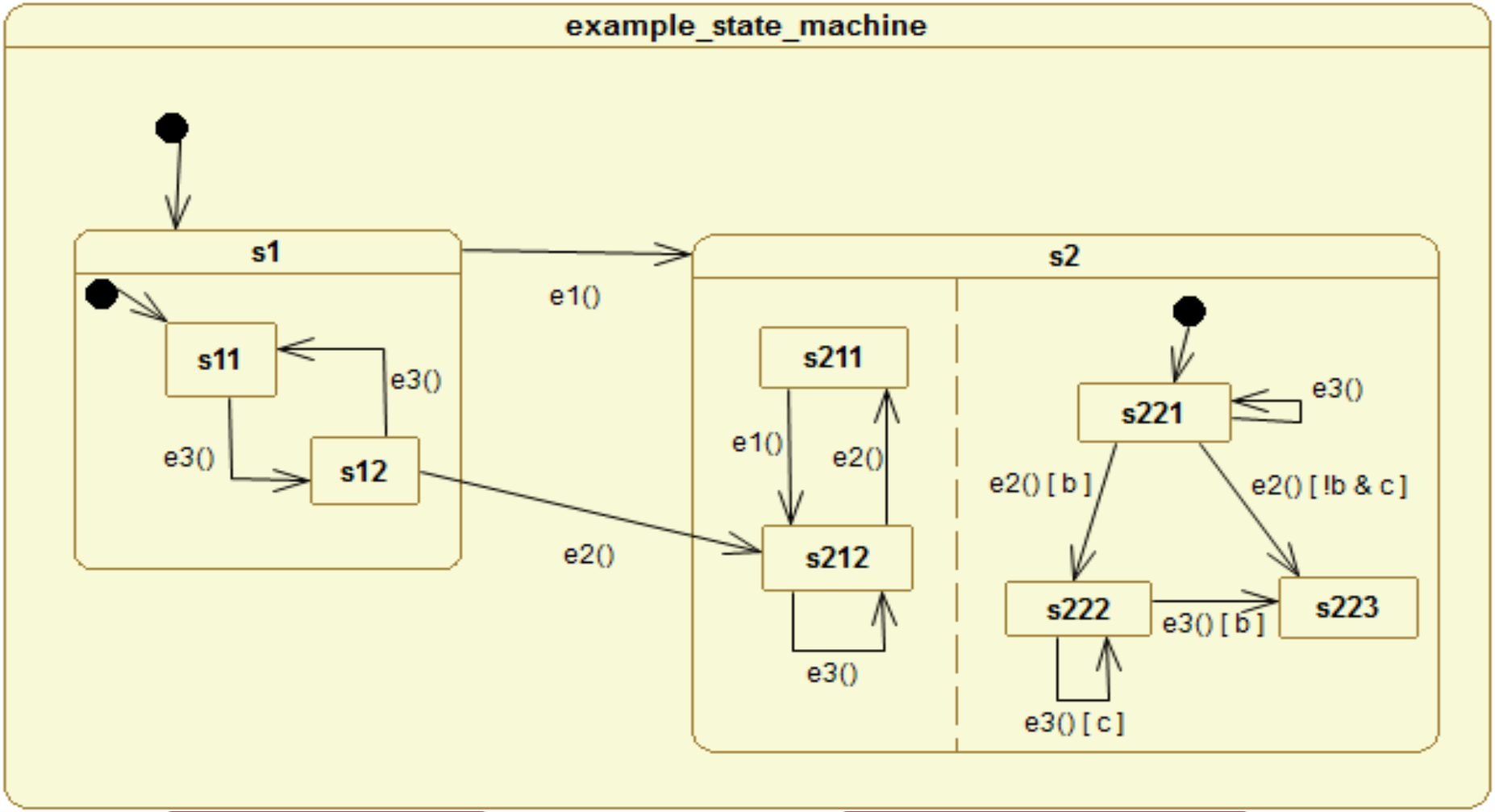


For embedded controllers: timeout

For every state configuration there should be a transition triggered by the TimeOut event



EXERCISE State machine review II.

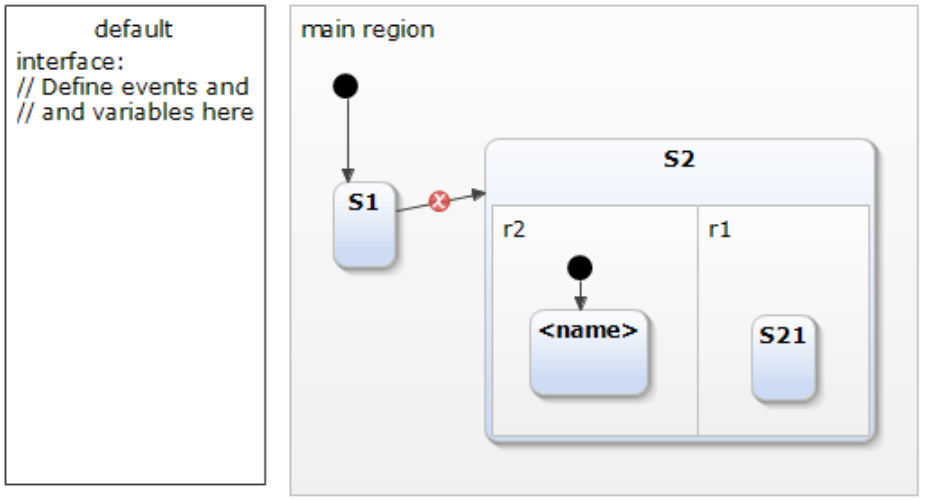


1. Check completeness

2. Check unambiguousness

Checking state machines (tool support)

Yakindu Statechart Tools



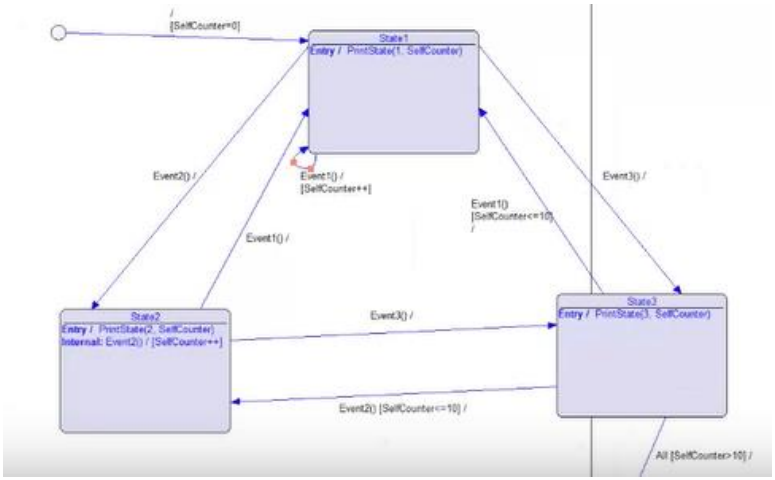
Tasks | Problems | Properties

4 errors, 1 warning, 0 others

Description	Resource	Path	L
Errors (4 items)			
A state must have a name.	default.sct	/yakindu-test	li
Node is not reachable.	default.sct	/yakindu-test	li
Region must have a 'default' entry.	default.sct	/yakindu-test	li
Target state has regions without 'default' entr	default.sct	/yakindu-test	li
Warnings (1 item)			
Missing trigger. Transition is never taken. Use	default.sct	/yakindu-test	li

https://www.youtube.com/watch?v=uO6MASCBPrg

IAR visualSTATE




```
Verification result log for all steps:


Conflicting transitions: (Error)
{
Event1:
State3: Event1() / -> State1
State3: All() [SelfCounter > 10] / -> Final1
}
{
Event2:
State3: Event2() / -> State2
State3: All() [SelfCounter > 10] / -> Final1
}
```

https://www.youtube.com/watch?v=05lTlymLugM

Summary

Static techniques

WHAT: Documents, code or other artefact

HOW: Without execution

USING: Manual examination (reviews) OR automated analysis (static analyses)

Properties of good requirements

- **Identifiable + Unique** (unique IDs)
- **Consistent** (no contradiction)
- **Unambiguous** (one interpretation)
- **Verifiable** (e.g. testable to decide if met)

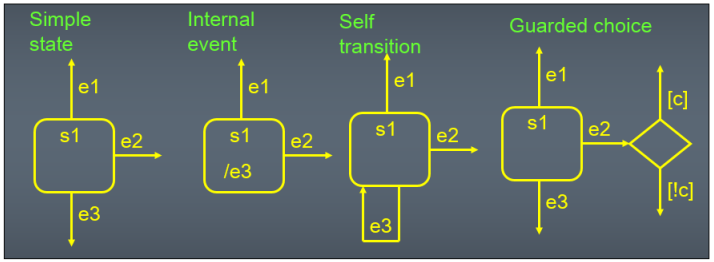
Captured with special statements and vocabulary

Typical review criteria

Completeness	• Functions • References
Consistency	• Internal and external • Traceability
Implement-ability	• Resources • Usability, Maintainability • Risks: budget, technical, environmental
Verifiability	• Specific • Unambiguous • Measurable

UML State Machines: Completeness

In every state configuration, for every event, for all possible evaluation of guards there is a defined transition.



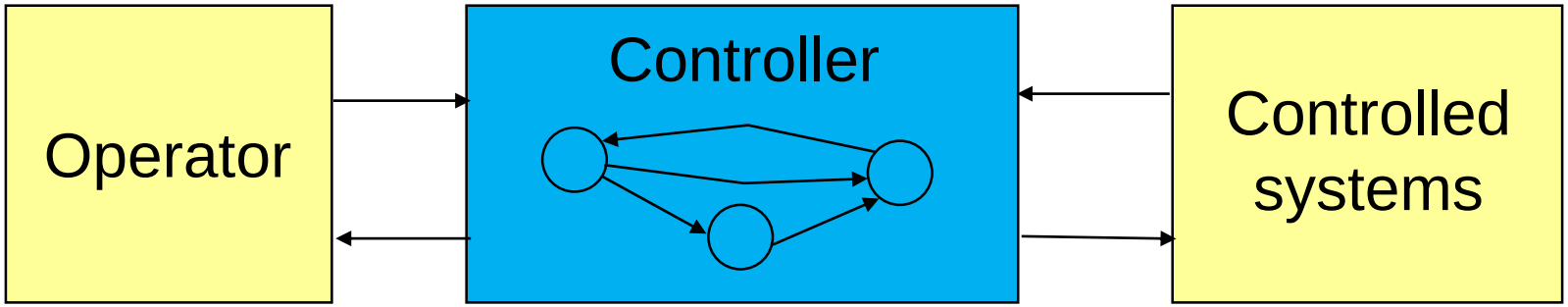
EXTRA MATERIAL: CRITERIA FOR REACTIVE SYSTEMS

Source: N. G. Leveson. “Safeware: System Safety and Computers”. Addison Wesley, 1995

Review criteria for reactive systems

- State definition
- Inputs (events)
- Outputs
- Outputs and triggers
- Transitions
- Human-machine interface

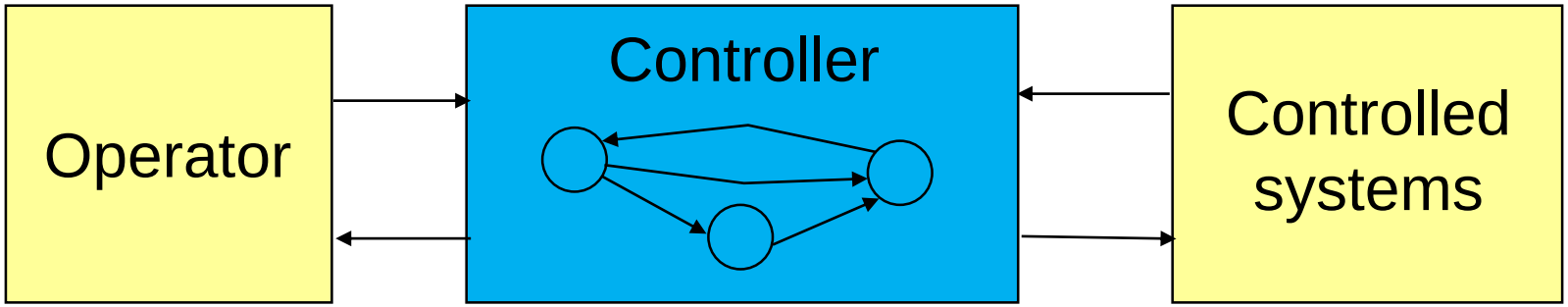
- Initial state is safe
- In case of missing input events there is a timeout and not external events



Review criteria for reactive systems

- State definition
- Inputs (events)
- Outputs
- Outputs and triggers
- Transitions
- Human-machine interfac

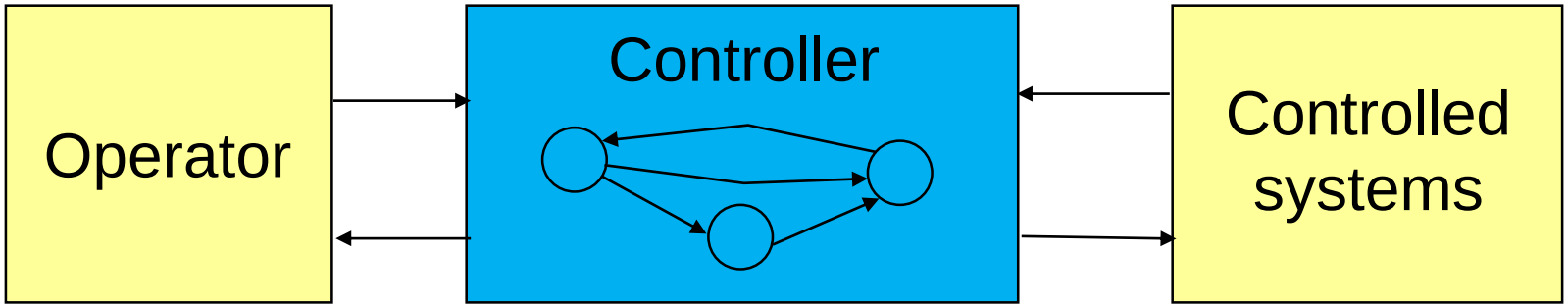
- Fore every input in every state there is a specified behavior
- Reactions are unambiguous (deterministic)
- Input validation (value, timeliness)
- Handing of invalid inputs is specified
- Rate of interrupts is limited



Review criteria for reactive systems

- State definition
- Inputs (events)
- Outputs
- Outputs and triggers
- Transitions
- Human-machine interface

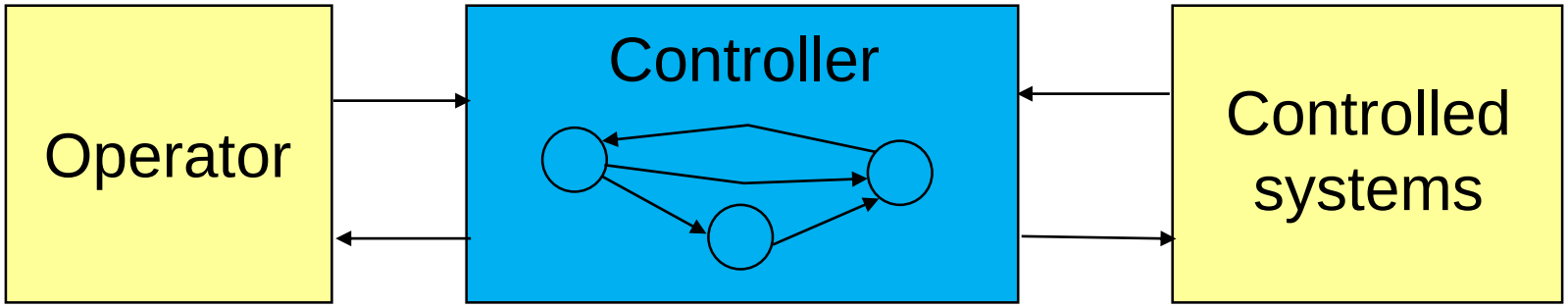
- Credibility checks are specified
- No unused outputs
- Processing rate of environment is respected



Review criteria for reactive systems

- State definition
- Inputs (events)
- Outputs
- Outputs and triggers
- Transitions
- Human-machine interface

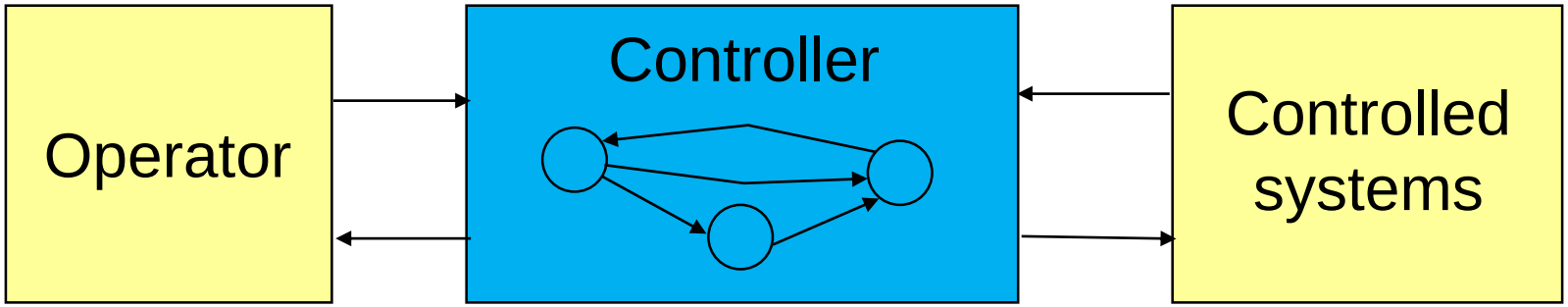
- Effect of outputs is checked through the inputs
- Control loop is stable



Review criteria for reactive systems

- State definition
- Inputs (events)
- Outputs
- Outputs and triggers
- Transitions
- Human-machine interface

- Every state is reachable statically
- Transitions are reversible (there is a way back)
- More than one transitions from dangerous to safe states
- Transitions from dangerous to safe states are confirmed



Review criteria for reactive systems

- State definition
- Inputs (events)
- Outputs
- Outputs and triggers
- Transitions
- Human-machine interface

Output events going to operator:

- Sequence is defined (with priority)
- Update rate is defined
- Rate is limited