Introduction

Istvan Majzik
majzik@mit.bme.hu

Budapest University of Technology and Economics
Dept. of Measurement and Information Systems

Synopsis

- Introduction
- Verification in the requirement phase
- Architecture verification and evaluation
- Verification of the detailed design
  - Classic techniques
  - Formal methods: model checking, equivalence checking
  - Advanced methods: formal verification of extra-functional properties and timed behavior, handling complex designs (large state spaces)
- Verification of the source code
  - Code review, abstract interpretation, symbolic execution
  - Classic techniques of proving program correctness
- Testing and test case generation
  - Test design at unit level
  - Integration and system testing
  - Model based testing and test case generation
- Validation and assessment
- V&V in the maintenance phases
- Integrated approaches

Contents of the lecture

- Motivation
  - What are the quality needs regarding software and what is offered by the software industry?
  - What is the role of software verification and validation techniques?
- Overview of the techniques of software V&V
  - What are the typical techniques in the development process?
- Development life cycle models
  - What is the role of V&V in the different life cycle models?
- The role of development standards
  - How systematic V&V is realized?

Motivation

What are the quality needs regarding software and what is offered by the software industry?

What is the role of software verification and validation techniques?

Expectations

- Service Level Agreements (SLA)
  - Availability (telco servers): 99,999% (5 min/year outage)
- Safety critical systems:
  - Tolerable hazard rate (THR)
  - Safety integrity levels (SIL)

SIL	Probability of dangerous failure per hour per safety function
1	$10^{-6} \leq PFH < 10^{-5}$
2	$10^{-7} \leq PFH < 10^{-6}$
3	$10^{-8} \leq PFH < 10^{-7}$
4	$10^{-9} \leq PFH < 10^{-8}$

15 years lifetime:
1 failure in case of
750 equipment

Operation without failure for approx. 11.000 years???

Different kinds of faults

Development phase

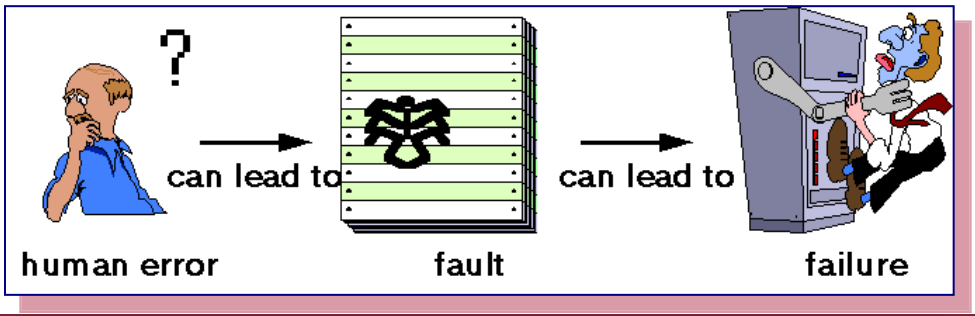
- Specification faults
- Design faults
- Implementation faults

V&V during design

Operational phase

- Hardware faults
- Configuration faults
- Operator faults

Fault tolerance (e.g. redundancy)



Software quality problems

„Defibtech issues a worldwide recall of two of its defibrillator products due to faulty self-test software that may clear a previously detected low battery condition.” (February 2007)

„Cricket Communications recalls about 285,000 of its cell phones due to a software glitch that causes audio problems when a caller connects to an emergency 911 call. (May 2008)”

Nissan recalls over 188,000 SUVs to fix brakes (Update) October 23, 2013

Nissan Motor Co. is recalling more than 188,000 Nissan and Infiniti SUVs worldwide to fix faulty brake control software that could increase the risk of a crash.

RECALLS Feb 12th 2014 at 9:15AM 67

Toyota recalling 1.9M Prius models globally for software update

Statistics for software projects

- Typical size of code
  - 10 kLOC … 1000 kLOC
- Development efforts:
  - Big but average software: 0.1 – 0.5 person months / kLOC
  - Safety critical software: 5-10 person months / kLOC
- Fault removal (review, testing, corrections):
  - 45 - 75% of the whole development efforts
- Change of fault density
  - 10 - 200 faults / kLOC occurring during development

Verification techniques

  - 0.1 - 10 faults / kLOC before operation

How many bugs do we have to expect?

DB Mobility Networks Logistics

How many „Bugs“ do we have to expect?

- Typical production type SW has 1 … 10 bugs per 1.000 lines of code (LOC).
- Very mature, long-term, well proven software: 0,5 bugs per 1.000 LOC
- Highest software quality ever reported :
  - *Less than 1 bug per 10.000 LOC*
  - At cost of more than 1.000 US$ per LoC *(1977)*
  - *US Space Shuttle with 3 m LOC* costing 3b US$ (out of 12b$ total R&D)

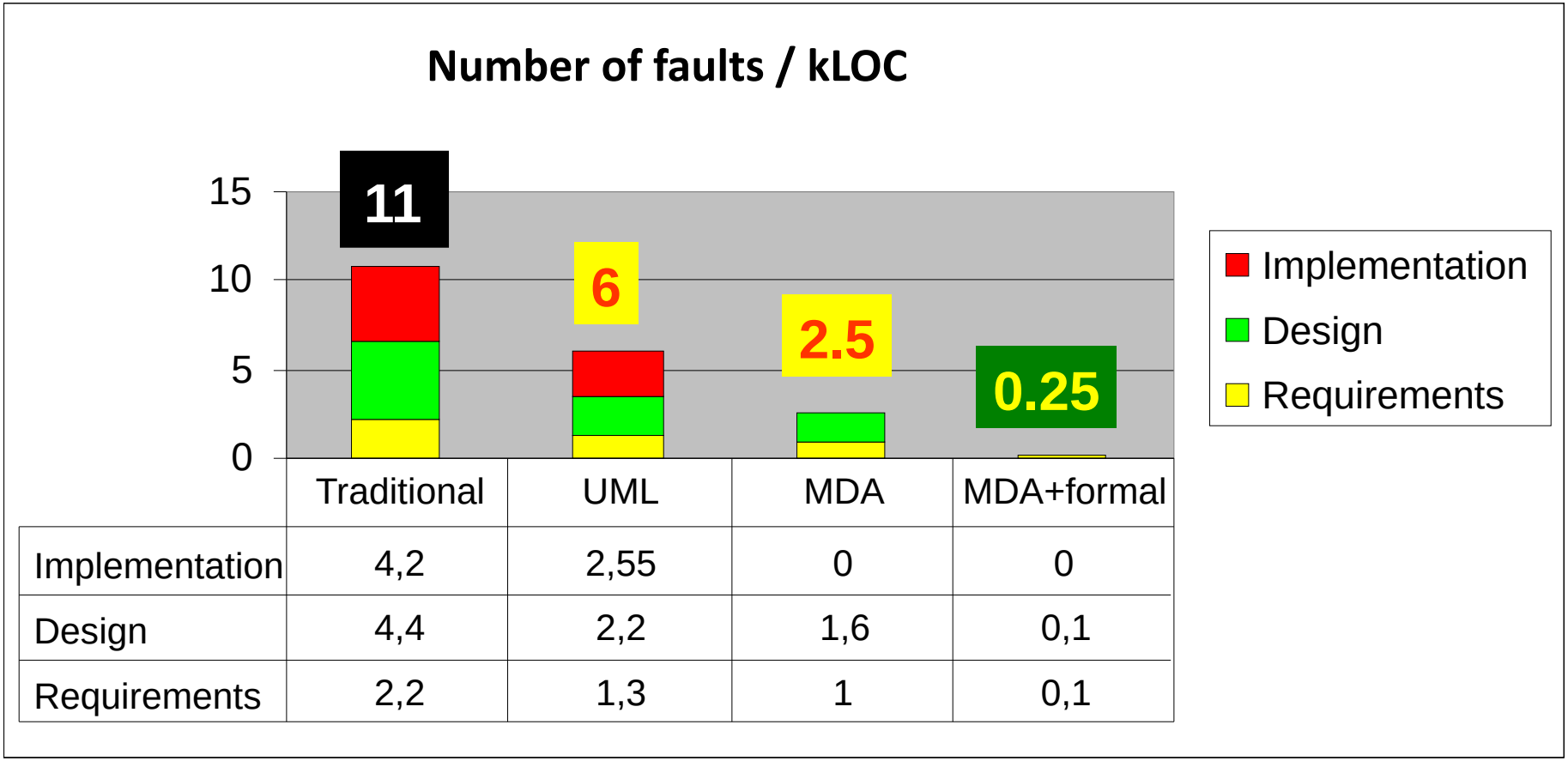
➔ Cost level not typical for the railway sector (< 100€/LoC)

- Typical ETCS OBU kernel software size is about 100.000 LOC or more
  - That means: 100 … 1.000 undisclosed defects per ETCS OBU
  - Disclosure time of defects can vary between a few days …. thousands of years

Source: K-R. Hase: „Open Proof in Railway Safety Software”, FORMS/FORMAT Conference, December 2-3, 2010, Braunschweig, Germany

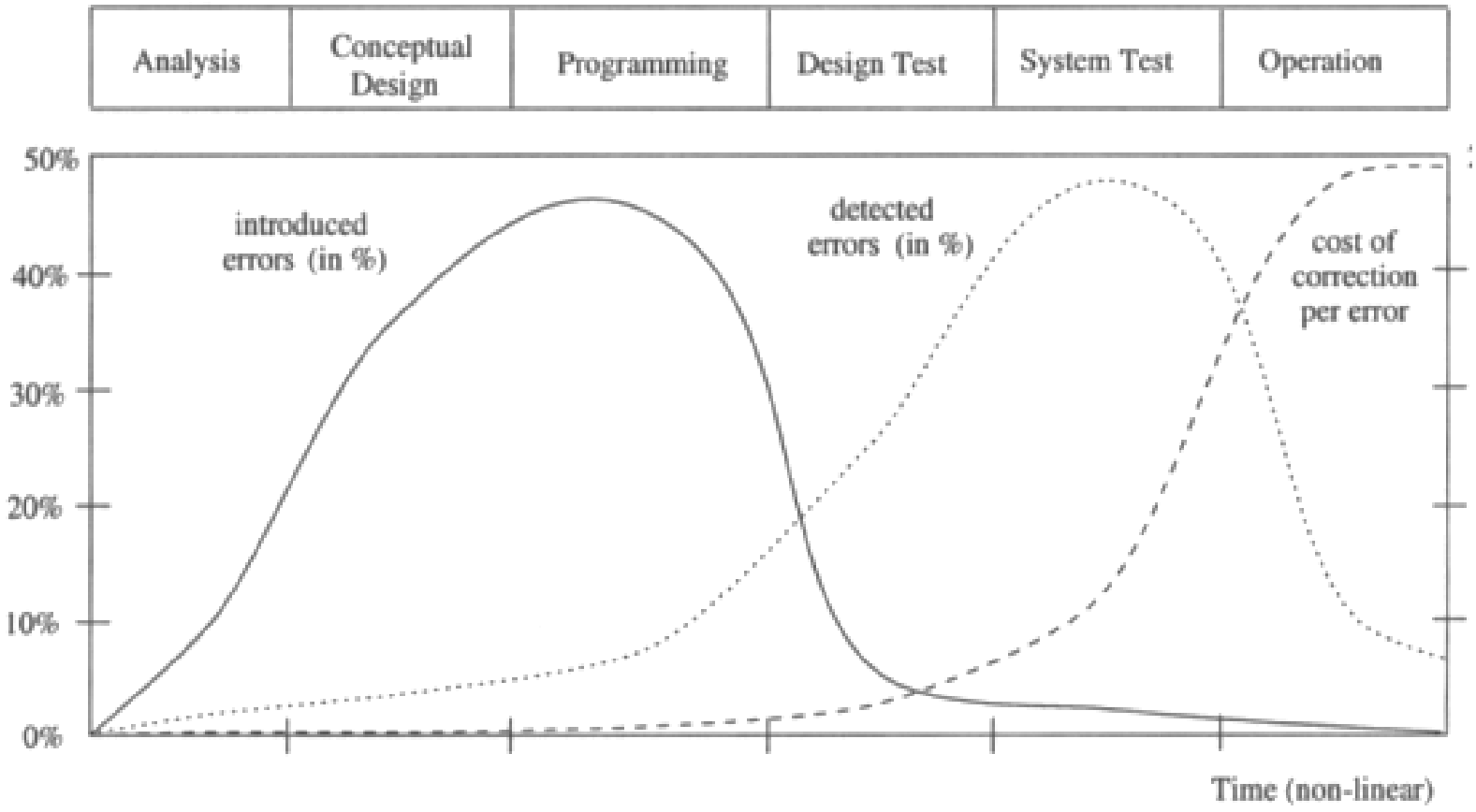
A study in Hungary

- Number of faults in 1 kLOC (embedded software):
  - Manual development and testing: ~ 10 faults
  - Tool-supported automated development: ~ 1-2 faults
  - Automated development with formal methods: < 1 faults



	Traditional	UML	MDA	MDA+formal
Implementation	4,2	2,55	0	0
Design	4,4	2,2	1,6	0,1
Requirements	2,2	1,3	1	0,1

Distribution and cost of bugs



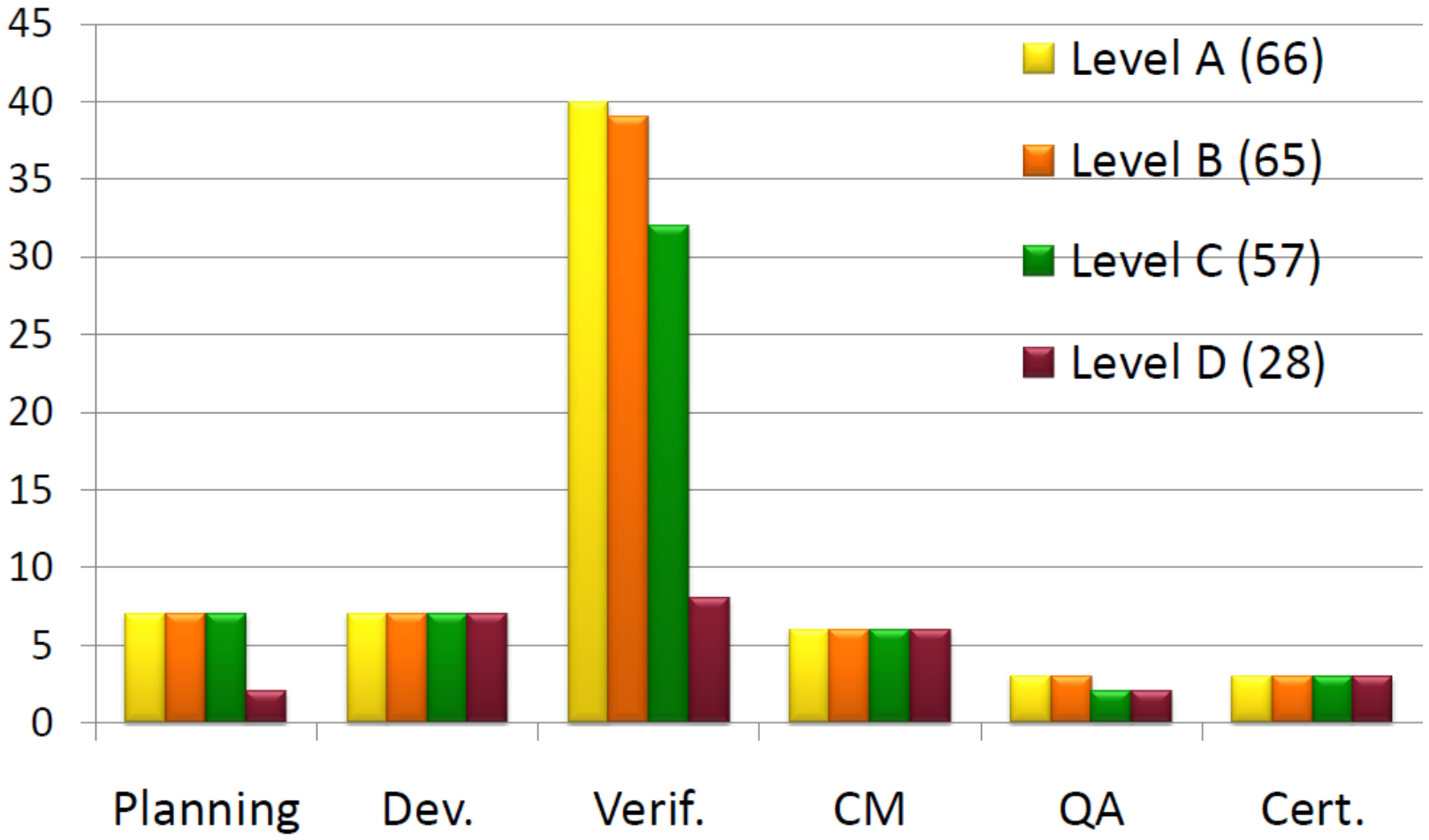
Early V&V reduces cost!

V&V: Verification and Validation

Verification	Validation
„Am I building the system right?”	„Am I building the right system?”
Check correctness and consistency of development phases	Check the result of the development
Conformance of designs/models and their specification	Conformance of the (finished) system and the user requirements
Objective (based on facts); can be automated	Subjective (influenced by user expectations); checking acceptance
Fault model: Design and implementation faults	Fault model: problems in the requirements are also included
Not needed if implementation is automatically generated from specification	Not needed if the specification is correct (very simple)

Example: Development of flight control SW

Objectives Distribution in DO-178B



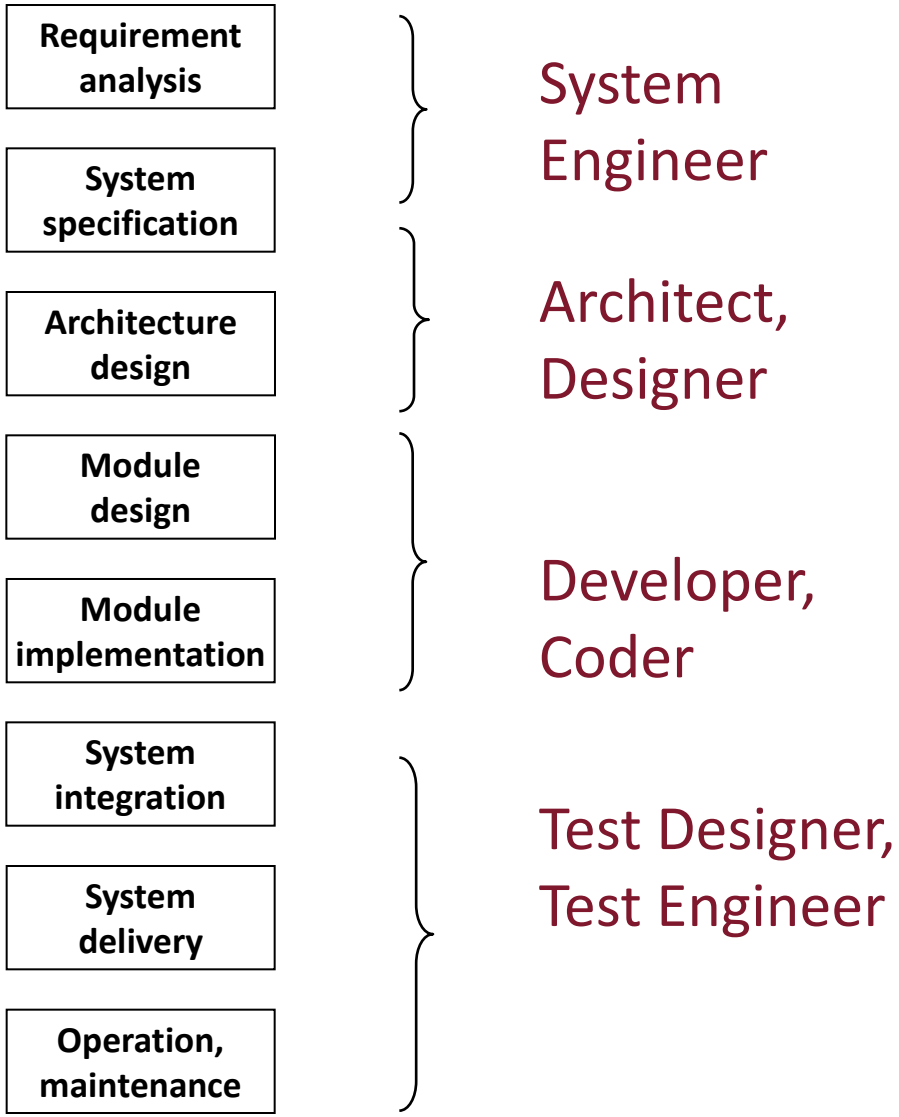
Overview of the techniques of software V&V

What are the typical techniques in the development process?

Who is concerned by V&V?

System Engineer	• Verifying requirement specification
Architect, Designer	• Modeling and verifying designs
Developer, Coder	• Verifying source code, unit testing
Test Designer	• Designing test processes and techniques
Test Engineer	• Test automation, integration and system tests
Safety Engineer	• Assessment w.r.t. development standards

What are the typical development steps?

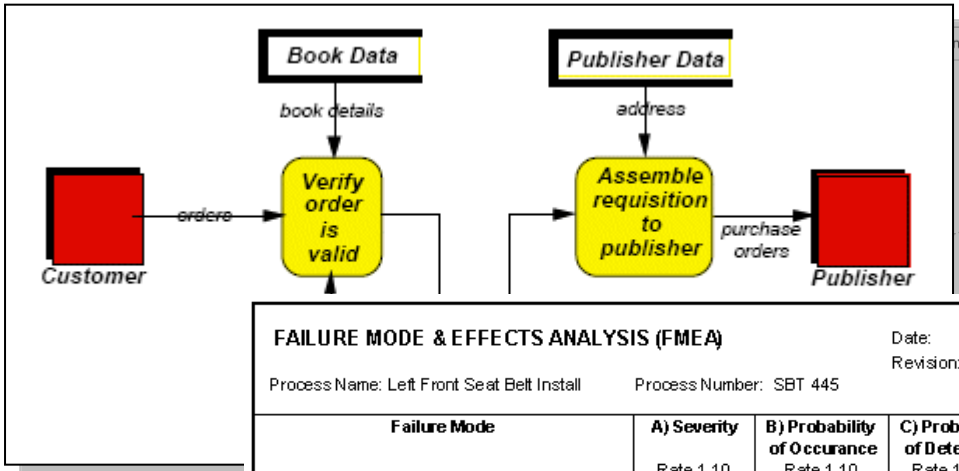


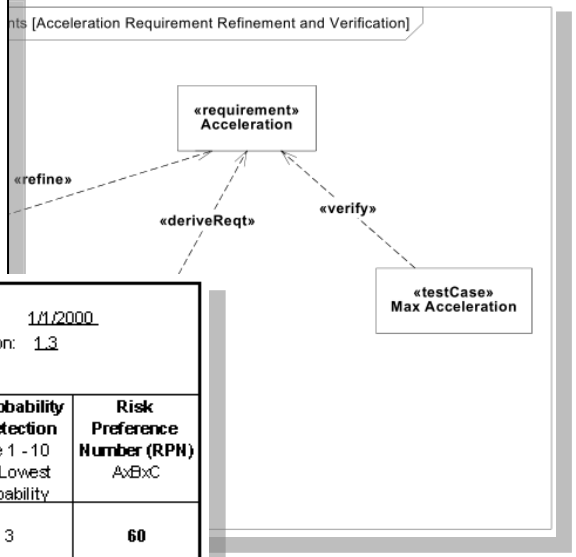
Schedule and sequencing depends on the lifecycle model (see later)

Requirement analysis

- **Requirement analysis**
- **System specification**
- **Architecture design**
- **Module design**
- **Module implementation**
- **System integration**
- **System delivery**
- **Operation, maintenance**

Task	V&V criteria	V&V technique
Defining functions, actors, use cases	- Risks - Criticality	- Checklists - Failure mode and effects analysis





FAILURE MODE & EFFECTS ANALYSIS (FMEA)

Date: 1/1/2000
Revision: 1.3

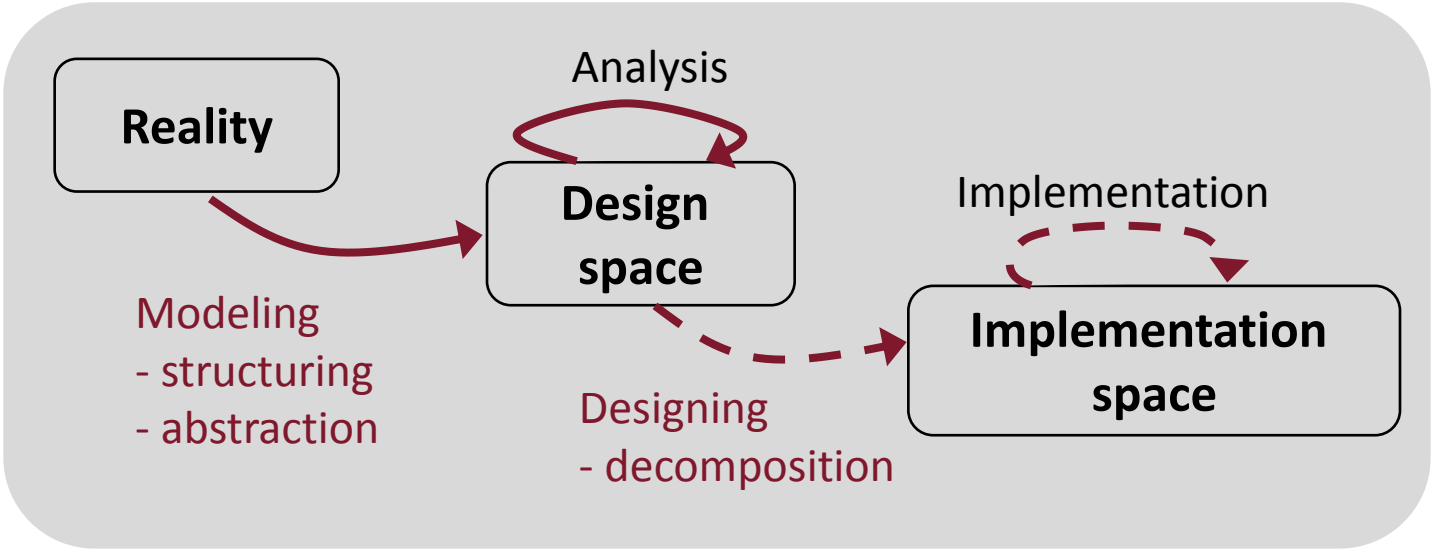
Process Name: Left Front Seat Belt Install Process Number: SBT 445

Failure Mode	A) Severity Rate 1-10 10 = Most Severe	B) Probability of Occurance Rate 1-10 10 = Highest Probability	C) Probability of Detection Rate 1 - 10 10 = Lowest Probability	Risk Preference Number (RPN) AxBxC
1) Select Wrong Color Seat Belt	5	4	3	**60**
2) Seat Belt Bolt Not Fully Tightened	9	2	8	**144**
3) Trim Cover Clip Misaligned	2	3	4	**24**

System specification

- Requirement analysis
- **System specification**
- Architecture design
- Module design
- Module implementation
- System integration
- System delivery
- Operation, maintenance

Task	V&V criteria	V&V technique
Defining functional and non-functional requirements	- Completeness - Consistency - Verifiability - Feasibility	- Reviews - Static analysis - Simulation



System specification

- Requirement analysis
- **System specification**
- Architecture design
- Module design
- Module implementation
- System integration
- System delivery
- Operation, maintenance

Task	V&V criteria	V&V technique
Defining functional and non-functional requirements	- Completeness - Consistency - Verifiability - Feasibility	- Reviews - St[illegible] analysis [illegible]tion

Review:

1. Assembling a checklist
2. Presentation by the developer
3. Answering the questions of reviewers
4. Discussion, preparing the review report

Types of peer review:

- Round robin: Different leader for reach module
- Walkthrough: The developer “guides” the reviewers
- Inspection: Based on a (formal) checklist

System specification

- Requirement analysis
- **System specification**
- Architecture design
- Module design
- Module implementation
- System integration
- System delivery
- Operation, maintenance

Task	V&V criteria	V&V technique
Defining functional and non-functional requirements	- Completeness - Consistency - Verifiability - Feasibility	- Reviews - Static analysis - Sim[illegible]ion

Example: Specification of an access control system (in Event-B):

Persons:	$prs \neq 0,\ p \in prs$	(set)
Buildings:	$bld \neq 0,\ b \in bld$	(set)
Authorization:	$aut \in prs \leftrightarrow bld$	(binary relation)
Situation:	$sit \in prs \rightarrow bld$	(complete function)
Invariant:	$sit \subseteq aut$	

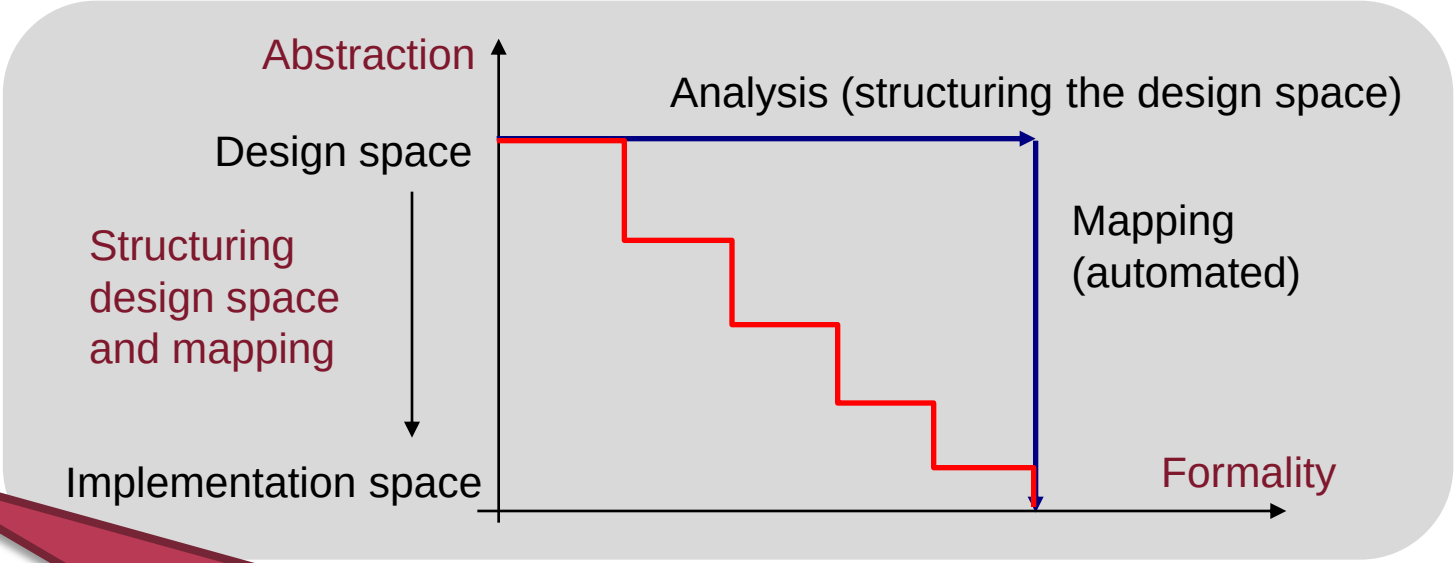
An event (change of situation):

$$pass = ANY\ p,b\ WHERE\ (p,b) \in aut \wedge sit(p) \neq b\ THEN\ sit(p) := b\ END$$

Automated analysis is possible: Checking invariant for each event

Architecture design

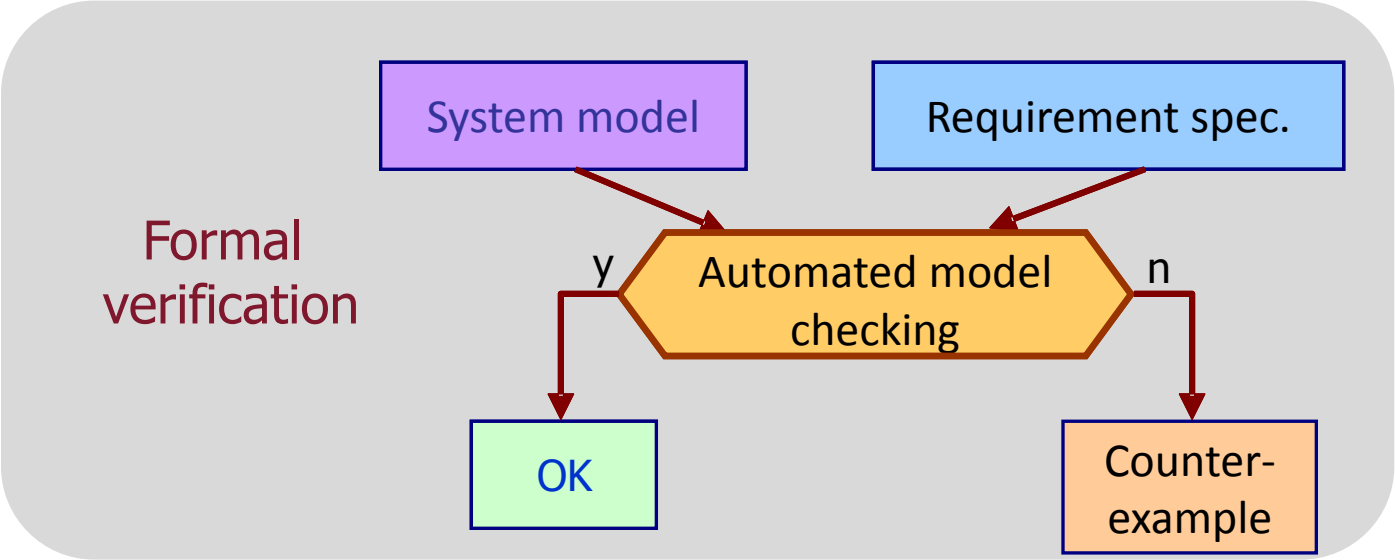
- Requirement analysis
- System specification
- **Architecture design**
- Module design
- Module implementation
- System integration
- System delivery
- Operation, maintenance



Task	V&V criteria	V&V technique
- Decomposing modules - HW-SW co-design - Designing communication	- Function coverage - Conformance of interfaces - Non-functional properties	- Static analysis - Simulation - Performance, dependability, security analysis

Module design (detailed design)

- Requirement analysis
- System specification
- Architecture design
- **Module design**
- Module implementation
- System integration
- System delivery
- Operation, maintenance



Task	V&V criteria	V&V technique
- Designing detailed behavior (data structures, algorithms)	- Correctness of critical internal algorithms and protocols	- Static analysis - Simulation - Formal verification - Rapid prototyping

Module implementation

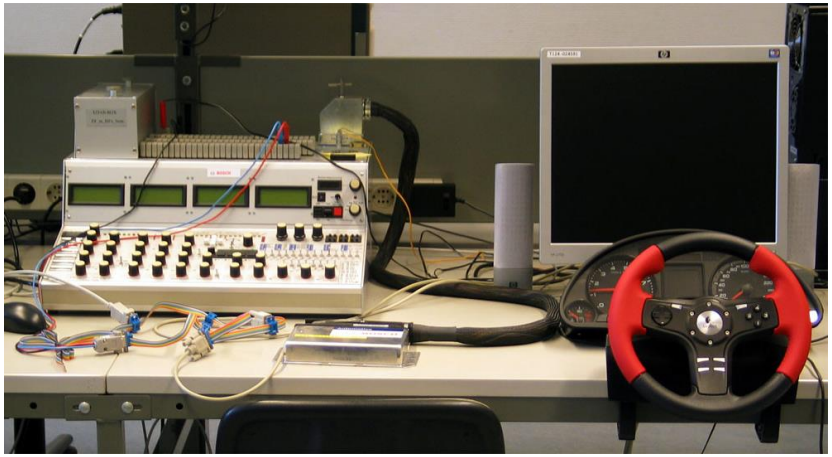
- Requirement analysis
- System specification
- Architecture design
- Module design
- **Module implementation**
- System integration
- System delivery
- Operation, maintenance

Task	V&V criteria	V&V technique
- Software implementation	Code is - Safe - Verifiable - Maintainable	- Checking coding conventions - Code reviews - Static code analysis
- Verifying module implementation	- Conformance to module designs	- Unit testing - Regression testing

System integration

- Requirement analysis
- System specification
- Architecture design
- Module design
- Module implementation
- **System integration**
- System delivery
- Operation, maintenance

Task	V&V criteria	V&V technique
- Integrating modules - Integrating SW with HW	- Conformance of integrated behavior - Correct communication	- Integration testing (incremental)



System delivery and deployment

- Requirement analysis
- System specification
- Architecture design
- Module design
- Module implementation
- System integration
- **System delivery**
- Operation, maintenance

Task	V&V criteria	V&V technique
- Assembling complete system	- Conformance to system specification	- System testing - Measurements, monitoring
- Fulfilling user expectations	- Conformance to requirements and expectations	- Validation testing - Acceptance testing - Alfa/beta testing

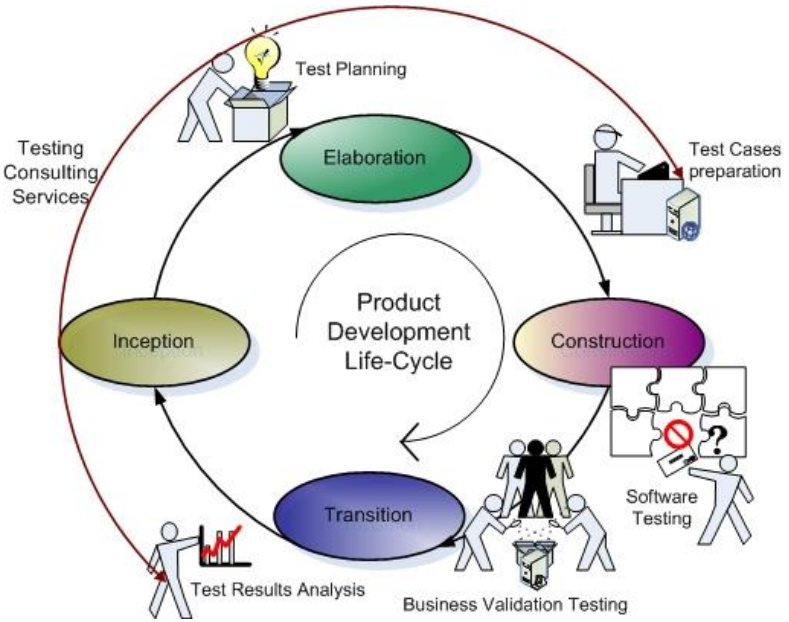
Operation and maintenance

- Requirement analysis
- System specification
- Architecture design
- Module design
- Module implementation
- System integration
- System delivery
- **Operation, maintenance**

Tasks during operation and maintenance:

- Failure logging and analysis (for failure prediction)
- V&V of modifications depending on the affected life cycle phases

"Mini-lifecycle" for each modification



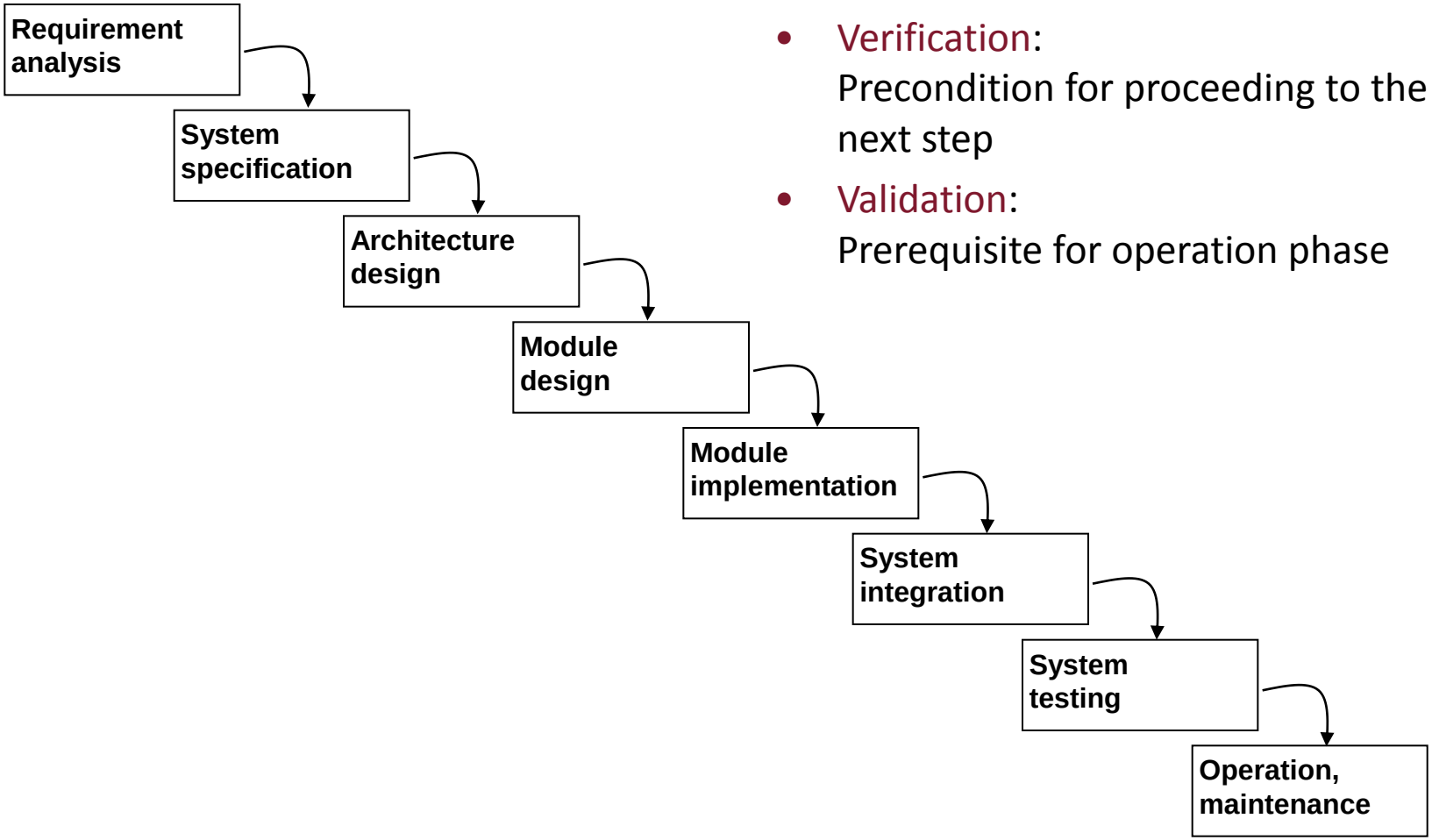
Development life cycle models

What is the role of V&V in the different life cycle models?

Development life cycle models

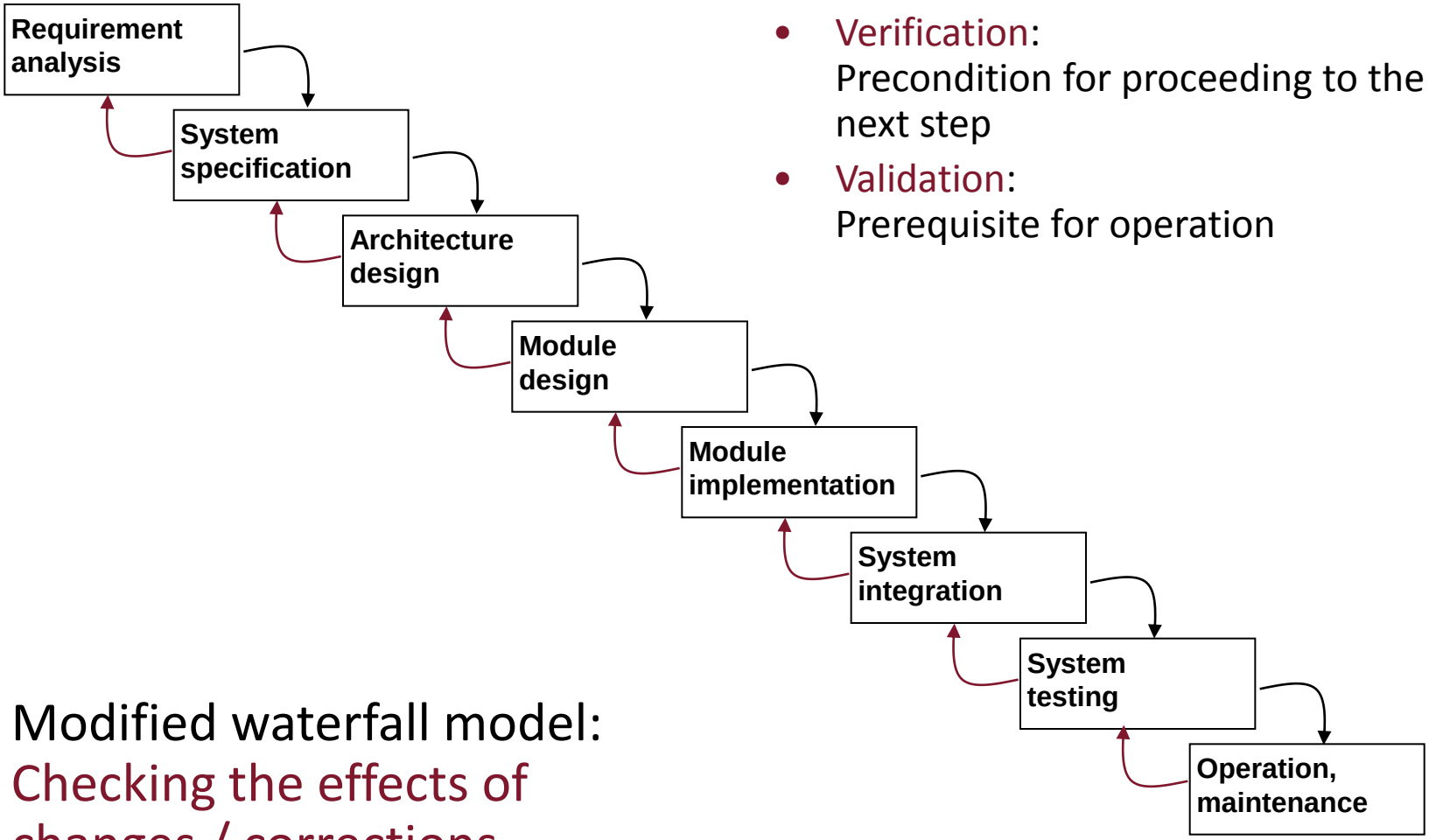
- The role of life cycle models
  - Handling the complexity of development
    - Dividing the development into phases, milestones
    - Basis for distributed / concurrent design and then integration
  - Change management
    - Handling the effects of requirement changes, modification and maintenance
    - Introduction of new methods and tools
- Generic models of software development:
  - Sequential development: Waterfall and V-model
  - Evolutionary development: Rapid application development
  - Iterative development: Spiral model
  - Model based (formal) development: 4G model
  - Iterative-incremental development: Unified Process

1. Waterfall model



- Verification:
  Precondition for proceeding to the next step
- Validation:
  Prerequisite for operation phase

1. Waterfall model

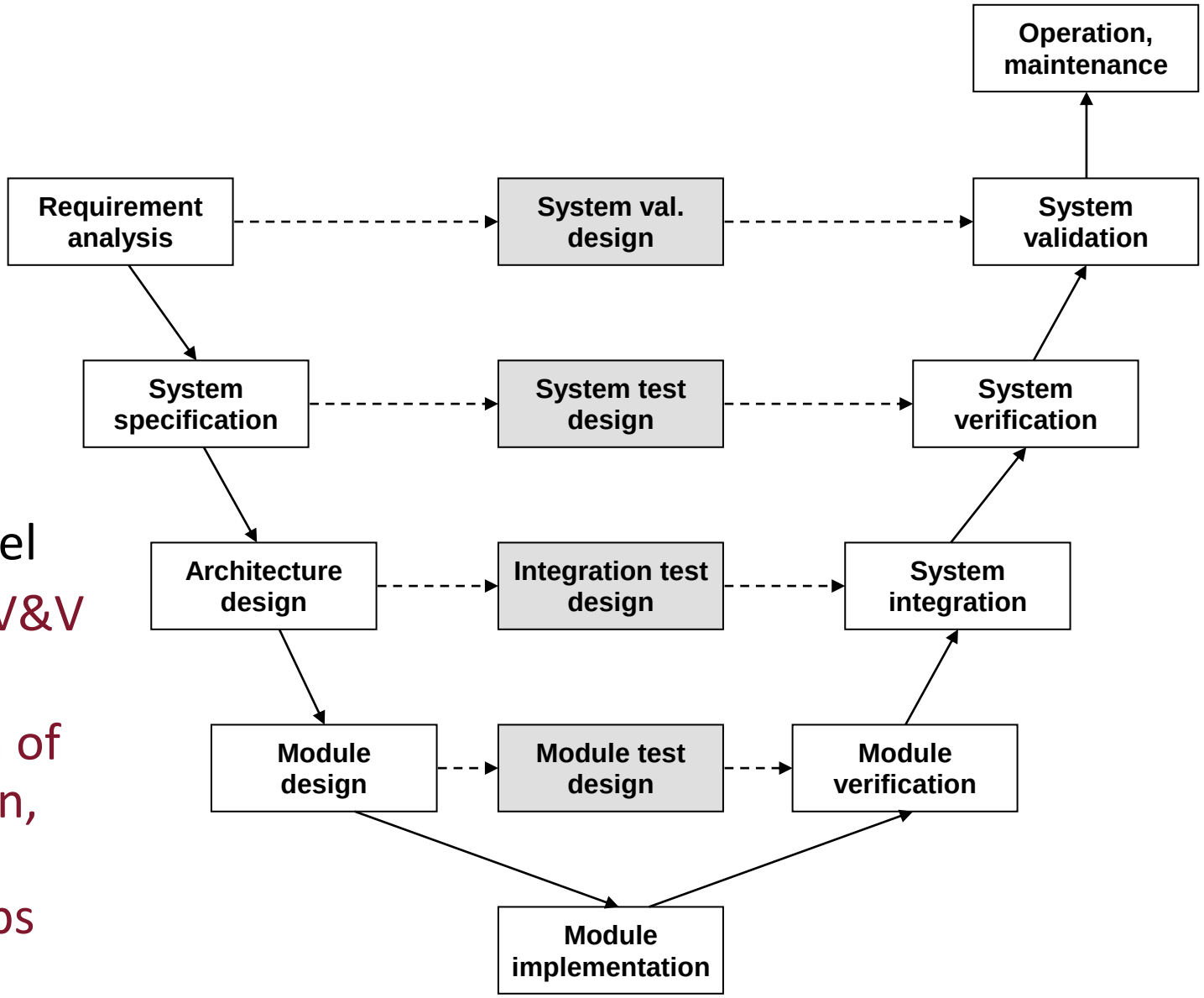


- Verification:
  Precondition for proceeding to the next step
- Validation:
  Prerequisite for operation

- Modified waterfall model:
  Checking the effects of changes / corrections
  (e.g., regression testing)

2. The V-model

- Based on the waterfall model
- Well-defined V&V for each step
- Precise design of the verification, testing and validation steps



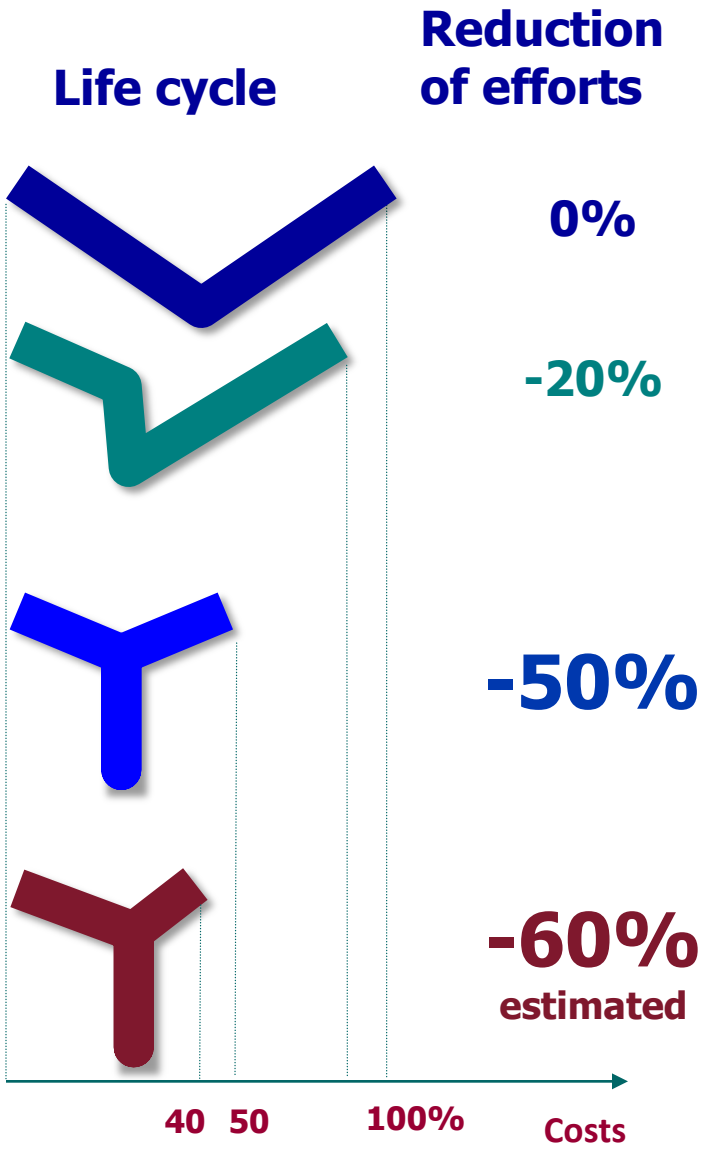
Model based design: From V to Y model

Manual coding

"Common" automated code generator is used

Certified automated code generator is used

Design using formal methods and tools



Classic method: Cleanroom Software Engineering

- Origin:
  - IBM proposal (1980s)
  - US military developments (1990s)
- Goal:
  - Verification based on formal models
  - Fault avoidance instead of removal
- Principles:
  - Use and verification of formal models
  - Incremental development with quality control (step-by-step increase of complexity)
  - Statistical testing based on formal models
    - Selecting the representative trajectories
    - Manual validation of modeling

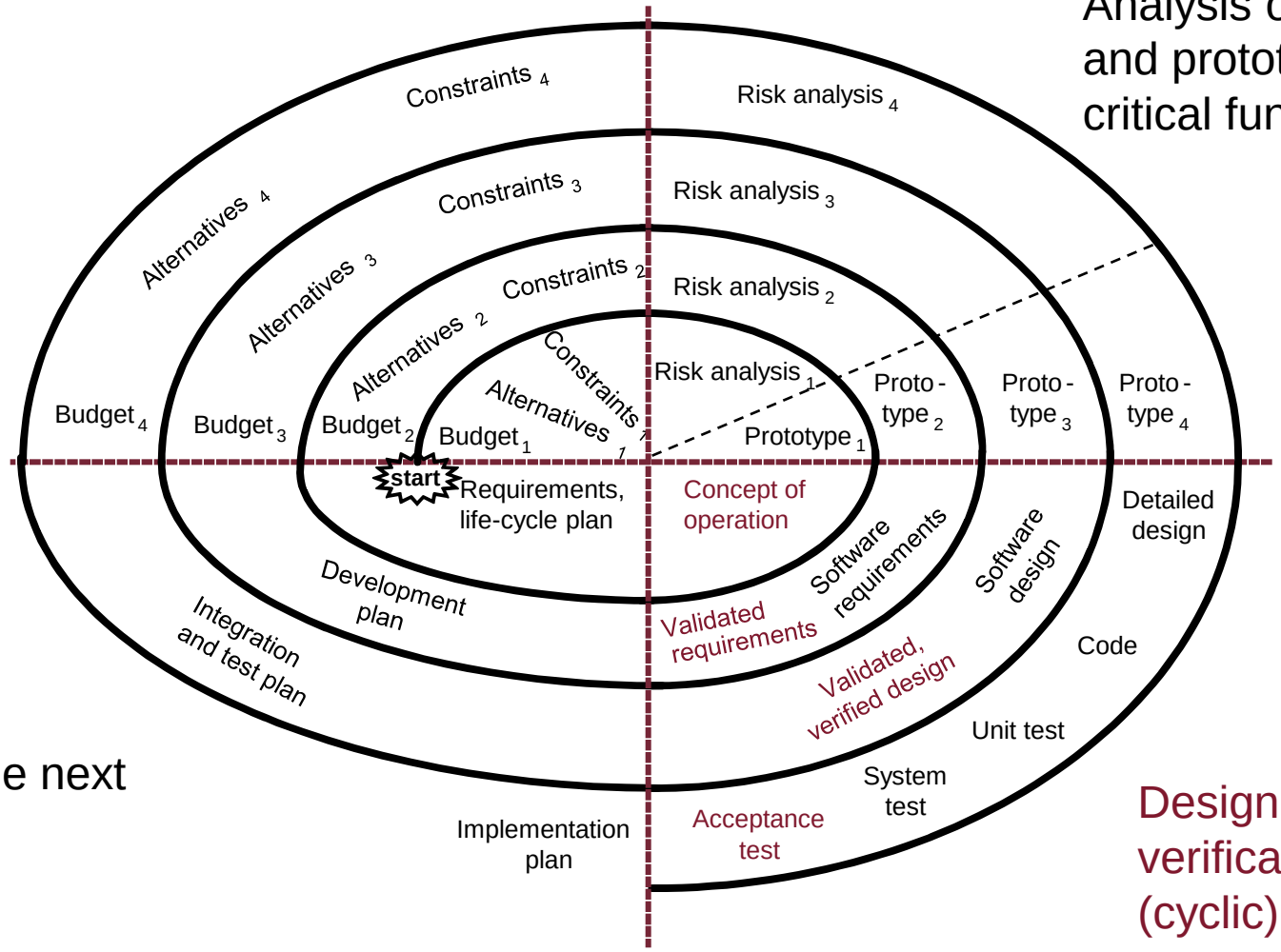
3. Evolutionary development (RAD)

- Rapid development of an initial implementation then refinement through several versions, based on user feedback
  - Explorative development: Discussed with users
    - First version: Based on known requirements
  - Rapid prototypes for the critical functions
    - Validation using the prototype, re-working the prototype
  - Can be applied in case of incompletely specified systems
- V&V characteristics:
  - Increased role of prototype testing
  - Increased role of integration testing
    - Adding new functions
  - Regression testing after modifications
    - Existing functions remain correct

4. Iterative development: Spiral model

Goals, alternatives, constraints

Analysis of risks and prototype for critical functions

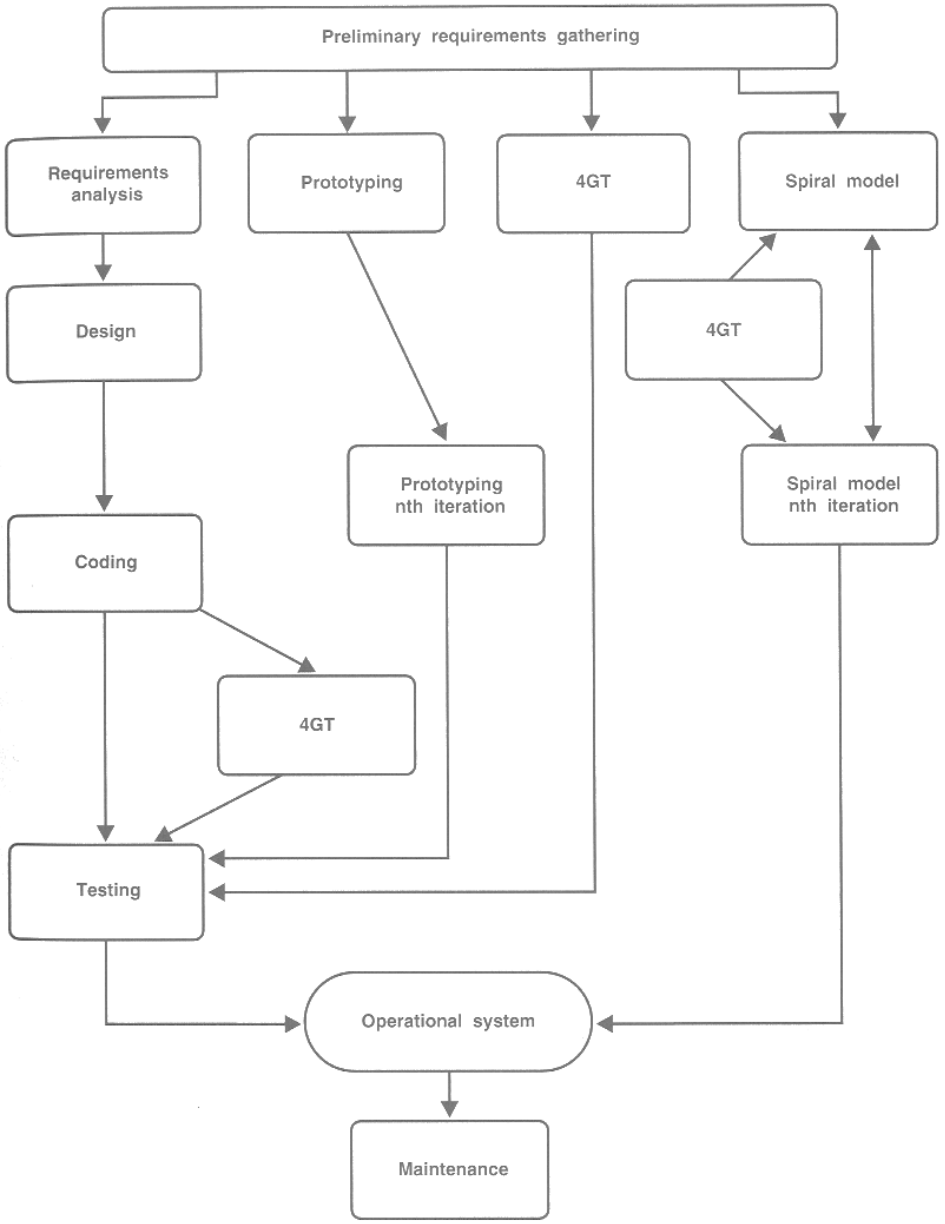


Planning the next phase

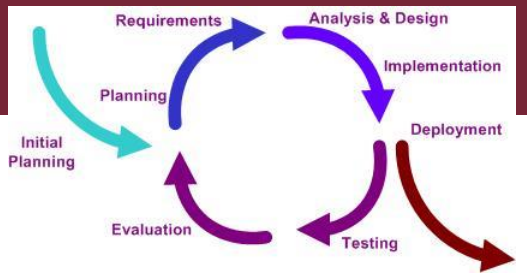
Design and verification (cyclic)

5. The “4G” model

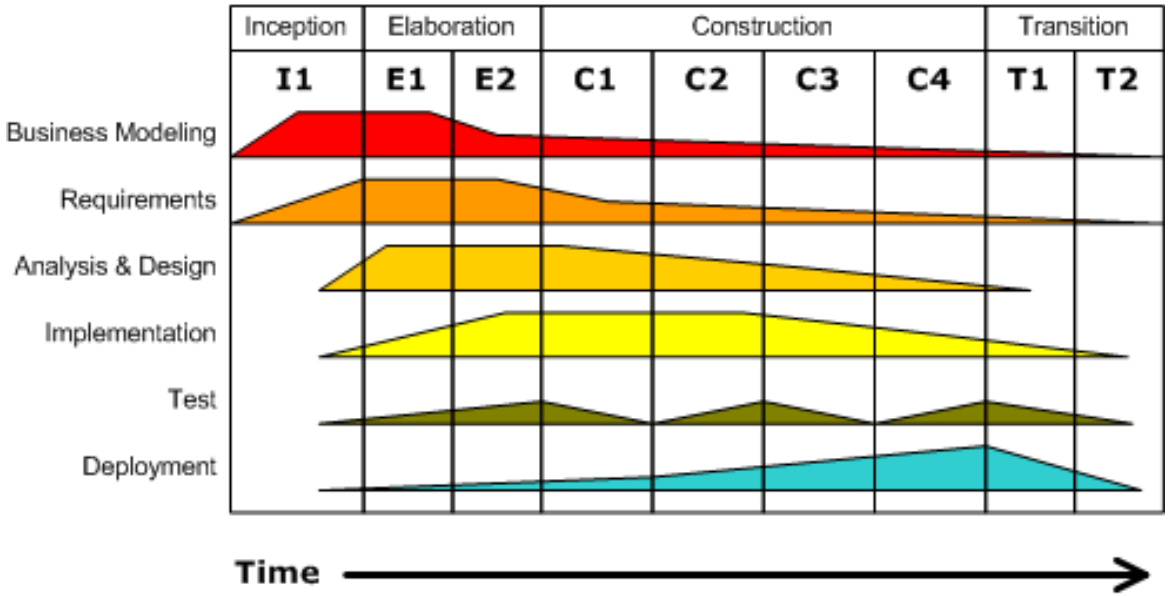
- Integration:
  - Well-specified requirements: “Traditional” development
  - Incompletely specified requirements: Rapid prototype development
  - Formally specified requirements: Model based development (with CASE tools), property preserving refinement
  - Iterative design
- Model based verification



6. Unified Process



- Incremental and iterative
  - Phases divided into iterations (bound in time)
  - Each iteration is a complete (mini) development cycle
  - Different focus of verification in each phase
    - Integration and regression testing is important



7. Agile software development

- Extreme Programming
  - Short iterations, focusing on operational code, regular (daily) integration and status tracking (developers, users)
    - Build frameworks
  - ”Test first programming” concept:
    - Functional tests based on “story card”
    - Testing after each modification (new function)
- Test Driven Development
  - Incremental, steps for each new function:
    1. Writing test for the new function (test will fail)
    2. Coding (for successful test)
    3. Refactoring of the code with re-testing
  - Uses automated unit testing

The role of development standards

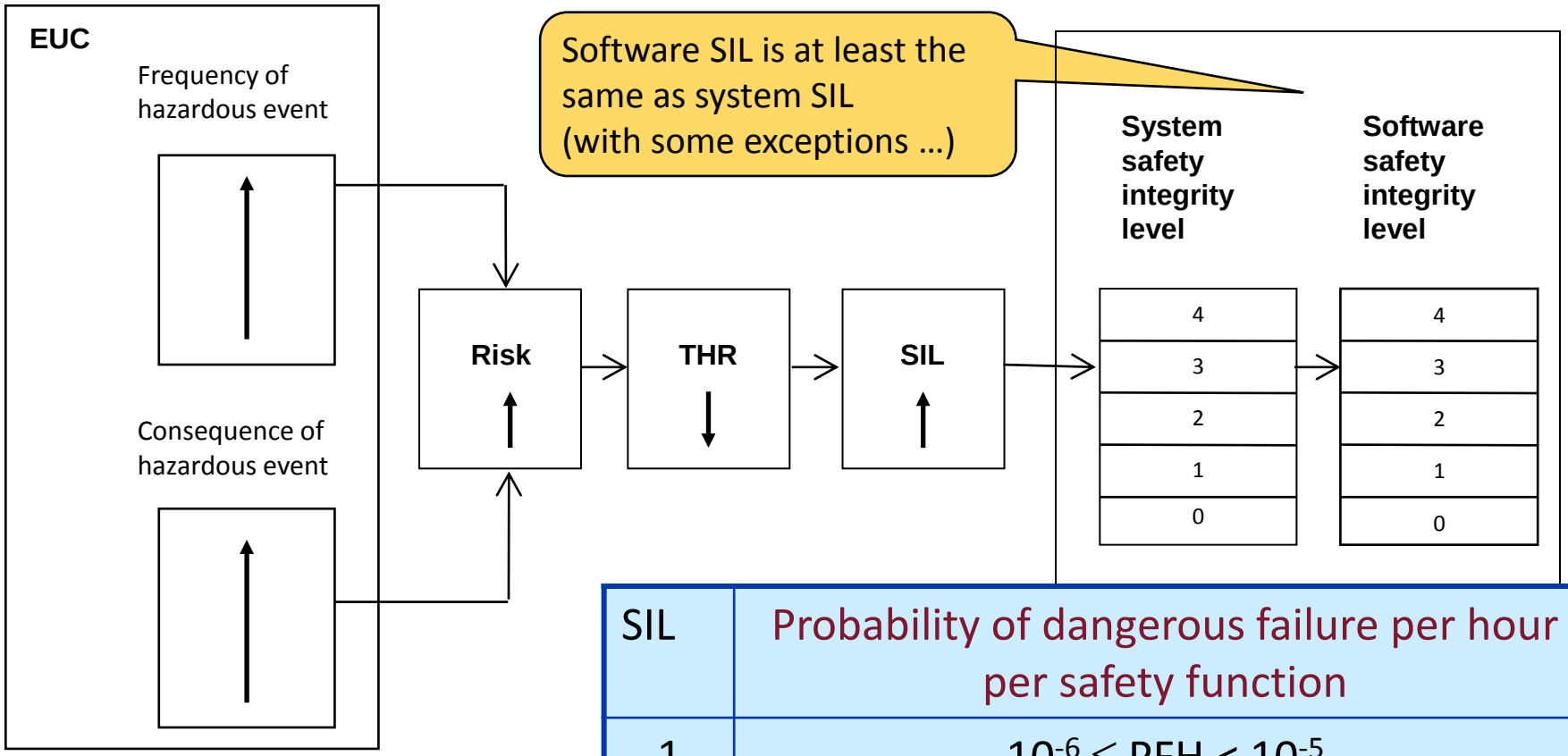
How systematic V&V is realized?

Use of standards: Safety critical systems

- Standards for development
  - IEC 61508: Functional safety in electrical / programmable electronic systems
  - EN 50128: Railway control software
  - ISO 26262: Automotive software
  - DO 178B: Airborne software
- Specification of safety functions
  - Functionality: Intended to achieve or maintain a safe state
  - Safety integrity: Probability of a safety-related system satisfactorily performing the required safety functions (under all stated conditions and within a stated period of time)
- Safety integrity levels
  - Safety integrity assignment to functions: Based on risk analysis (of failures)
    - Continuous operation: Tolerable rate of failures
    - On demand operation: Tolerable probability of failure
  - Tolerable Hazard Rate:
    - Categories based on numerical ranges: SIL 1, 2, 3, 4

Determining SIL

- Hazard identification and risk analysis -> Target failure measure



Risk analysis
-> Function THR
-> Function SIL
-> (Sub)system SIL

SIL	Probability of dangerous failure per hour per safety function
1	$10^{-6} \leq PFH < 10^{-5}$
2	$10^{-7} \leq PFH < 10^{-6}$
3	$10^{-8} \leq PFH < 10^{-7}$
4	$10^{-9} \leq PFH < 10^{-8}$

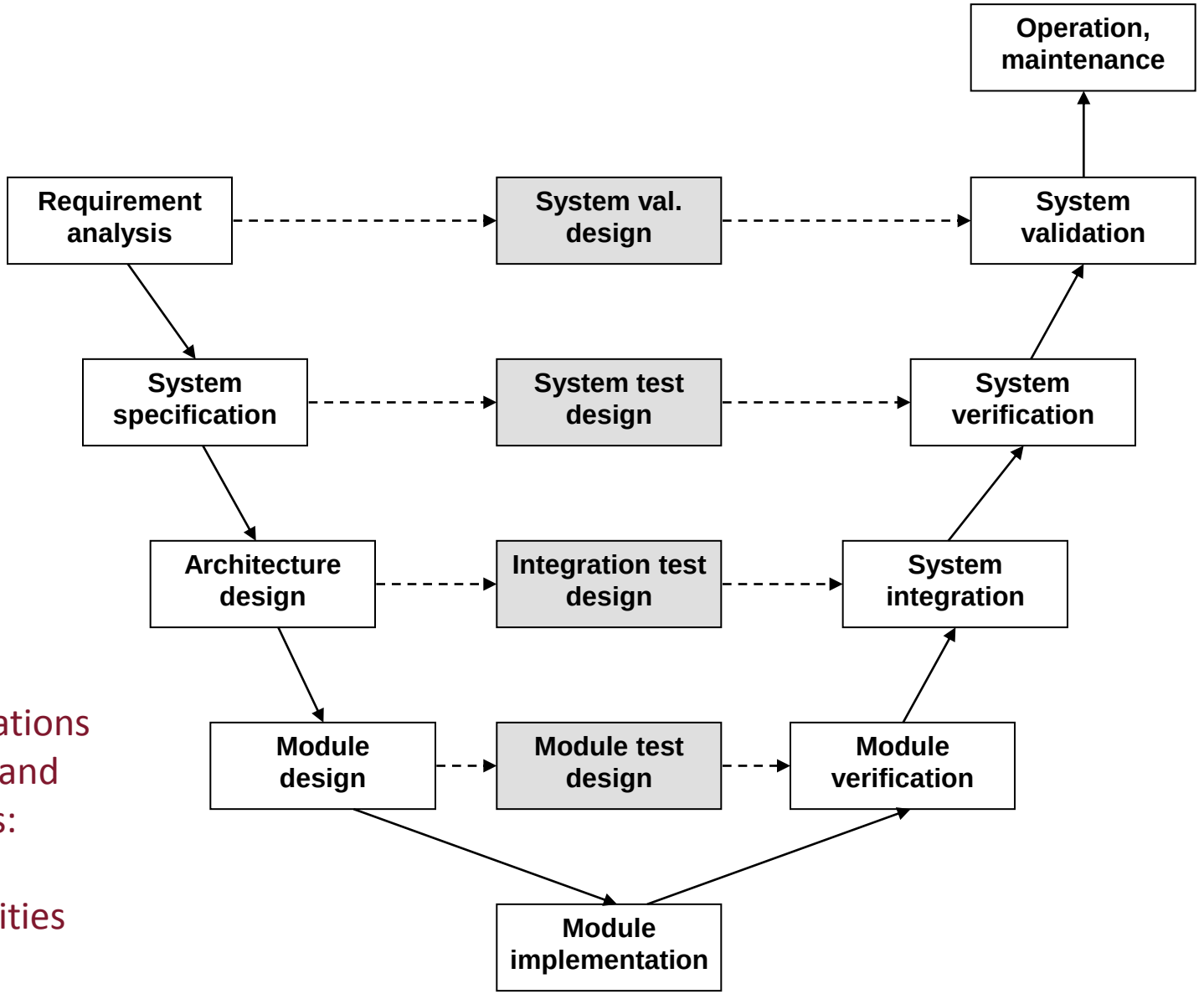
Demonstrating SIL requirements

- Safety case:
  - Documented demonstration that the product complies with the specified safety requirements (functional + safety integrity)
  - Evidence is based on verification and validation
- Random failure integrity (for hardware):
  - Quantitative approach: Based on statistics, experiments
  - Computation of system failure rate using component fault rate data from reliability handbooks
- Systematic failure integrity (for software):
  - Quantitative approach is not possible (missing reliability data)
  - Qualitative approach: Prescribing rigor in the development
    1. Well-defined development process (life cycle)
    2. Mandatory / recommended techniques and measures
    3. Organizational structure: Independence of persons / roles
    4. Precise documentation

1. The development process (life cycle)

- Typically a static development process (e.g., V-model)
  - Well-defined steps
  - Requirements and environment known in advance
- Strict rules for proceeding to the next step:
  Important to verify the results of development
  - High costs of late corrections (esp. during operation)
  - The risk caused by remaining failures may be high
- Characteristics:
  - Evidences collected for the safety case
  - Assessment (independent review)
  - Certification and supervision by safety authorities, based on the development standard

Typical life-cycle model: V-model



Well-defined relations between design and verification steps: Planning of the verification activities

2. Techniques and measures

- Goal: Preventing the introduction of systematic faults and controlling the residual faults
- SIL determines the set of techniques to be applied as
  - M: Mandatory
  - HR: Highly recommended (rationale behind not using it should be detailed and agreed with the assessor)
  - R: Recommended
  - ---: No recommendation for or against being used
  - NR: Not recommended
- Combinations of techniques is allowed
  - E.g., alternative or equivalent techniques are marked
- Hierarchy of techniques (references to sub-tables)

Example: Testing techniques (EN 50128)

- Software design and implementation:

TECHNIQUE/MEASURE		Ref	SWS ILO	SWS IL1	SWS IL2	SWS IL3	SWS IL4
14.	Functional/ Black-box Testing	D.3	HR	HR	HR	M	M
15.	Performance Testing	D.6	-	HR	HR	HR	HR
16.	Interface Testing	B.37	HR	HR	HR	HR	HR

- Functional / black box testing (D3):

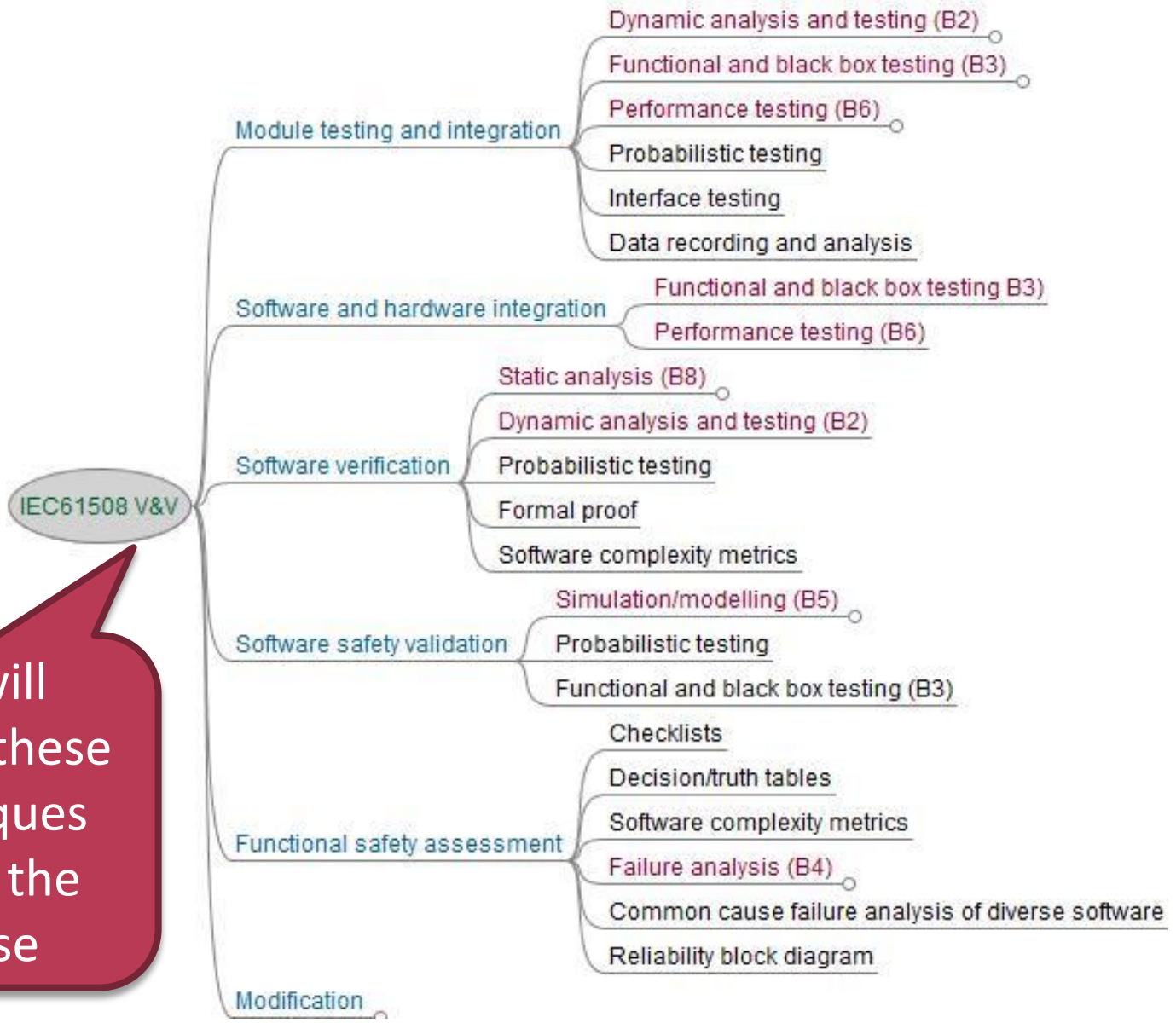
1.	Test Case Execution from Cause Consequence Diagrams	B.6	-	-	-	R	R
2.	Prototyping/Animation	B.49	-	-	-	R	R
3.	Boundary Value Analysis	B.4	R	HR	HR	HR	HR
4.	Equivalence Classes and Input Partition Testing	B.19	R	HR	HR	HR	HR
5.	Process Simulation	B.48	R	R	R	R	R

Example: Testing techniques (EN 50128)

- Performance testing (D6):

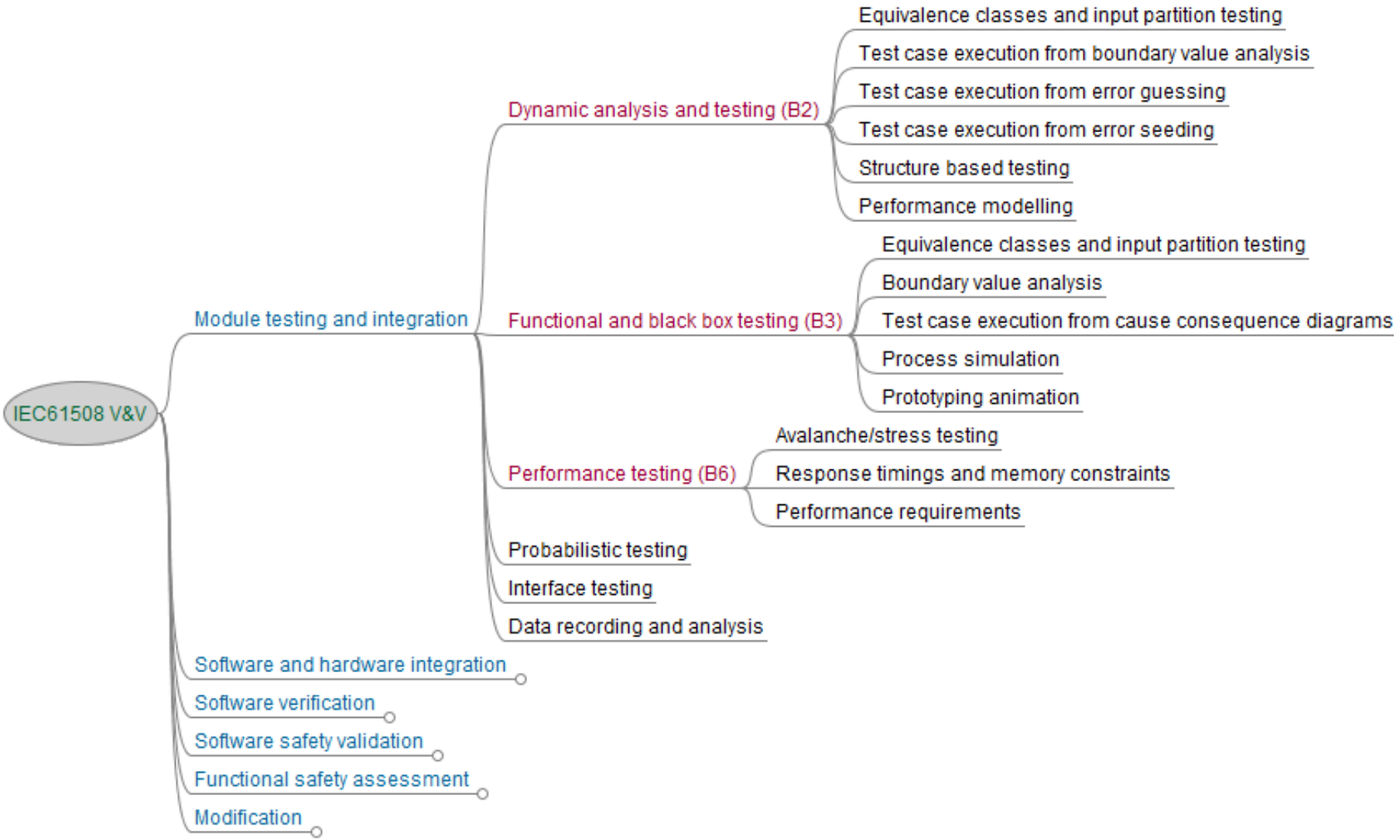
TECHNIQUE/MEASURE		Ref	SWS ILO	SWS IL1	SWS IL2	SWS IL3	SWS IL4
1.	Avalanche/Stress Testing	B.3	-	R	R	HR	HR
2.	Response Timing and Memory Constraints	B.52	-	HR	HR	HR	HR
3.	Performance Requirements	B.46	-	HR	HR	HR	HR

Example: Hierarchy of V&V methods (IEC 61508)

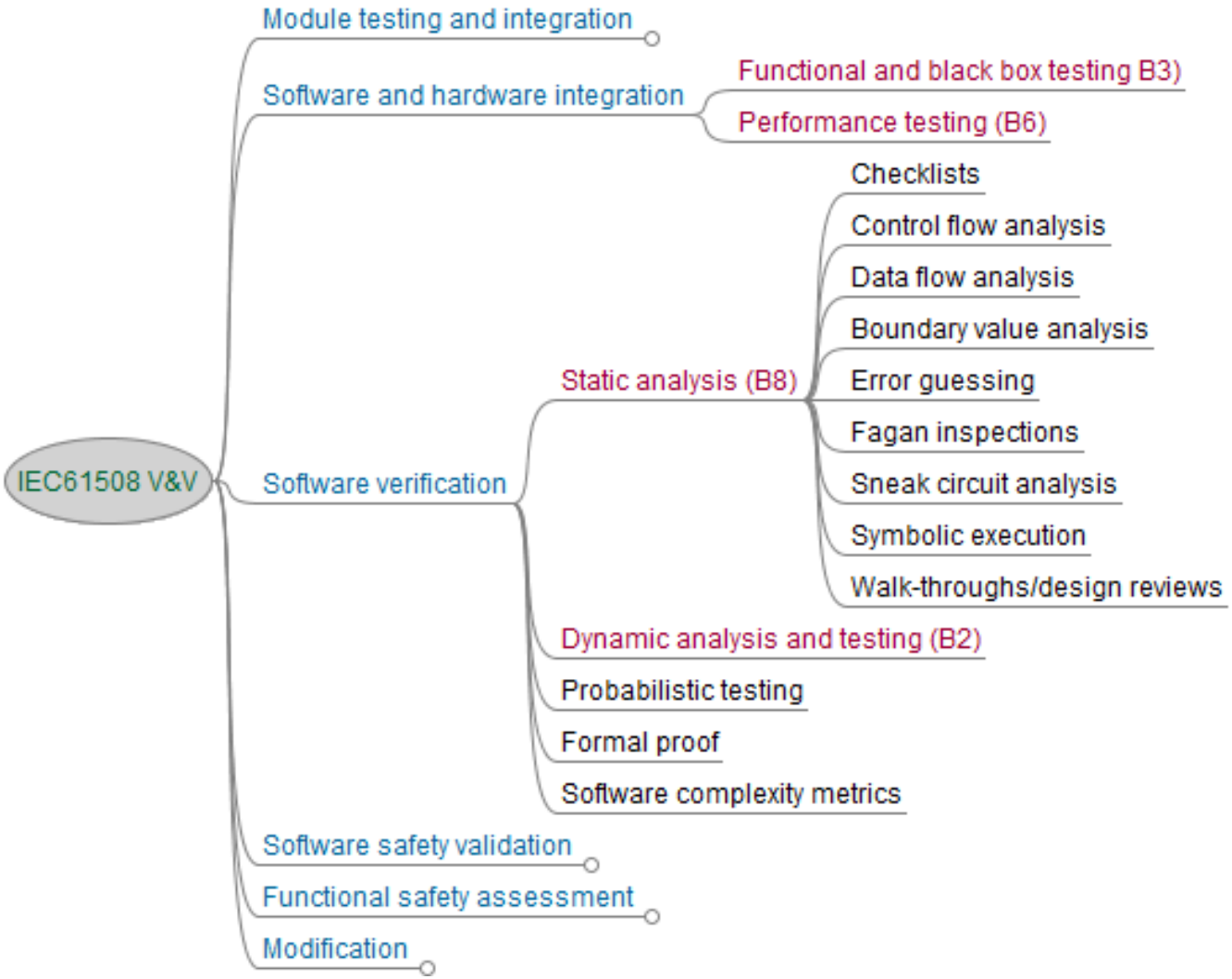


We will discuss these techniques during the course

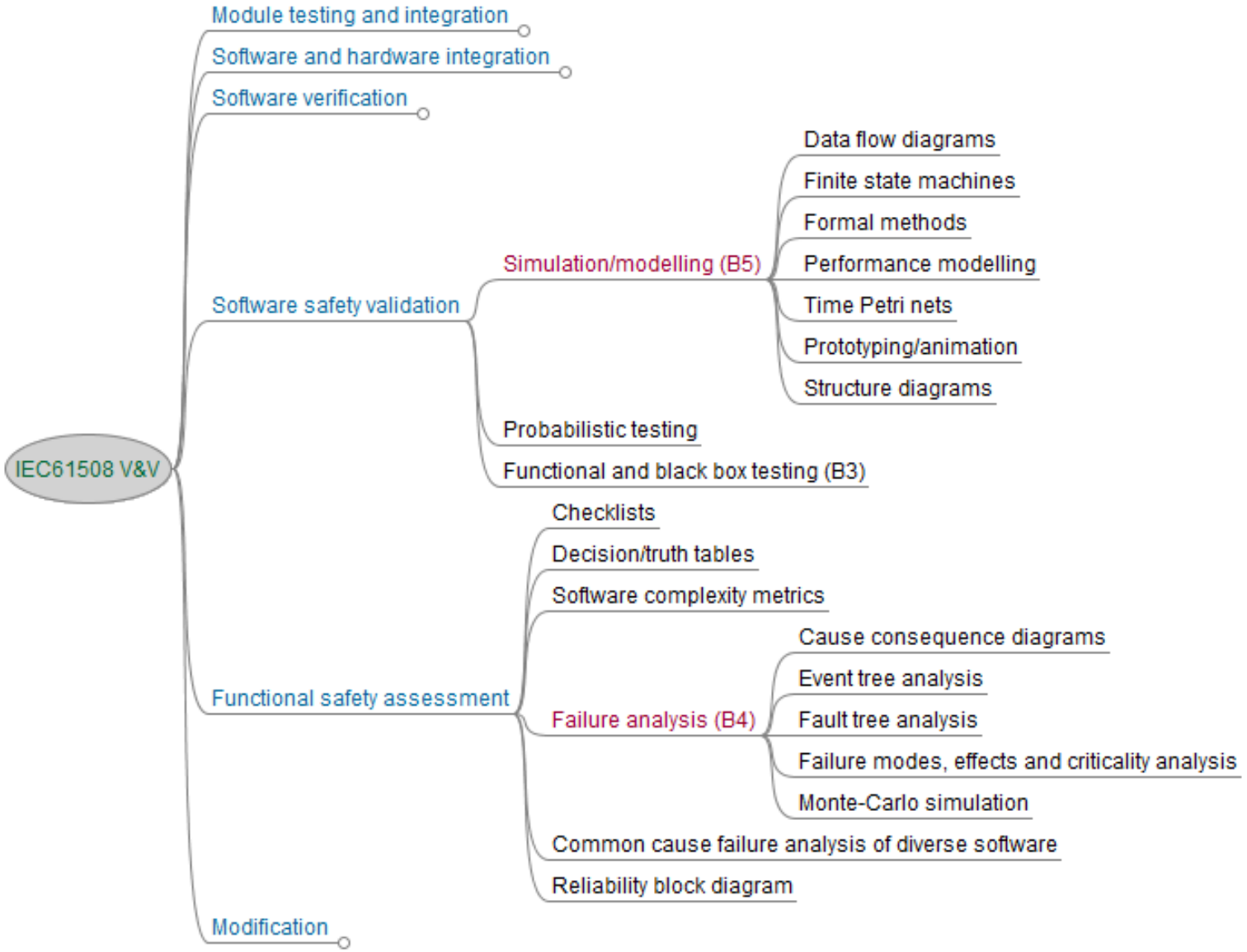
Example: Hierarchy of V&V methods (IEC 61508)



Example: Hierarchy of V&V methods (IEC 61508)



Example: Hierarchy of V&V methods (IEC 61508)

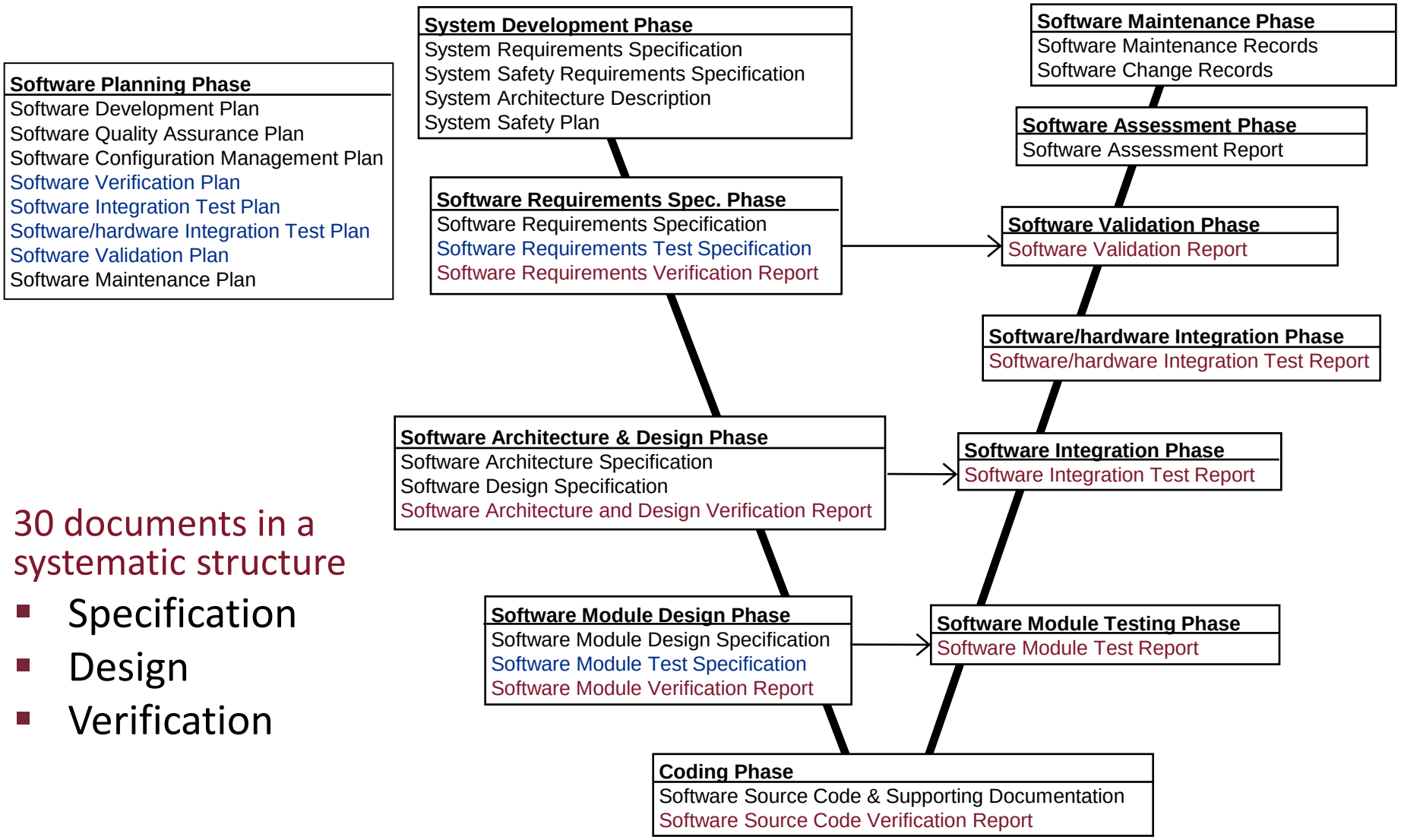


3. Precise documentation

- Type of documentation
  - Comprehensive (overall lifecycle)
    - E.g., Software Verification Plan
  - Specific (for a given lifecycle phase)
    - E.g., Software Source Code Verification Report

- Document Cross Reference Table
  - Determines documentation for a lifecycle phase
  - Determines relations among documents
- Traceability of documents is required
  - Relationship between documents is specified ("based on", "includes")
  - Terminology, references, abbreviations are consistent
- Merging documents is allowed
  - If responsible persons (authors) shall not be independent

Example: Document structure (EN50128)



30 documents in a systematic structure

- Specification
- Design
- Verification

Example: Document cross reference table (EN50128)

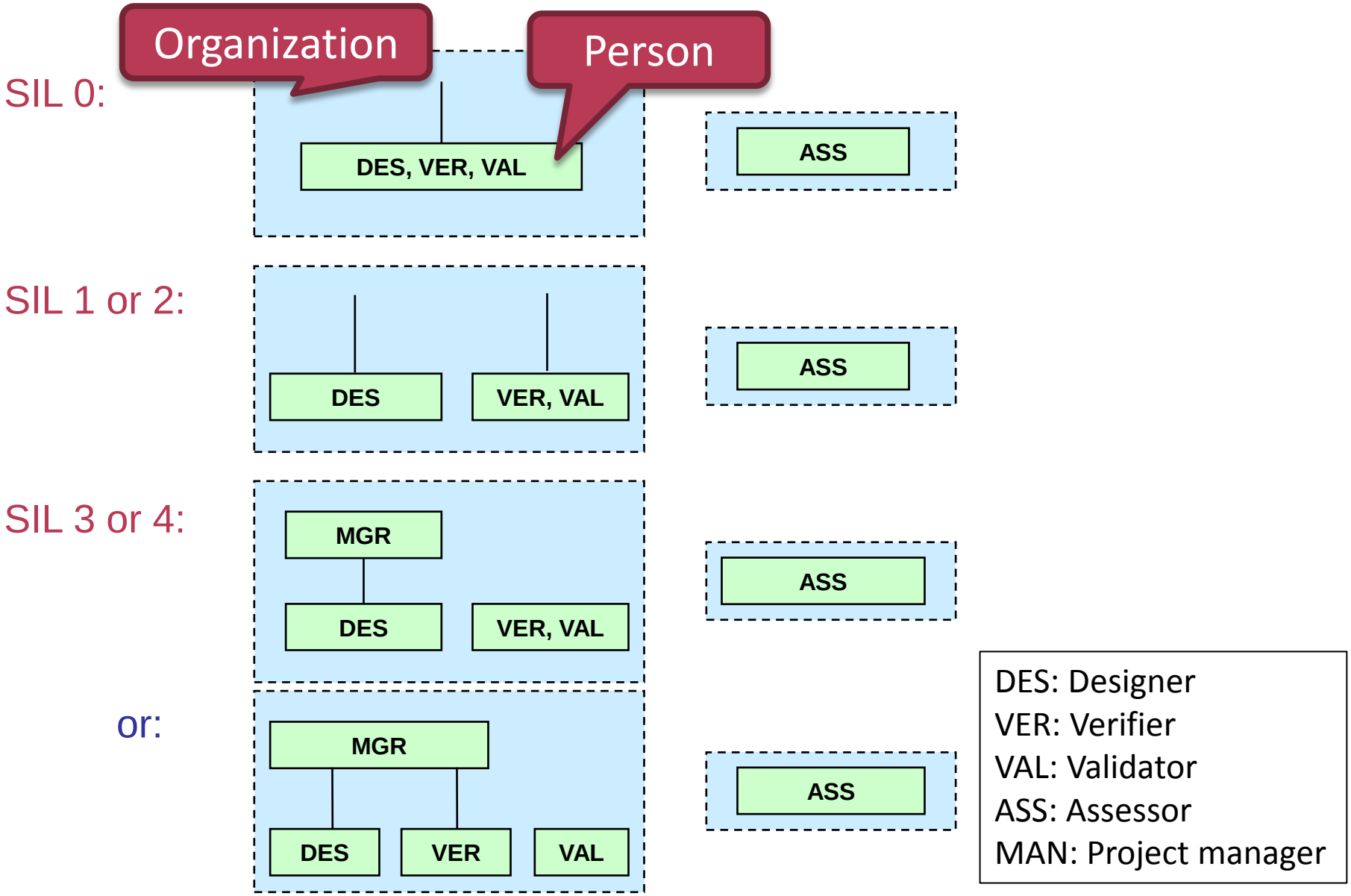
- ■ Creation of a document
- ◆ Use of a document in a given phase

clause	8	9	10	11	12	13	14	15	16	
title	SRS	SA	SDD	SVer	S/H I	SVal	Ass	Q	Ma	
PHASES *(*)=in parallel with other phases*										**DOCUMENTS**
SW REQUIREMENTS	■	◆	◆	◆	◆	◆	◆			Sw Requirements Specification
	■			◆	◆	◆	◆			Sw Requirements Test Specification
				■						Sw Requirements Verification Report
SW DESIGN		■	◆	◆	◆	◆	◆			Sw Architecture Specification
			■	◆	◆	◆	◆			Sw Design Specification
				■						Sw Arch. and Design Verification
SW MODULE DESIGN			■	◆	◆	◆	◆			Sw Module Design Specification
			■	◆	◆	◆	◆			Sw Module Test Specification
				■						Sw Module Verification Report
CODE			■	◆	◆	◆	◆			Sw Source Code
				■		◆	◆			Sw Source Code Verification Report
MODULE TESTING			■	◆						Sw Module Test Report
SW INTEGRATION				■						Sw Integration Test Report
										Data Test Report
SW/HW INTEGRATION					■					Sw/Hw Integration Test Report
VALIDATION *(*)*						■				Sw Validation Report

4. Organization and independence of roles

- Safety management
  - Quality assurance
  - Safety Organization (responsible persons)
- Competence shall be demonstrated
  - Training, experience and qualifications
- Independence of roles:
  - DES: Designer (analyst, architect, coder, unit tester)
  - VER: Verifier
  - VAL: Validator
  - ASS: Assessor
  - MAN: Project manager
  - QUA: Quality assurance personnel

Example: Responsibilities (EN 50128)



Summary

- Motivation
  - What are the quality needs regarding software and what is offered by the software industry?
  - What is the role of software verification and validation techniques?
- Overview of the techniques of software V&V
  - What are the typical techniques in the development process?
- Development life cycle models
  - What is the role of V&V in the different life cycle models?
- The role of development standards
  - How systematic V&V is realized?