

# 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**?

# 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	$10^{-6} \leq \text{PFH} < 10^{-5}$
2	$10^{-7} \leq \text{PFH} < 10^{-6}$
3	$10^{-8} \leq \text{PFH} < 10^{-7}$
4	$10^{-9} \leq \text{PFH} < 10^{-8}$

15 years lifetime:  
1 failure in case of  
750 equipment

Operation without  
failure for approx.  
11.000 years???

# Different kinds of faults

## Development phase

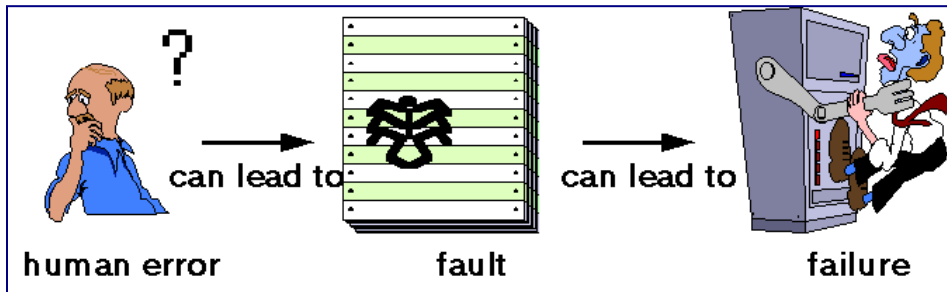
- Specification faults
- Design faults
- Implementation faults

**V&V during development**

## Operational phase

- Hardware faults
- Configuration faults
- Operator faults

**Fault tolerance (e.g. redundancy)**



# Software quality problems due to development faults

„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 code size of complex applications
  - 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?

## 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

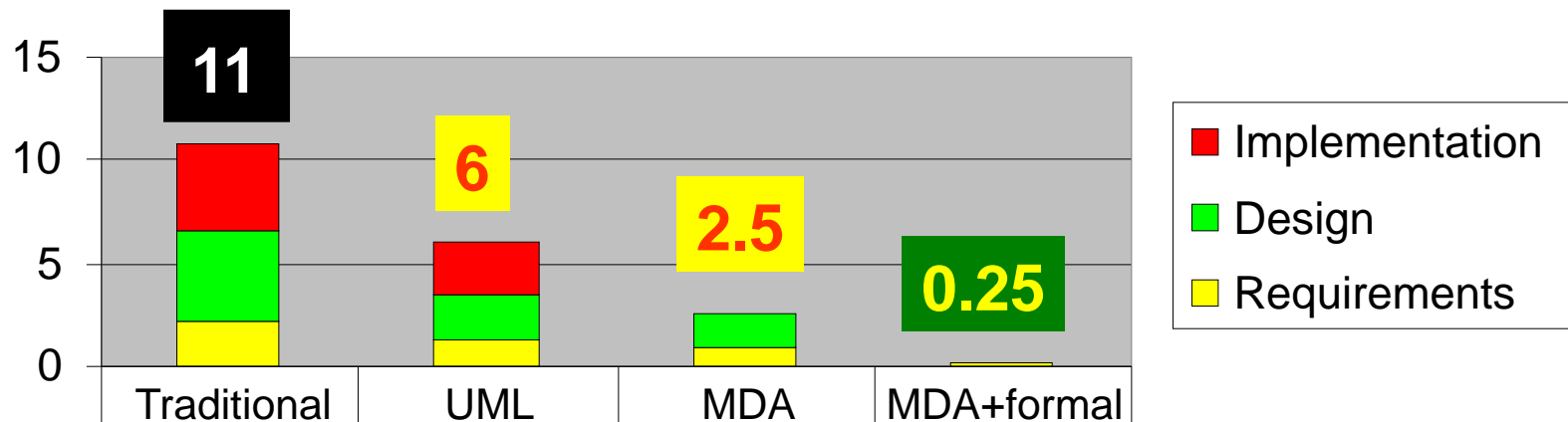


Source: K-R. Hase: „Open Proof in Railway Safety Software“, FORMS/FORMAT Conference, Braunschweig, Germany

# A study in Hungary

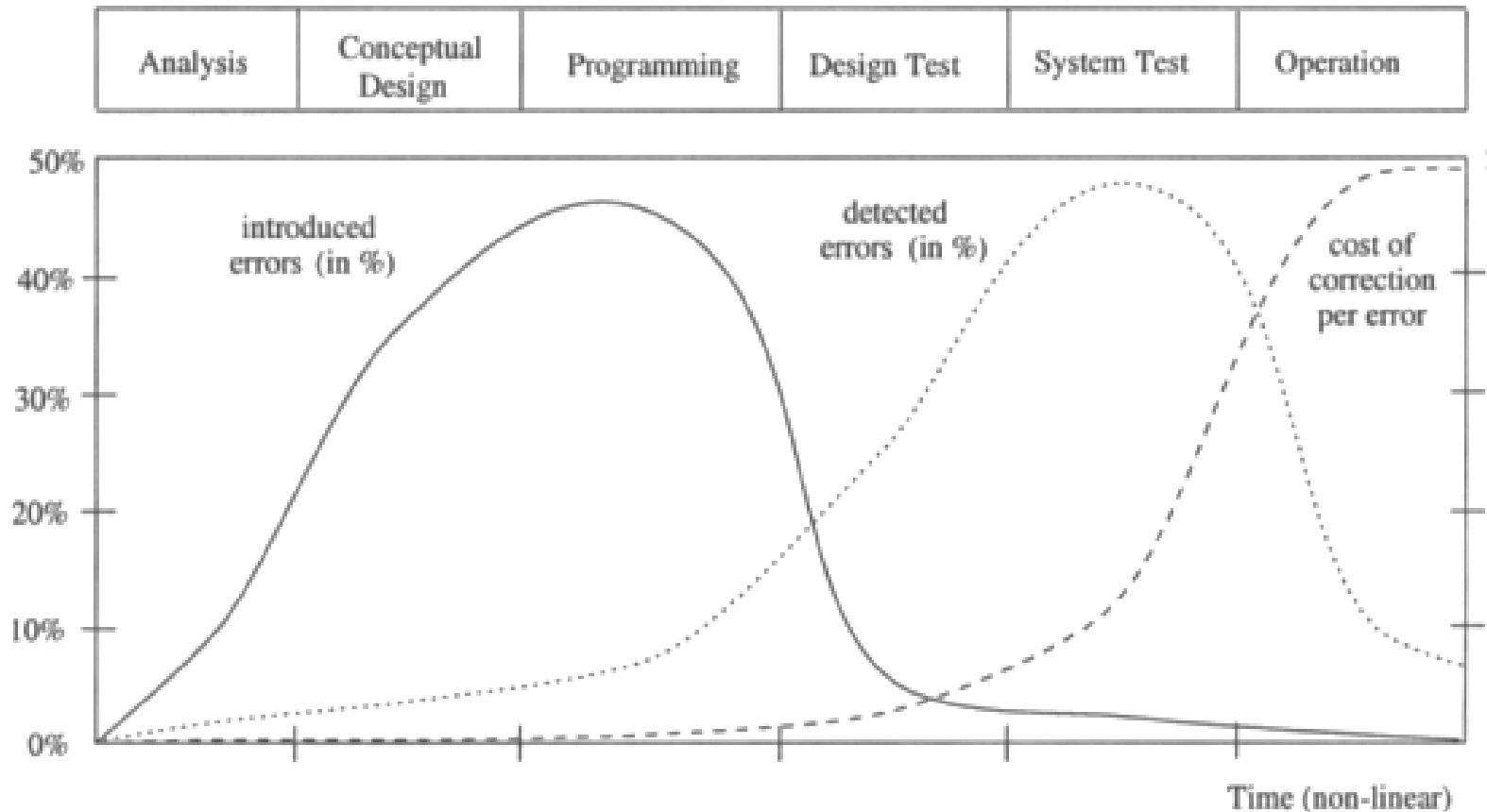
- Number of faults in 1 kLOC (embedded software):
  - Manual development and testing: ~ 10 faults
  - Tool-supported automated development: ~ 1-2 faults
  - Automated development with formal methods: < 1 faults

Number of faults / kLOC



	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



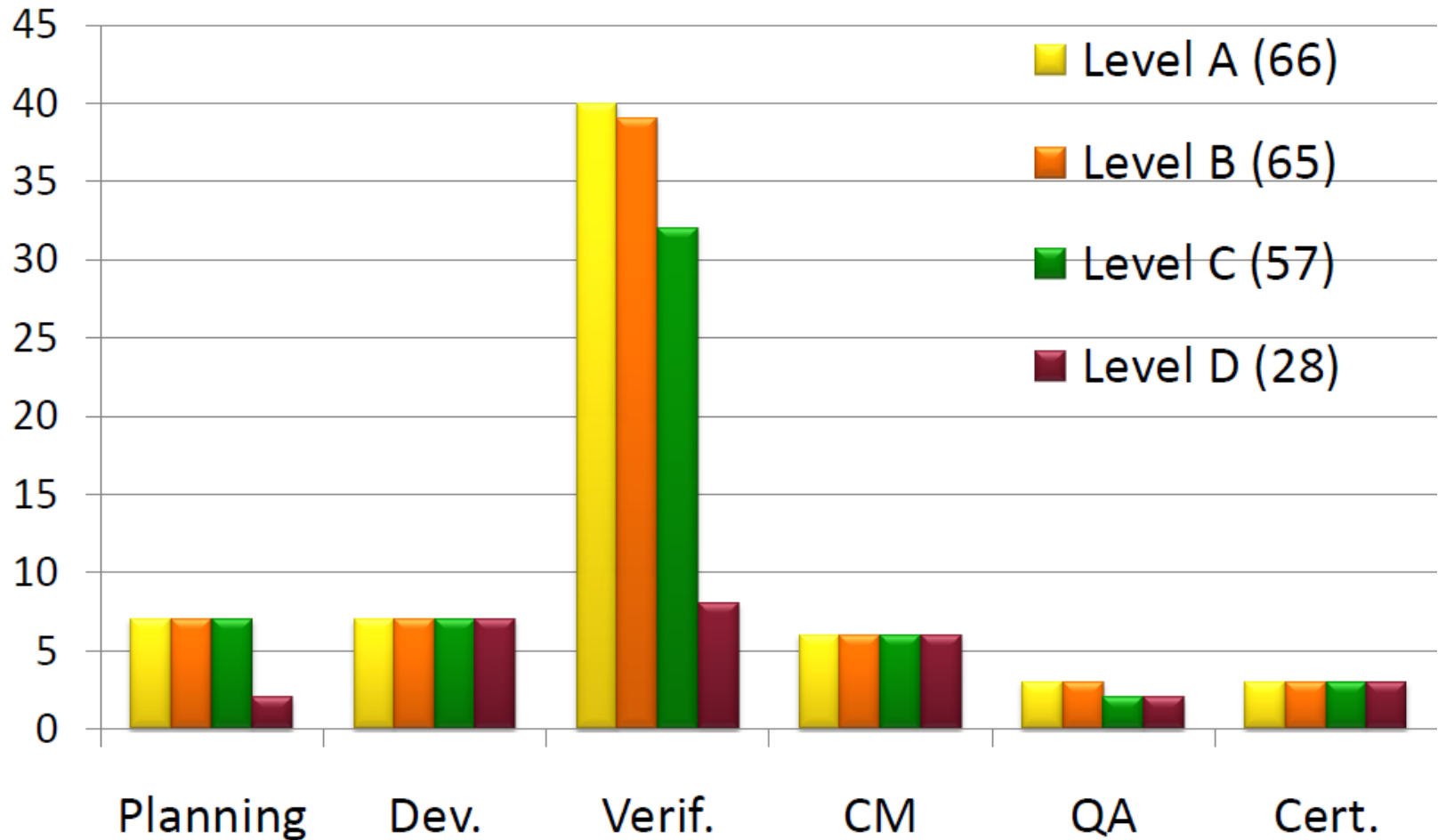
Early V&V reduces cost!

# V&V: Verification and Validation

<b>Verification</b>	<b>Validation</b>
„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

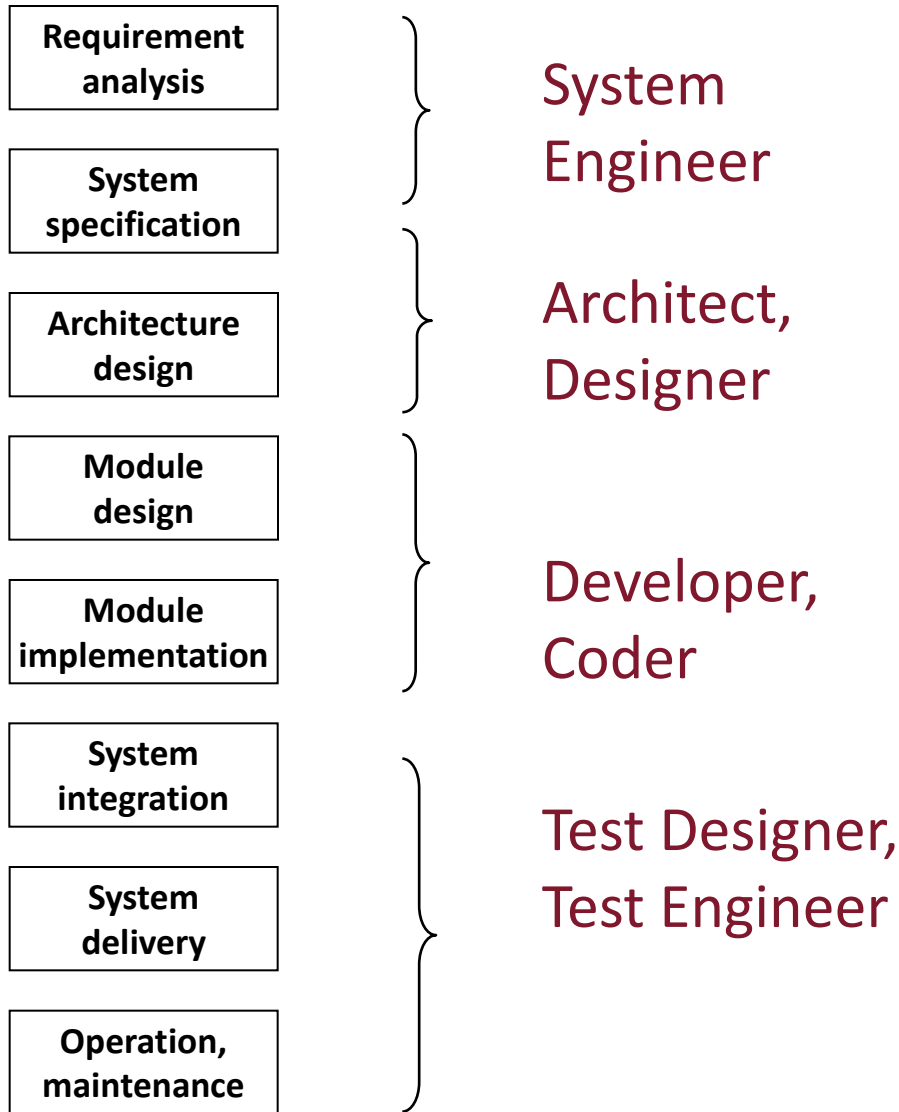
Test Engineer

- Test automation, integration and system tests

Safety Engineer

- Assessment w.r.t. development standards

# What are the typical development steps?



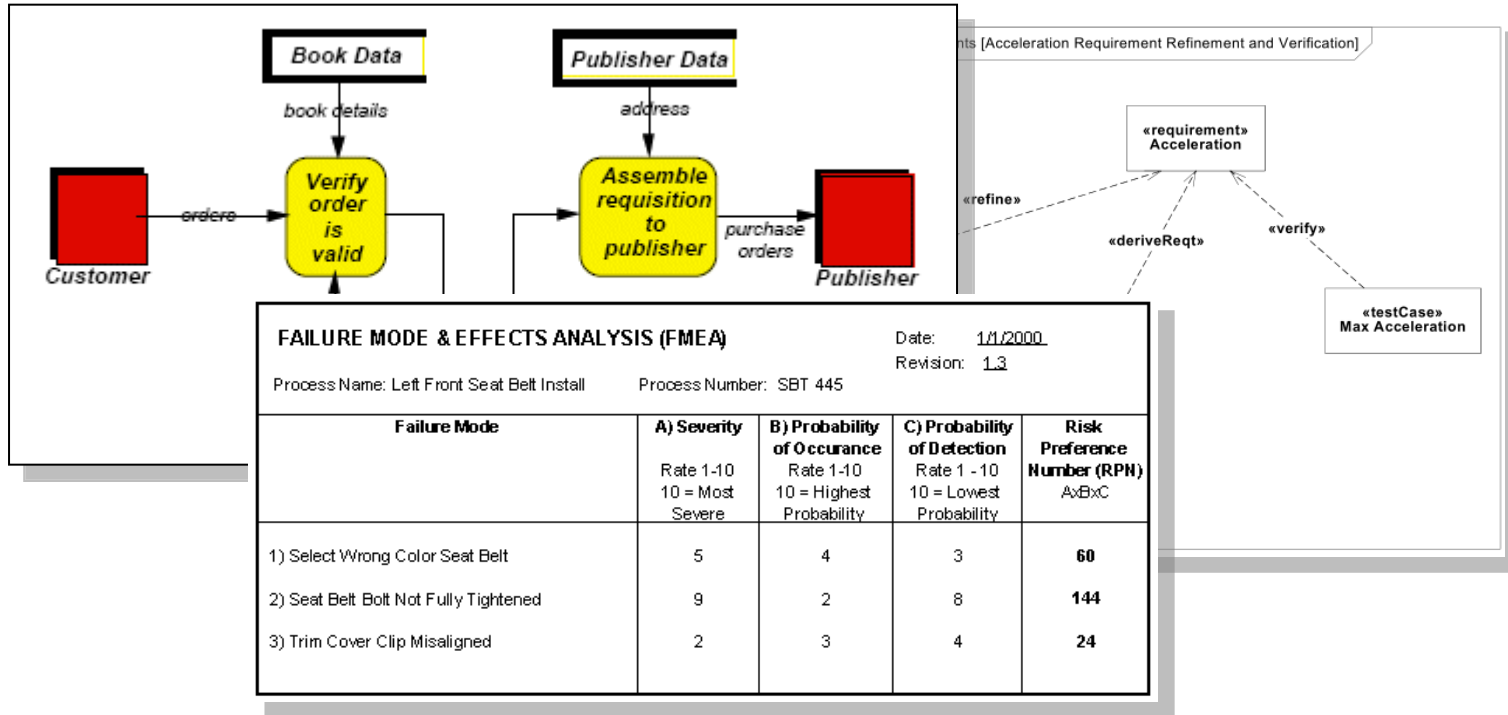
Schedule and sequencing depends on the lifecycle model (see later)



# 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	<ul style="list-style-type: none"> <li>- Risks</li> <li>- Criticality</li> </ul>	<ul style="list-style-type: none"> <li>- Checklists</li> <li>- Failure mode and effects analysis</li> </ul>



# System specification

Requirement analysis

System specification

Architecture design

Module design

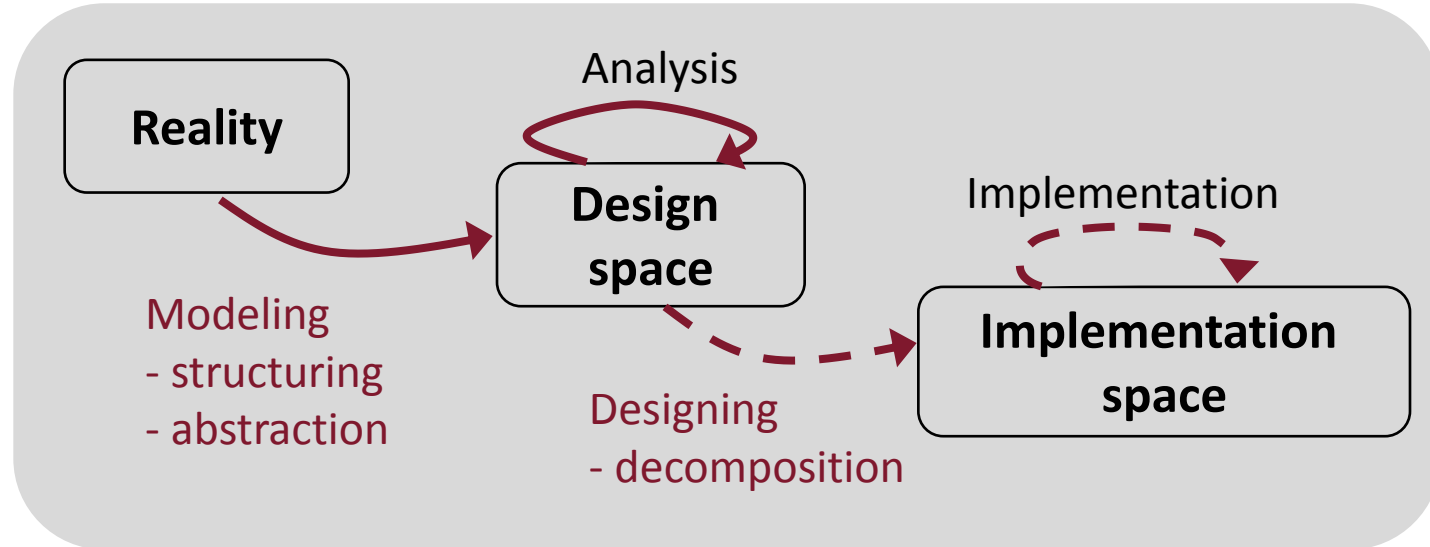
Module implementation

System integration

System delivery

Operation, maintenance

Task	V&V criteria	V&V technique
Defining functional and non-functional requirements	<ul style="list-style-type: none"><li>- Completeness</li><li>- Consistency</li><li>- Verifiability</li><li>- Feasibility</li></ul>	<ul style="list-style-type: none"><li>- Reviews</li><li>- Static analysis</li><li>- Simulation</li></ul>



# 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	<ul style="list-style-type: none"><li>- Completeness</li><li>- Consistency</li><li>- Verifiability</li><li>- Feasibility</li></ul>	<ul style="list-style-type: none"><li>- Reviews</li><li>- Static analysis</li><li>- Test case generation</li></ul>

## 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 each 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	<ul style="list-style-type: none"> <li>- Completeness</li> <li>- Consistency</li> <li>- Verifiability</li> <li>- Feasibility</li> </ul>	<ul style="list-style-type: none"> <li>- Reviews</li> <li>- Static analysis</li> <li>- Simulation</li> </ul>

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

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

An **event** (change of situation):

```
pass = ANY p,b WHERE (p,b) ∈ aut ∧ sit(p) ≠ b
      THEN sit(p) := b END
```

**Automated analysis** is possible: Checking invariant for each event

# Architecture design

Requirement analysis

System specification

Architecture design

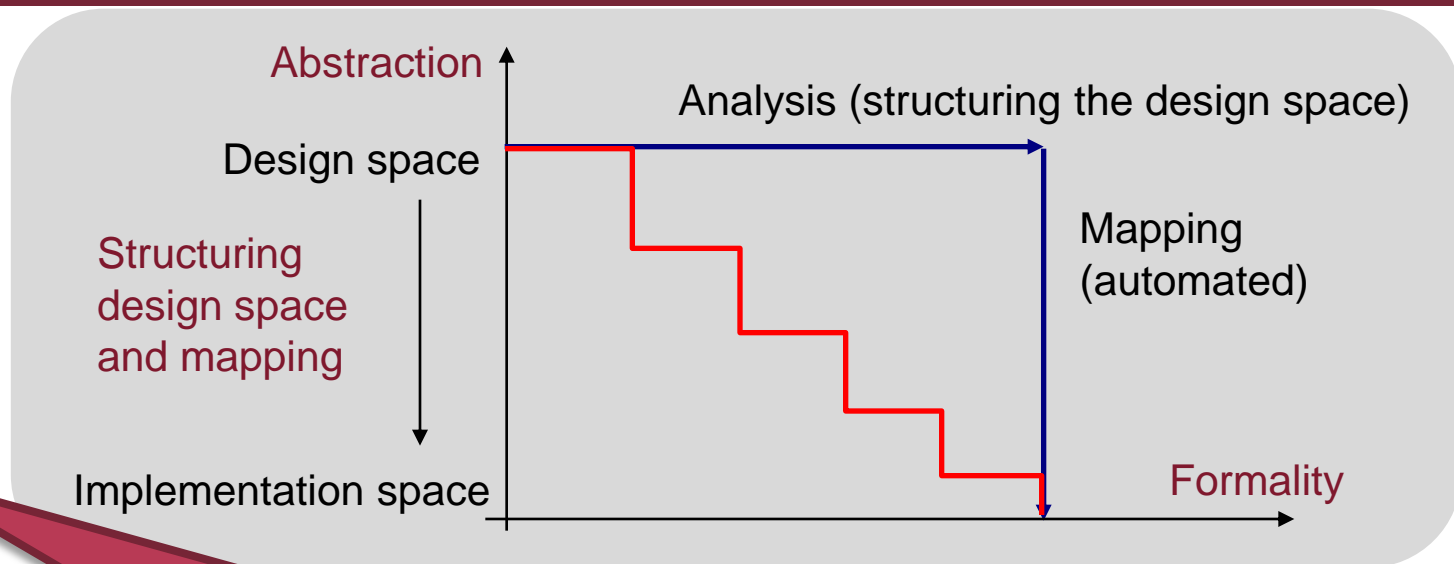
Module design

Module implementation

System integration

System delivery

Operation, maintenance



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

# Module design (detailed design)

Requirement analysis

System specification

Architecture design

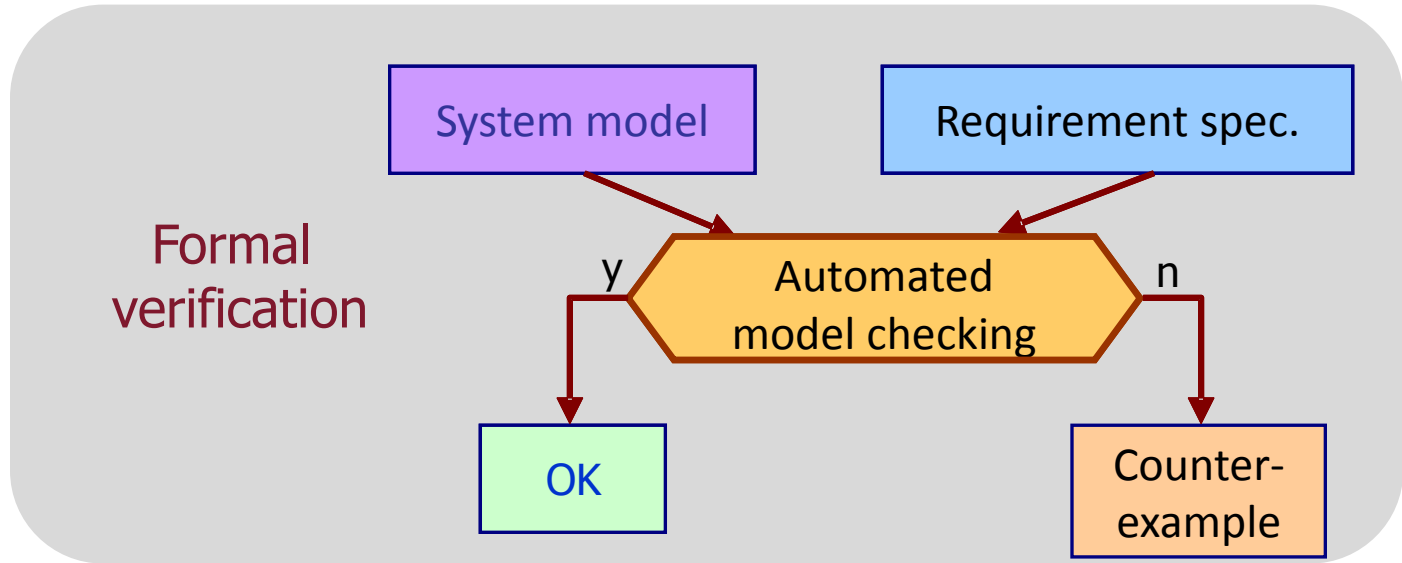
Module design

Module implementation

System integration

System delivery

Operation, maintenance



Task	V&V criteria	V&V technique
- Designing detailed behavior (data structures, algorithms)	- Correctness of 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

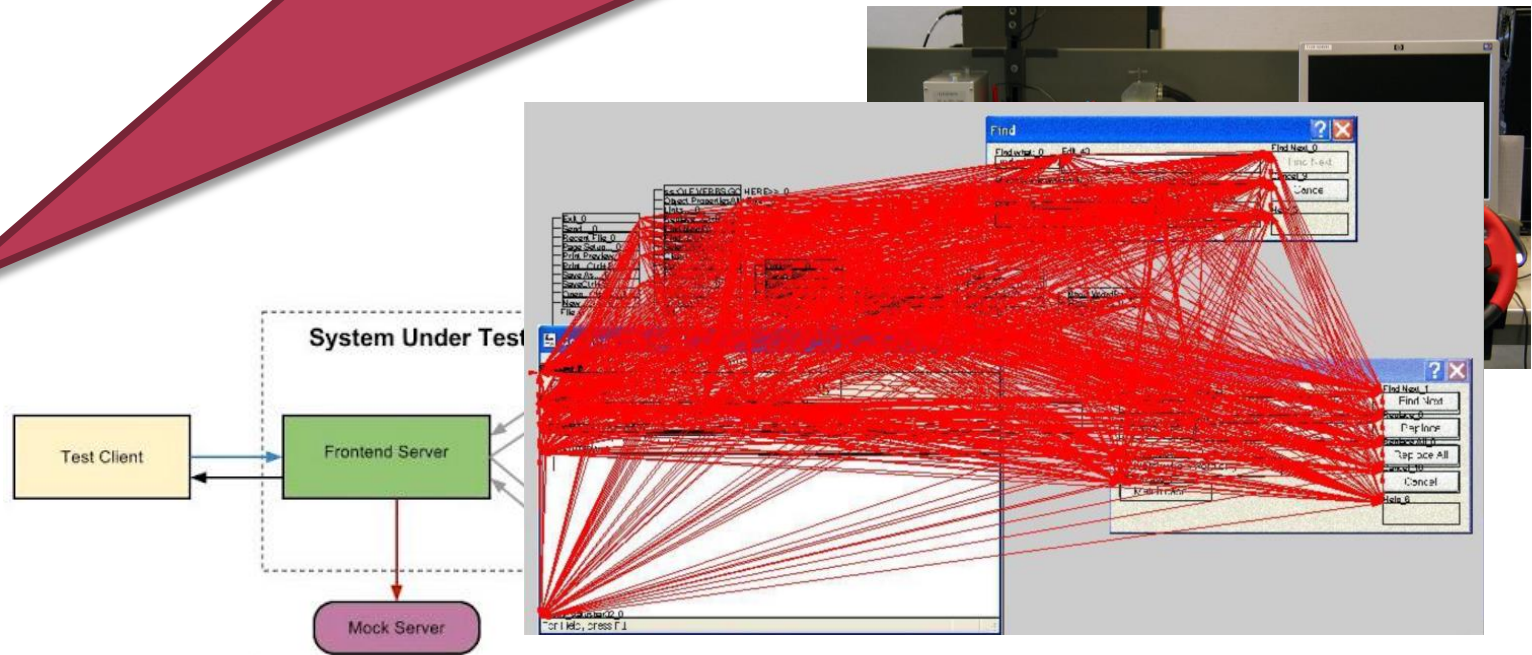
Module implementation

**System integration**

System delivery

Operation, maintenance

Task	V&V criteria	V&V technique
<ul style="list-style-type: none"> <li>- Integrating modules</li> <li>- Integrating SW with HW</li> </ul>	<ul style="list-style-type: none"> <li>- Conformance of integrated behavior</li> <li>- Correct communication</li> </ul>	<ul style="list-style-type: none"> <li>- Integration testing (incremental)</li> </ul>





# 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
- Satisfying 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

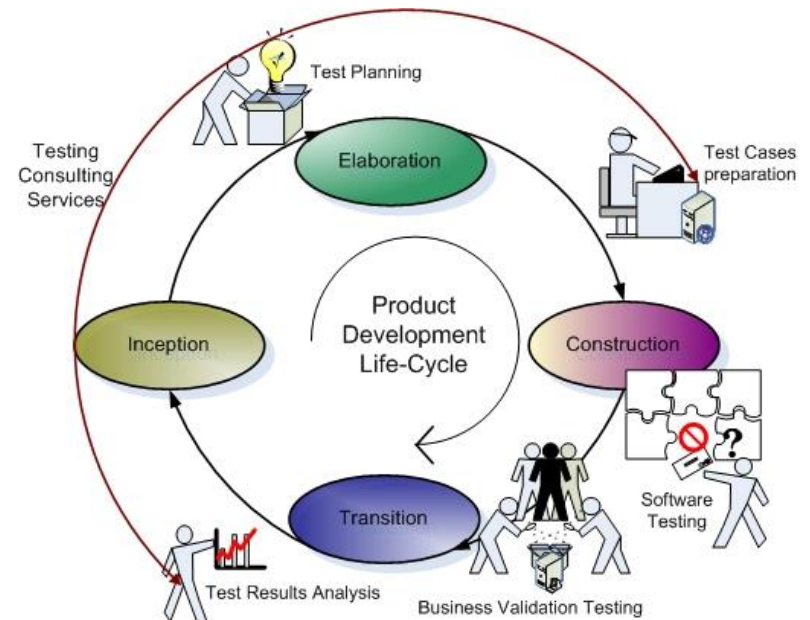
System delivery

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



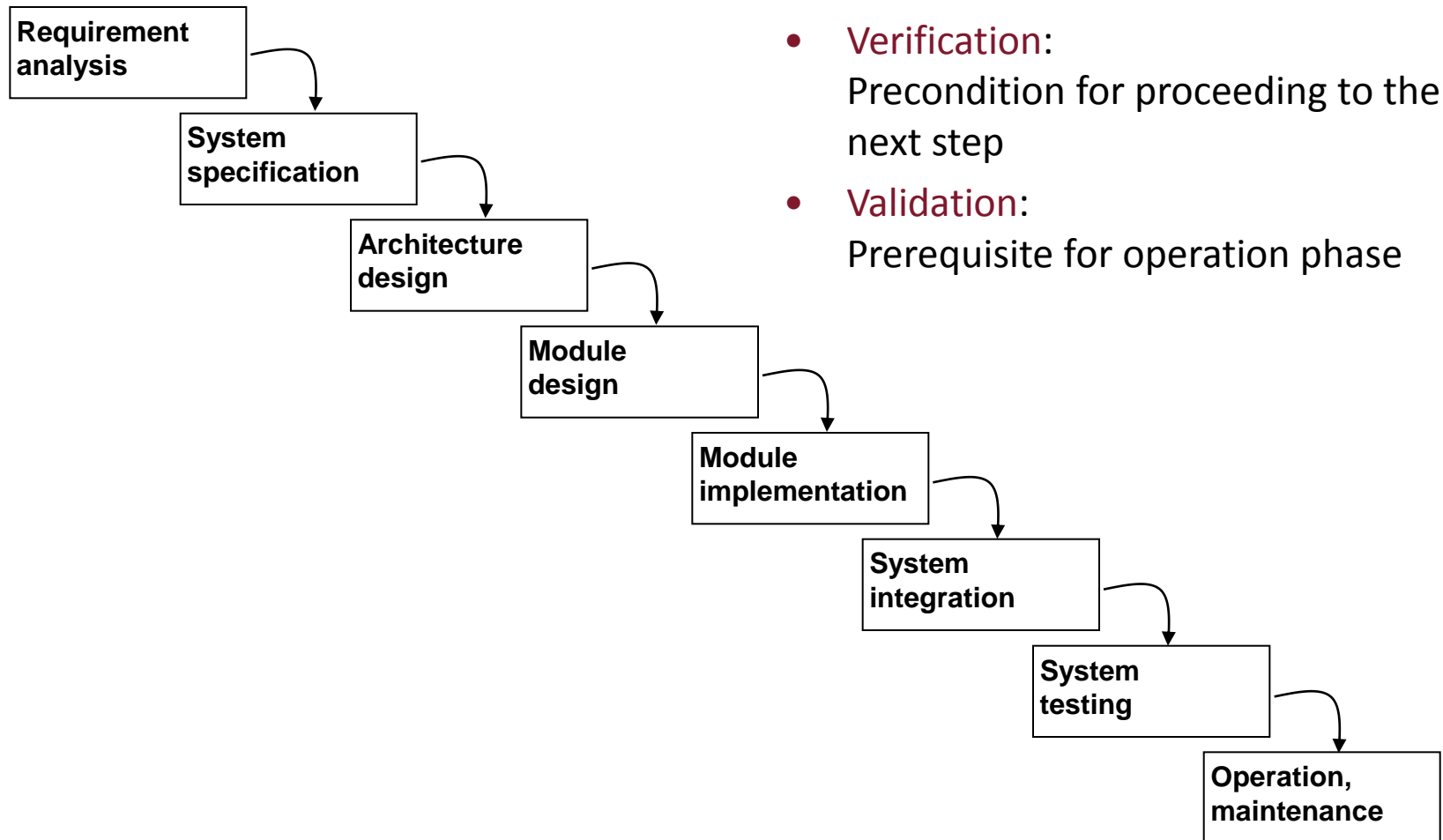
# Development life cycle models

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

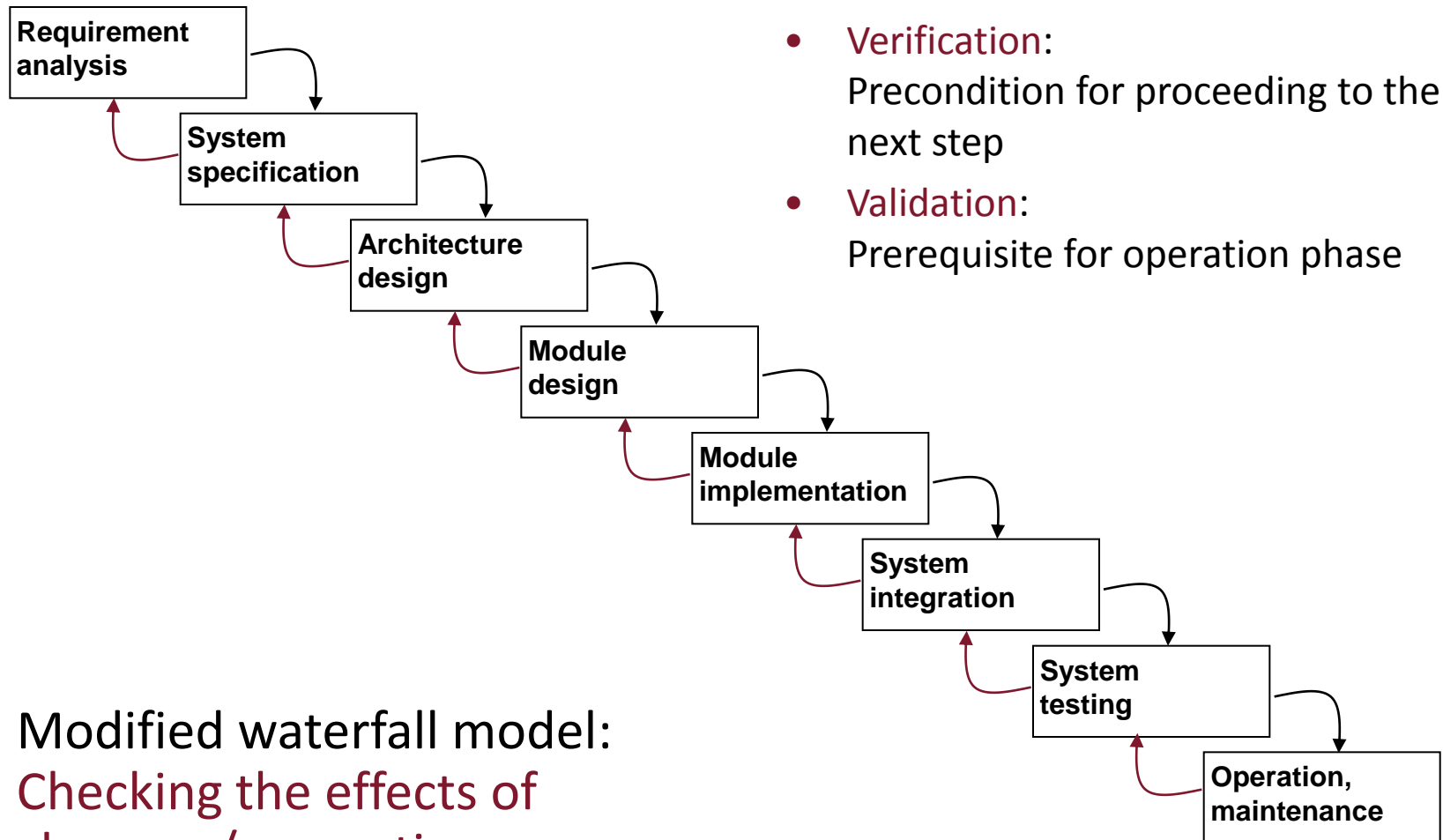
# Development life cycle models

- 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 development: 4G model
  - Iterative-incremental development: Unified Process

# 1. Waterfall model

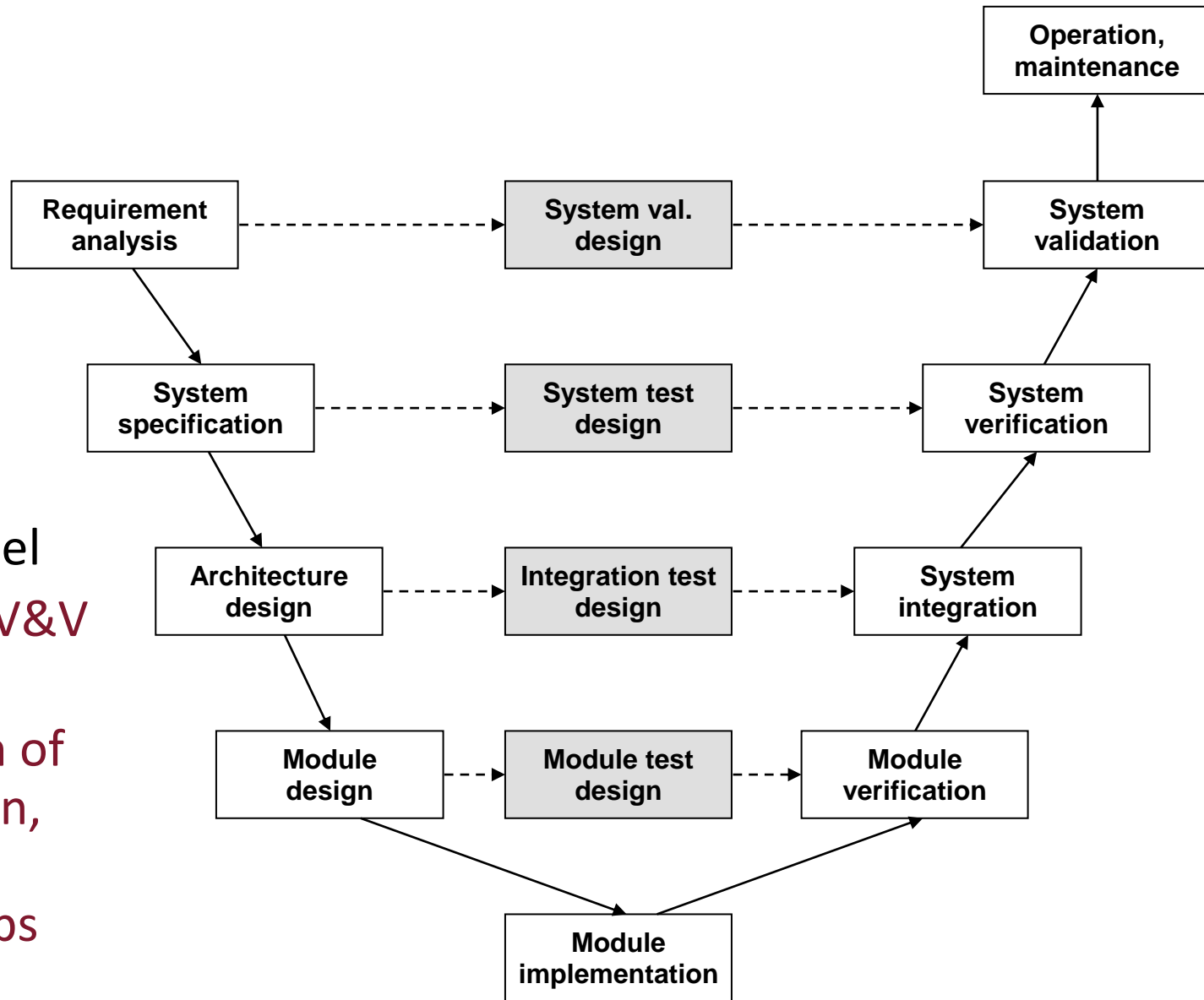


# 1. Waterfall model



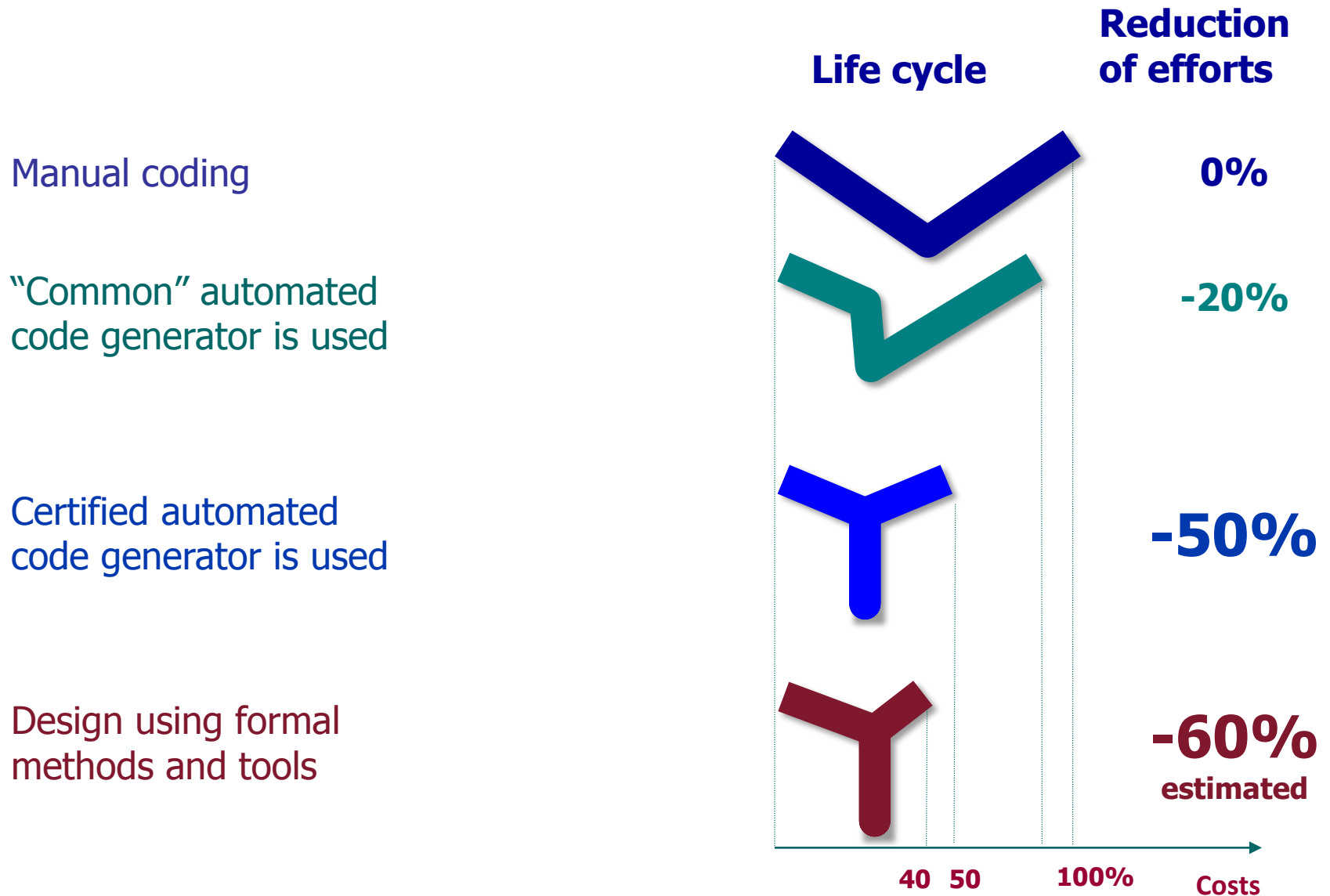
- Modified waterfall model:  
Checking the effects of changes / corrections  
(e.g., regression testing)

# 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





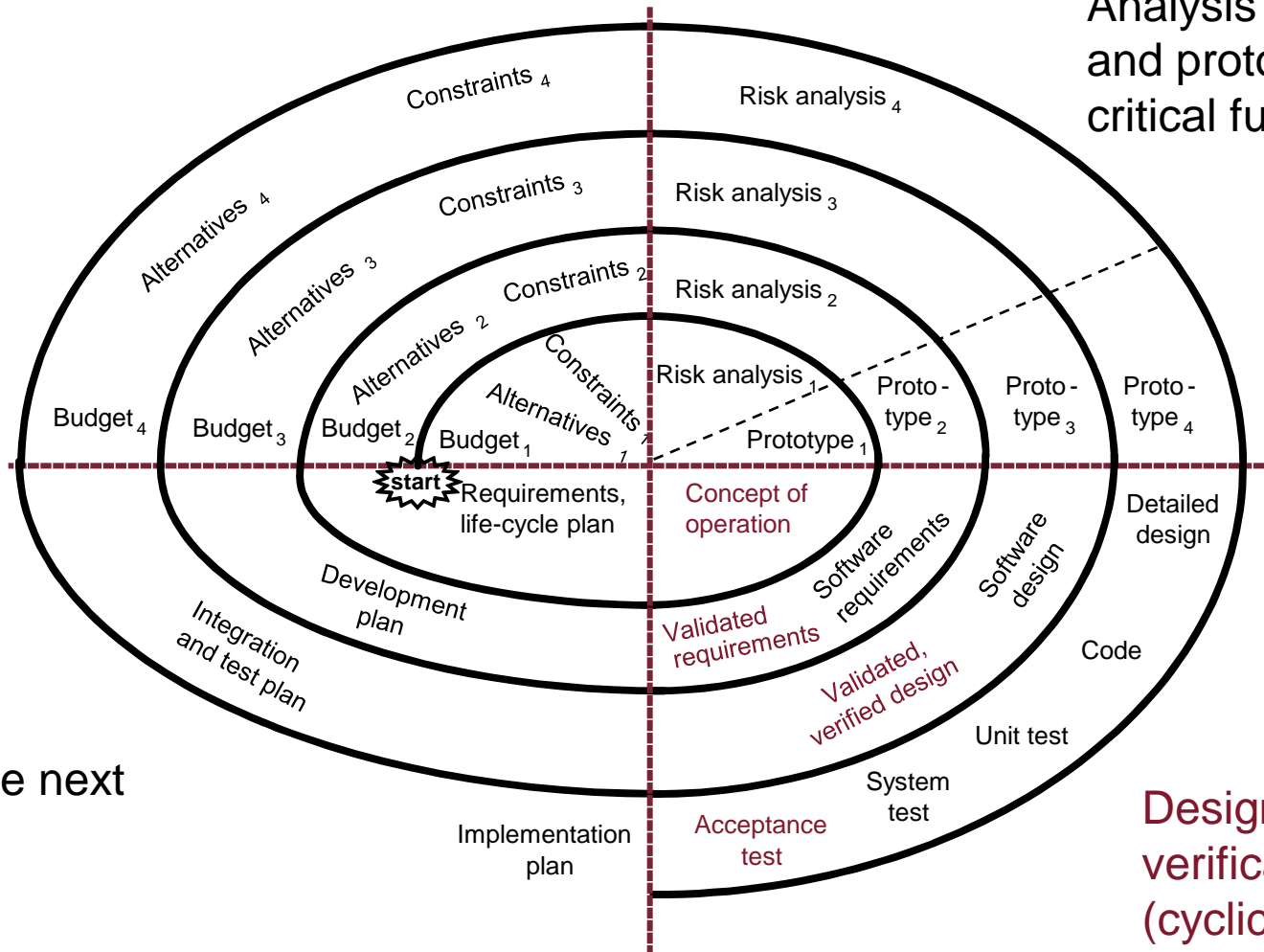
# 3. Evolutionary development (RAD)

- 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

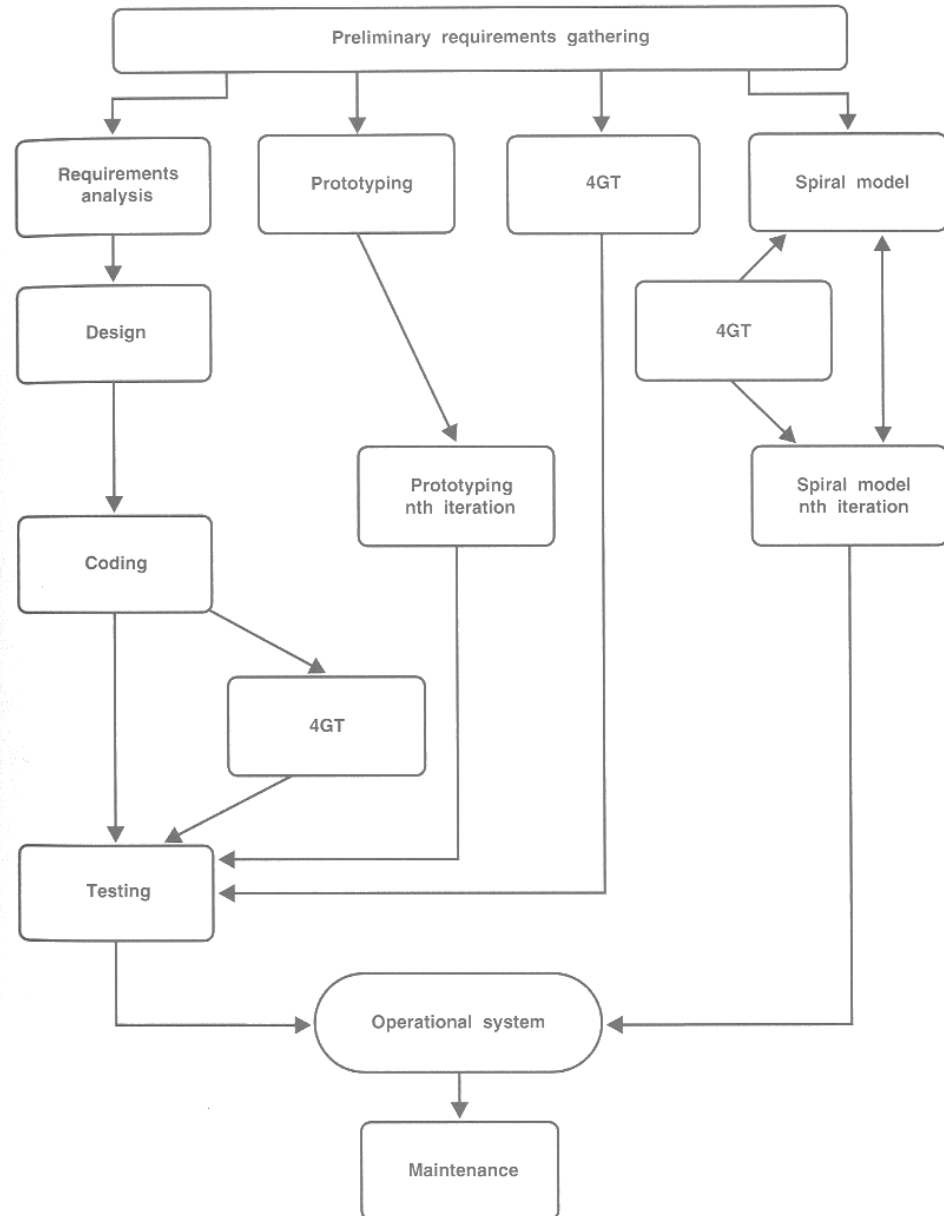


Planning the next  
phase

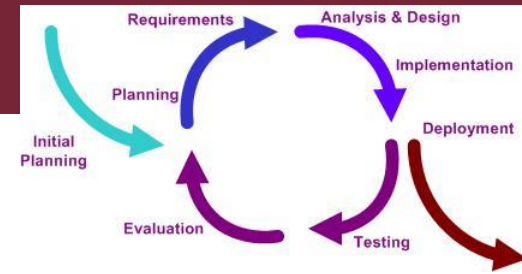
Design and  
verification  
(cyclic)

# 5. The “4G” model

- Model based development
  - CASE tools
  - Property preserving refinement
  - Model based verification
- Integration of approaches
  - Well-specified requirements: “Traditional” development
  - Incompletely specified requirements: Rapid prototype development
  - Formally specified requirements: Model based development
  - With iterative design

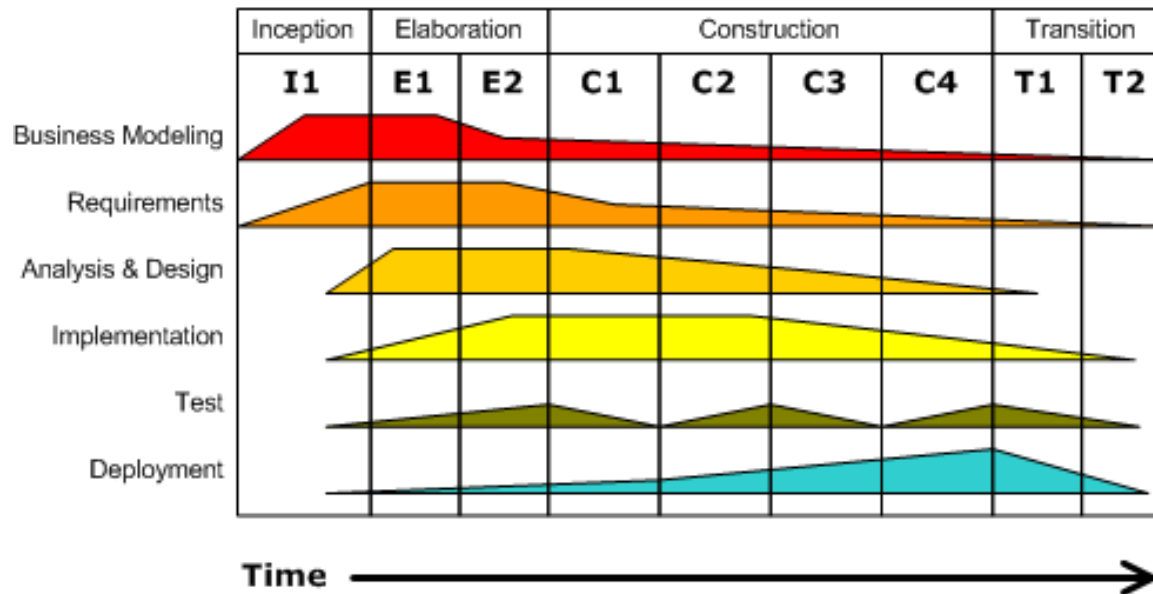


# 6. Unified Process



## ■ 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)
  - Using build frameworks, testing is included
- "Test first programming" concept:
  - Functional tests based on "story card"
  - Testing after each modification (new functions)

## ■ 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