System Engineering over time

Notes on Model Based Engineering

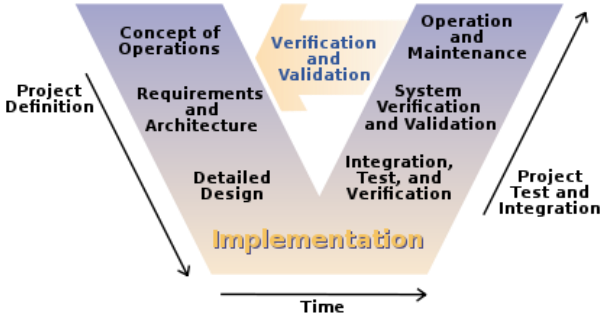
Johan Lukkien

Disclaimer

“In a system consisting of several components, it is essential that such components are jointly designed and tuned to serve the overall system goals. Although accepted in theory it is rarely done in practice” (PhD thesis Creusen)

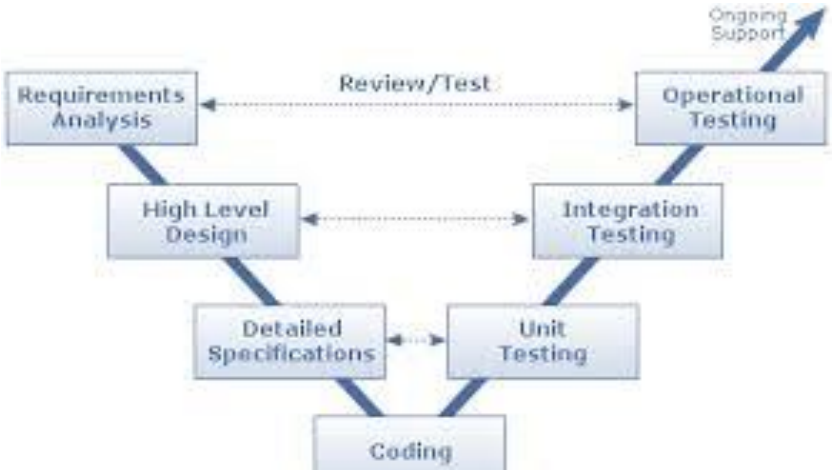
- dependencies
- standards
- interfaces
- evolution
- models

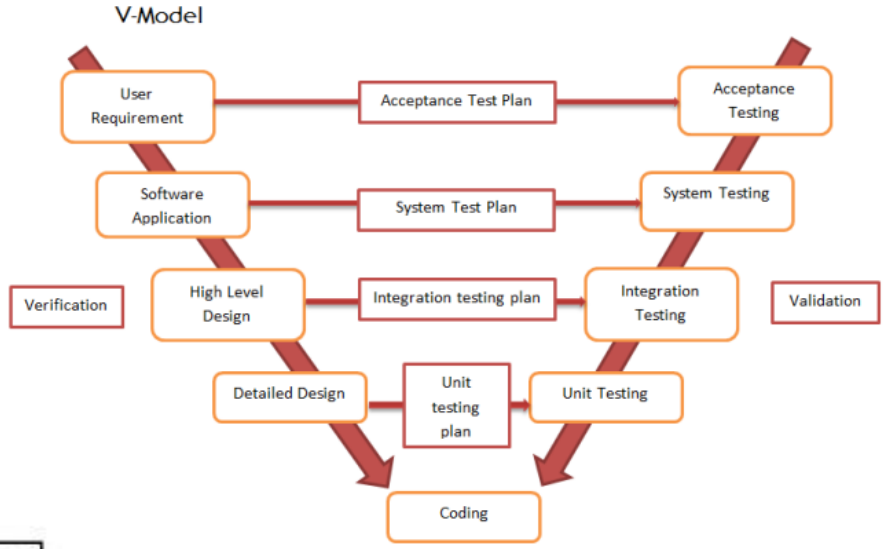
Theory and practice

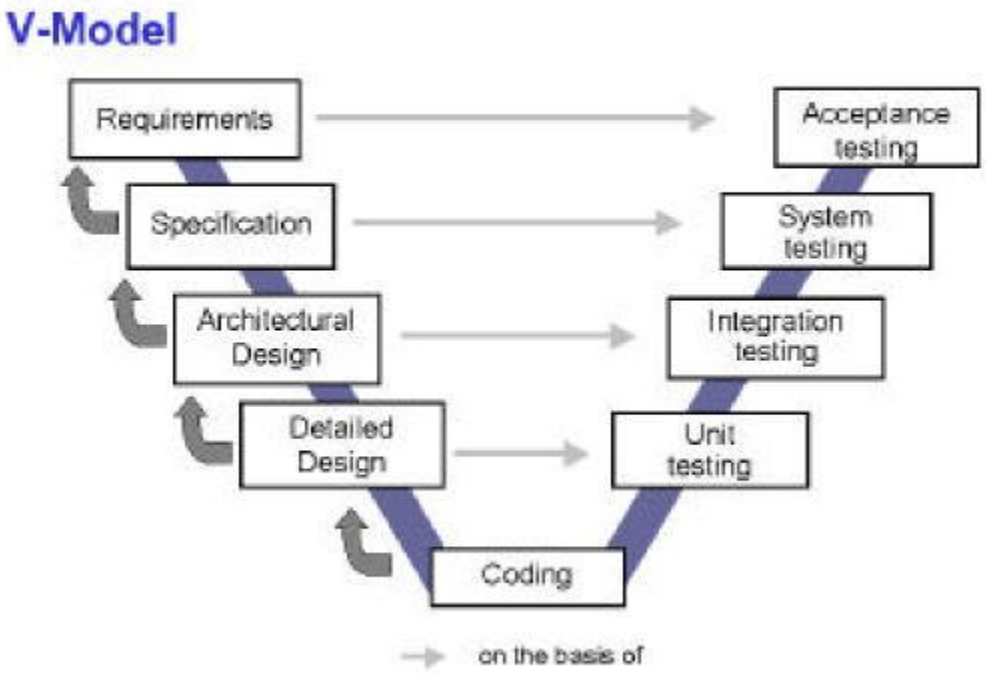


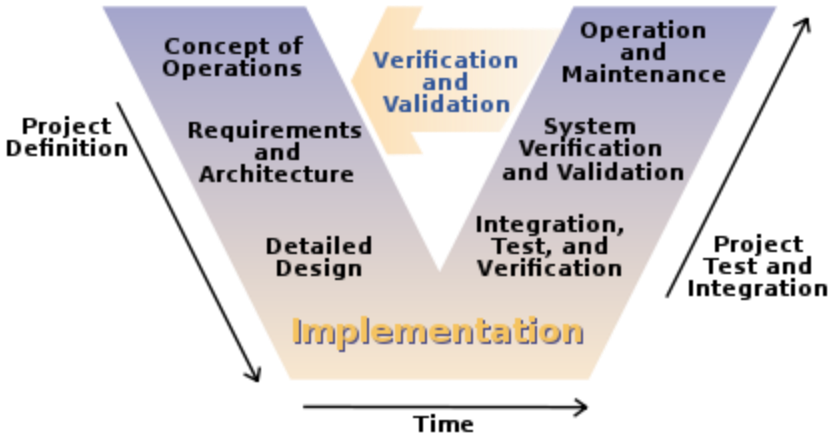
- Theory: run the V down and up again and your system is ready to use
- Practice: If this ever happens, then only once . After the first version, evolution is the normal mode of operation.
- Hence, concentrate on aspects of evolution, and define engineering processes and artifacts accordingly as evolution steps of an existing system.
- Questions:
  - which evolution steps, which categories of steps are typical?
  - what are typical problems encountered?
  - co-evolution, with multiple parts and parties involved
  - include evolution of engineering processes

V? Which V?


Ongoing Support
Requirements Analysis
Review/Test
Operational Testing
High Level Design
Integration Testing
Detailed Specifications
Unit Testing
Coding


V-Model
User Requirement
Acceptance Test Plan
Acceptance Testing
Software Application
System Test Plan
System Testing
Verification
High Level Design
Integration testing plan
Integration Testing
Validation
Detailed Design
Unit testing plan
Unit Testing
Coding


V-Model
Requirements
Acceptance testing
Specification
System testing
Architectural Design
Integration testing
Detailed Design
Unit testing
Coding
on the basis of


Concept of Operations
Verification and Validation
Operation and Maintenance
Project Definition
Requirements and Architecture
System Verification and Validation
Detailed Design
Integration, Test, and Verification
Project Test and Integration
Implementation
Time

Engineering workflows

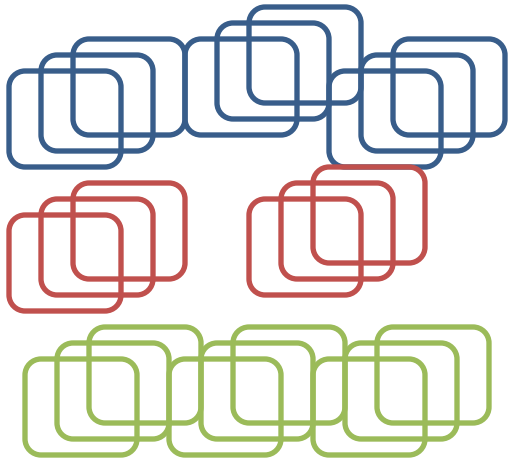
- New insights, new ICT techniques, new tools, new methodologies have a deep impact on engineering workflows

- This is often not recognized at first, and is a root cause for slow (or no) adoption
  - the workflow is not adjusted resulting in people arguing that the new insights do not apply
  - the workforce is not trained properly

- Questions:
  - what is the actual workflow, and what is the aimed-for one, and why?
  - is there a ‘holistic’ view on the entire process?
  - how is the transition plan?

Set of models for each component

Models target *aspects*



analysis construction

- Represent components, and system (partly) in the virtual world (virtual prototyping)
- Identify *variation points* and features supporting *architecture variability* (product families)
- Analyze component and system properties

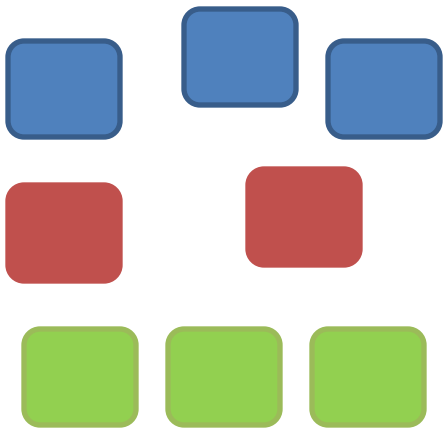
consistency under evolution by extraction (abstraction) / generation (refinement)

Component-based system

Software

(Computer) Hardware

Plant



evolution

Component evolution, result of

- technology progress (versions)
- replacement: end-of-life/-manufacturing

System evolution, result of

- upgrade
  - configuration change
  - component evolution and implied dependencies
- new generation, new components
  - instances in a product family

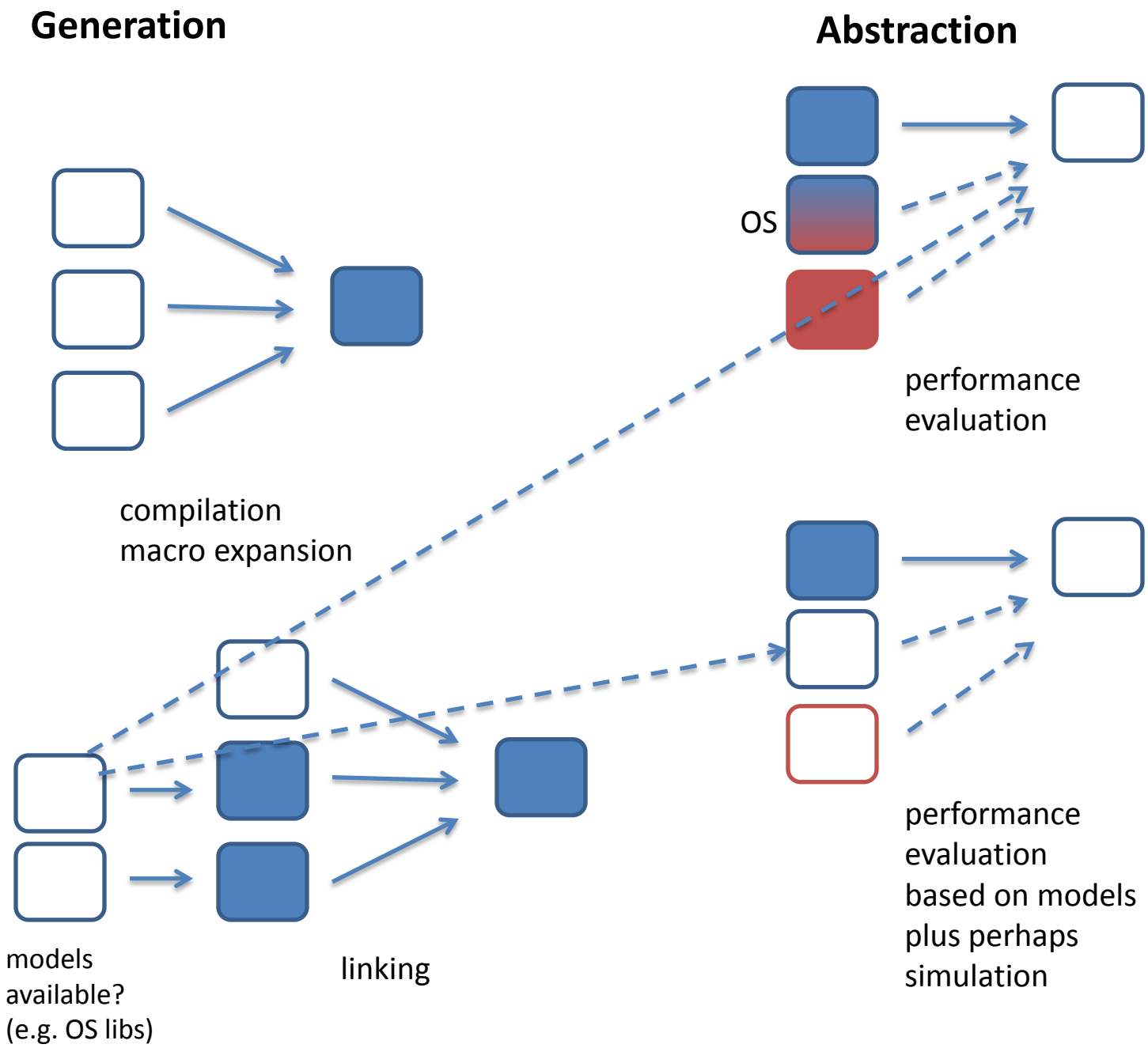
Component

- In software, it is some object that conforms to a component model (CORBA, DCOM, ….)
  - however, not very successful in developing a design discipline for software
  - … while composing components in the sense of ‘adding subsystems together’ has been very successful
- We consider a component as an element of the implementation domain: any unit of deployment with well-defined interfaces
- Models give views on these components
- The mapping: component-model(s) depends on the model transformation, and, especially for software, it is not always 1-1 as the picture suggests

Models of components

	static	dynamic
descriptive	structural and behavioral properties	describe (evaluate) properties of executions
executable	sufficient information to construct the component	study execution details dependent on inputs

- developed
  - during design
  - from implementation


Generation
Abstraction
OS
performance
evaluation
compilation
macro expansion
performance
evaluation
based on models
plus perhaps
simulation
models
available?
(e.g. OS libs)
linking

Challenges

- How to use modeling, simulation, transformation, specialization techniques to support the process of evolution?
  - Which representations help?

- Can we measure the improvements?
  - define metrics, and measurements
  - zero measurement, measure where the problems are

Dependencies

- Parnas' principles:
  - The developer of a software component must provide the intended user with all the information needed to make effective use of the services provided by the component, and should provide *no* other information.
  - The implementor of a software component must be provided with all the information necessary to carry out the given responsibilities assigned to the component, and should be provided with *no* other information.

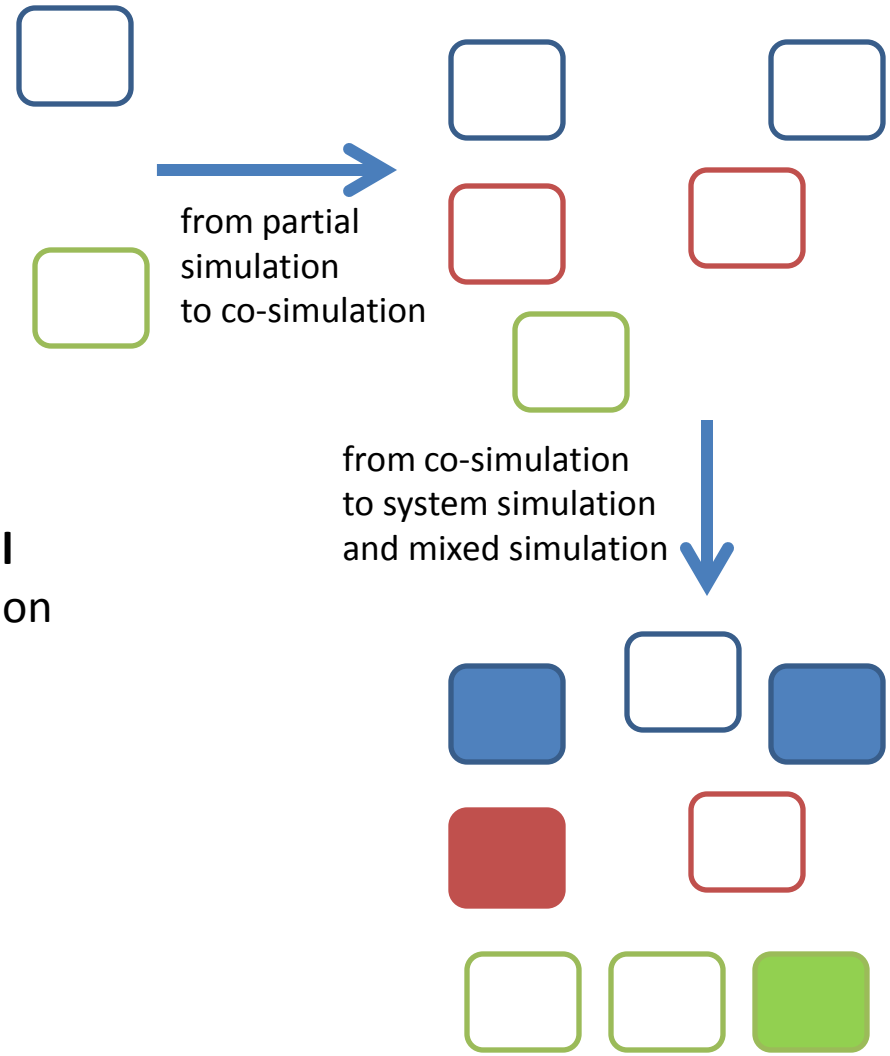
- Rather successful for *functional* properties, to separate specification and implementation
- However, difficult to evolve
  - knowledge of the interface of a component may be laid down in control structures or untested assumptions of a dependent component
- Very difficult for *emergent* properties (performance, dependability, …)
  - brittle, non-continuous
  - needs information about emergent properties at interfaces

What (which models) to maintain?

- Models that are correct abstractions of the actual components
  - inevitably, depending on (assuming) a *context,* i.e., other models
    - e.g. OS version, scheduling policies, hardware details, but also input characteristics
  - have been abstracted from the actual component (and verified as such), or from verified models of that and other components
  - or can be mechanically transformed into the component
- Verification paths for such models
- Models developed during requirement analysis are often superseded by later choices and insights
- For dynamic, executable models
  - input traces
  - run traces
  - maintain interfaces (APIs, events)

The use of simulation

- Classical, during analysis
  - Analyze requirements
  - Evaluate, predict properties of the system beforehand
    - obtain configuration information
  - Early feedback

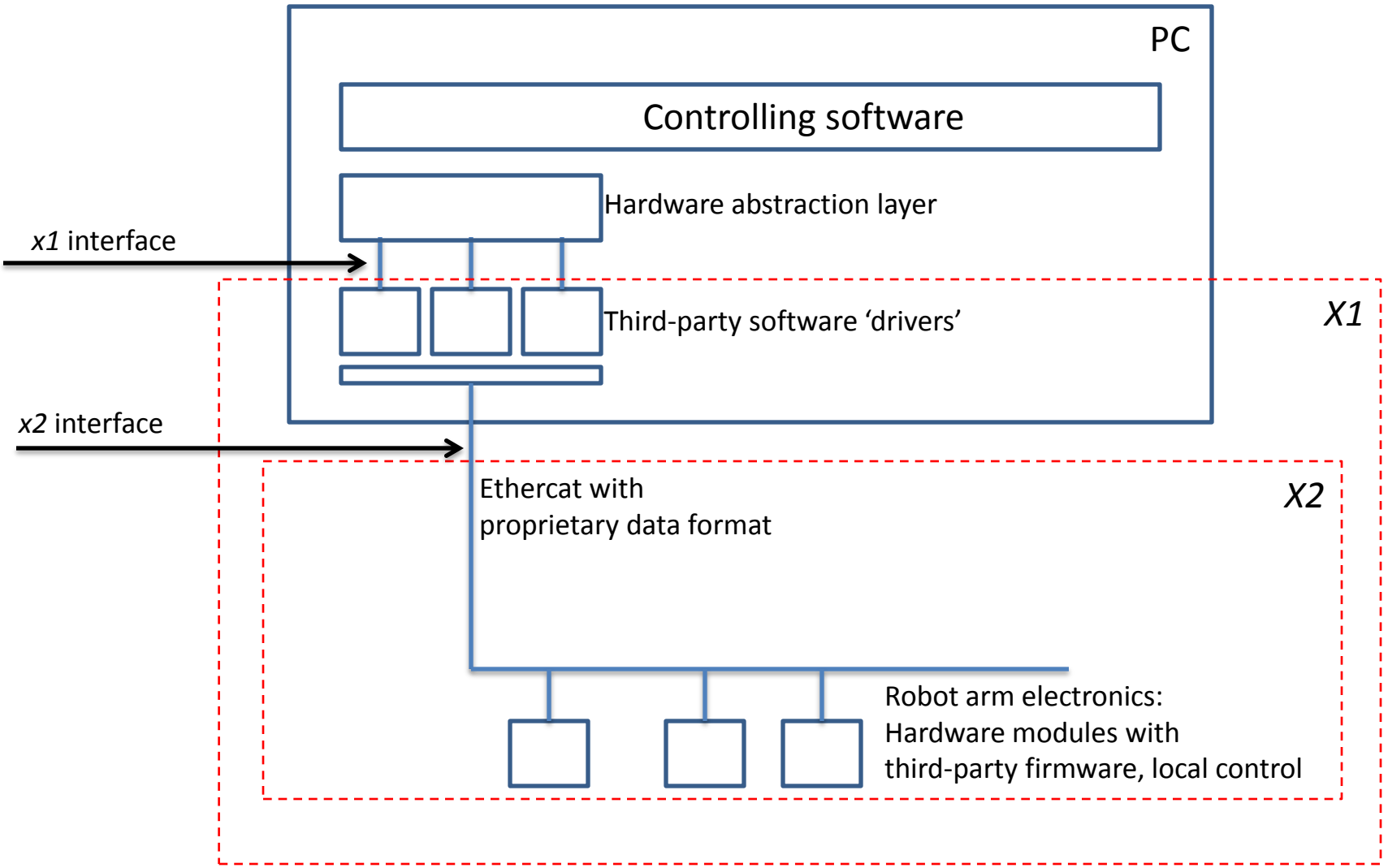


- More recently, during construction
  - Analyze **the actual system, the actual components**, understand the operation
    - look from the inside
  - obtain test cases, traces
  - perform dangerous or expensive test cases virtually
  - fault insertion
  - … improve the entire process

Maintaining interfaces

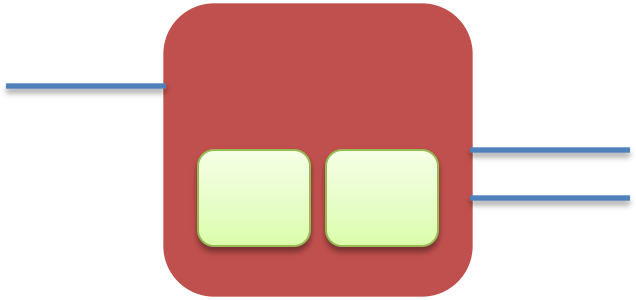
- Admits to ‘cut’ along an interface (or set of interfaces)
  - simulate one half, retain the other half

Example


PC
Controlling software
Hardware abstraction layer
x1 interface
Third-party software 'drivers'
X1
x2 interface
Ethercat with
proprietary data format
X2
Robot arm electronics:
Hardware modules with
third-party firmware, local control

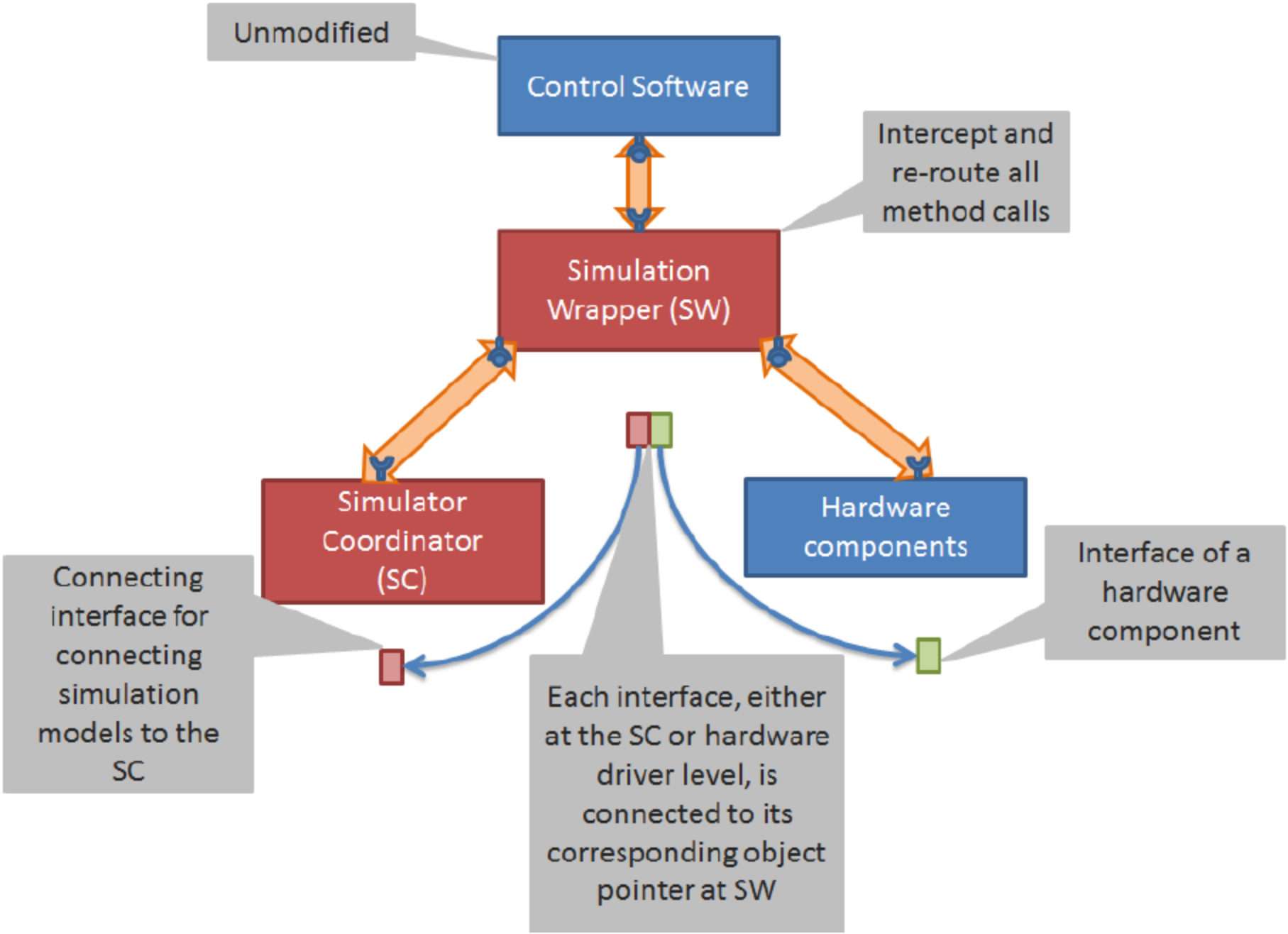
Simulation setup patterns

- Embed models (of the plant) into a special hardware setup to generate the right signals (*x2* interface).

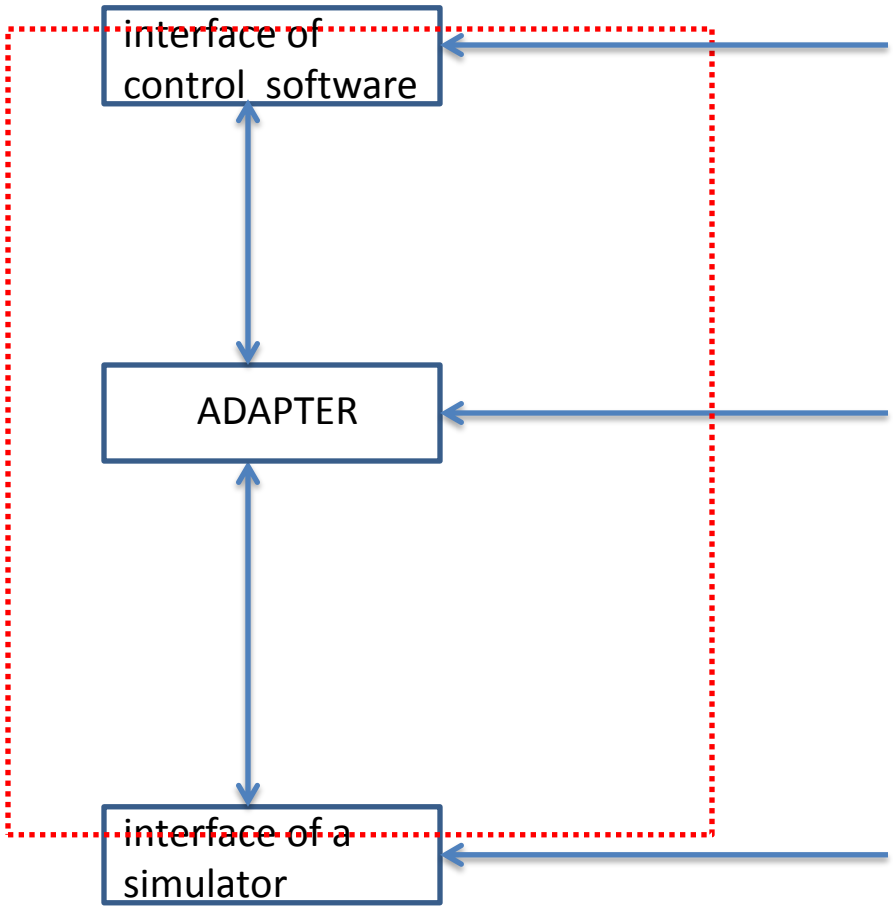


- Embed models into the runtime environment of a tool (e.g. Matlab).
  Need to connect to the ‘dangling’ interfaces of the remainder of the system (*x1* interface).

- Generate a runnable component that represents the model, with a standardized interface, e.g. FMI.
  Need to connect to the dangling interfaces.


Unmodified
Control Software
Intercept and re-route all method calls
Simulation Wrapper (SW)
Simulator Coordinator (SC)
Hardware components
Connecting interface for connecting simulation models to the SC
Each interface, either at the SC or hardware driver level, is connected to its corresponding object pointer at SW
Interface of a hardware component

Simulator Coordinator



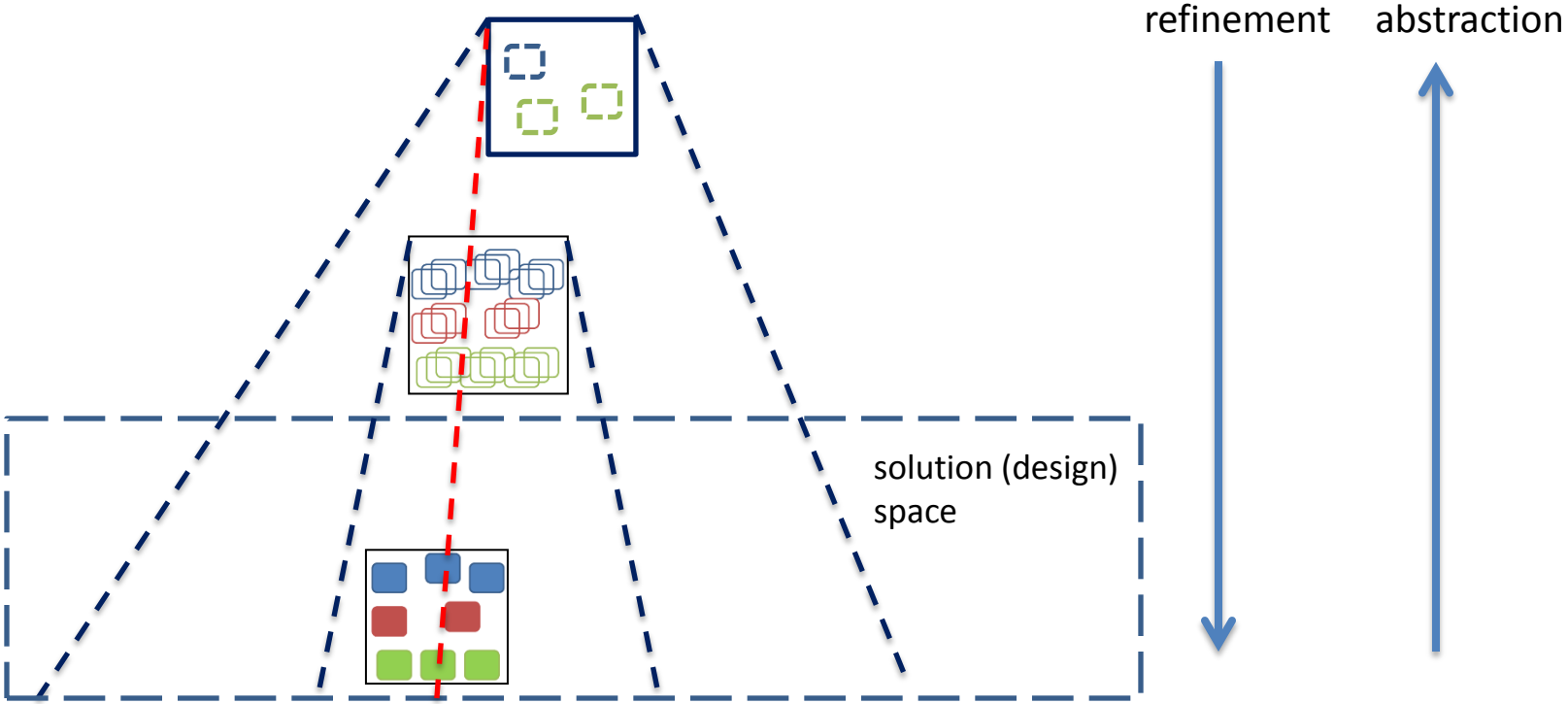
- connects to control software (data format, control flow)
- lives in real-time (although…)

- data format differences
- resolve control flow, may need to store some state
- manage real time and simulation time

- advance time (step)
- observe (internal) state
- adapt state
- pass events
- generate events

Concluding

- We need to design for evolution
- Models are first class citizens
  - models increase the abstraction level away from coding
  - models are produced during design but also extracted from the product
  - management
    - models more diverse, more expressive than just code
    - scalability requires automation in handling
    - consistency
- Traditional concerns of large code bases will move to model repositories
  - e.g. how to refactor models?
- Flexible simulation setups
  - combine / integrate simulators, patterns
  - need automation: e.g. generate based on interface specifications


refinement
abstraction
solution (design)
space