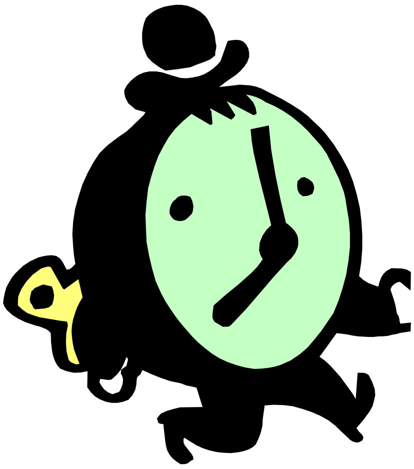
Real-Time Systems Programming

Summer-Semester 2002
Lecture 5
25 April 2002



Design Stages

The 5-Minute Review Session

1) What is *system engineering*?

2) What is *software engineering*?

3) What is the *V-Model*?

4) How does the *3-V Model* differ from it?

5) What are possible uses of a *system model*?

Designing RT Systems

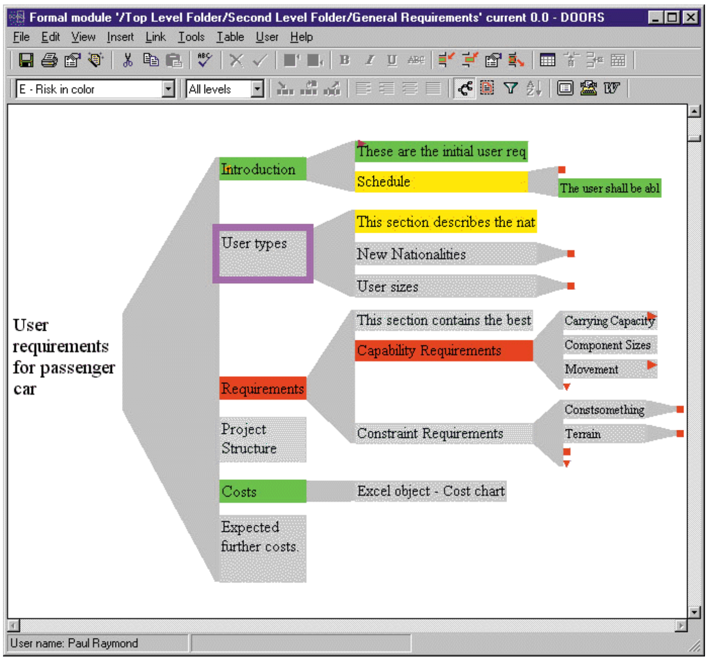
- In the following, will discuss these design stages:
  - Requirement specification
  - Architectural Design
  - Implementation
  - Validation/Testing
  - Prototyping

Requirement Specification

- Projects typically start with informal specification of ***objectives***
- Next, should perform extensive analysis of ***requirements***
  - Defines functionality of system
- For RT systems, requirements should include
  - *Temporal* behavior
  - *Dependability* requirements
  - Behavior in case of component *failure*
- Requirements also define later *acceptance tests*!

Requirement Specification

- Some structured/formal approaches used in requirements analysis:
  - *ALBERT* (Agent-oriented Language for Building and Eliciting Requirements for Real-Time Systems)
  - *VDM*, *Z*, *B*
  - *DOORS*

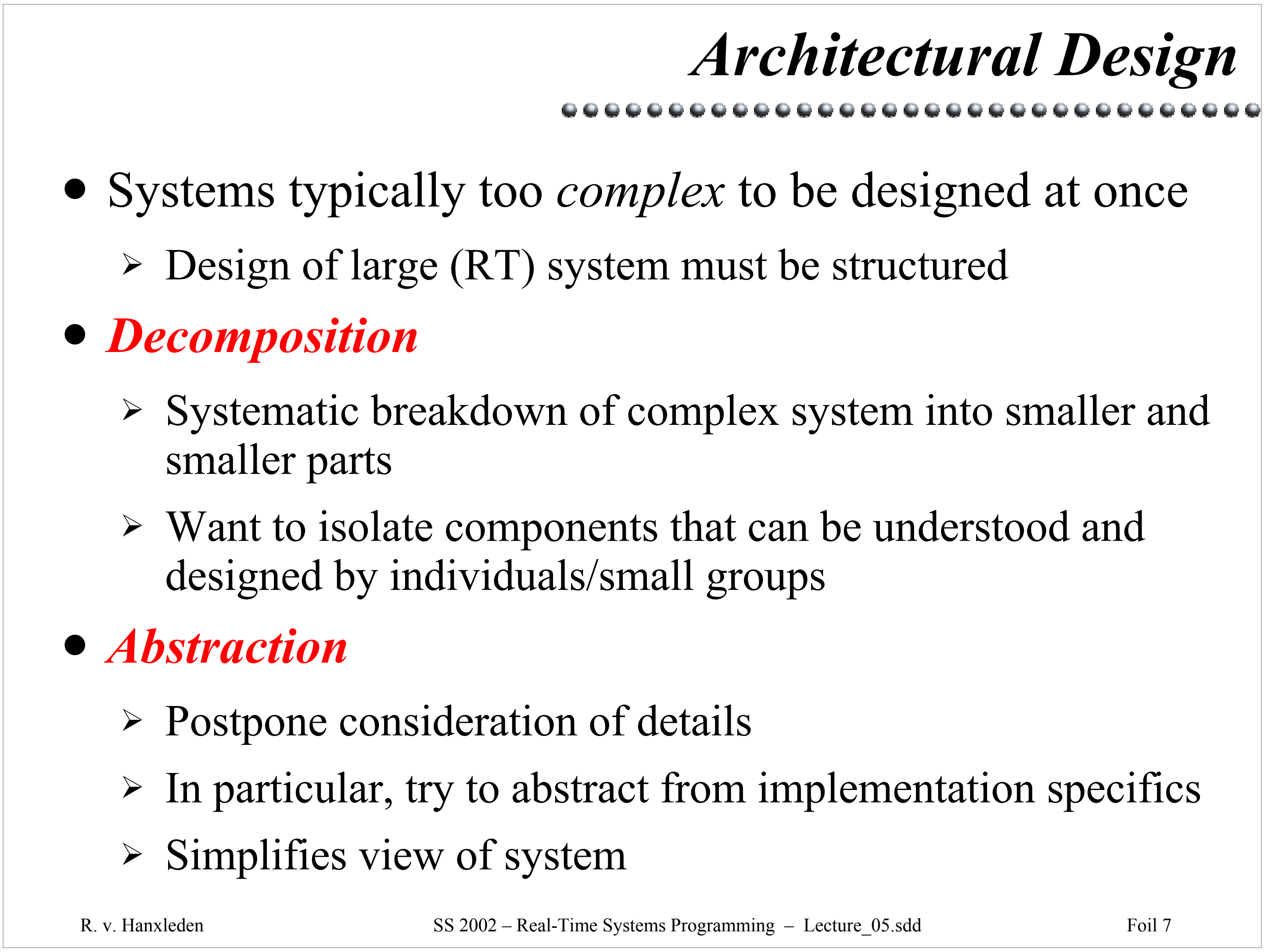

Formal module '/Top Level Folder/Second Level Folder/General Requirements' current 0.0 - DOORS
File Edit View Insert Link Tools Table User Help
E - Risk in color
All levels
User requirements for passenger car
Introduction
These are the initial user req
Schedule
The user shall be abl
User types
This section describes the nat
New Nationalities
User sizes
This section contains the best
Capability Requirements
Carrying Capacity
Component Sizes
Movement
Requirements
Constsomething
Terrain
Constraint Requirements
Project Structure
Costs
Excel object - Cost chart
Expected further costs.
User name: Paul Raymond

Telelogic AB, 2002

- ***ALBERT:***
  - E. Dubois, P.D. Bois, J. Zeippen, A formal requirements engineering method for real-time, concurrent, and distributed systems, *Proceesings of the Real-Time Systems Conference RTS '95, Paris, France,* 1995
- ***VDM:***
  - C. Jones, *Systematic Software Development using VDM*, Prentice Hall, 1986
- ***Z:***
  - M. Spivey, *The Z Notation – A Reference Manual*, Prentice Hall, 1989, http://spivey.oriel.ox.ac.uk/~mike/zrm/
- ***B:***
  - The B-Book: Assigning Programs to Meanings, J.-R. Abrial, Cambridge University Press, 1996
  - http://www.afm.sbu.ac.uk/b/
- ***DOORS:***
  - http://www.telelogic.com

Requirement Specification

- Requirement analysis must characterise
  - ***System under development*** (***SUD***)
  - ***Environment*** where SUD will be deployed
    - Physical characteristics
    - Max. interrupt rates
    - *etc.*
- Requirement specification is typically ***most critical*** design phase!
  - ... and, at the same time, often the phase done rather carelessly – so better pay attention here ...



Architectural Design

- Systems typically too *complex* to be designed at once
  - Design of large (RT) system must be structured
- ***Decomposition***
  - Systematic breakdown of complex system into smaller and smaller parts
  - Want to isolate components that can be understood and designed by individuals/small groups
- ***Abstraction***
  - Postpone consideration of details
  - In particular, try to abstract from implementation specifics
  - Simplifies view of system

- Taken together, the two complementary approaches of *decomposition* and *abstraction* form the basis of most common SW engineering methodologies

Encapsulation

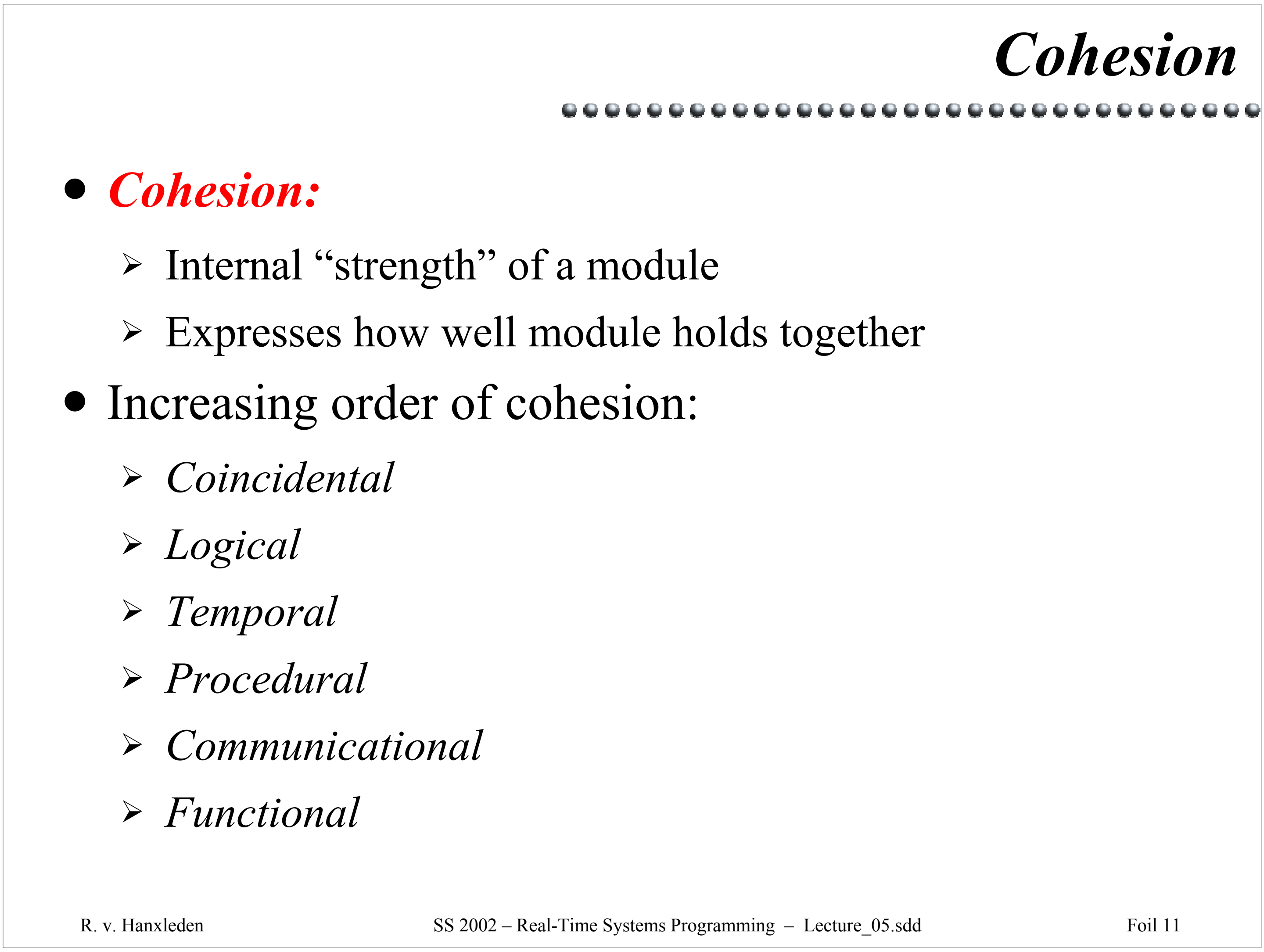
- *Decomposition* of system:
  - Specification/design of *subsystems*
- *Abstraction*:
  - Subsystem must have well-defined roles
  - Interfaces must be unambiguous
- ***Compositional*** decomposition:
  - Entire system can be verified just in terms of subsystem specifications

Encapsulation

- *Sequential* programs particularly amenable to compositional methods
  - ***Simula*** – introduced **class** construct
  - ***Modula 2*** – uses **module** structures
  - OO-languages (***C++***, ***Java***, ***Eiffel***) – build upon class construct
  - ***Ada*** – combination of modules and type extensions
- ***Objects***
  - Provide abstract interface
  - In *concurrent* environments, require extra facilites – namely, the **process** abstraction

How to Decompose a System?

- *Objects* and *processes:*
  - Provide good encapsulation facilities for system decomposition
- The remaining, critical question:
  - Into which parts should a system be decomposed?
- ***Approach:***
  - Evaluate prospective decompostions wrt certain *metrics*
- Here, strive for decompositions that have
  - Strong *coherence*
  - Loose *coupling*



Cohesion

- ***Cohesion:***
  - Internal “strength” of a module
  - Expresses how well module holds together
- Increasing order of cohesion:
  - *Coincidental*
  - *Logical*
  - *Temporal*
  - *Procedural*
  - *Communicational*
  - *Functional*

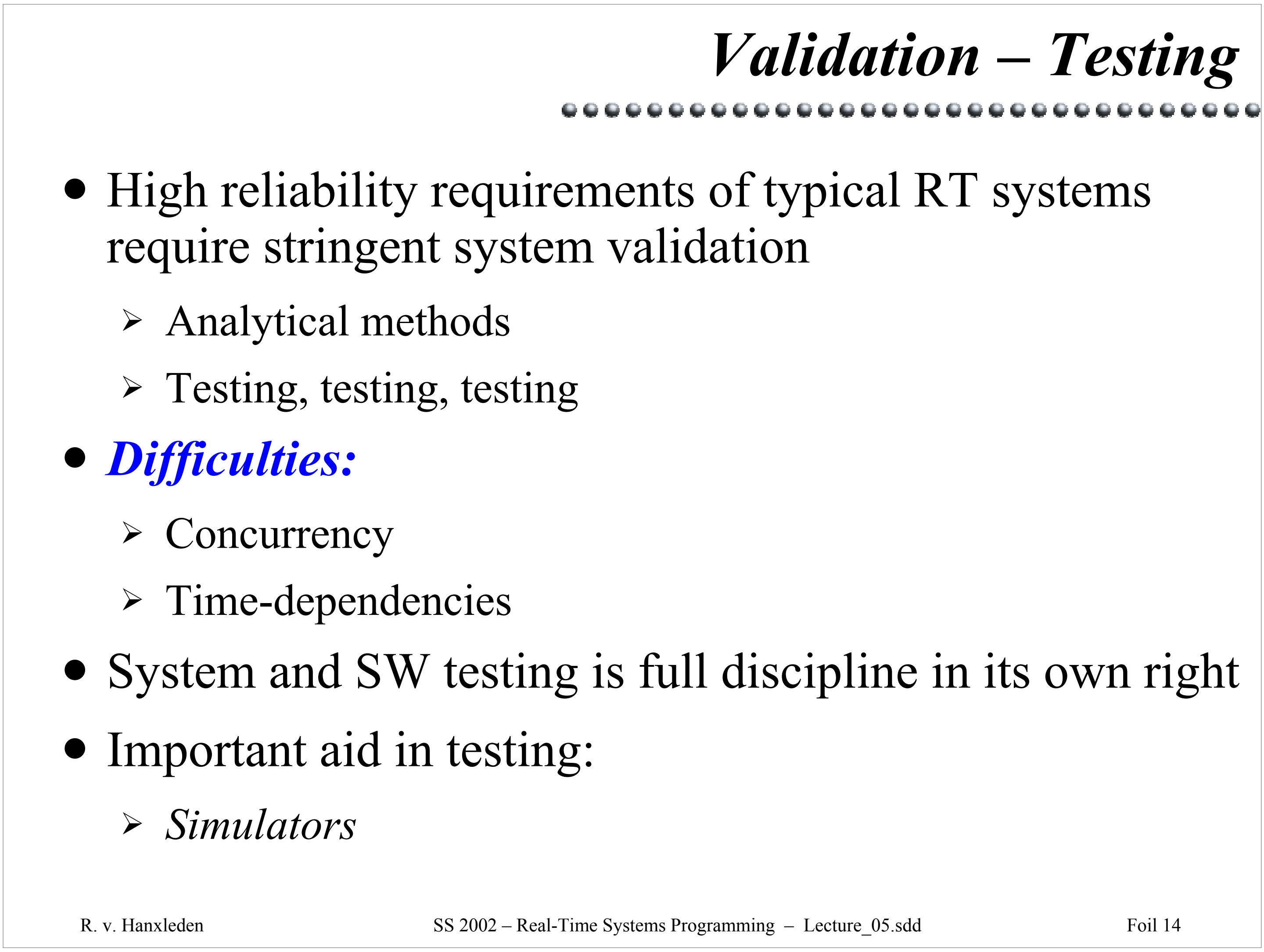
- See S. Allworth and R. Zobel, *Introduction to Real-Time Software Design*, MacMillan, 1987
- *Coincidental*
  - E.g., modules written in same month
- *Logical:*
  - Elements related in terms of overall system, but not in terms of actual SW (e.g., all device drivers)
- *Temporal*
  - Elements executed at similar times, e.g. start-up routines
- *Procedural*
  - Elements used in same program section; e.g., user interface
- *Communicational*
  - Elements working on same data structure
- *Functional*
  - Contribute to single system function

Coupling

- ***Coupling:***
  - Measure of interdependence of program modules
- Modules exchanging control information:
  - *High/tight* coupling
- Modules exchanging data:
  - *Loose* coupling
- Degree of coupling typically also expresses how easy it is to replace a component within system

Implementation

- System design should lead naturally to implementation
- However, proper design in practice still requires knowledge of what is possible at implementation level
    - ➢ This also true when using auto-coding!
- Implementation languages for real-time systems are a focus of this course
    - ➢ Will subsequently give language overview
- ***One aim:*** To sharpen your vision to distinguish the essentials from syntactic embellishments



Validation – Testing

- High reliability requirements of typical RT systems require stringent system validation
  - Analytical methods
  - Testing, testing, testing
- ***Difficulties:***
  - Concurrency
  - Time-dependencies
- System and SW testing is full discipline in its own right
- Important aid in testing:
  - *Simulators*

- *See* http://www.mtsu.edu/~storm/

Simulators

- ... are programs that imitate environment in which RT SW is embedded
  - I/O
  - Interrupts
- ***Advantages:***
  - Reproducibility
  - Can stop and resume operations
  - Safety
- Costs are often significant

Prototyping

- ... is the construction of a “mock-up” of the final system
  - Must be cheaper than building final system
  - Often does not exhibit dependability characteristics of final system
- Helps to ensure that requirements specification is ...
  - ... really what customer desired
  - ... complete

Summary

- The *requirements specification* is a critical phase in system development
  - For RT systems, requirements should include their *temporal* behavior, *dependability* requirements, and the behavior in case of component *failure*
- The *architectural design* of a system should result in a decomposition that exhibits *strong coherence* and *loose coupling*
- The *validation* of a RT system is often complicated by their *concurrent nature* and their RT dependencies
- *Testing* is a crucial part of validation, often aided by *simulators*

To Go Further

- ***System Design***
  - ☞ [Burns and Wellings 2001] – Chapter 2
- ***Software Testing***
  - ☞ http://www.mtsu.edu/~storm/