Real-Time Systems Programming

Summer-Semester 2002
Lecture 3
18 April 2002

System Engineering

The 5-Minute Review Session

1) What is a *real-time system*?
2) What are the *characteristics* of real-time systems?
3) What is an *embedded system*?
4) What types of *deadlines* do we distinguish?

- As you probably already know, the chances of transferring some material into long term memory increase significantly if you see this material not just once
- This kind of review session will be a regular fixture of the lecture – the main purposes being
  - to remind you of where we left off during the last lecture
  - to recall some of the concepts that will be needed in this and subsequent lectures
  - and perhaps also, to give a bit of an incentive to have a look at earlier class notes
- This warm-up really should not take significantly more than 5 minutes – but this also depends on you ... ☺

Overview

1) Temporal Requirements
2) System Design

Where are we?

1) ***Temporal Requirements***
   - ***The RT controller designer's view***
   - ***The system designer's view***
   - ***The synergetic approach***
2) System Design

Temporal Requirements

Where do temporal requirements come from ?

- Determining these requirements mostly done in ad-hoc fashion – no unified, objective method yet
- Different views:
  - RT controller design
  - Controlled system design
  - Synergistic approach

The RT Controller Designer's View

- Start from pre-existing functional and timing specification, provided by the customer
- Focus on
  - Task scheduling
  - Operating systems
  - Communication protocols etc.
- Prevalently a binary view of timeliness:
  - Time-value function assumed to be a step function
  - Most scheduling algorithms based on this assumption
  - No concept of timing tolerance or graceful degradation

The System Designer's View

- Tend to use (mathematical) model to describe behavior of controlled system
- Focus on
  - ***Stability analysis*** of controlled processes
  - ***Grace time***: tolerance against controller malfunctioning
  - ***Reversibility***: ability to recover from erroneous commands
  - ***Safe state***: non-dangerous system attitude, reachable by passive means

The Synergetic Approach

- Models the controlled system and the controller as single unit
- Concept of ***accomplishment levels*** exhibited by total system
- Obtaining hard deadlines by considering
  - Allowed state-space boundaries
  - Admissible inputs
  - Actual state-space placement as function of time
- *... this is still research!*

Where are we?

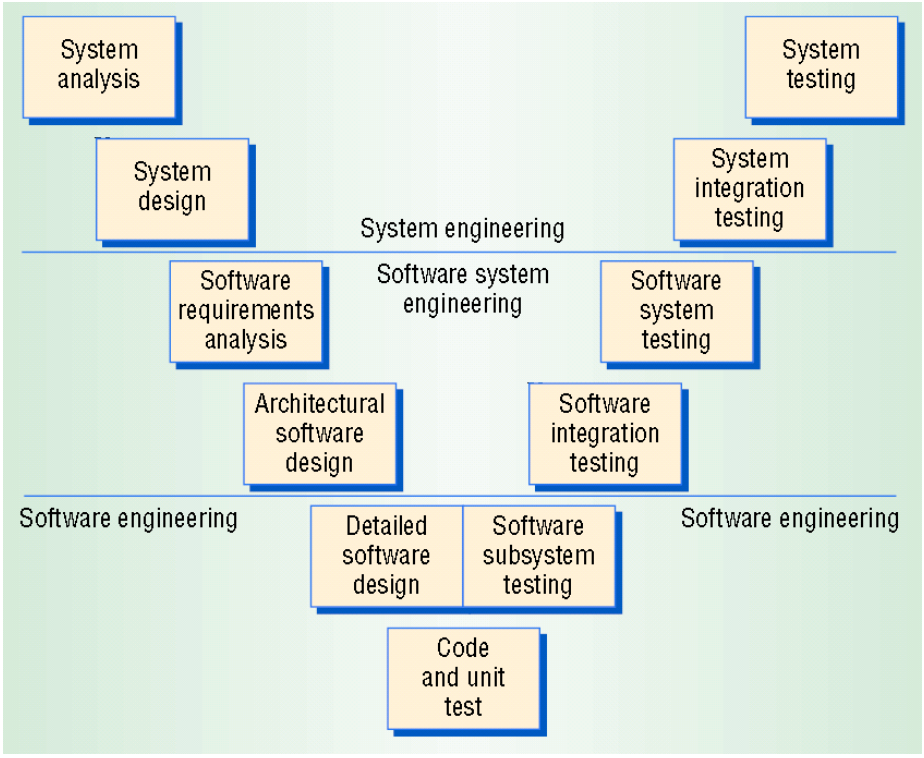
1) Temporal Requirements
2) ***System Design***
   - ***System engineering***
   - ***Software (System) engineering***
   - ***Notation***
   - ***Requirements***

Designing RT Systems

- Most important stage in development of RT system:
  - Consistent *system design* that satisfies requirement specification
- This is no different in non-RT systems
  - However, RT systems often pose fundamental design problems due to their ***complexity***

Designing RT Systems

- A hierarchy of design levels:



IEEE Micro

- *See* Richard H. Thayer, Software System Engineering: A Tutorial, *Computer*, April 2002 (Vol. 35, No. 4)

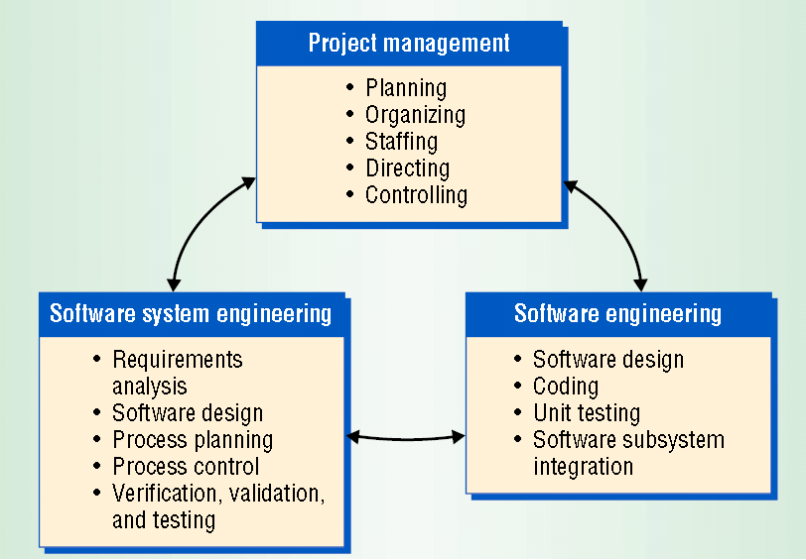
Designing RT Systems

- ***System engineering*** produces *documents*, not hardware or software
  1. Problem definition
  2. Solution analysis
  3. Process planning
  4. Process control
  5. Product evaluation

Designing RT Systems

- ***Software engineering:***
  - Methods and tools to ensure that
    - SW production process is *managable*
    - Constructed programs are *reliable* and *correct*
- SW engineering per se not focus of this course
  - Will only briefly review some general concepts
- Will focus on language and OS primitives that allow *realisation* of designs

- ***Software system engineering:***
  - Can be viewed as separate from SW engineering
  - Establishes base line for SW engineering and evaluates results



IEEE Micro

Designing RT Systems

- Design is typically done top-down
  - However, need understanding at higher level of what is feasible at lower levels
- Essentially, design methods are series of transformations
  - requirements $\Rightarrow$ executable code

Notations

- Each design stage must be documented in a certain *notation*
- Distinguish
  - *Informal*
  - *Structured*
  - *Formal*
- Also distinguish
  - *Graphical*
  - *Textual*

- See
  - [Burns and Wellings 2001], Chapter 2
  - J. Mcdermid, Assurance in high integrity software, *in* C. Sennett (ed.), *High Integrity Software*, Pitman, 1989

Notations

- ***Informal*** notations:
  - Natural language
  - (Imprecise) diagrams
  - ***Pro:*** understood by large group of people
  - ***Con:*** interpretations may vary

Notations

- ***Structured*** notations:
  - Often use *well-defined* graphical representation
  - Can be quite rigorous
  - However, cannot be analyzed or manipulated
- ***Formal*** notations:
  - Have mathematical basis
  - ***Pro:*** Can prove certain properties
  - ***Con:*** Typically hard to understand by the uninitiated
- Dependability requirements of RT systems cause move towards structured and formal notations

- Once again, this is a taxonomy that gives room for debate on where exactly to place certain items. For example, how would you classify a UML class diagram? Arguments can be made either way. Nevertheless, it is helpful to be aware of these broad classifications, and the trade-offs to be made when choosing a notation

Requirements

- According to Gomaa (1994), an RT system design method should be capable of
  1. Structuring system in concurrent tasks
  2. Supporting reusability through information hiding
  3. Defining behavior with finite-state machines
  4. Analysing RT properties of design

- *See* H. Gomaa, Software design methods for the design of large-scale real-time systems, *Journal of Systems and Software,* **25**(2):127-146, 1994

Summary

- A significant characteristic of RT systems are their *temporal requirements*
  - However, a systematic process for deriving these requirements does not exist yet
  - In practice, different approaches are the *RT controller designer's approach*, and the *system designer's approach*

Summary

- The *system* design is a critical aspect in the development of *complex* RT applications
- Can distinguish
  - System engineering
  - System SW engineering
  - SW engineering
- SW engineering is concerned with the transformation of requirements to executable code
- Distinguish informal, structured, and formal notations for the different design stages
- In this course, will focus on language and OS primitives that allow *realisation* of designs

To Go Further

- ***Temporal Requirements:***
  - ☞ A. P. Magalhães, “A Survey on Estimating the Timing Constraints of Hard Real-Time Systems,” *Design Automation for Embedded Systems*, vol. 1, no. 3, pp. 213-230, July 1996, Kluwer Academic Publishers, Boston
- ***System Design***
  - ☞ [Burns and Wellings 2001]
  - ☞ Richard H. Thayer, Software System Engineering: A Tutorial, *Computer*, April 2002 (Vol. 35, No. 4)