Software Development Life Cycle

Gruia-Catalin Roman

October 2014
Department of Computer Science
University of New Mexico

Overview

- Software Engineering
- Basic Concepts
- System Development Life-cycle
- Software Development Methodologies
- Total System Design

Software Engineering

- ***Software engineering*** is a discipline concerned with the formulation, evaluation, and application of systematic development, maintenance, enhancement, and analysis processes to systems which include substantial software components
- The term was first introduced in 1968 at a NATO conference in Garmisch, West Germany
- The discipline emerged in response to a generally perceived software crisis
  - rising software costs
  - inadequate software quality
  - increased software criticality

Current State of Affairs

- Better understanding of basic software development methods
- Success in narrow application areas (e.g., compilers, user-oriented programming)
- Difficulties in addressing total system issues
- Practically no usage of formal methods
- Limited application of basic proven techniques
- Limited tool availability and usage

Basic Concepts

- Requirements
- Design
- Implementation

- Formality
- Abstraction
- Modeling

Requirements

- Functional requirements define the interactions between a component and its environment, i.e., *what* the component must do
- Non-functional requirements restrict the range of possible realizations by defining a set of acceptability criteria, e.g., performance, cost, etc.
  - The severity of the non-functional requirements significantly affects the complexity of the realization
  - Assumptions regarding the component's environment can significantly reduce the complexity of the realization

Design

- A design defines the structure and the behavior of some component and establishes requirements for its subcomponents
- The design is affected by the component's requirements and by the available technology
- The design must be functionally correct and must satisfy all non-functional requirements
- The design must be feasible given current technology

Implementation

- An implementation is an operational component manufactured in accordance with the design
- The implementation must exhibit the same basic structure and behavior as the design

Example:
Message Traffic Merging

- Functional Requirements
  - Messages arriving on lines 1 and 2 must be forwarded on line 3
  - The environment guarantees that no new messages arrive before the current message is completely forwarded
- Design
  - Software solution: one task with two interrupt-driven entries
  - Hardware solution: one *OR* gate

Example: Image Blurring

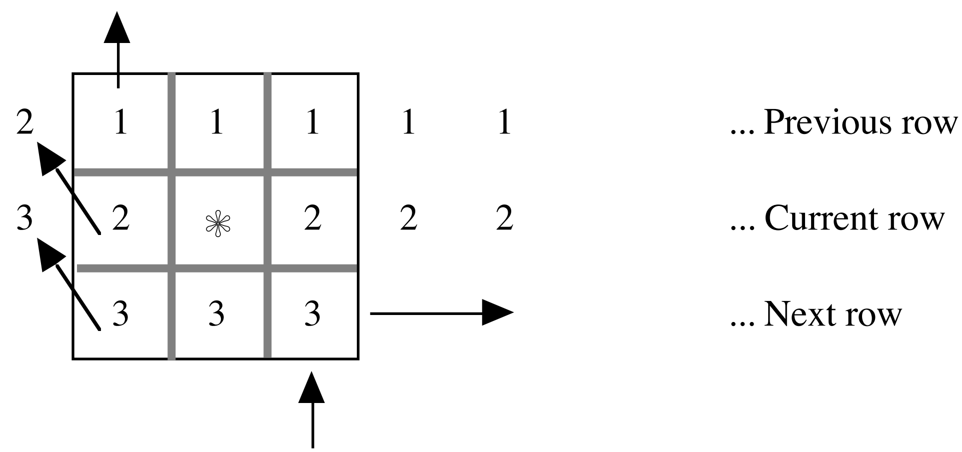
- Functional Requirements
  - Blur an image I consisting of N x N pixels stored in row major form

$$i,j : 1<i,j<N :$$
$$I'(i,j) = (I(i-1,j)+I(i+1,j)+I(i,j-1)+I(i,j+1))/4$$

  - The size N is bound by N_{max}
  - The input and output images are stored on disk

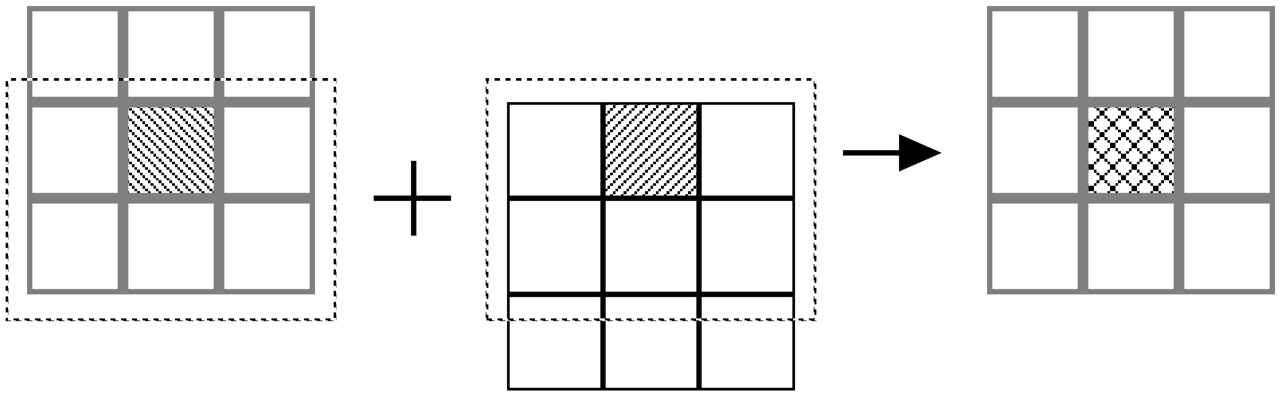
Example: Image Blurring

- <u>Design</u>
  - Limited main memory *(2*N_{max}+K)*



Example:
Image Blurring

- <u>Design</u>
  - Pipelined architecture



i,j : 1<i,j<N : I'(i,j) := I'(i,j) + I(i-1,j)

Formality

- Formal notation is essential to precise communication
- Typical informal statement of the problem
  - *Sort N numbers*
- Sample formal statement of the problem
  - *Let N_i and M_i be the input and output values, respectively, with $(1 \leq i \leq K)$*

$$i,j : 1 \leq i,j \leq K : i < j \Rightarrow M_i < M_j$$

- <u>Question</u>: *Is this what was intended?*

Abstraction

- An abstraction is a set of objects that share some common properties
  - Redness—all red objects
  - People—all human beings
  - Requirement specification—all correct realizations
  - Design diagram—all systems sharing that particular structure
- Abstractions hide details and focus on essential common properties
- Commonly used abstractions in software development
  - *procedural abstractions* (e.g., pre- and post-assertions)
  - *data abstractions* (e.g., abstract data types)
- All such abstractions must have finite representations

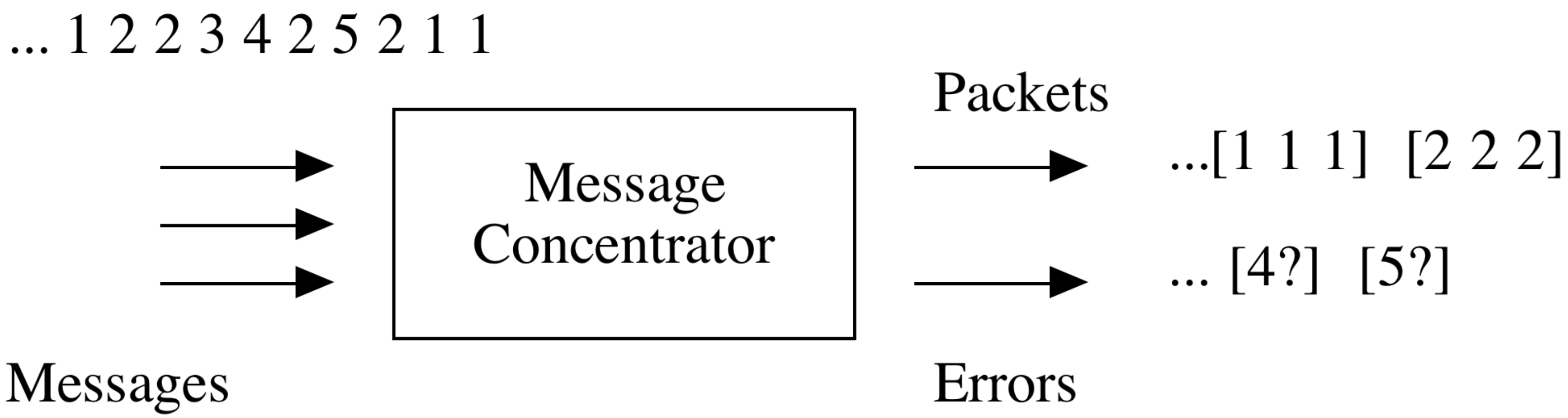
Working with Abstractions

- Most software development work involves the creation and manipulation of hierarchies of abstract objects
- Successful software development often depends upon
  - Identifying a clean abstraction of the problem (simple to understand, easy to name)
  - Finding the right abstraction
  - Working at the right level of abstraction
- Illustrations: human interfaces and the analysis of structure charts

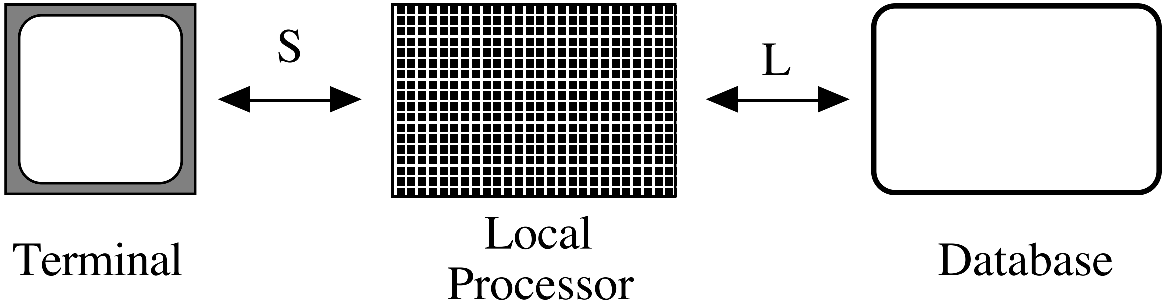
Modeling

- A model is an abstraction whose purpose is to facilitate analysis of complex objects
- An object M is a valid model for an object X with respect to some property P if M is an abstraction of X and if M allows one to make correct judgments about the property P of X by studying M
- Properties of interest in software development include
  - behavior
  - data usage patterns
  - response time
- Commonly used modeling techniques include
  - mathematical models
  - discrete event simulation
  - functional simulation
  - benchmarking

Example:
Message Concentrator



Example:
Response Time Estimation



Request_size/L + DB_time + Max(Reply/L, Reply/S)

System

- A <u>system</u> is a software/hardware aggregate
- A typical system involves concurrent processing across a set of locally or geographically distributed processors
- System level issues involving software/hardware tradeoffs, concurrency, and real-time are the premier challenges in today's software development
- Premature hardware commitment is a common failure point for many systems

Life Cycle

- The <u>system development life-cycle</u> consists of all development, operation, maintenance, enhancement, and evaluation activities related to a system from the time of its inception up to its retirement or replacement
- Short-range focus on development alone fails to recognize the substantial (often dominant) costs incurred during the remainder of a system's life
- Software development must be treated as capital investment

Software Development Methodologies

- A methodology is a mode of procedure to be followed in solving a given class of problems
- A methodology exploits particular features of the problem and problem solver through the use of selected techniques and tools
- Effective methodologies are application, organization, and technology specific
- Software engineering is concerned with methodologies for software development

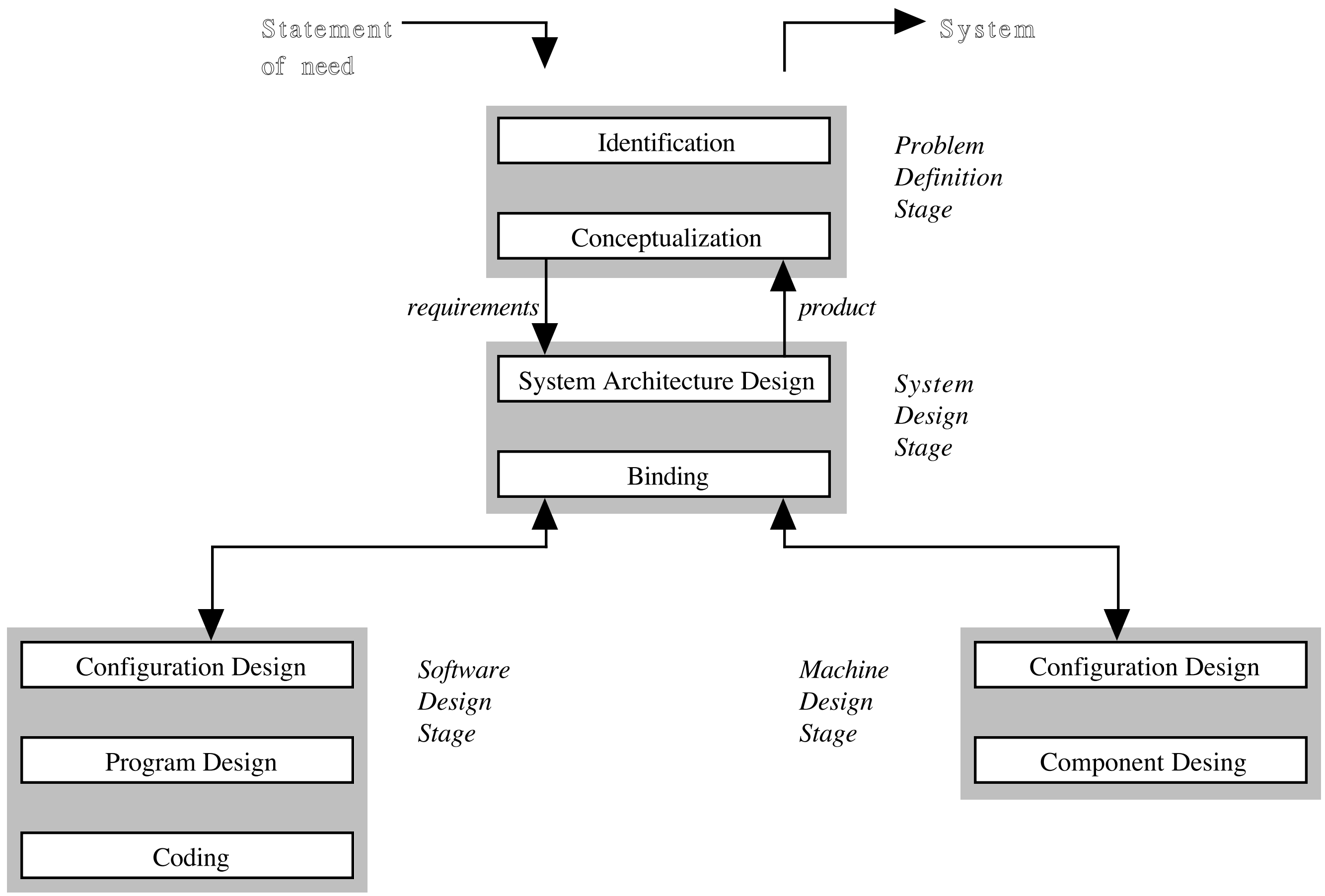
Methodological Objectives

- Quality assurance (e.g., technical reviews)
- Control (e.g., structured design processes)
- Planning (e.g., measurement and evaluation)
- Productivity (e.g., reusability of design know-how)

Total System Design Framework

- The Total System Design (TSD) Framework is an abstraction over a class of methodologies
- The TSD Framework emphasizes a balanced treatment of the role of software and hardware in system development
- The TSD Framework identifies fundamental tasks (called stages, phases, and steps) that must be accomplished during system design but not the exact sequencing of design activities
- The criterion for separation into stages and phases is the *technical expertise required to perform the particular activity*
- The TSD Framework may be used as the basis for methodology development

TSD Framework Structure



Problem Definition Stage

- Identification Phase
  - It explores and elaborates the statement of need
- Conceptualization Phase
  - It formalizes the functional and non-functional requirements
  - It specifies all external interfaces

System Design Stage

- System Architecture Design Phase
  - It defines the software and hardware architectures and the allocation of software to hardware
  - It specifies all (software and hardware) interfaces across machines
- Binding Phase
  - It generates the software and hardware requirements for custom, customized, and off-the-shelf components

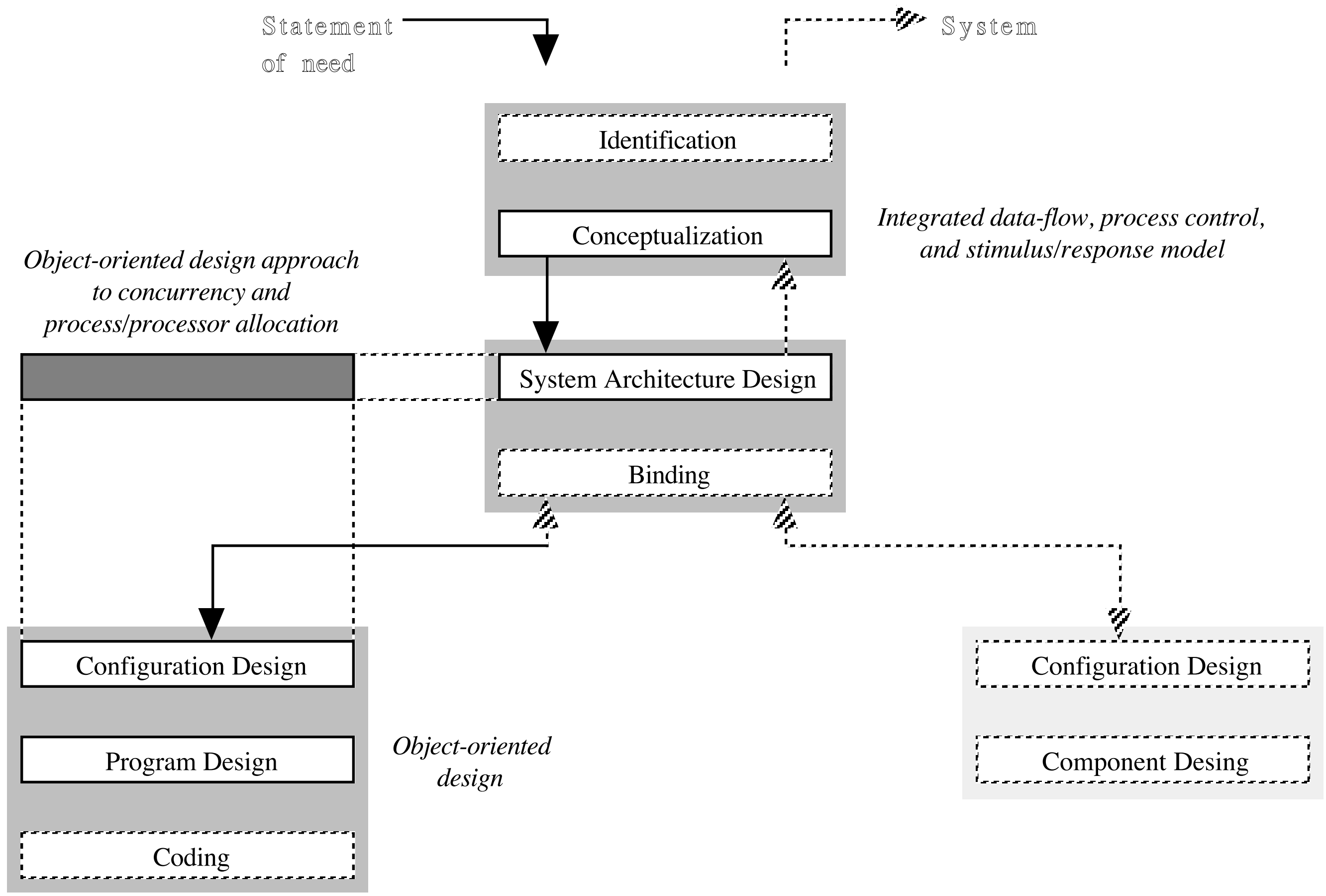
Software Design Stage

- <u>Software Configuration Design Phase</u>
  - It defines the architecture of the concurrent software residing on each processor
- <u>Program Design Phase</u>
  - It specifies the data organization and algorithms for all sequential software components
- <u>Coding Phase</u>
  - It manufactures the code

Illustration: Real Time Control

- High-speed and low-level input/output
- Specialized hardware and devices
- Concurrent processing
- Interrupt-driven and periodic processing
- Critical timing constraints
- Custom architectures consisting of off-the-shelf processors

Custom Designed Methodology



Reading List

- Roman, G.-C. et al., "A Total System Design Framework," *Computer* 17, No. 5, May 1984, pp. 15-26.
- Wiener, R., and Sincovec, R., *Software Engineering with Modula-2 and Ada*, John Wiley & Sons, 1984.