Computer Systems: Evaluating Correctness & Performance - I

Presenter: Sandeep K. S. Gupta

Reference:

-Software Engineering, Ian Sommerville, 7th Ed..

-Computer Organization and Design: The Hardware/Software Interface, David A. Patterson and John L. Hennessy

- An Engineering Approach to Computer Networking, S. Keshav

Arizona State University

School of computing, informatics, and decision systems engineering

Agenda

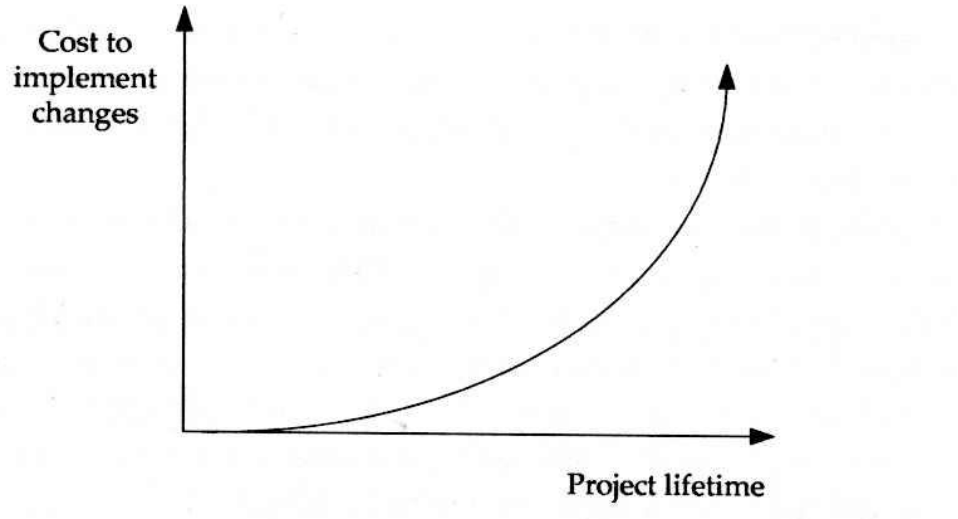
- Computer System Design Process
- Requirement Specification
- Constraints & Standards
- Computer System Evaluation
  - Correctness
  - Performance
- Models (Abstractions of System)
- Verification and Validation
- Testing & Debugging
- Static Analysis

System Design Process

- Steps
  - Problem Identification
  - Research Phase
  - Requirements Specification
  - Concept Generation
  - Design Phase
  - Prototyping Phase
  - System Integration
  - Maintenance Phase
- System Evaluation
- System Evolution (Optimization)

Cost of Design Change

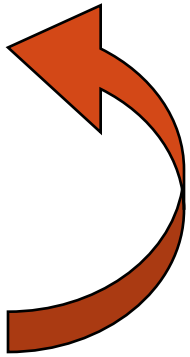
- Costs increases exponentially as the project life increases



Source: Design for ECE Engineers, Ford & Coulston

Performance analysis and tuning

- Steps
  - measure
  - characterize workload
  - build a system model
  - analyze
  - implement



System Requirement Specification

- System Specification – A comprehensive functional and technical description of the system
- Requirement Specification
  - Precise and clear description of the system and its external interface
    - Functional
    - Performance –throughput, latency etc.
    - Quality: reliability, portability etc.
- Requirement versus Design
  - "What" versus "How"
  - Requirement analysis – goal is "understanding"
  - Design analysis – goal is "optimization"
  - Usually, Customer is interested only in the requirements and not design.

Example System Requirements

- Performance and Functionality
  - Will identify skin lesions with a 90% accuracy
  - Should be able to measure within 1mm
- Reliability
  - Operational 99.9% of the time
  - MTBF (Mean Time Between Failure) of 10 years
    - Failure rate = 1/MTBF
- Energy
  - Average power consumption of 2 watts
  - Peak current draw of 1 amp

Constraints

- Economic
- Environmental
- Ethical and Legal
- Health and Safety
- Manufacturability
- Political and Social – FDA, language?
- Sustainability

Standards

- Examples – RS-232, TCP/IP, USB
- Types
  - Safety
  - Testing
  - Reliability
  - Communications
  - Documentation
  - Programming Languages

Various Methods of "Evaluating" Computer System

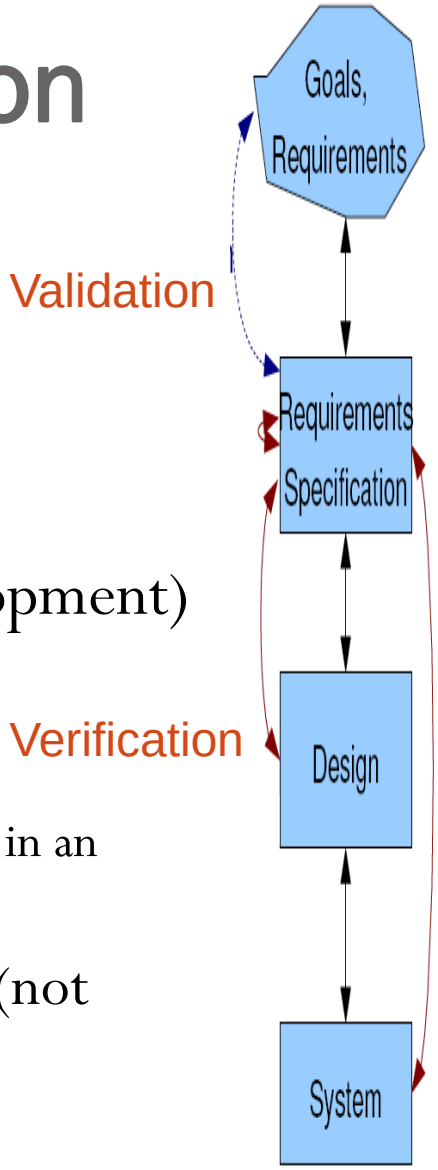
- Correctness
  - Verification & Validation
  - Testing
  - Formal Methods
  - Debugging
- Performance Evaluation
  - Simulation (Emulation)
  - Analytical Modeling
    - Queuing Models
  - Experimental Evaluation
    - Performance Metrics
    - Performance Measurement
  - Performance Tuning

Models

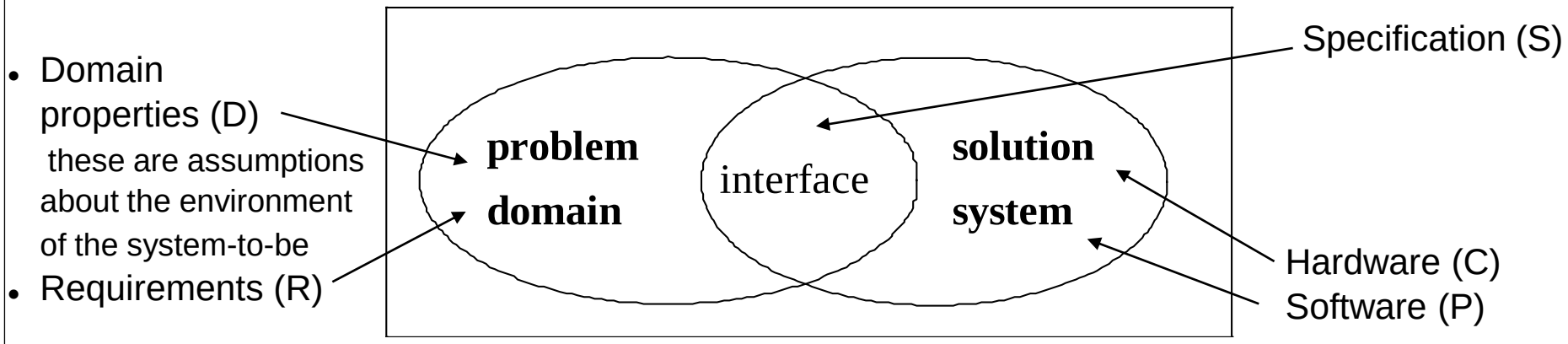
- Model: A model is an abstraction of a system or a process
- Types:
  - Behavioral model: abstracts system or a process as a black box, with inputs, outputs, and a transfer function specifying the relation between them.
  - Architectural model: abstracts system in terms of its components and connections between them. The connections may indicate flow of data or control signals from one component to the other or simply a subcomponent relationship. E.g. client-server, peer-to-peer
  - Formal model: abstracts system operation in terms of discrete states and transitions between them. Formal models are of different types – finite state automata, timed automata, and hybrid systems.

Verification versus Validation

- Verification: " Are we building the system right"
  - The system should conform to its specification
- Validation: "Are we building the right system"
  - The system should do what the "user" really wants
- V&V process (applied to each stage of system development)
  - Objectives
    - Discovery of defects in the system
    - Assessment of whether or not the system is useful and useable in an operation situation
  - Establishes whether the system is "fit" for the purpose (not necessarily free of defects.)



Formal View [1]



- **Validation question** (do we build the right system?) : if the domain-to-be (excluding the system-to-be) has the properties D, and the system-to-be has the properties S, then the requirements R will be satisfied.

$$D \text{ and } S \Rightarrow R$$

- **Verification question** (do we build the system right?) : if the hardware has the properties H, and the software has the properties P, then the system requirements S will be satisfied.

$$C \text{ and } P \Rightarrow S$$

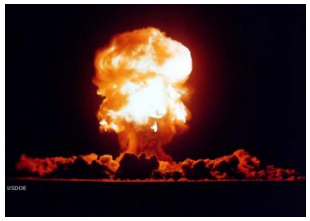
- Conclusion:

$$D \text{ and } C \text{ and } P \Rightarrow R$$

[1] M. Jackson, 1995

Verification

- Two well established approaches to verification
  - Model Checking
  - Theorem Proving
- Model checking
  - Build a finite model of system and perform an exhaustive search
  - Completely Automatic
  - Produces counter examples to show subtle error in design
  - State Explosion problem



- Theorem Proving
  - Mechanization of a logical proof
  - Process of finding a "proof" from the "axioms" of the system
  - As opposed to model checking can deal with infinite state space.

Verification and formal methods

- Formal methods can be used when a mathematical specification of the system is produced.
- They are the ultimate static verification technique.
- They involve detailed mathematical analysis of the specification and may develop formal arguments that a program conforms to its mathematical specification.

Formal methods: Pros & Cons

- Pros:
  - Producing a mathematical specification requires a detailed analysis of the requirements and this is likely to uncover errors.
  - They can detect implementation errors before testing when the program is analyzed alongside the specification.
- Cons:
  - Require specialized notations that cannot be understood by domain experts.
  - Very expensive to develop a specification and even more expensive to show that a program meets that specification.
  - It may be possible to reach the same level of confidence in a program more cheaply using other V & V techniques

Computer System Testing

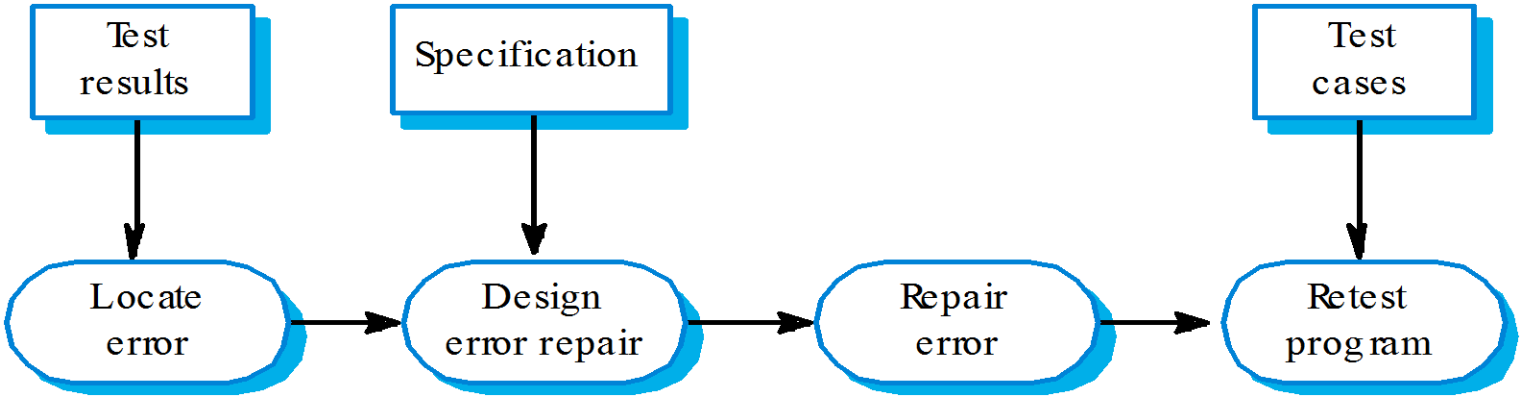
- Evaluating system in a simulated or real environment using carefully selected inputs.
- Goals
  - Detect faults
  - Establish confidence in the system
  - Evaluated system properties
    - Performance
      - CPU, Memory, Network etc.
    - Reliability
    - Security
    - Usability

Types of Testing

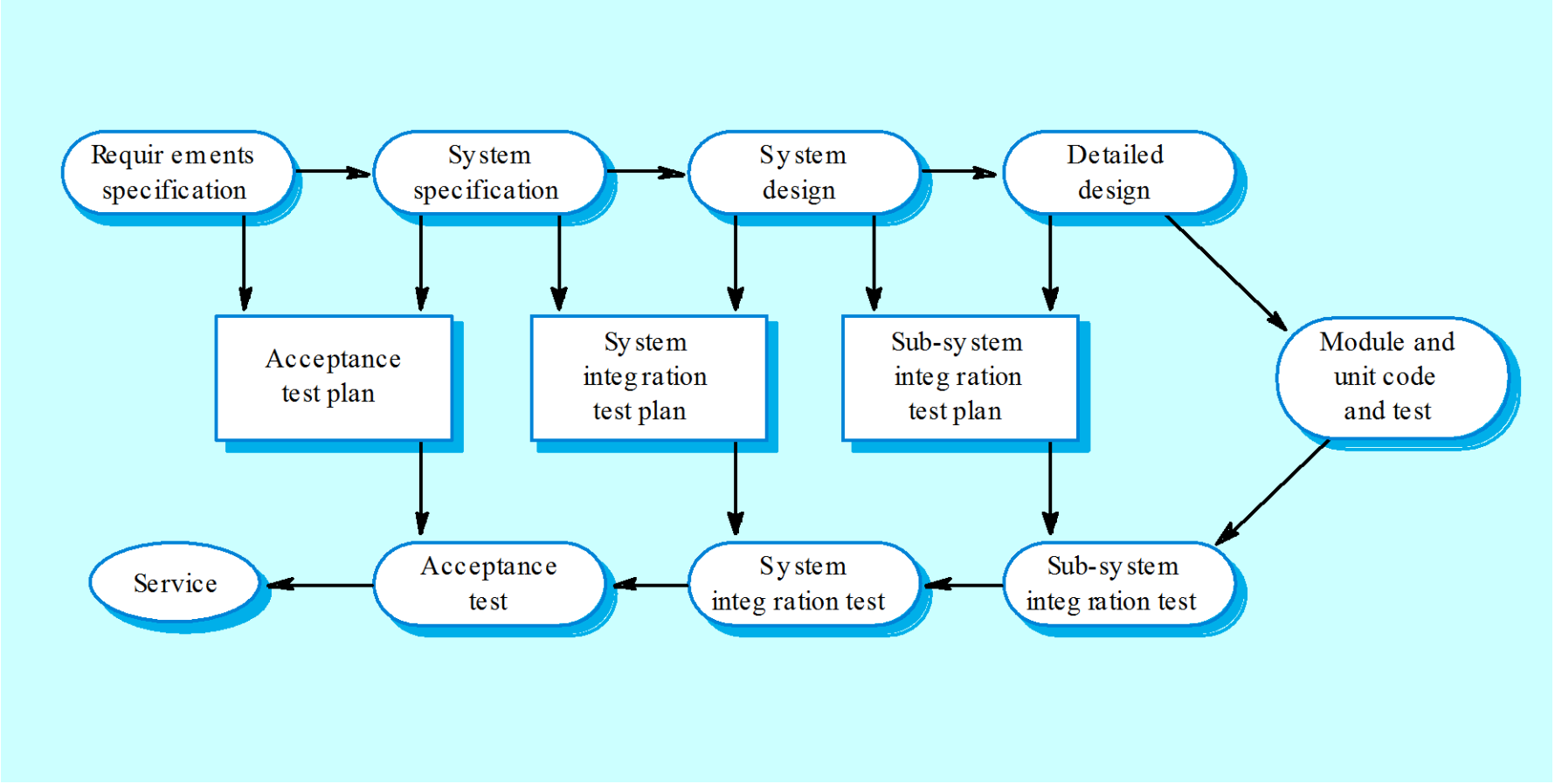
- Defect testing
  - Tests designed to discover system defects.
  - A successful defect test is one which reveals the presence of defects in a system.
- Validation testing
  - Intended to show that the software meets its requirements.
  - A successful test is one that shows that a requirements has been properly implemented.

Testing and debugging

- Defect testing and debugging are distinct processes.
- Verification and validation is concerned with establishing the existence of defects in a program.
- Debugging is concerned with locating and repairing these errors.
- Debugging involves formulating a hypothesis about program behaviour then testing these hypotheses to find the system error.
- Debugging Process:



Example: The V-model of development



Automated static analysis

- Static analysers parse the program text and try to discover potentially erroneous conditions.

Fault class	Static analysis check
Data faults	Variables used before initialisation Variables declared but never used Variables assigned twice but never used between assignments Possible array bound violations Undeclared variables
Control faults	Unreachable code Unconditional branches into loops
Input/output faults	Variables output twice with no intervening assignment
Interface faults	Parameter type mismatches Parameter number mismatches Non-usage of the results of functions Uncalled functions and procedures
Storage management faults	Unassigned pointers Pointer arithmetic

Stages of static analysis

- Control flow analysis. Checks for loops with multiple exit or entry points, finds unreachable code, etc.
- Data use analysis. Detects uninitialised variables, variables written twice without an intervening assignment, variables which are declared but never used, etc.
- Interface analysis. Checks the consistency of routine and procedure declarations and their use

Example: LINT static analysis

```
138% more lint_ex.c
#include <stdio.h>
printarray (Anarray)
 int Anarray;
{    printf("%d",Anarray);  }

main ()
{
 int Anarray[5]; int i; char c;
 printarray (Anarray, i, c);
 printarray (Anarray) ;
}

139% cc lint_ex.c
140% lint lint_ex.c

lint_ex.c(10): warning: c may be used before set
lint_ex.c(10): warning: i may be used before set
printarray: variable # of args. lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) :: lint_ex.c(10)
printarray, arg. 1 used inconsistently lint_ex.c(4) :: lint_ex.c(11)
printf returns value which is always ignored
```

Use of static analysis

- Particularly valuable when a language such as C is used which has weak typing and hence many errors are undetected by the compiler,
- Less cost-effective for languages like Java that have strong type checking and can therefore detect many errors during compilation.

Summary

- System Development involves many distinct steps
- Each step has its own theory and tools
- It is important to keep the overall system goals to determine important of each step of system development process.
- Some requirements can be quantified while others cannot!
  - Hence there is both art and science behind system development