Object Oriented Analysis and Design with Java

UE19CS353

Dr H L Phalachandra and Prof. Sudeepa Roy Dey
Department of Computer Science and Engineering

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

PES UNIVERSITY ONLINE

UE19CS353: Object Oriented Analysis and Design with Java

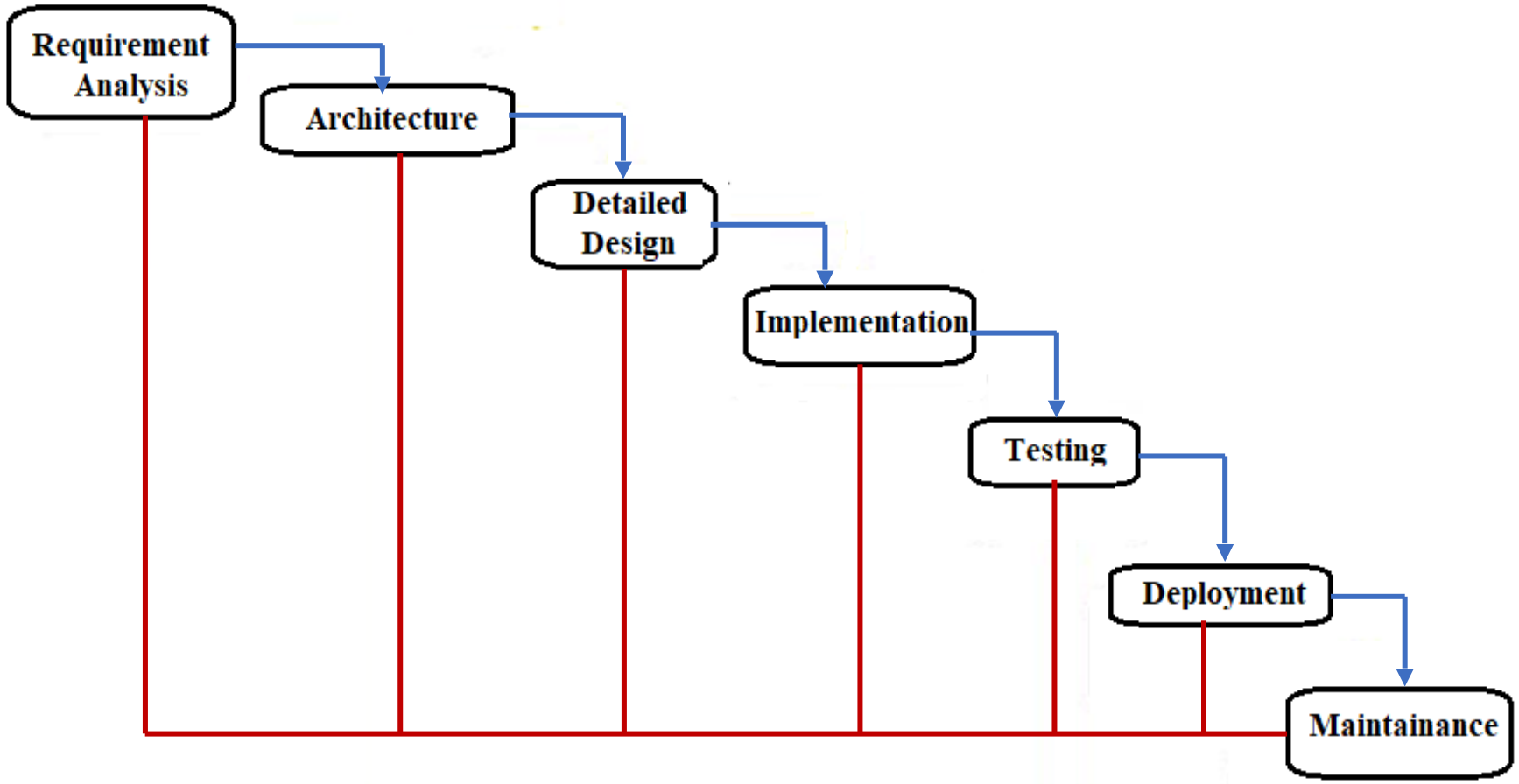
Recap on SDLC/Architecture Techniques

Prof Sudeepa Roy Dey
Department of Computer Science and Engineering

Agenda

- Recap of SDLC model
- Discussion on OOAD as part of SDLC
- Recap of Architecture & design characteristics/techniques

Waterfall Model



Waterfall Model

Advantages

- Simple
- Clear Identified phases as shown and easy to departmentalize and control
- Easy to manage due to the rigidity of the model
- Each phase has specific deliverables and a review process

Dis-Advantages

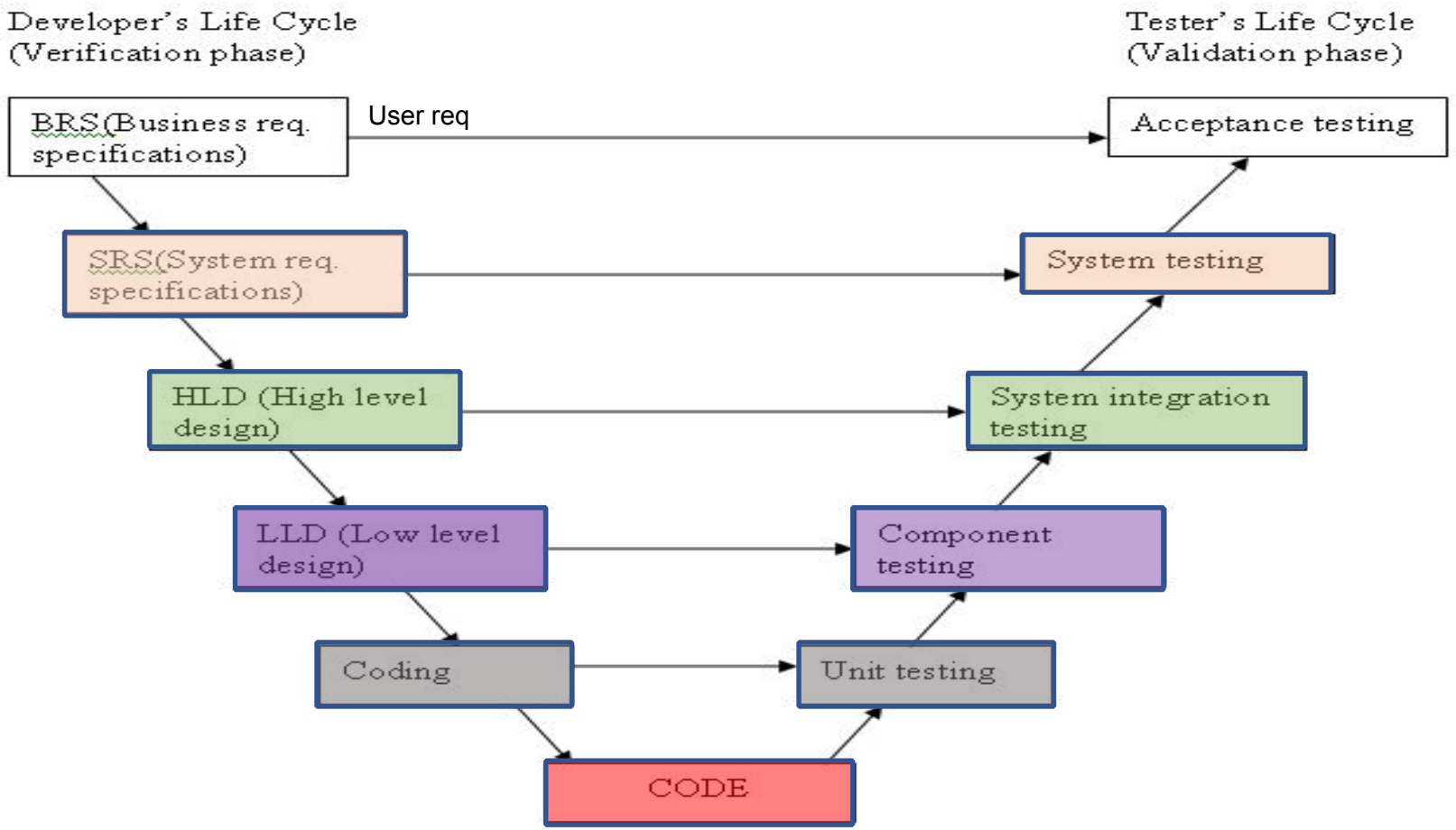
- Assumes requirements (both user & hardware) can be frozen
- Difficult to accommodate change and hence not very flexible
- Sequential. Move from one phase to another only after the completion of

Waterfall Model

Usage

- Used (in pure form) for short projects where requirements are well understood
- Variants are used at a high level in large projects
- Product definition is stable and technology understood
- Variants used in large globally developed projects with other LC

V Model


Developer's Life Cycle
(Verification phase)
Tester's Life Cycle
(Validation phase)
BRS(Business req. specifications)
User req
Acceptance testing
SRS(System req. specifications)
System testing
HLD (High level design)
System integration testing
LLD (Low level design)
Component testing
Coding
Unit testing
CODE

V Model

Advantages

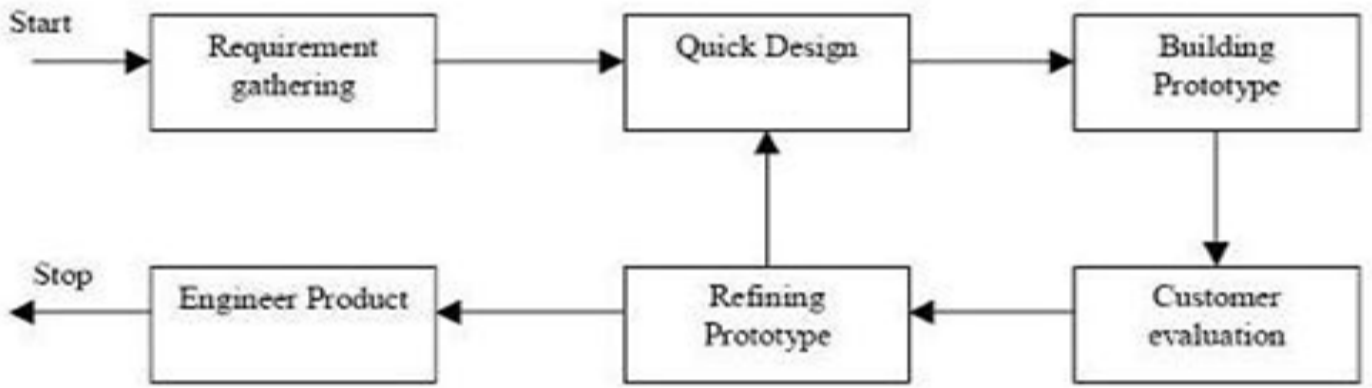
- Similar to waterfall model
- Test development activities like test design and static testing can happen before coding and the formal testing lifecycle
- Higher probability of success

Dis-Advantages

- Similar to waterfall model

Prototyping

- Prototyping should be a relatively a cheap process (subset, non production quality etc.)
- Instead of freezing the requirements before a design or coding can proceed, a complete system prototype (with no or very less details) is built and to understand the requirements
- Throw-away and evolutionary prototypes



Prototyping Model

Prototyping

Advantages :

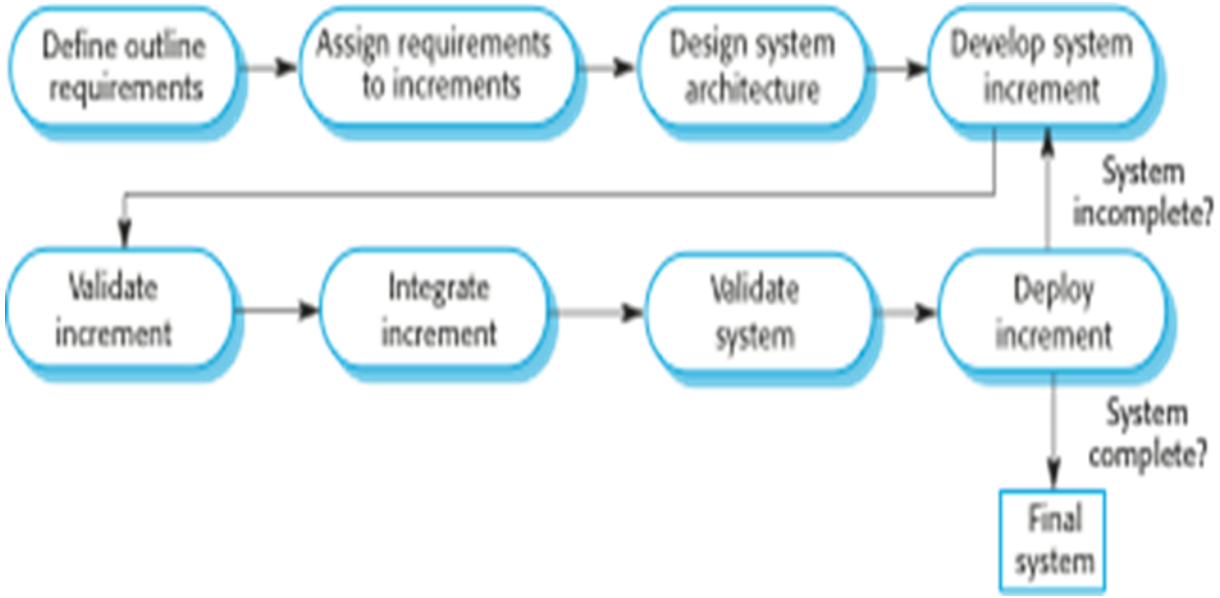
- Active involvement of the users
- Provides better risk mitigation (cost, understanding of the user requirements)
- Problems/defects are detected earlier and hence reduces time and cost
- Missing functionality easily identified and hence resulting system is full featured
- Systems are mostly more stable

Dis-Advantages:

- Practically, this methodology may increase the complexity of the system as scope of the system may expand beyond original plans
- Performance of the resulting system may not be optimal

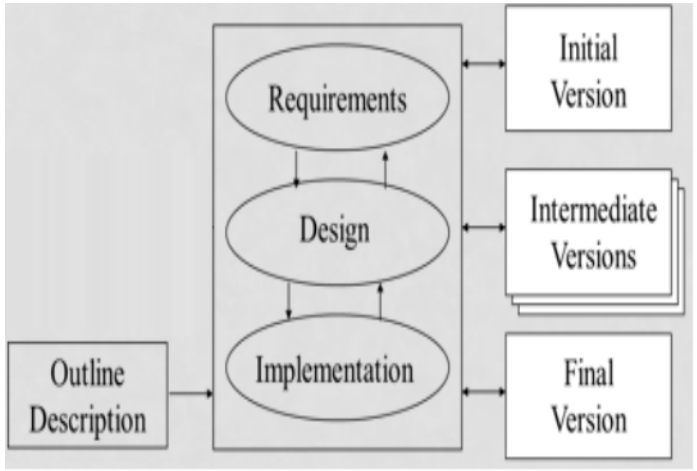
Object Oriented Analysis and Design with Java

Incremental Model



- The partitioned requirements can each have a development lifecycle
- Models like waterfall could be used/employed for each of the partition/increment
- Leveraged prototype for additional features would be a form of incremental model

Iterative Model (Evolutionary)

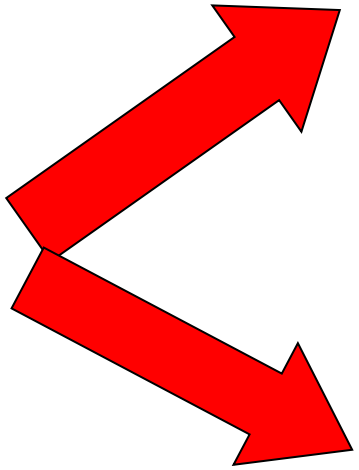


- Initial implementation attempts to start with a skeleton of the product, followed by refinement through user feedback and evolution until system is complete.
- Built with dummy modules
- Rapid prototyping
- Successive refinement

Recollect the differences and usages

Context of Software Design (or where does it fit in)

- Concept and Feasibility Study
- Requirements
- Design
- Construction (Implementation)
- Testing

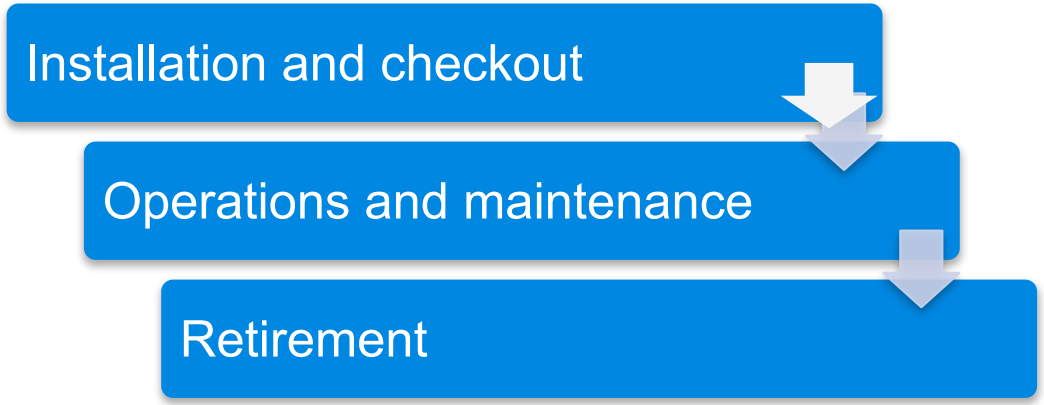


Architectural Design

Architectural design describes how software is decomposed and organized into components (the software architecture) [IEEE 1471-00]

Detailed Design

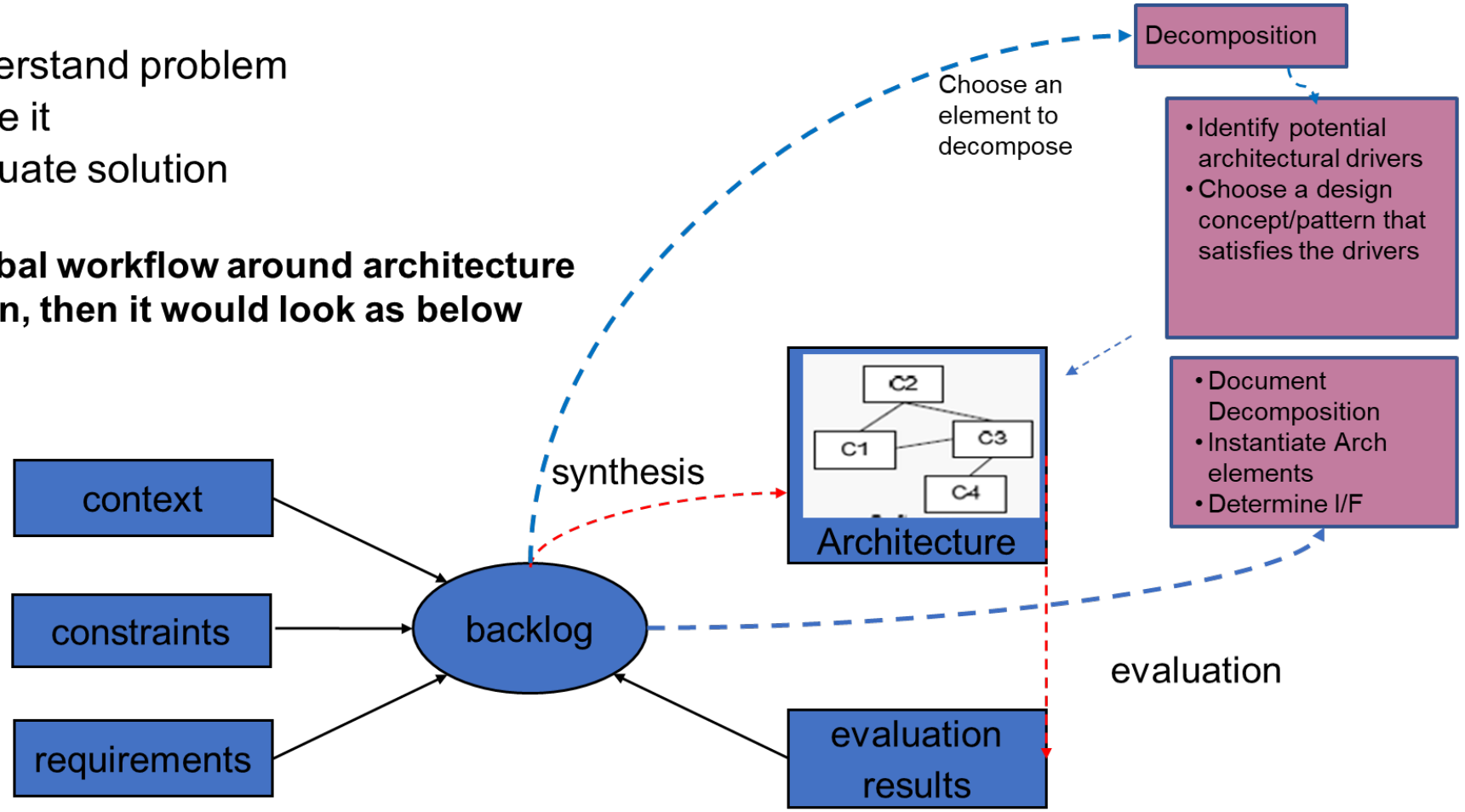
Detailed design describes the specific behavior of components identified during architectural design



Generalized model for Architecting

- Understand problem
- Solve it
- Evaluate solution

A global workflow around architecture design, then it would look as below



Common themes of an Architecture : Decomposition

The first step typically would involve decomposition of the problem to individual Modules (a set of code or data units) or Components

There are different approaches of Decomposition

1. **Decomposition based on Layering**
   - Ordering the system in to different **Layers** where by each layer provides some service to the upper layer while consuming some service from the layer underneath. Can be looked at as ordering of Abstractions
2. **Decomposition based on Distribution among "COMPUTATIONAL RESOURCES" based on or due to system having**
   - Dedicated task owning a thread of control and hence some processes not needing to wait for this process
   - Many clients needing access
   - Needing Greater fault isolation
   - Distribution for separation of concern with redundancy which allows for high availability

Common themes of an Architecture : Decomposition

3. **Decomposition based on Exposure**
   - On how the component is **Exposed** and consumes other components.
   - This could be based on the components - Service offered, logic and Integration
4. **Decomposition based on Functionality**
   - Grouping within the problem domain and separating concerns based on the **Functionality**. Eg. login module, customer module, inventory module etc. Done with the mindset of an Operational process
4. **Decomposition based on Generality**
   - Considering **Generality** or trying to understand components which can be used in other places as well. (Not to overdo)

Object Oriented Analysis and Design with Java

Common themes of an Architecture : Decomposition

6. Decomposition based on Volatility

- Identifying the sections or parts in the larger system which may change frequently and keeping them together. Eg. UI

With any or all of the approaches being used for Decomposition

- Keep things together **(Coupling)** that work together **(Cohesion)**
- Keep things away that works away.

"Look for **Low** coupling and **high** cohesion”

Common themes of an Architecture : Decomposition

Other ways of looking at Approaches for Decomposing into smaller components

1. **Divide and conquer** - Divide a complex problem into smaller simpler problems
2. **Stepwise refinement** - Start with a simple solution and enhance it in steps
3. **Top-down approach** - Start with an overview of the system, then detail the subsystems
4. **Bottom-up approach** - Specify individual elements in detail and then compose them together
5. **Information hiding** – E.g. Encapsulation, separation of interface from implementation

Design Principles

Detailed design can involve

- Further Decomposition (post architecture) of the components being developed if necessary.
- Description of the behavior of components/sub-systems identified as part of Architectural design
- Description of how the interfaces will actually be realized using the appropriate algorithms and data structures
- Description on how the system will facilitate interaction with the user through the user interface
- The use of appropriate structural and behavioral design patterns
- Consideration of Maintenance and Reuse as couple of its goals

Techniques that Enable Design Principles

1. **Abstraction**
   - Focus on essential properties
   - Needs to expose only the relevant functionality to the outside world
   - These could be procedural abstraction or data abstraction
2. **Modularity, coupling and cohesion**
   - Modularity is the degree or the extent to which the larger module can be decomposed
   - Modules are best to be self contained
   - Two designs for the same problem could be decomposed into different components/ modules using different Structural criteria (theme for coming up with individual components/modules and their interconnections)

Techniques that Enable Design Principles…(cntd.)

Cohesion which is the extent to which the component/modules are dependent on/ fit into or are related to each other, when trying to address a specific responsibility **(Strong is Good)**

- Cohesion could be ..
  - Adhoc/coincidental
  - Logical (Input routines)
  - Temporal (Initialization sequence)
  - Sequential (Sequence of events)
  - Procedural (Read, Print etc)
  - Functional (Contribute to the same function)

Techniques that Enable Design Principles…(cntd.)

Coupling indicates how strongly the modules are connected to other modules.

(Loose is Good)

Coupling based on how the elements of the components are dependent on each other could be looked at as

- Content coupling : One component directly affects the working of another. To be Avoided
- Common coupling :Two components sharing data
- External coupling : Components communicating through external medium like a file
- Control coupling : One component controls the other through passing control information
- Stamp coupling : Complete data structures are passed. Clear visibility to the common data structure
- Data coupling : Only simple data is passed between components

Techniques that Enable Design Principles…(cntd.)

3. Information hiding

- Need to know
- Each module has a secret
- Design involves a series of decisions and each such decision, need to consider who needs to know & who can be kept in the dark

Information Hiding could be through

Encapsulation

- A information hiding strategy which hides data and allows access to the data only through specific functions e.g. Class

Separation of interface and implementation

- An information hiding strategy which involves defining a component by specifying a public interface (known to the clients) but separating the details of how the component is actually realized
- Enables the implementation to change independently of interface

Techniques that Enable Design Principles…(cntd.)

4. Limiting complexity

- Complexity refers to the amount of effort required for building its solution
  You could look at measuring certain aspects of the software (lines of code, # of if-statements, depth of nesting, …)
- Use these numbers as a criterion to assess a design, or to guide the design
- Interpretation:
  higher value => higher complexity => higher effort required (= worse design)

Two kinds of module complexity:

- Intra-modular: complexity of the single module
  - Measures based on size e.g. Counting lines of code
- Inter-modular: between modules
  - Measures based on size and Measures based on Structure
    Eg. Local flow, Global flow etc. fan-in, fan-out

5. Hierarchical structure – Views the whole structure as a hierarchy

THANK YOU

Prof Sudeepa Roy Dey

Department of Computer Science and Engineering

sudeepar@pes.edu