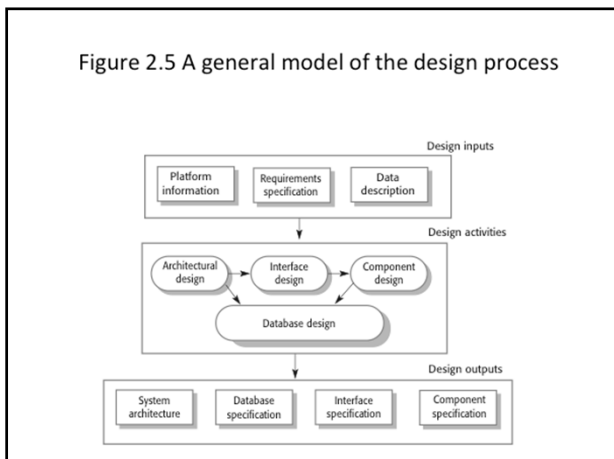
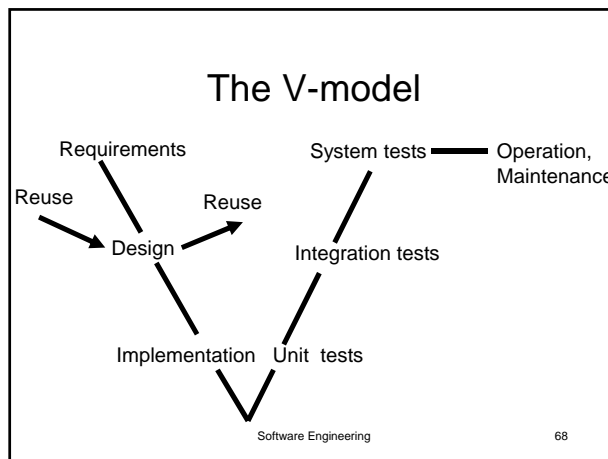


## Design

Design process [2.2.2]  
Architectures [6]  
Reuse [16]  
Component-based design [17]



## Design methods

- "Structured methods" 70's/80's
  - Asking the right questions
  - Models
    - decomposition, architecture
    - levels of abstraction (formality, detail)
    - focus on an aspect
  - Notations ... UML
  - Tools

Software Engineering 70

## Architecture is based on non-functional requirements [6.1 18.1]

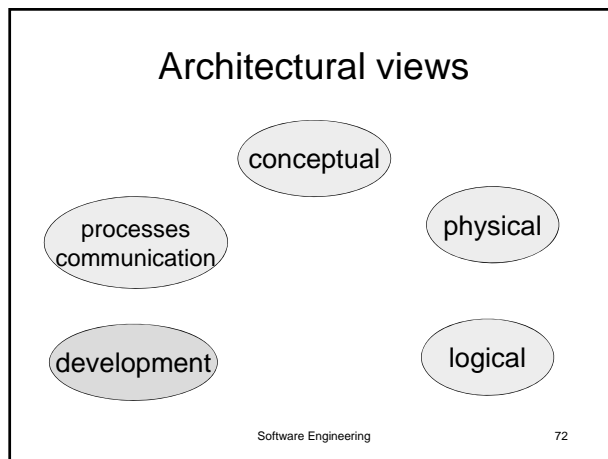
- Performance - Scalability
- Robustness
- Reliability - Availability
- Safety - Security
- Simplicity - Maintainability
- Reuse - Reusability

physical

decomposition

logical

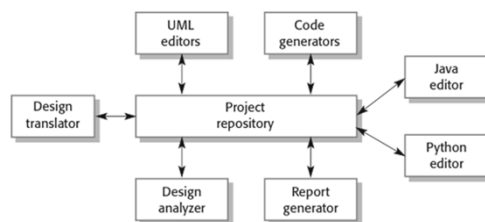
Software Engineering 71



### Architectural patterns

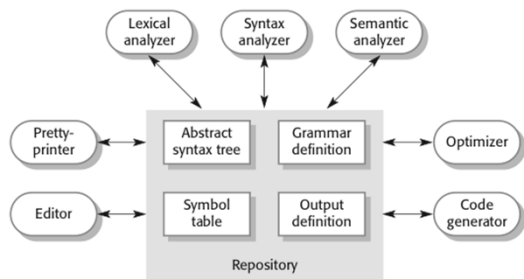
- Understanding legacy systems
- Common understanding
- Reuse of designs

### Repository architecture [6.3.2]

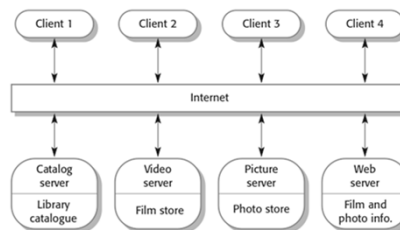


- + easy to share data
- + independent applications
- fixed data representation
- security
- bottleneck: repository

Figure 6.20 A repository architecture for a language processing system



### Client-server architecture [6.3.3]



- + easy to extend
- + security (implement in server)
- bottleneck: network

### Logical 3-tier [18.3.3]

- Presentation
- Computation (Business logic)
- Data

### Logical 3-tier [18.3.3]

Example: stocks in the bank

- Presentation Menus, graphs, etc.
- Computation (Business logic) Authentication  
Computation  
Verifying orders
- Data (time series) How much you own  
Value of stocks

### Physical implementation

- Presentation
- Computation
- Data

thin client   fat client   physical 3-tier

Software Engineering 79

### Physical implementation

- Presentation
- Computation
- Data

thin client   fat client   physical 3-tier

- + simple client (browser, COTS)
- + security
- server load

Software Engineering 80

### Physical implementation

- Presentation
- Computation
- Data

thin client   fat client   physical 3-tier

- + simple client
- + security
- server load
- need to install client
- security (raw data)
- + server load

Software Engineering 81

### Physical implementation

- Presentation
- Computation
- Data

thin client   fat client   physical 3-tier

- Games
- In-house systems (CAD, CASE)
- need to install client
- security (raw data)
- + server load

Software Engineering 82

### Physical implementation

- Presentation
- Computation
- Data

thin client   fat client   physical 3-tier

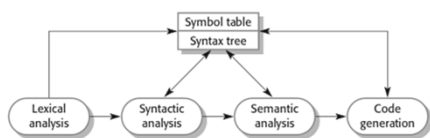
- + scalable
- + best of two worlds
- double network delays
- complex

Software Engineering 83

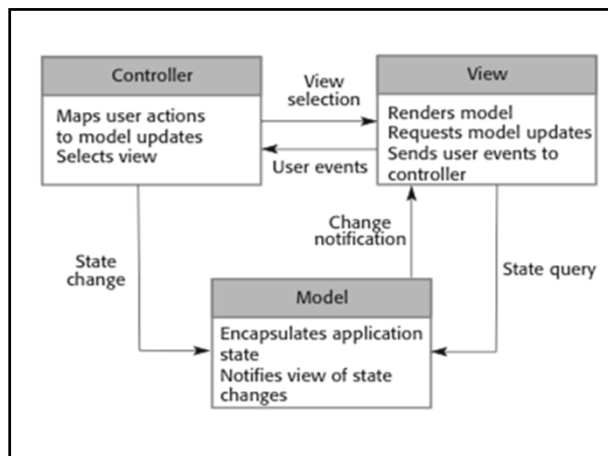
### Layered architecture [6.3.1]

- + reuse:
  - + share lower layers
  - + replace layer
- performance
- need to bypass layers

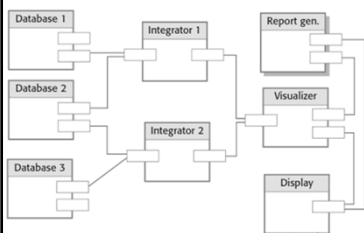
### Pipe and filter [6.3.4]



### Process view



### Distributed components [18.3.4]



- CORBA - Common Object Request Broker Architecture
- EJB - Enterprise Java Beans
- Microsoft - COM, COM+, .NET

### Reuse [16]

- System (COTS)
- Configurable system
  - Product line (ERP)
- Large subsystem
  - wrapped legacy system
  - frameworks, services
- Libraries
- Components

Software Engineering

88

### Level

### Patterns

- System (COTS)
- Configurable system
  - Product line (ERP)
 Architecture
- Large subsystem
  - wrapped legacy system
  - frameworks, services
 Model-based engineering
- Libraries
- Components

Software Engineering

89

### Reuse also

- Requirements
- User interface
  - standardization improves usability
  - documentation
- Test suites
  - regression testing

Software Engineering

90

## Reuse - why? [Fig.16.1]

- Less work
  - shorter time to market
  - effective use of specialists
  - cheaper
- You know it
  - less process risk
  - dependability [warning: Ariane 5 Fig.17.9]
- Standards compliance

Software Engineering

91

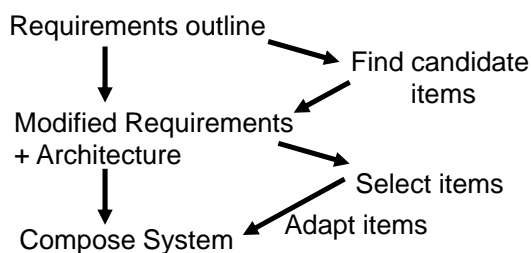
## Reuse - problems [Fig.16.2]

- Need to invest in reusable items
- Finding reusable items
  - searching, understanding, adapting
- Lack of control
  - dependence on supplier, support
  - maintenance
  - "not invented here"
  - tool support for integration

Software Engineering

92

## Reuse - process [17.2]



Software Engineering

93

## Component-Based Software Engineering (CBSE) essentials

- ✧ Independent components specified by their interfaces.
- ✧ Component standards to facilitate component integration.
- ✧ Middleware that provides support for component interoperability.
- ✧ A development process that is geared to reuse.



Chapter 17 Software reuse - Slide by Sommerville

94

## CBSE problems

- ✧ Component trustworthiness - how can a component with no available source code be trusted?
- ✧ Component certification - who will certify the quality of components?
- ✧ Emergent property prediction - how can the emergent properties of component compositions be predicted?
- ✧ Requirements trade-offs - how do we do trade-off analysis between the features of one component and another?



Chapter 17 Software reuse - Slide by Sommerville

95

## Component development for reuse

- ✧ Specially constructed by generalising existing components.
- ✧ Component reusability
  - stable domain abstractions
  - hide state representation
  - as independent as possible
  - publish exceptions through the component interface
- ✧ Trade-off between reusability and usability
  - More general interface = greater reusability = more complex = less usable.



Chapter 17 Software reuse - Slide by Sommerville

96

## Changes for reusability

- ✧ Remove application-specific methods.
- ✧ Change names to make them general.
- ✧ Add methods to broaden coverage.
- ✧ Make exception handling consistent.
- ✧ Add a configuration interface for component adaptation.
- ✧ Integrate required components to reduce dependencies.

