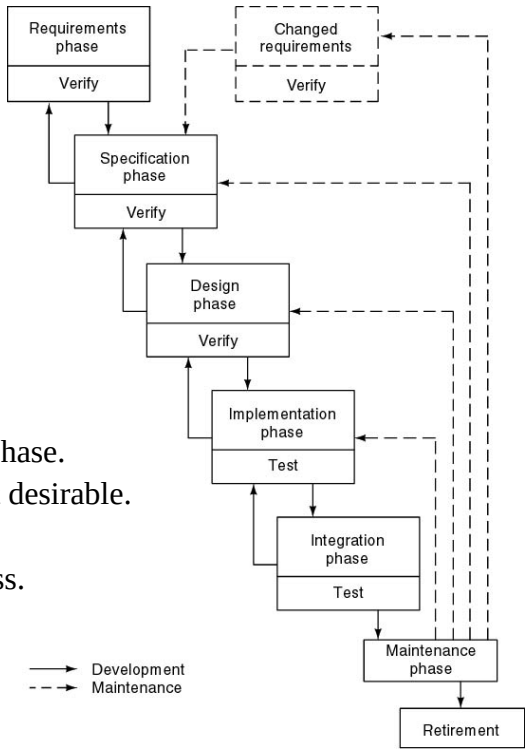
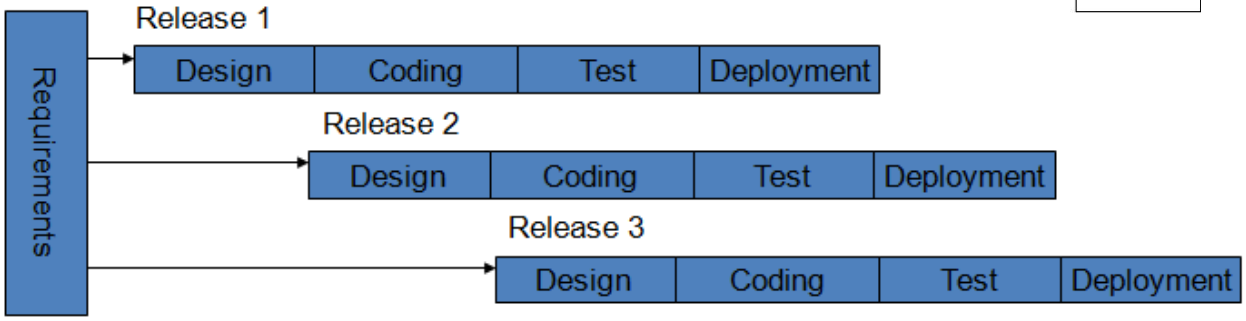
CS 2212, Software Engineering – Final Exam Content:

_____ (Topic – Process/Cost Models) _____

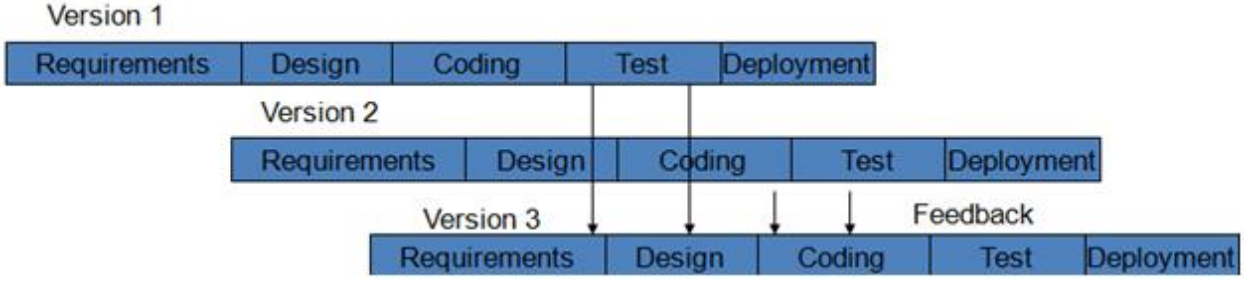
- *Waterfall Model:*
  - Characterized by:
    - Phases/Sequences.
    - Feedback Loops.
    - Documentation driven aspects.
  - Advantages:
    - Documentation.
    - Easier maintenance.
  - Disadvantages:
    - Often not feasible in practice.
    - Customer involvement only in first phase.
    - Complete execution of all phases not desirable.
    - Process is difficult to control.
    - Product is available late in the process.



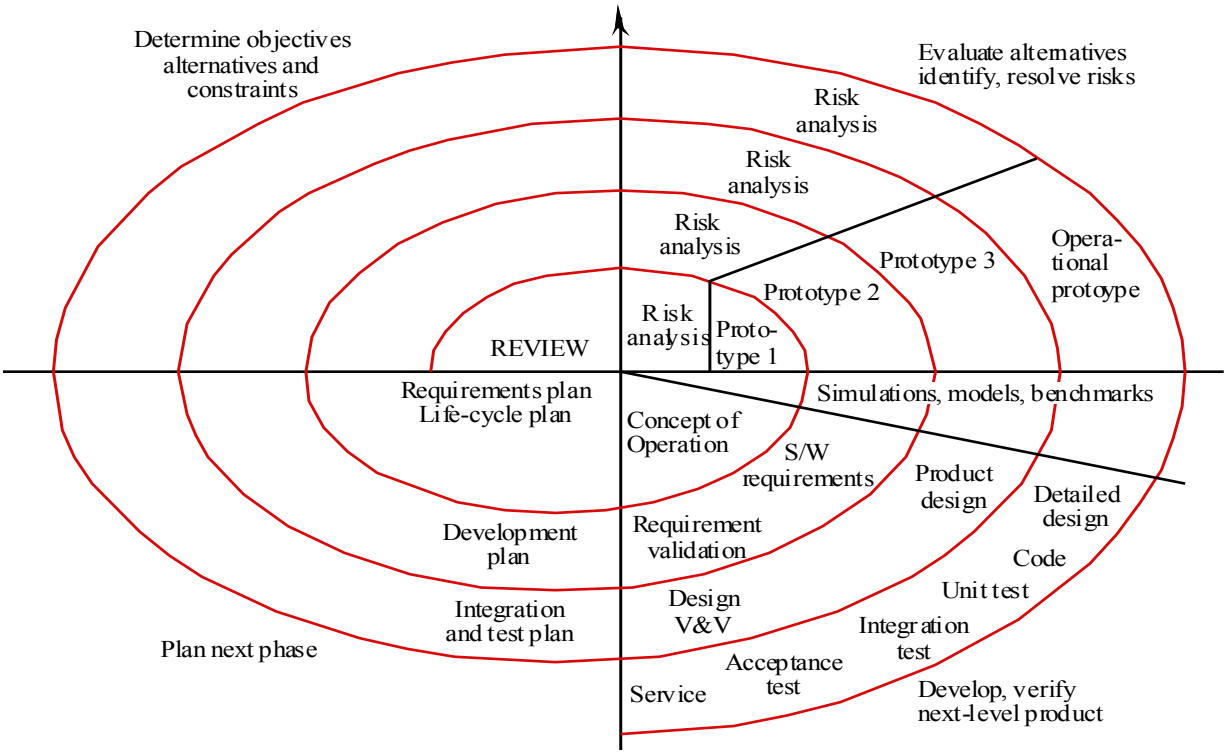
- *Incremental Model:*
  - Each release adds more functionality.



- *Evolutionary Model:*
  - New versions implement new and evolving requirements.



- Spiral Model:



- *The 4 main variables of a project:*
  - Development cost.
  - Time.
  - Quality.
    - These 3 are "bad: control variables.
  - Scope.
    - Only real control variable.

- <u>Functional Points</u> = a measure of the functionality of a system.
  - Proposed by Albrecht in 1979.
  - Scoring Process:
    - Counting FPs – through informational domain.
    - Adjusting FPs – through assessing software complexity.
    - Applying an empirical relationship to come up with LOC or P-months based on adjusted FPs.
  - Must be performed manually.

Counting the information domain

Measurement parameter	Count		Weighting factor Simple	Av.	Complex	
Number of user inputs	_____	x	3	4	6	= _____
Number of user outputs	_____	x	4	5	7	= _____
Number of user inquiries	_____	x	3	4	6	= _____
Number of files	_____	x	7	10	15	= _____
Number of ext. interfaces	_____	x	5	7	10	= _____
Count Total ----------→						= _____

- Adjusting Functional Points:

Answer the following questions using a scale of [0-5]: 0 not important; 5 absolutely essential. We call them influence factors (F_i).

1. Does the system require reliable backup and recovery?
2. Are data communications required?
3. Are there distributed processing functions?
4. Is performance critical?
5. Will the system run in an existing, heavily utilized operational env.?
6. Does the system require on-line data entry?
7. Does the on-line data entry require the input transaction to be built over multiple screens or operations (user efficiency)?
8. Are the master files updated on-line?
9. Are the inputs, outputs, files, or inquiries complex?
10. Is the internal processing complex?
11. Is the code designed to be reusable?
12. Is installation included in the design?
13. Is the system designed for multiple installations?
14. Is the application designed to facilitate change and ease of use by the user?

- D

- Calculate the empirical relationship:
  - Functional point = count total x [0.65 + 0.01 x (sum of the 14 F_i)].

- Example project calculations:

# of user inputs: {on, off, ext. number} (3) x simple (3)	= 9
# of user outputs: {tone} (1) x simple (4)	= 4
# of user inquiries: 0 x simple	= 0
# of files: {mapping table} (1)x simple (7)	= 7
# of external interfaces: {memory map} (1) x simple (5)	= 5
Total count	= 25

- Total of FPs = 25
- $F_4 = 4$, $F_{10} = 4$, other F_i's are set to 0. Sum of all F_i's = 8.
- FP = 25 x (0.65 + 0.01 x 8) = 18.25
- Lines of code in C = 18.25 x 128 LOC = 2336 LOC
- In the past, students have implemented their projects using about 2500 LOC.

- Organic mode: PM = 2.4 (KDSI) $^{1.05}$
- Semi-detached mode: PM = 3 (KDSI) $^{1.12}$
- Embedded mode: PM = 3.6 (KDSI) $^{1.2}$

- PM = Person months
- KDSI = Kilo Delivered Source Instructions

COCOMO example

- Organic mode project, 32KLOC
  - PM = 2.4 (32) $^{1.05}$ = 91 person months
  - TDEV = 2.5 (91) $^{0.38}$ = 14 months
  - N = 91/15 = 6.5 people

_____ (Topic – Design Principles) _____

- Cohesion = the measure of the strength of functional relatedness of elements within a module.
    - High Cohesion = means a module should encapsulate some well defined, coherent piece of functionality.

- Coupling = a measure of how closely connected two modules are.
    - Low Coupling = means minimizing the amount of dependencies between modules.

- Levels of Coupling:
    1) Data Coupling (best)
        - Two modules interact with each other by passing data as a parameter.
    2) Stamp Coupling
        - Multiple modules sharing the same data structure but work on different aspects of it.
    3) Control Coupling
        - Two modules are control modules if they decide the function of other modules.
    4) Common Coupling
        - When multiple modules have read and write access to global data.
    5) Content Coupling (worst)
        - When a module can directly access or modify the contents of another module.

_____ (Topic – Component Diagrams) _____

- *Component Diagram:*
    - Model the structure of a system as a collection of components interacting using interfaces.
    - A component is a system unit that offers specific functionality, and if designed well has high cohesion and low coupling.

- *Hierarchy of Units:* System → Subsystem → Component (Composite or Class)

- *Deployment Diagrams:*
    - Show how the components will be physically deployed.

_____ (Topic – Architecture Styles) _____

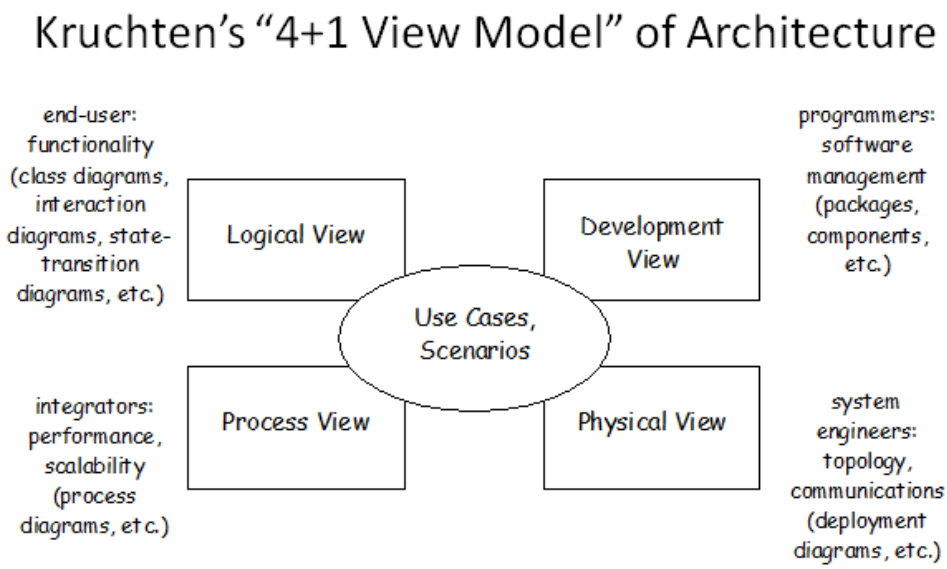
- Architecture definition - by Shaw and Garlan:
    - Abstractly, software architecture involves the description of elements from which systems are built, interactions among those elements, patterns that guide their composition, and constraints on these patterns. In general, a particular system is defined in terms of a collection of components and interactions among these

components. Such a system may in turn be used as a (composite) element in a larger system design.

- Architecture definition - by Bushmann et al:
    - Software architecture is a description of the subsystems and components of a software system and the relationships between them. Subsystems and components are typically specified in different views to show the relevant functional and non-functional properties of a software system. The software architecture of a system is an artifact. It is the result of the software development activity.

- Architectural design allows for decomposition of a system into interacting components.

- Good architecture:
    - Consistent techniques and design.
    - Resilient to change.
    - Offers guidance through the entire product lifetime.



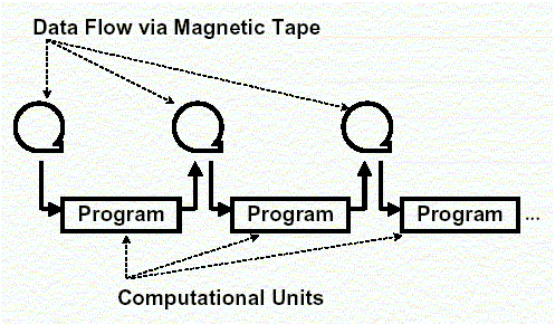
- Architecture is composed of components and connectors.

- Architectural style defines a family of architectures with common topologies, semantics, and vocabulary.

- *Data Flow Style:*
    - Availability of data controls computation.
    - The structure of the design is determined by motion of data from component to component.

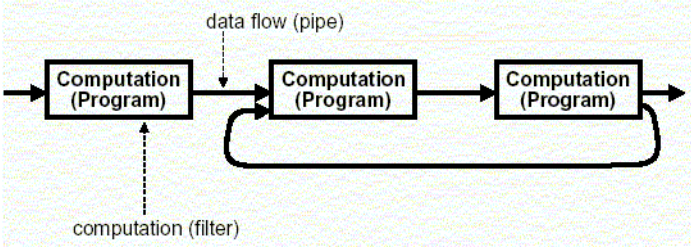
- Pattern of data flow is explicit.
- Only form of communication between components is through data.
- Data Flow Components:
  - Interface – input and output ports.
  - Computational Model – read data from input ports, compute, write data to output ports.
- Data Streams:
  - Uni-directional.
  - Computational Model – transport data from writer roles to reader roles.
- Systems:
  - Arbitrary graphs.
  - Computational Model – function compositions.
- Data can flow linearly, arbitrarily, or cyclically.

- *Batch Sequential (Data Flow Architecture):*



  - Components are independent programs.
  - Connectors are a type of media (historically mainframes and magnetic tape).
  - Each step runs to completion before the next step. Uses block scheduling CPU processing time.
  - Limited disk space.
  - Uses periodic reports from periodic data updates.
  - Typical application – payroll computations or CRA tax returns.

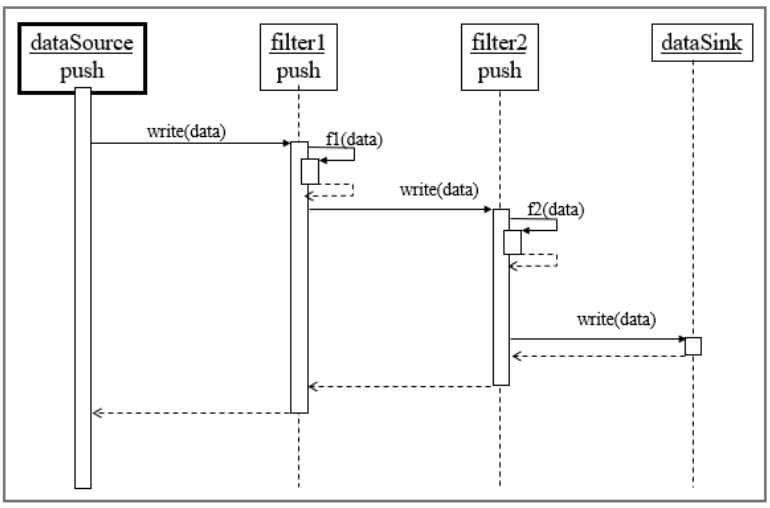
- *Dataflow Network / Pipes and Filters (Data Flow Architecture):*



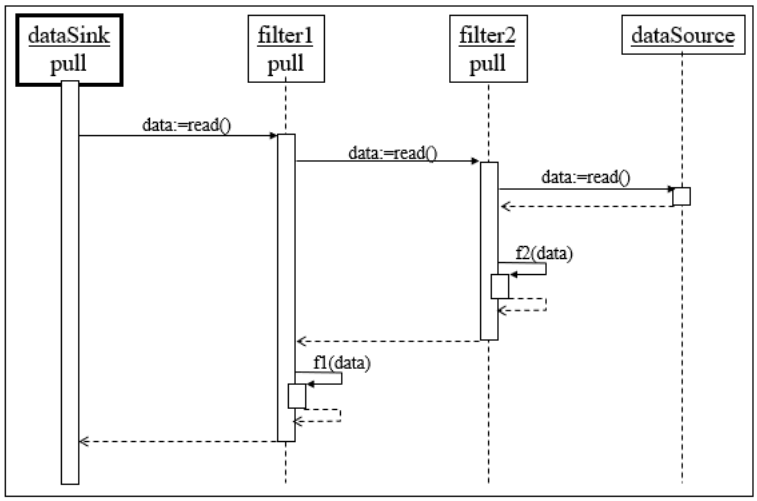
  - Processed incrementally.
  - Each component is encapsulated in a filter component, with data passed through pipes between adjacent filters.
  - Data is processed as it arrives, instead of being gathered then processed.
  - Output from one filter to the input of the other.
  - Commonly follows a linear pipeline sequence.
  - Filters are independent.
  - Examples:
    - Image/Signal processing.
    - Video/Voice streaming.
    - Lex compiler (scans, parses, generates code).

- Data pulling and pushing – if more than one filter is pushing or pulling data upstream or downstream, synchronization is needed.

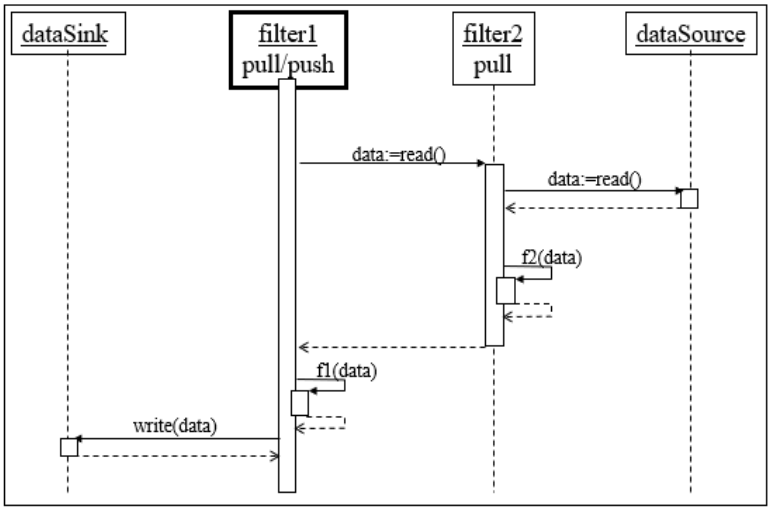
- Push:



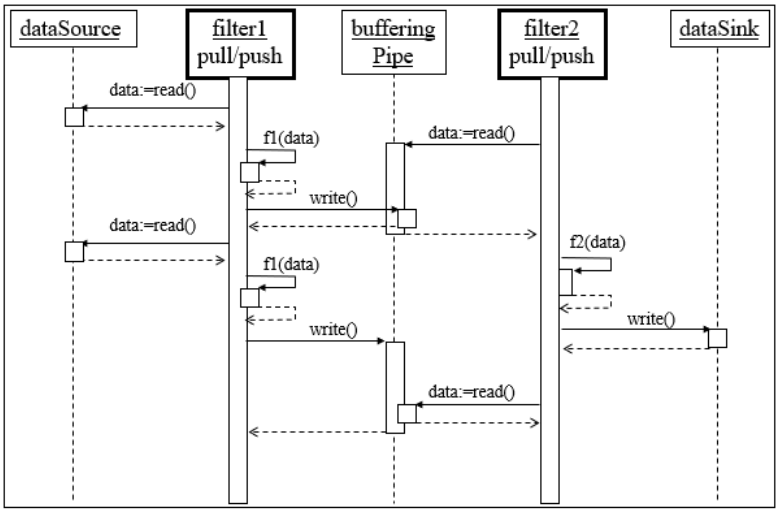
- Pull:



- Push/Pull Mix:



- Active Filtering, synchronizing and buffering:



- *Pipe and Filter – Strengths:*
  - Reusable.
  - Easy maintenance.
  - Potential for parallelism.

- *Pipe and Filter – Weaknesses:*
  - Can degenerate into batch processing.
  - Sharing global data is expensive and limited.

- Designing incremental filters is hard.
- Not useful for interactive applications
- Error handling is a problem since the filter consumes 75% of the input. Only solution is restarting.

Batch Sequential	Pipe-and-Filter
♦ course grained ♦ high latency ♦ external access to input ♦ no concurrency ♦ non-interactive	♦ fine grained ♦ results starts processing ♦ localized input ♦ concurrency possible ♦ interactive awkward but possible

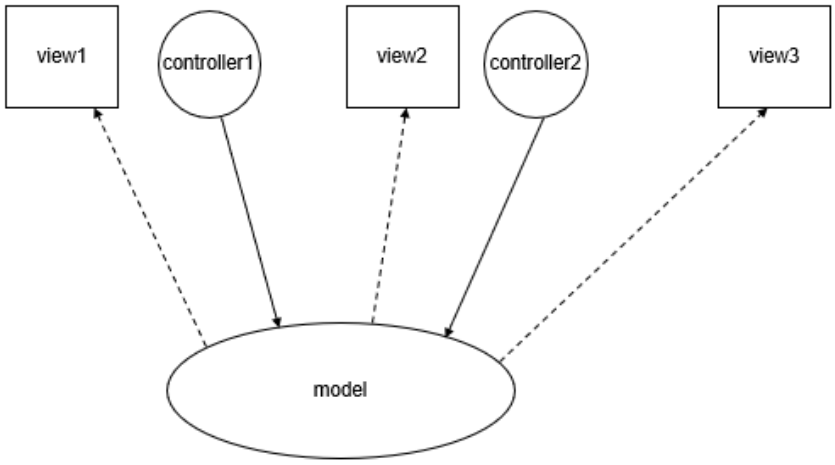
- *Data Abstraction / Object Orientation:*
  - Widely used in architectural styles.
  - Objects preserve integrity of representation.
  - Strengths:
    - Change implementation without affecting clients.
    - Breaks problems into interacting components.
  - Weaknesses:
    - Objects must know each other's identity to interact.
    - When an identity changes, objects it invokes must change.

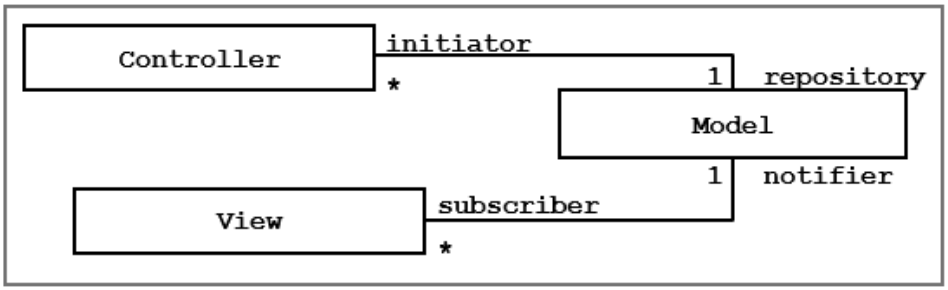
- *Event Based Invocation:*
  - Components communicate using a generalized observer design pattern style of communication.

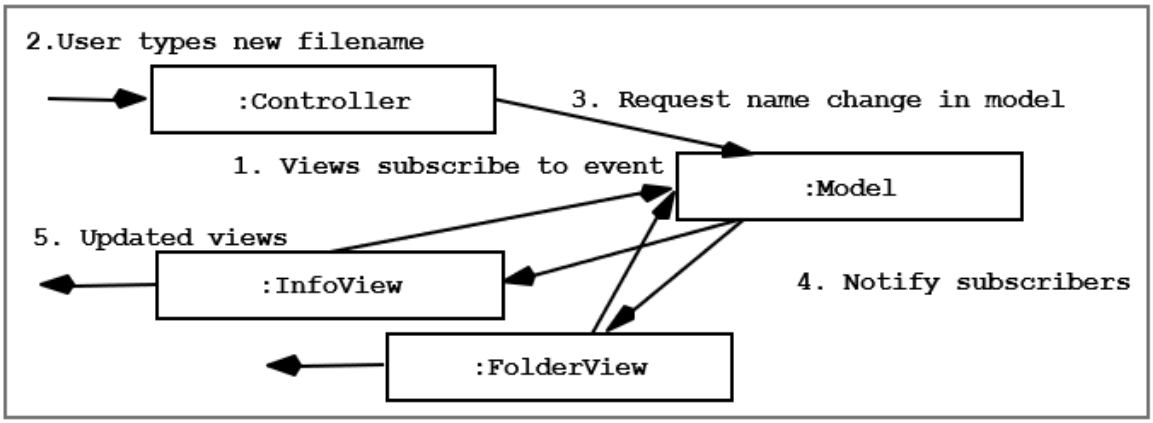
- *Implicit Invocation:*
  - Components register interest in an event by associating a procedure with the event. The event is then announced, and the system implicitly invokes all procedures registered for the event.
    - Example – Editor announces it has finished editing a module and the compiler registers for such announcements and automatically re-compiles the module.
    - Shaw and Garlan example.
  - Pros – strong reusability.
  - Cons – loss of control.

- *Model View Controller:*

- Splits the system into 3 components:
  - Model – of core functionality and data.
  - Viewer – user information.
  - Controller(s) for user input.



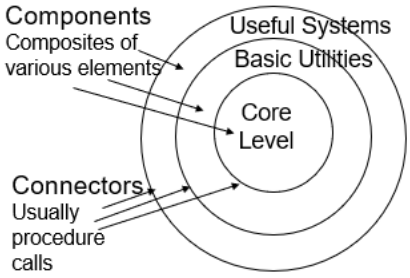




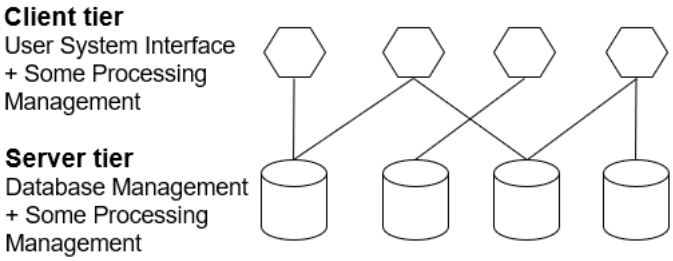
- Blackboard Architecture = cooperating "partial solution solvers" collaborating but not following a pre-defined strategy.
  - Useful for AI, and modern compilers

- Layered Systems = a layered system is organized hierarchically with each layer providing service to the layer above it and serving as a client to the layer below.

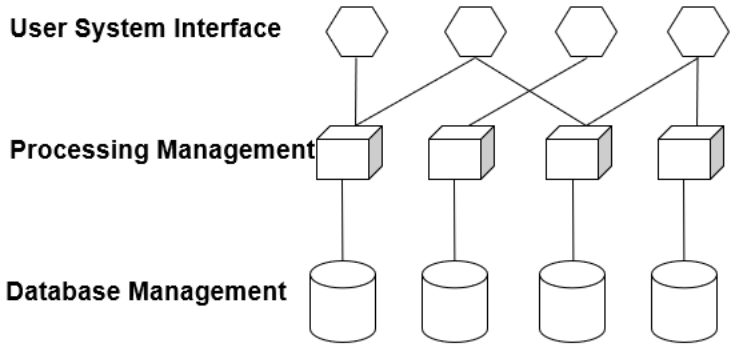
- Onion Skin Model:



- Two Tier Client Server Architecture Design:
  - Developed in the 1980's to decouple.
  - Improved maintainability scaling up to 100 users.



- Three Tier Client Server Architecture Design:
  - Developed in the 1990's to overcome the previous limitations.
  - New middle tier provides process management to execute business logic and rules.



_____ (Topic – Design Patterns) _____

- X
- X

Found in Week 6 Lecture 1

Found in Week 7 Lecture 1

Found in Week 6 Lecture 2

_____ (Topic – Reliability) _____

- Failure Intensity
- Cumulative Failures
- Resource Utilization
- Reliability Models

Found in Week 8 Lecture 1