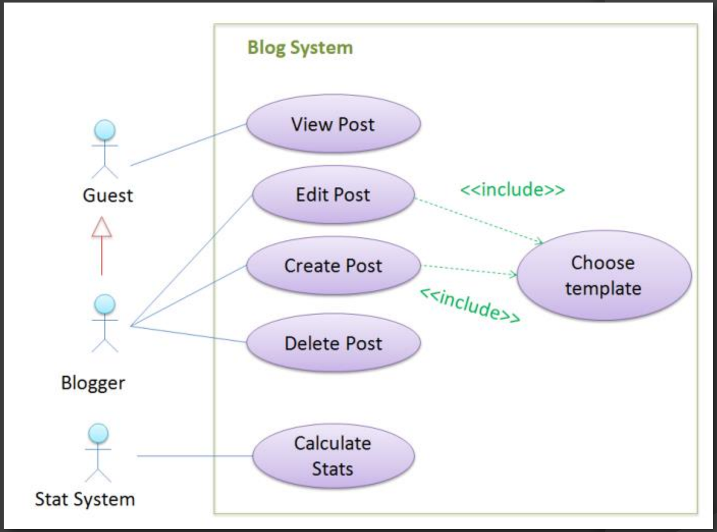
MODELS, VIEWS & DIAGRAMS

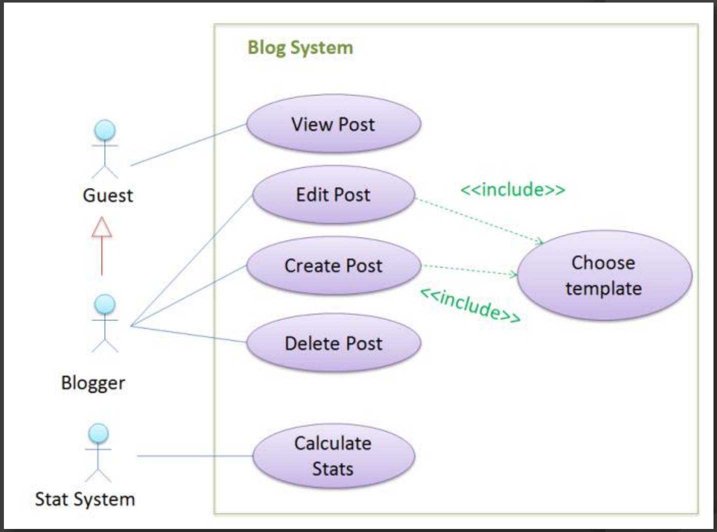
Lecture 13

Models

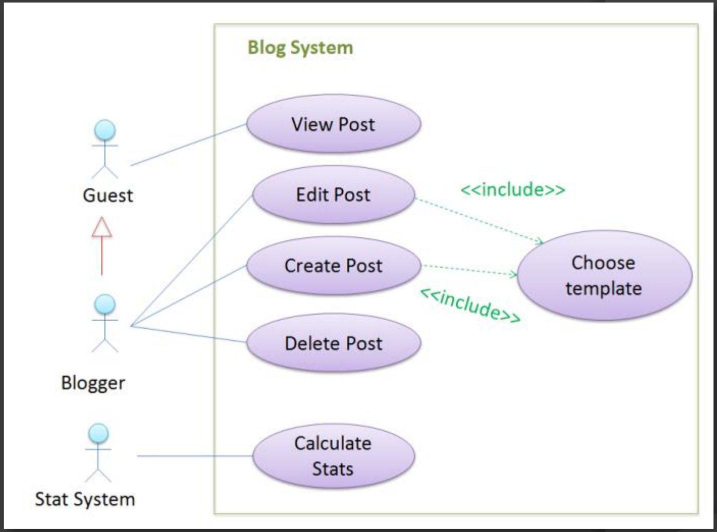
What word is common to all three?



What word is common to all three?

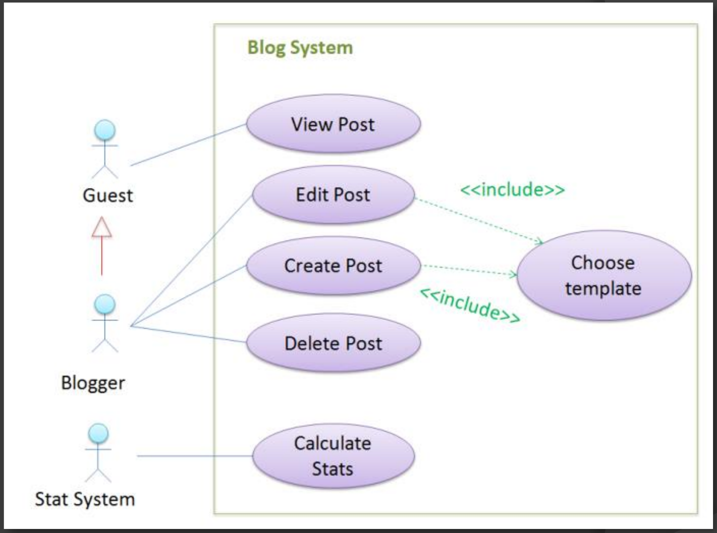


models



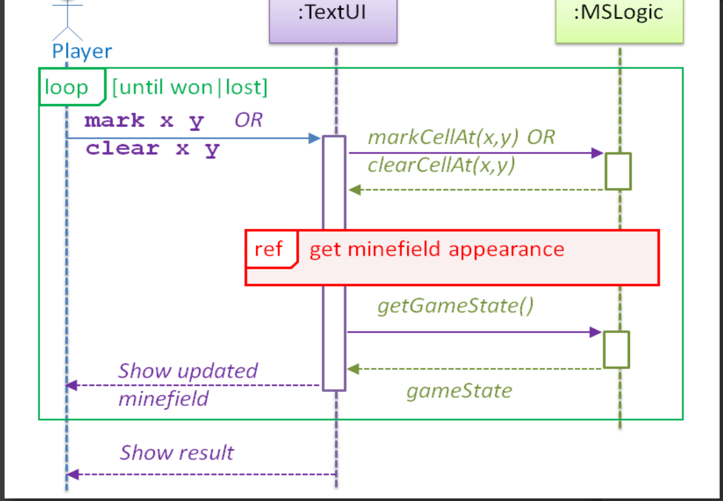
models

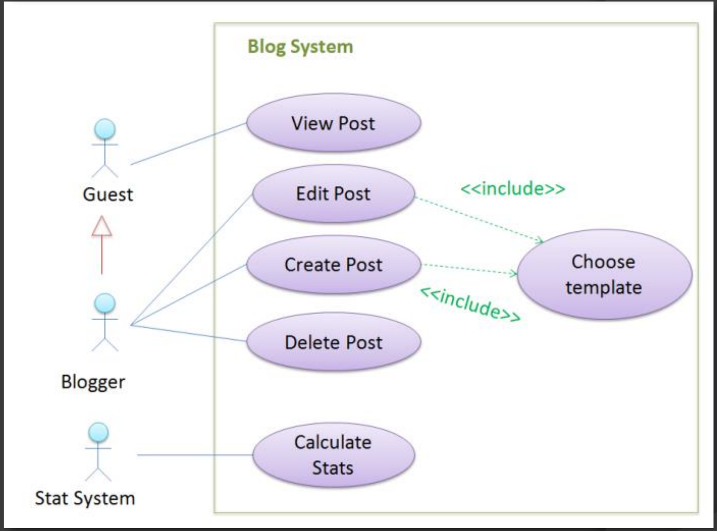
… are **not the real thing!**

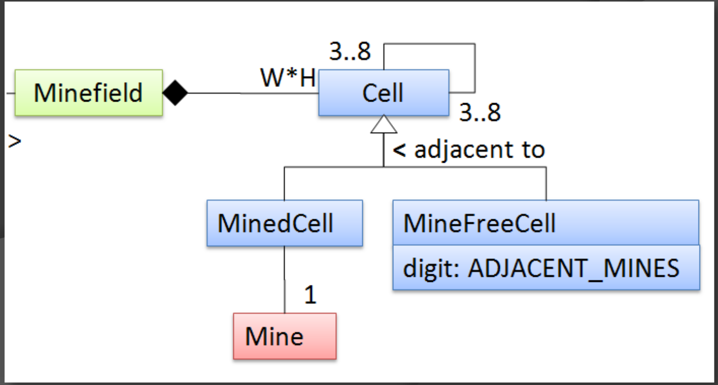


models

… are **simpler than** the real thing!

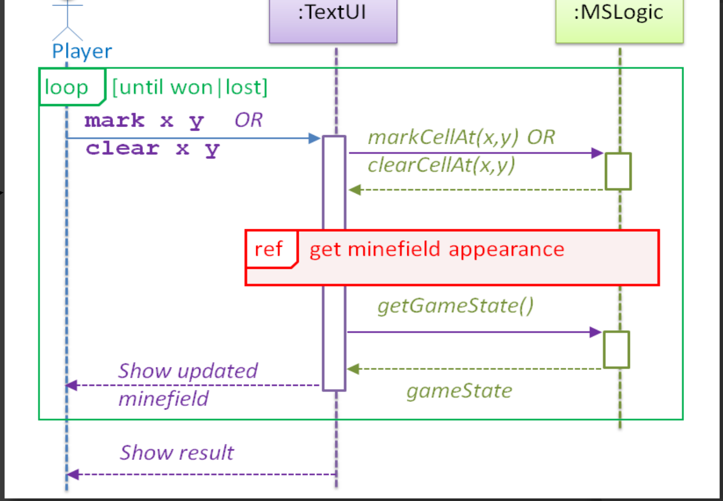


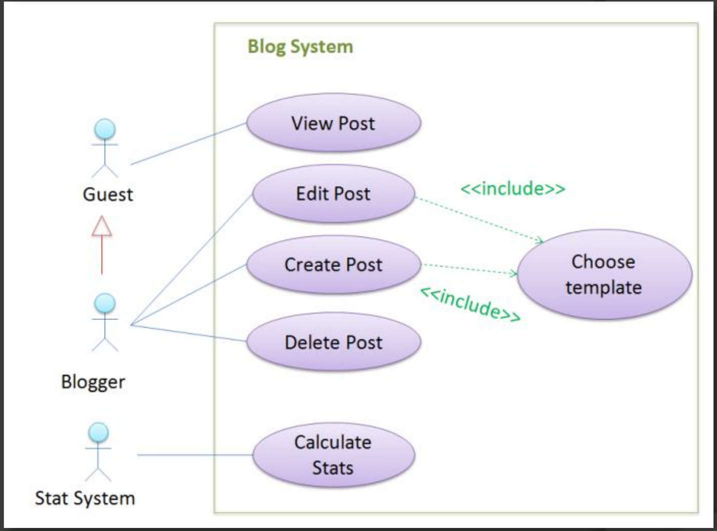


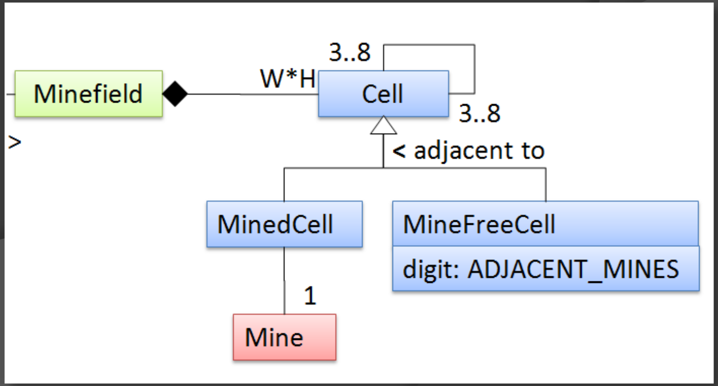


models

… capture different aspects

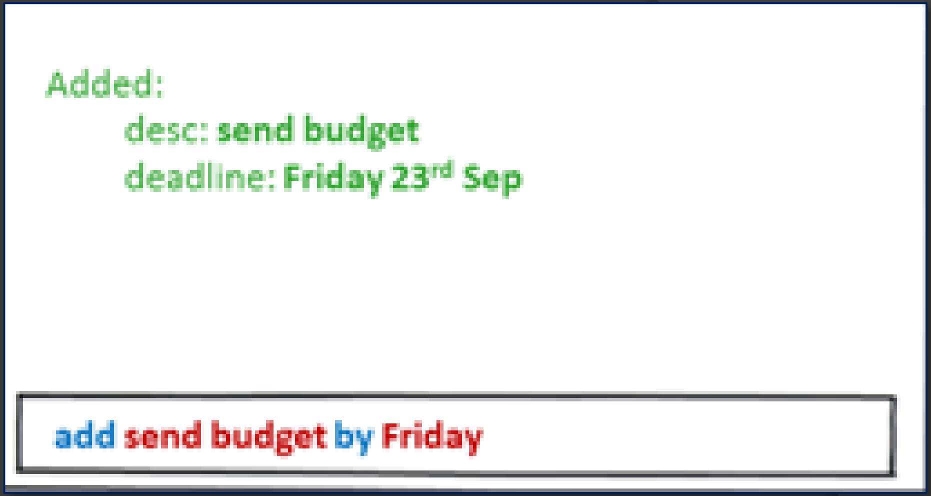






models

… combat complexity



Added:
    desc: send budget
    deadline: Friday 23rd Sep

add send budget by Friday

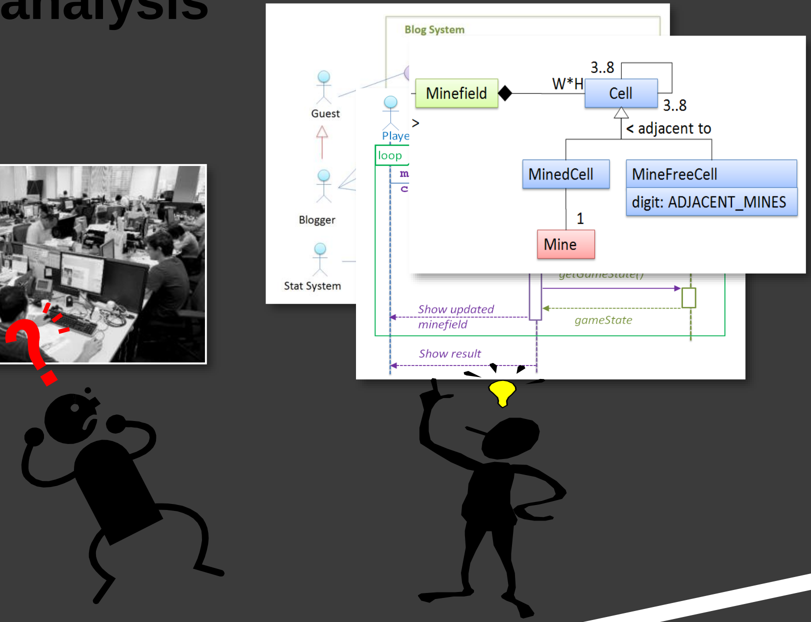
models

… are indispensable for bigger roles in projects

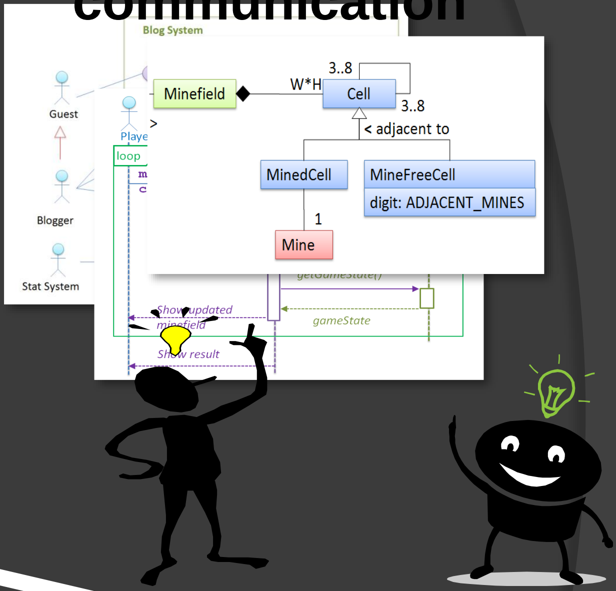
models **for what?**

models for what?

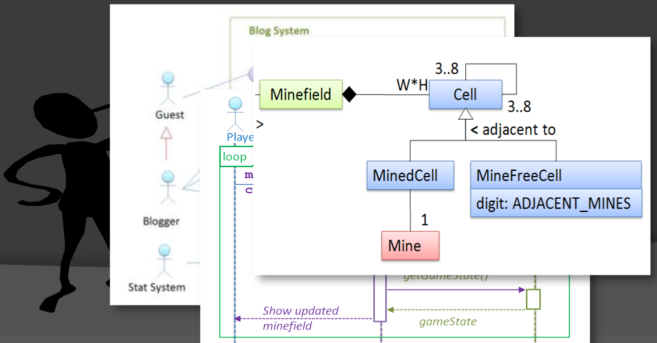
i. for analysis

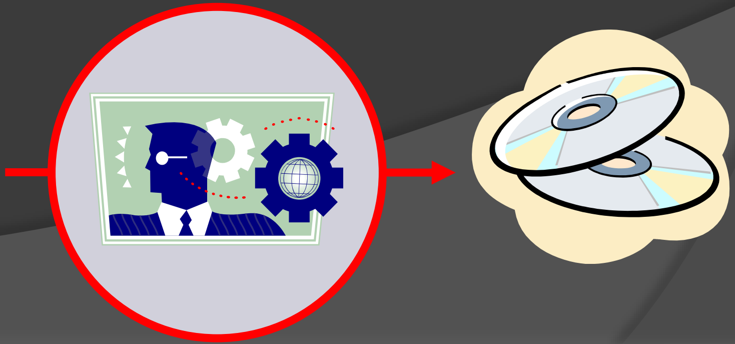


ii. for communication

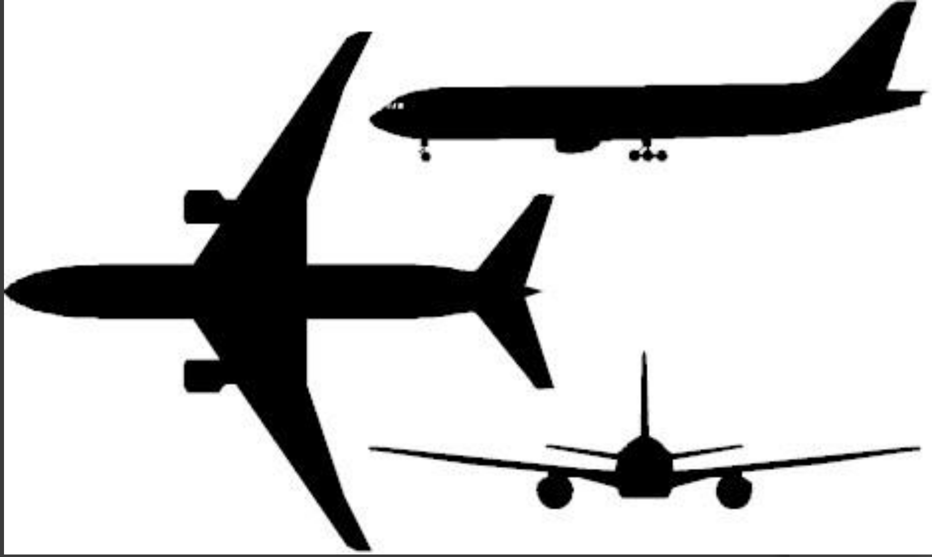


iii. as a blueprint



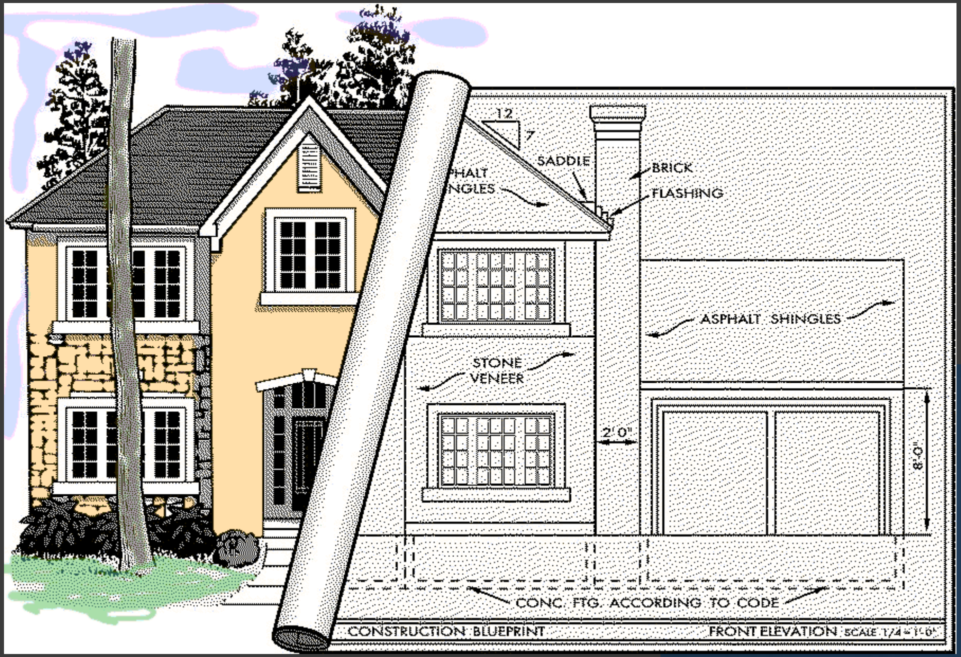


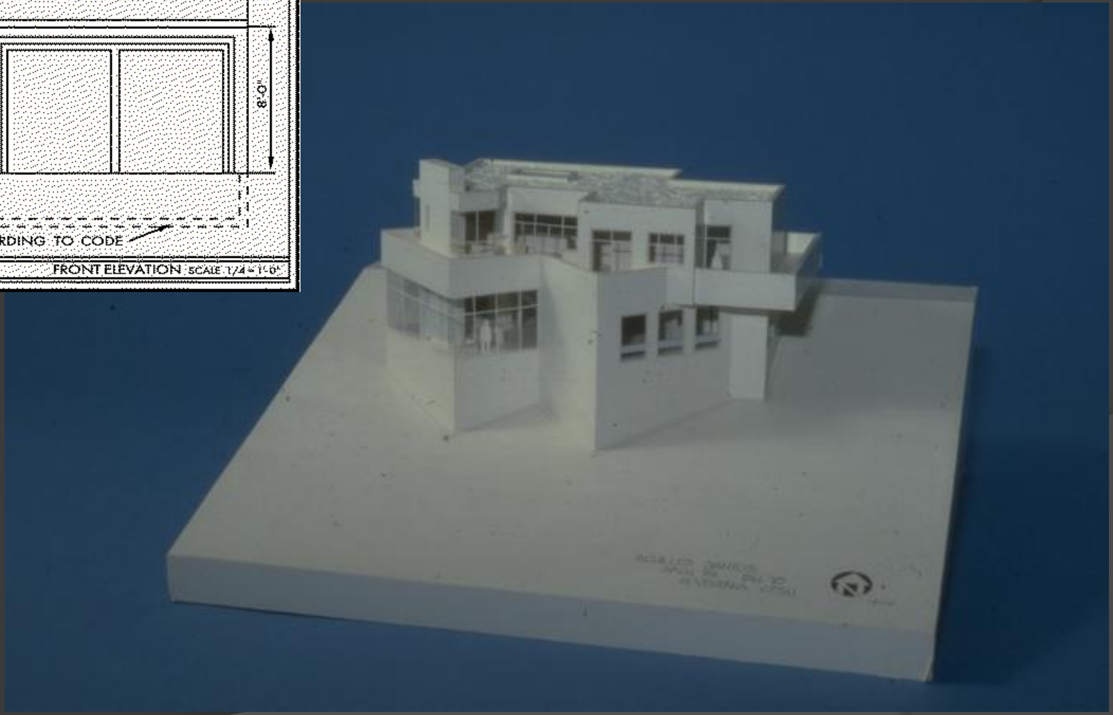
Modeling



- Based on abstraction
  - Looking only at relevant information
  - Hiding details
- The idea of modelling is fundamental to UML
- Like a map, a model represents something else
- A useful model has the right level of detail and represents only what is important for the task in hand
- Many things can be modelled: bridges, traffic flow, buildings, economic policy

Modeling a House





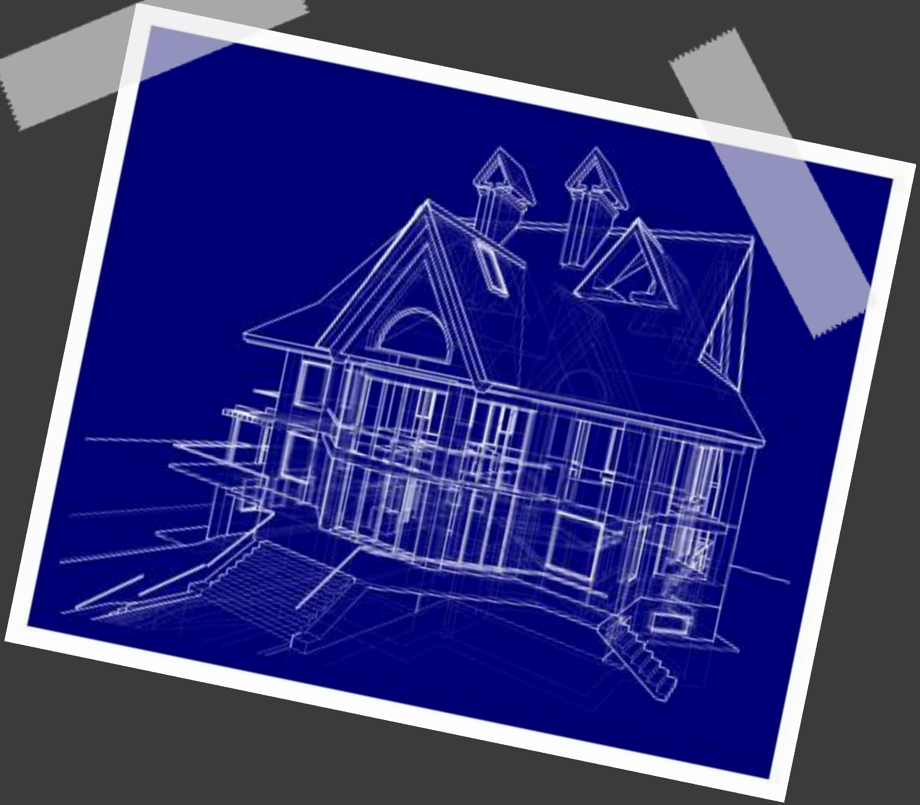
Diagrams vs Models

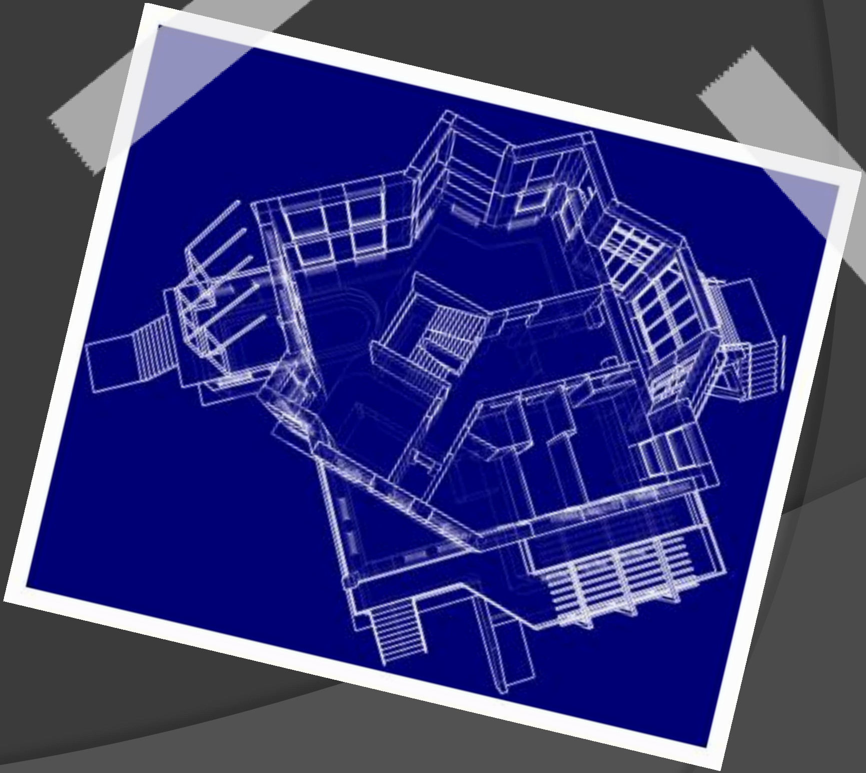
- A diagram illustrates some aspect of a system.
- A model provides a complete view of a system at a particular stage and from a particular perspective, e.g., Analysis model, Design model.
- A model may consist of a single diagram, but most consist of many related diagrams and supporting data and documentation.

Views

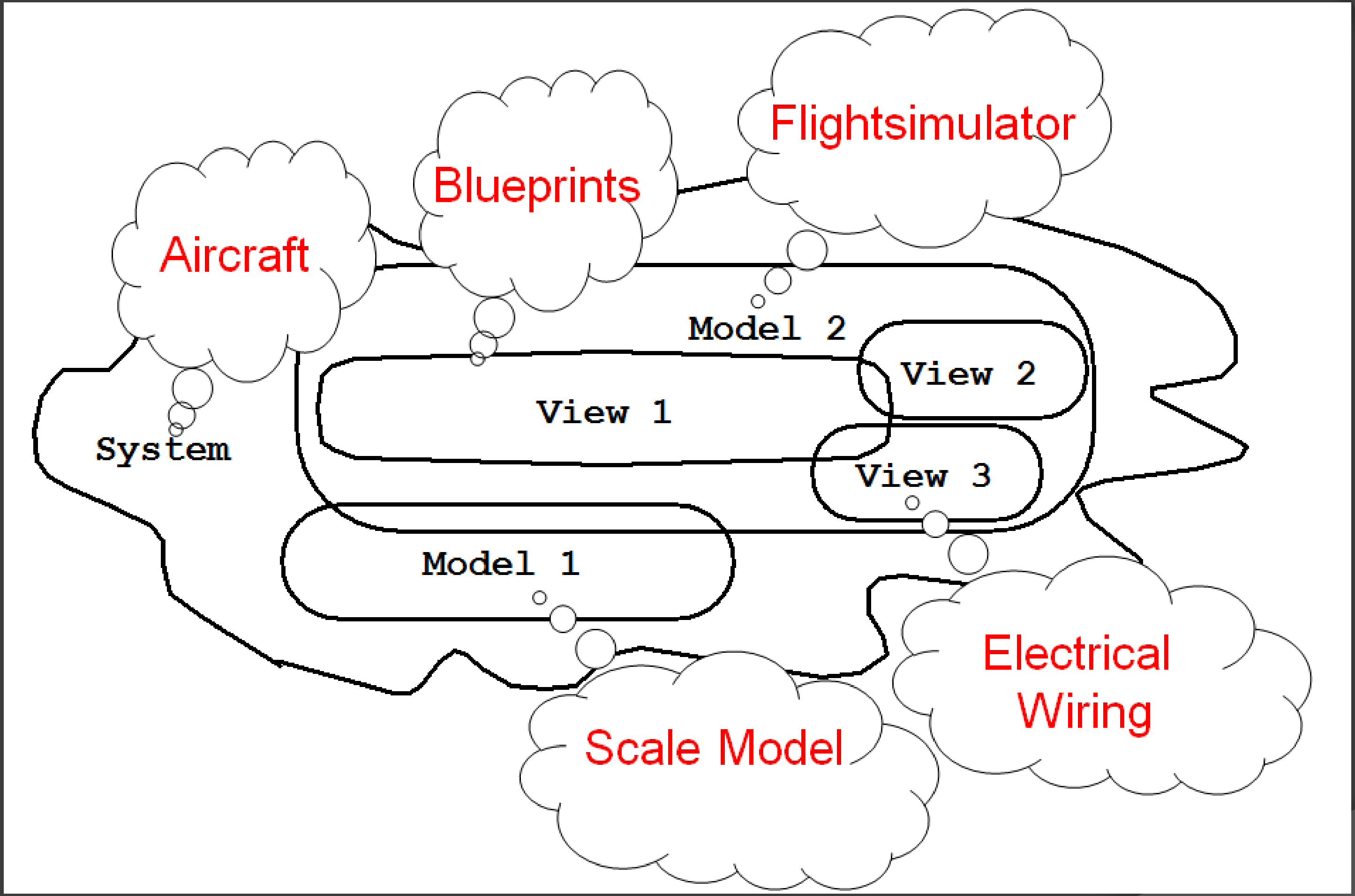
- A view is a subset of UML modelling constructs that represents one aspect of a system.
- Create multiple views
  - Each view has information that is unique
  - Each view has information that appears in other views
  - Common information is consistent

Views





Systems, Models and Views



UML Views

- Structural classification
- Dynamic behaviour
- Physical layout
- Model management

UML views and diagrams

Major View	**Diagram**	**Concepts**
structural	class diagram	association, class, dependency, generalization, interface, realization
	internal structure	connector, interface, part, port, provided interface, role, required interface
	collaboration diagram	connector, collaboration, collaboration use, role
	component diagram	component, dependency, port, provided interface, realization, required interface, subsystem
	use case diagram	actor, association, extend, include, use case, generalization

UML views and diagrams cont.

Major View	Diagram	Concepts
dynamic	state machine diagram	completion transition, do activity, effect, event, region, state, transition, trigger
	activity diagram	action, activity, control flow, control node, data flow, exception, expansion region, fork, join, object node, pin
	sequence diagram communication diagram	occurrence specification, execution specification, interaction, lifeline, message, signal
		collaboration, guard condition, message, role, sequence number

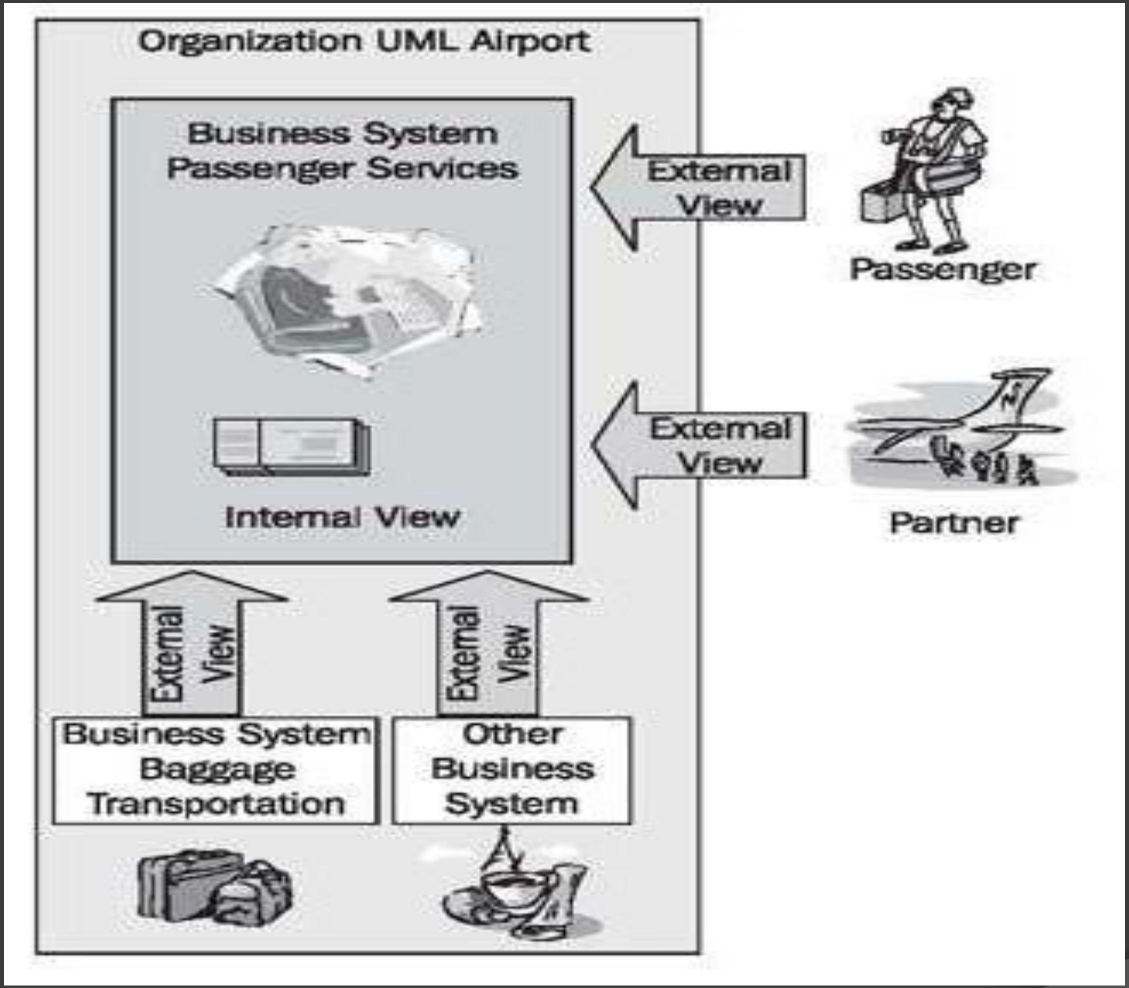
UML views and diagrams cont.

Major View	**Diagram**	**Concepts**
physical	deployment diagram	artifact, dependency, manifestation, node
model management	package diagram	import, model, package constraint, profile, stereotype, tagged value

One Model—Two Views

- External View
- Internal View

External and internal view



External and internal view of the business system

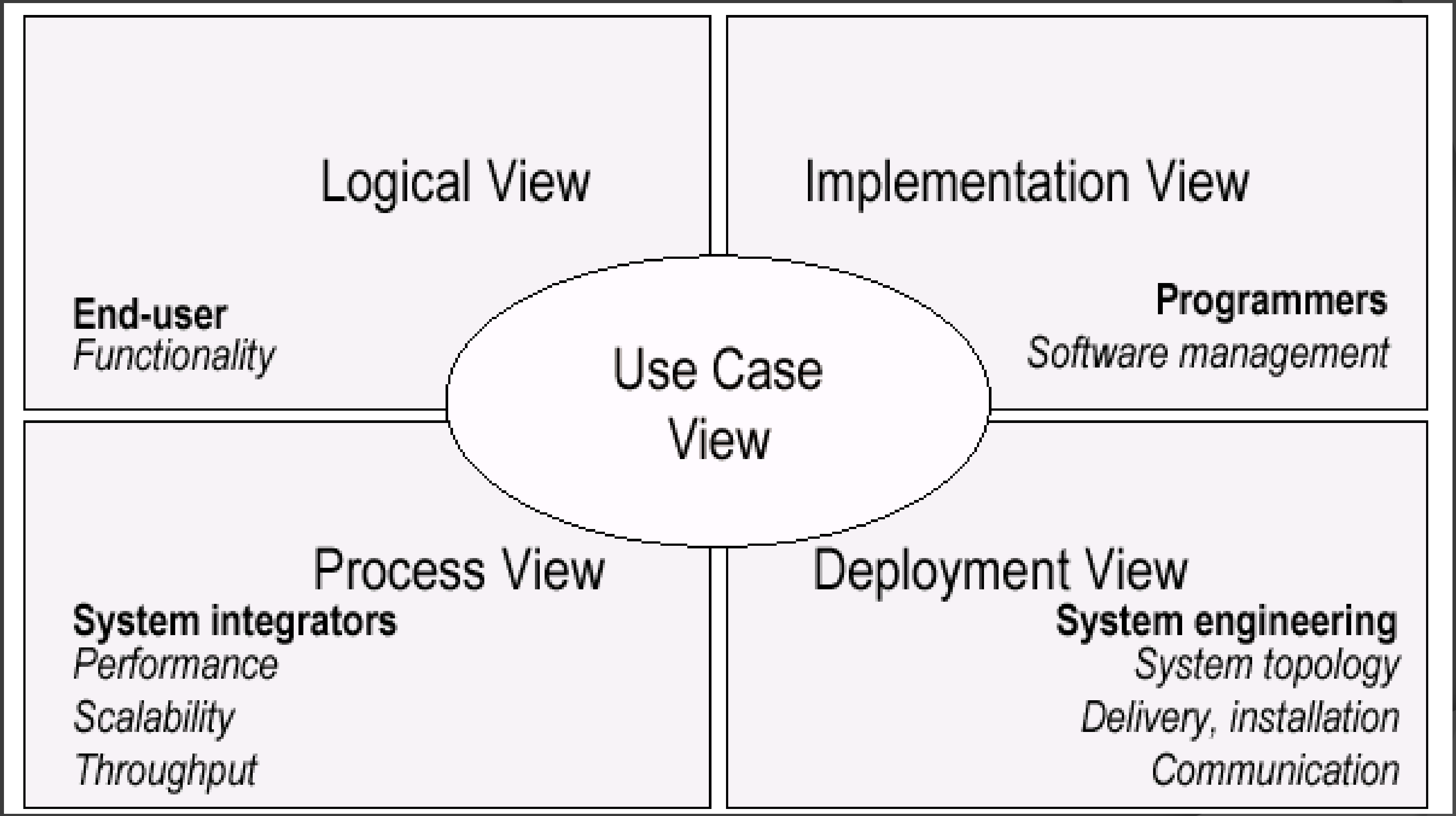
What is View Model ?

- A view model in systems engineering or software engineering is a framework.
- It defines a coherent set of views to be used in the construction of a system architecture or software architecture.
- A view is a representation of a whole system from the perspective of a related set of concerns.
- Viewpoint modeling has become an effective approach for dealing with the inherent complexity of large distributed systems.

Intent of 4+1 view model

- To come up with a mechanism to separate the different aspects of a software system into different views of the system.
- But why???? -> Different stakeholders always have different interest in a software system.

4+1 View Model of Architecture



How Many Views?

- Views should to fit the context
  - Not all systems require all views
  - Single processor: drop deployment view
  - Single process: drop process view
  - Very small program: drop implementation view
- A system might need additional views
  - Data view, security view, …

Use Case View

- The use-case view describes the functionality the system should deliver, as perceived by external actors.
- An actor interacts with the system; the actor can be a user or another system.
- The use-case view is used by customers, designers, developers, and testers; it is described in use-case diagrams, sometimes with support from activity diagrams.
- The desired usage of the system is described as a number of use cases in the use-case view, where a use case is a generic description of a function requested.

Central View

- The use-case view is central, because its contents drive the development of the other views.
- The final goal of the system is to provide the functionality described in this view—along with some nonfunctional properties.
- Hence, this view affects all the others.

Logical View

- The logical view describes how the system’s functionality is provided.
- Mainly for designers and developers.
- In contrast to the use-case view, the logical view looks inside the system.
- It describes both the static structure (classes, objects, and relationships) and the dynamic collaborations that occur when the objects send messages to each other to provide a given function.
- Properties such as persistence and concurrency are also defined, as well as the interfaces and the internal structure of classes.

Implementation view

- The implementation view describes the main modules and their dependencies.
- It is mainly for developers
- Consists of the main software artifacts. The artifacts include different types of code modules shown with their structure and dependencies.

Process View

- The process view deals with the division of the system into processes and processors. This aspect allows for efficient resource usage, parallel execution, and the handling of asynchronous events from the environment.
- This view must also deal with the communication and synchronization of these threads.

Deployment view

- Finally, the deployment view shows the physical deployment of the system, such as the computers and devices (nodes) and how they connect to each other.
- The various execution environments within the processors can be specified as well.

Why is it called the 4 + 1 instead of just 5?

- **The use case view has a special significance.**
- **When all other views are finished, it's effectively redundant.**
- **However, all other views would not be possible without it.**
- **It details the high levels requirements of the system.**
- **The other views detail how those requirements are realized.**

4+1 View model came before UML

- **It's important to remember the 4 + 1 approach was put forward two years before the first introduction of UML.**
- **UML is how most enterprise architectures are modeled and the 4 + 1 approach still plays a relevance to UML today.**
- **UML has 13 different types of diagrams - each diagram type can be categorized into one of the 4 + 1 views.**
- **UML is 4 + 1 friendly!**

Is it important?

- It makes modeling easier.
- Better organization with better separation of concern.
- The 4 + 1 approach provides a way for architects to be able to prioritize modeling concerns.
- The 4 + 1 approach makes it possible for stakeholders to get the parts of the model that are relevant to them.

Use Case View

Use Case Analysis is a technique to capture business process from user's perspective.

Static aspects in use case diagrams; Dynamic aspects in interaction (state chart and activity) diagrams.

Design View

Encompasses classes, interfaces, and collaborations that define the vocabulary of a system., Supports functional requirements of the system.

Static aspects in class and object diagrams; Dynamic aspects in interaction diagrams.

Process View

Encompasses the threads and processes defining concurrency and synchronization, Addresses performance, scalability, and throughput.

Static and dynamic aspects captured as in design view; emphasis on active classes.

Implementation View

Encompasses components and files used to assemble and release a physical system.

Addresses configuration management.

Static aspects in component diagrams; Dynamic aspects in interaction diagrams.

Deployment View

Encompasses the nodes that form the system hardware topology, Addresses distribution, delivery, and installation.

- Static aspects in deployment diagrams; Dynamic aspects in interaction diagrams..

Homogenization of the system

Homogenization of the system

- The word homogenize means to blend into a smooth mixture, to make homogeneous.
- In parallel design process, several stimuli with the same purpose or meaning are defined by several designers. These stimuli should be consolidated to obtain as few stimuli as possible. It is called homogenization.
- Homogenize - to change (something) so that its parts are the same or similar.

Why Homogenize?

- As more use cases and scenarios are developed it is necessary to make the model homogeneous. This is especially true if multiple teams are working on different parts of the model.
- Since use cases and scenarios deal with the written word, people may use different words to mean the same thing or they may interpret words differently. This is just the natural maturation of the model during the project life cycle.
- Homogenization does not happen at one point in the life cycle—it must be on-going.

Why Homogenize?

- Homogenization does not happen at one point in the life cycle—it must be on-going.
- Projects that wait until the end to synch up information developed by multiple groups of people are doomed to failure.
- The most successful projects are made up of teams that have constant communication mechanisms employed. Communication may be as simple as a phone call or as formal as a scheduled meeting—it all depends on the project and the nature of the need to talk.
- The only thing that matters is the fact that the teams are not working in isolation.

Elements of Homogenization

- Combining Classes
- Splitting Classes
- Eliminating Classes
- Consistency Checking
- Scenario Walk-Through
- Event Tracing
- Documentation Review

Combining Classes

- Examine each class along with its definition.
- Also examine the attributes and operations defined for the classes, and look for the use of synonyms.
- Once you determine that two classes are doing the same thing, choose the class with the name that is closest to the language used by the customers.

Combining Classes

- Pay careful attention to the control classes created for the system.
- Initially, one control class is allocated per use case. This might be overkill—control classes with similar behavior may be combined.
- Examine the sequencing logic in the control classes for the system. If it is very similar, the control classes may be combined into one control class.

Splitting Classes

- Classes should be examined to determine if they are following the golden rule of OO, which states that a class should do one thing and do it really well.
- Example :
- A StudentInformation class contains:
  - Info about student Actor
  - And info about the courses that student has successfully completed.
- This is better modeled as two classes—StudentInformation and Transcript,

Eliminating Classes

- A class may be eliminated altogether from the model.
- **This happens when:**
  - The class does not have any structure or behavior
  - The class does not participate in any use cases

- Guideline , examine control classes.
- Lack of sequencing responsibility may lead to the deletion of the control class. This is especially true if the control class is only a passthrough— that is, the control class receives information from a boundary class and immediately passes it to an entity class without the need for sequencing logic.

Consistency Checking

- Consistency checking is needed since the static view of the system, as shown in class diagrams, and the dynamic view of the system, as shown in use case diagrams and interaction diagrams, are under development in parallel.
- Because both views are under development concurrently they must be cross-checked to ensure that different assumptions or decisions are not being made in different views.

Consistency Checking

- The team should be composed of a cross-section of personnel— analysts and designers, customers or customer representatives, domain experts, and test personnel

Scenario Walk-Through

- A primary method of consistency checking is to walk through the high-risk scenarios as represented by a sequence or collaboration diagram.
- Since each message represents behavior of the receiving class, verify that each message is captured as an operation on the class diagram.
- Verify that two interacting objects have a pathway for communication via either an association or an aggregation.

Scenario Walk-Through

- For each class represented on the class diagram, make sure the class participates in at least one scenario. For each operation listed for a class, verify that either the operation is used in at least one scenario or it is needed for completeness.
- Finally, verify that each object included in a sequence or collaboration diagram belongs to a class on the class diagram.

Event Tracing

- **For every message shown in a sequence or collaboration diagram,**
  - Verify that an operation is responsible to send the message in the sending class
  - Verify that an operation is responsible to receive the message in the receiving class and handles it properly.
  - Verify that there exists some kind of association between sending and receiving classes on the class diagram.
  - Verify each message in the state transition diagram.

Documentation Review

- Each class should be documented!
- Check for uniqueness of class names and review all definitions for completeness.
- Ensure that all attributes and operations have a complete definition.
- Finally, check that all standards, format specifications, and content rules established for the project have been followed.