CS 40107: Object Oriented Modeling and Design

Lecture 2

Introduction to Modeling

Software Engineering revisited

Engineering…

- …creates cost-effective solutions to practical problems by applying

  scientific knowledge to building things in the service of humankind”

Software Engineering:

- the “things” contain software (??)

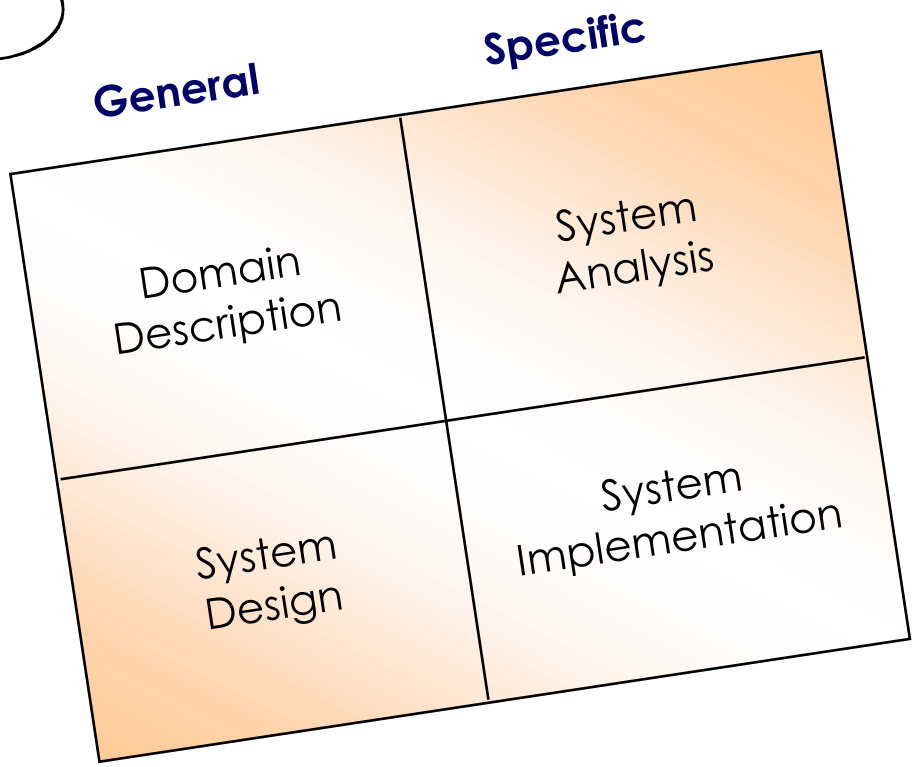
BUT:

- pure software is useless!
  - …software exists only as part of a system
- software is invisible, intangible, abstract
- there are no physical laws underlying software behaviour
- there are no physical constraints on software complexity
- software never wears out
- software can be replicated perfectly

The System Issue

System Concept

- General
- People



System Representation

- Technology
- Specific

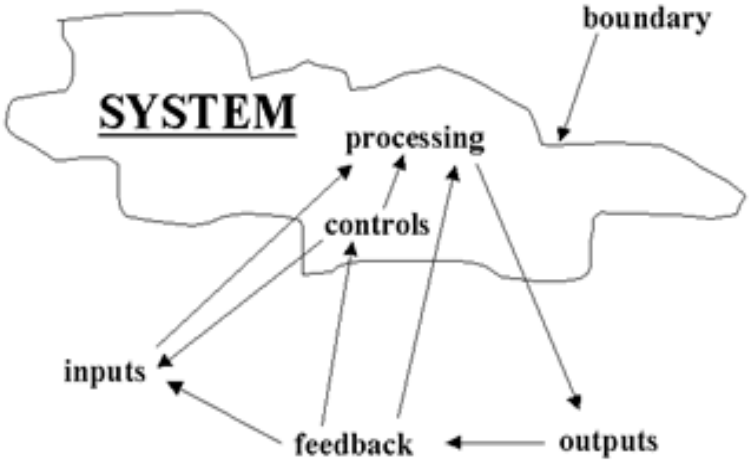
System

- ANSI/EIA-632-1999: "An aggregation of end products and enabling products to achieve a given purpose."
- IEEE Std 1220-1998: "A set or arrangement of elements and processes that are related and whose behavior satisfies customer/operational needs and provides for life cycle sustainment of the products."
- ISO/IEC15288:2008: "A combination of interacting elements organized to achieve one or more stated purposes."
- NASA Systems Engineering Handbook:
  - The combination of elements that function together to produce the capability to meet a need. The elements include all hardware, software, equipment, facilities, personnel, processes, and procedures needed for this purpose.
  - The end product (which performs operational functions) and enabling products (which provide life-cycle support services to the operational end products) that make up a system."

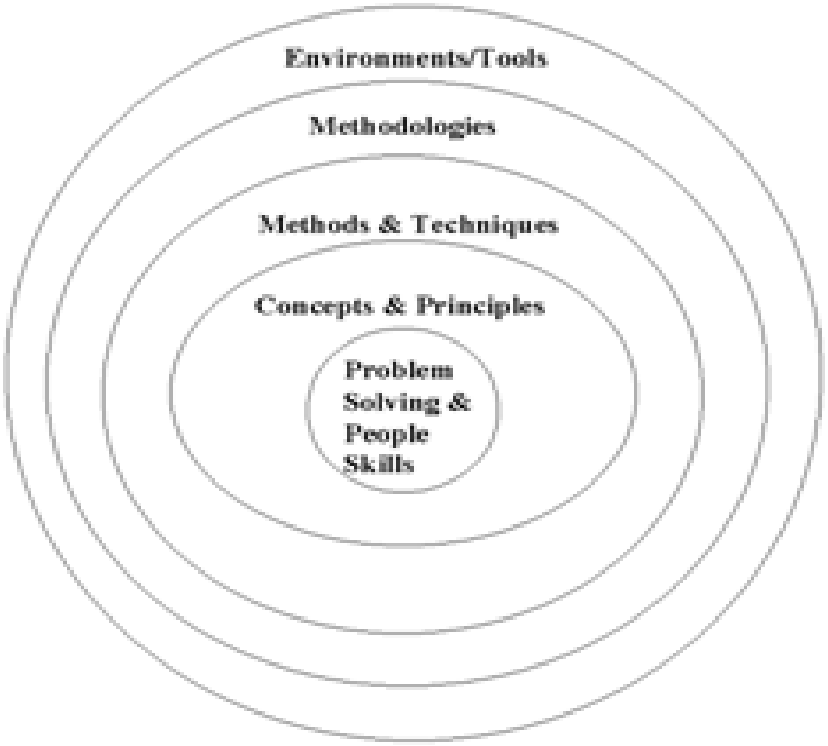
System

- INCOSE Systems Engineering Handbook: "homogeneous entity that exhibits predefined behavior in the real world and is composed of heterogeneous parts that do not individually exhibit that behavior and an integrated configuration of components and/or subsystems."
- INCOSE:
  - "A system is a construct or collection of different elements that together produce results not obtainable by the elements alone.
  - The elements, or parts, can include people, hardware, software, facilities, policies, and documents; that is, all things required to produce systems-level results.
  - The results include system level qualities, properties, characteristics, functions, behavior and performance. The value added by the system as a whole, beyond that contributed independently by the parts, is primarily created by the relationship among the parts; that is, how they are interconnected."

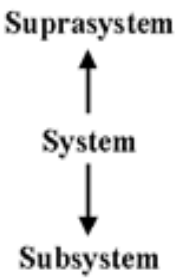
A Sample Cross-Section

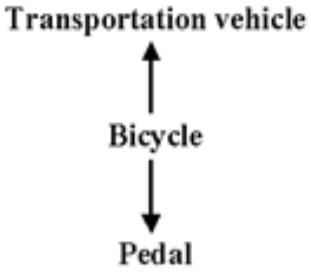


a) Systems Model with Six Components

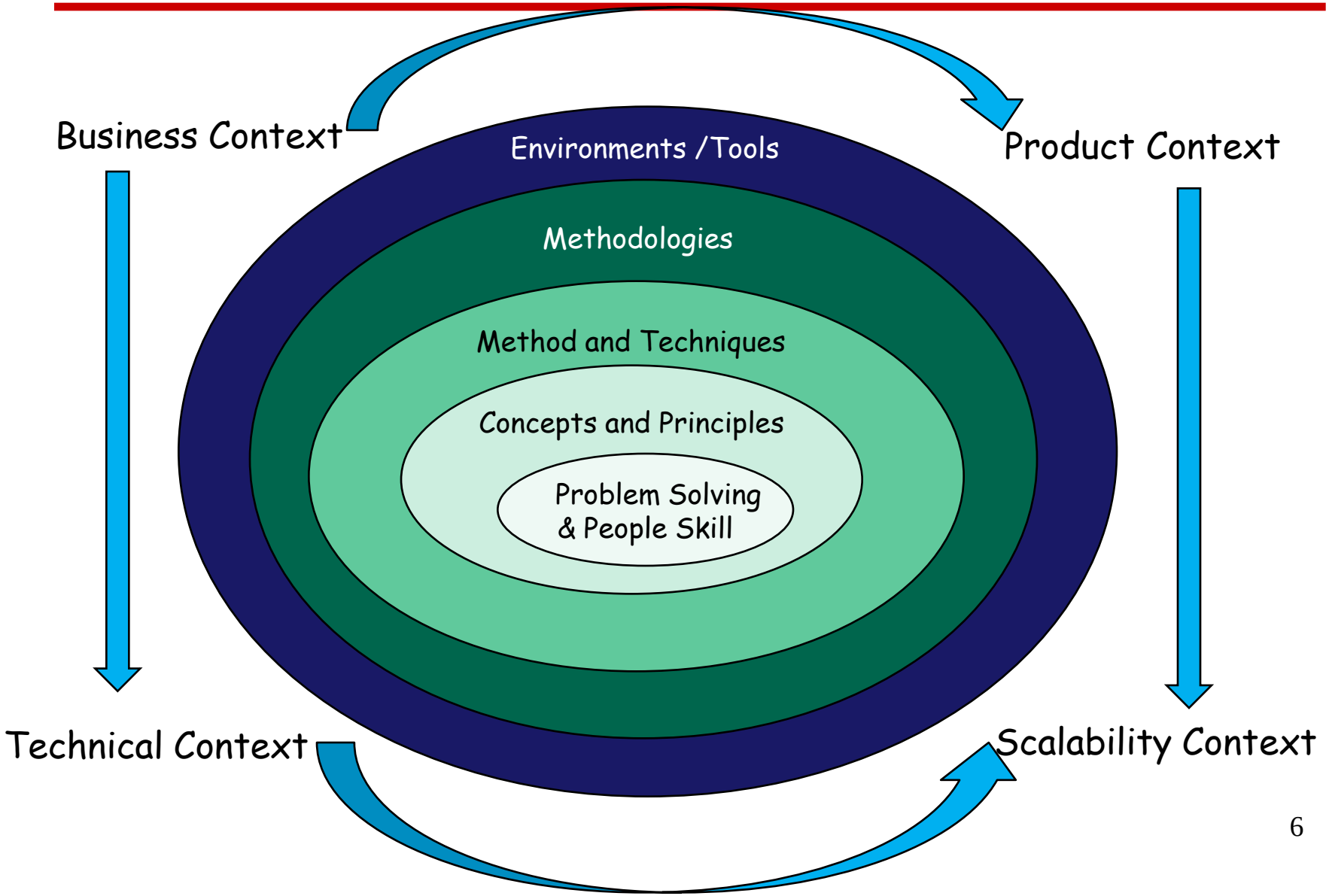


plus:
- Knowledge of functional business areas
- verbal and written communication skill
- work experience in systems analysis and design

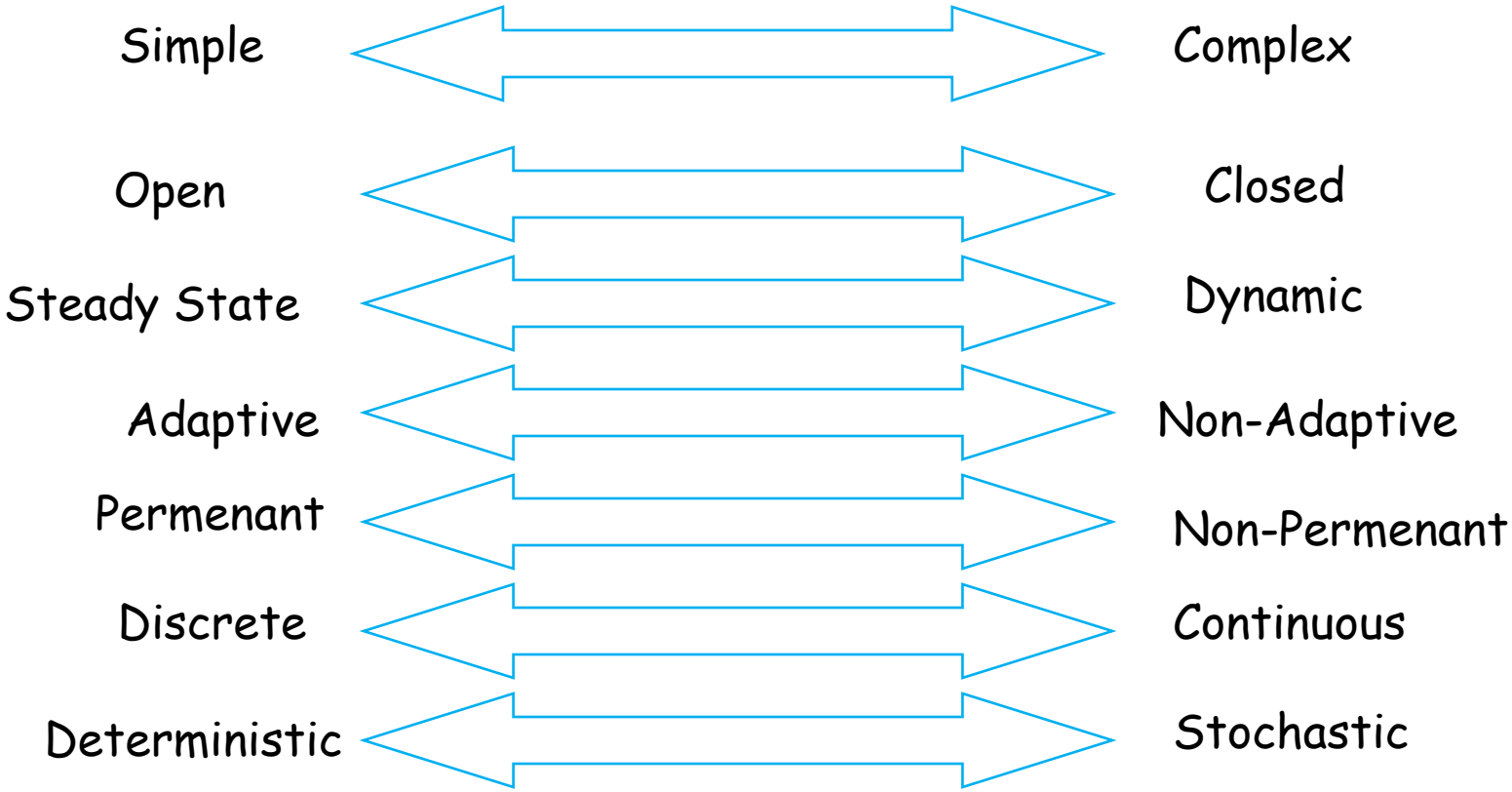




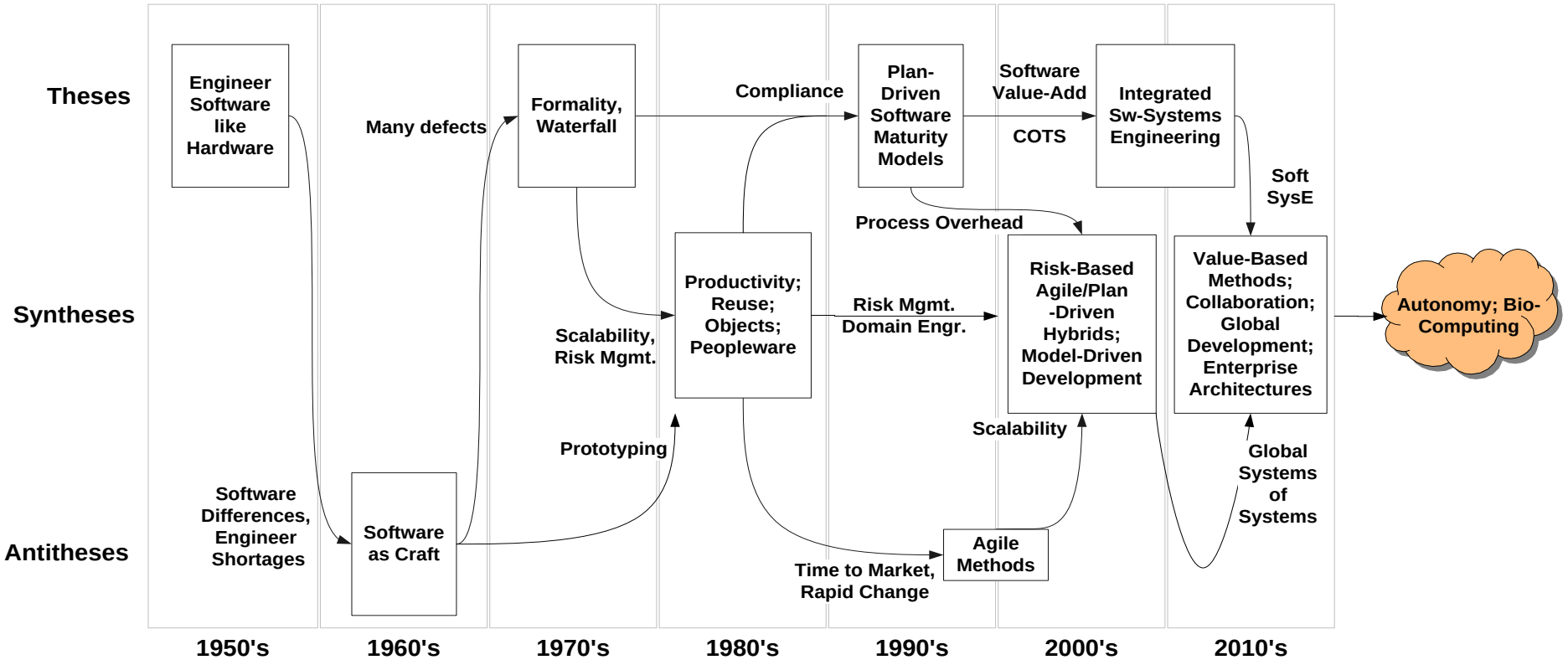
Product Evolution



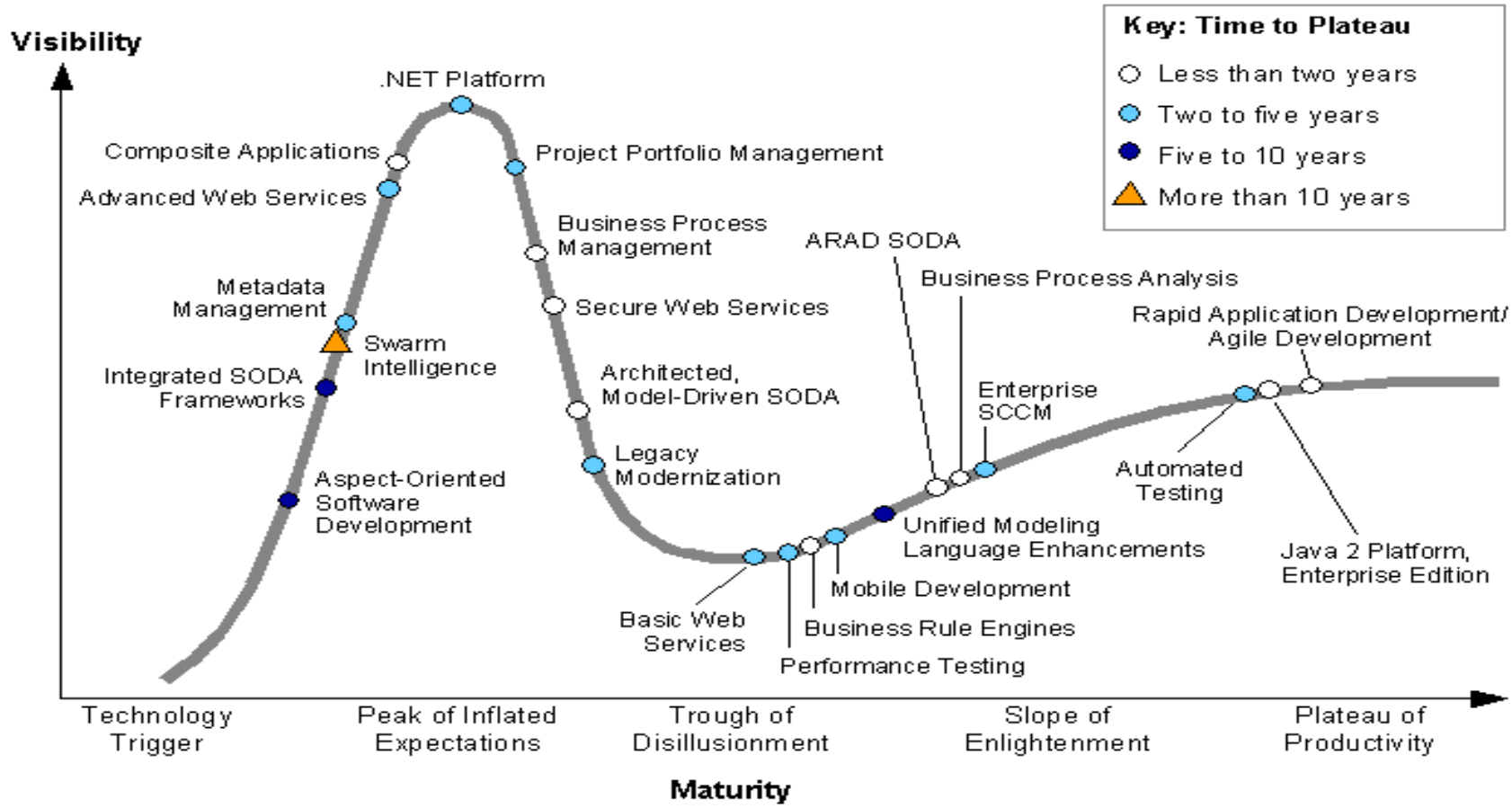
System Types



SE Evolution



MDA Adoption Story

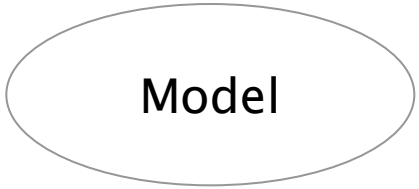


Acronym Key

ARAD architected, rapid application development
SCCM software change and configuration management
SODA service-oriented development of applications

Models of software systems

Models can provide a way to specify clearly system design, architecture, function



Resolve ambiguities using a clearly specified and standard set of modelling tools

Models aid communication between client and designer

Models aid understanding of

- Functionality of the system
- How well a software system matches the desired process

Business process

Software system

Models of Interest

- **model** is a pattern, plan, representation (especially in miniature), or description designed to show the main object or workings of an object, system, or concept.
  - Business model, a framework expressing the business logic of a firm
  - Causal model, an abstract model that uses cause and effect logic
  - Computer model, a computer program which attempts to simulate an abstract model of a particular system
  - Data model a description of the structure of a database
  - Mathematical model, an abstract model that uses mathematical language
  - Model (abstract), an abstraction or conceptual object used in the creation of a predictive formula
  - Model Driven Engineering, the systematic use of models in engineering
  - Model theory, study of the representation of mathematical concepts
  - Morphological modelling, a problem-solving technique used for problems with which causal modelling does not function well
  - Scientific modelling, the process of generating abstract models
  - Similitude (model), in engineering, used in the scientific testing of physical models
  - Working Model, engineering software

Models of software systems

Models can be used to describe different aspects of a system

- All models are developed from a particular perspective
  - **External** perspective shows context of system and relationship to environment (e.g. other systems in the business).
  - **Behavioural** perspective shows dynamic and functional aspects of the system (e.g. how data structures are processed).
  - **Structural** perspective shows the structure of the system and data (e.g. relationships between components of the system and data structures).
- Models may be **abstract** (simplified) or **concrete** (detailed).

Mental Models

- A mental model is a kind of internal symbol or representation of external reality, hypothesized to play a major role in cognition and decision-making. Once formed, mental models may replace carefully considered analysis as a means of conserving time and energy.
- Mental model is generally:
  - found on hardly qualifiable, impugnable, obscure, or incomplete facts
  - **Flexibility** is considerably variable in positive as well as in negative sense
  - effects as **information filter** - causes selective perception , perception of us only selected parts of information
  - compared with the complexities surrounding the world is **very limited** and even when the model is extensive and in accordance with a certain reality in the derivation of logical consequences of it we are very limited. We must take into account such as restrictions on working memory - ie. well-known rule on the maximum number of elements that we are suddenly able to remember, or failure of the principles of logic, etc.
  - source of information, which can not find anywhere else are available at any time and can be used, if other routes are possible, which is linked with the fact that it is not always clearly understood by the other and the process of interpretation can be interpreted in different ways

How we Decide

- Three basic forms are used:
  - **Hexagons**- through them we can express the essential dependencies in the population. Items that are sharing the edge are related
  - **Casual Loop Diagrams**- are used to display tendency and a direction of information connections and the resulting causality
  - **Flow diagram**- a most perfect way to express a dynamic system

What we do:

- **System dynamics** - extending our mental models through the creation of , which are clear, easily communicating and can be compared with each other.
- **Systemic thinking**- seeking the means to improve the mental models and thereby improve the quality of dynamic decisions that are based on mental models

Types of models

Abstract models

Hide/Suppress Principle

- Abstract models provide an overview (abstraction) of an entire system, and shows the most important aspects.
- Details are not included.
- Abstract models are most useful in the requirements analysis and design stages.

Concrete models

Expose/Elaborate Principle

During design, models become

- Less abstract and more concrete.
- More formal.
- More detailed.

These models represent the system, and maintain all (most) of the information about the entity that is being modelled. .

This type of model is important in the design, implementation, and testing stages of software development.

Role of models and Types

- Modelling is used for design of software systems before coding begins.
- Models are essential for large projects, and valuable for smaller ones.
- Diagrams and models are used in other areas of engineering.
- Models aid documentation.
- Models enable modular architecture to be developed.
- Modular approach enables code re-use.
- CASE tools enable code generation from model diagrams.

Natural language models.

- Useful for gathering requirement details.
- Natural language is inherently ambiguous, aim for structured approach.
- Can be over flexible and long.

Diagrammatic models.

- Can be ad hoc, or follow a specific notation with clearly defined syntax.
- Can show static or dynamic relationships and behaviours.

Formal descriptions

- State based models using formal algebraic and logical approach.

Combinations

Complementary models

- **Context models** show the relationships of the system to other systems, and are used to establish the system boundaries.
- **Behavioural models** show the processes that are supported by the system.
- **Data flow diagrams** show the flow of information from one process to another. Can be easily understood by client.
- **Statechart models** show how the states of the system change in response to external or internal stimuli.
- **Object models** show attributes, methods and relationships of object classes within the system.

What do We Model? (cont'd)

- Elements of the architectural style
  - Inclusion of specific basic elements (e.g., components, connectors, interfaces)
  - Component, connector, and interface types
  - Constraints on interactions
  - Behavioral constraints
  - Concurrency constraints
  - …

What do we model?

- Static and Dynamic Aspects
  - Static aspects of a system *do not* change as a system runs
    - e.g., topologies, assignment of components/connectors to hosts, …
  - Dynamic aspects *do* change as a system runs
    - e.g., State of individual components or connectors, state of a data flow through a system, …
  - This line is often unclear
    - Consider a system whose topology is relatively stable but changes several times during system startup
- Important distinction between:
  - Models of dynamic aspects of a system (models do not change)
  - Dynamic models (the models themselves change)

What do We Model? (cont'd)

- Functional and non-functional aspects of a system
  - Functional
    - "The system prints medical records"
  - Non-functional
    - "The system prints medical records *quickly* and *confidentially*."
- Architectural models tend to be functional, but like rationale it is often important to capture non-functional decisions even if they cannot be automatically or deterministically interpreted or analyzed

Important Characteristics of Models

- Ambiguity
  - A model is **ambiguous** if it is open to more than one interpretation
- Accuracy and Precision
  - Different, but often conflated concepts
    - A model is **accurate** if it is correct, conforms to fact, or deviates from correctness within acceptable limits
    - A model is **precise** if it is sharply exact or delimited

Accuracy vs. Precision

Inaccurate and imprecise: incoherent or contradictory assertions

(a)

(b)

Accurate but imprecise: ambiguous or shallow assertions

Inaccurate but precise: detailed assertions that are wrong

(c)

(d)

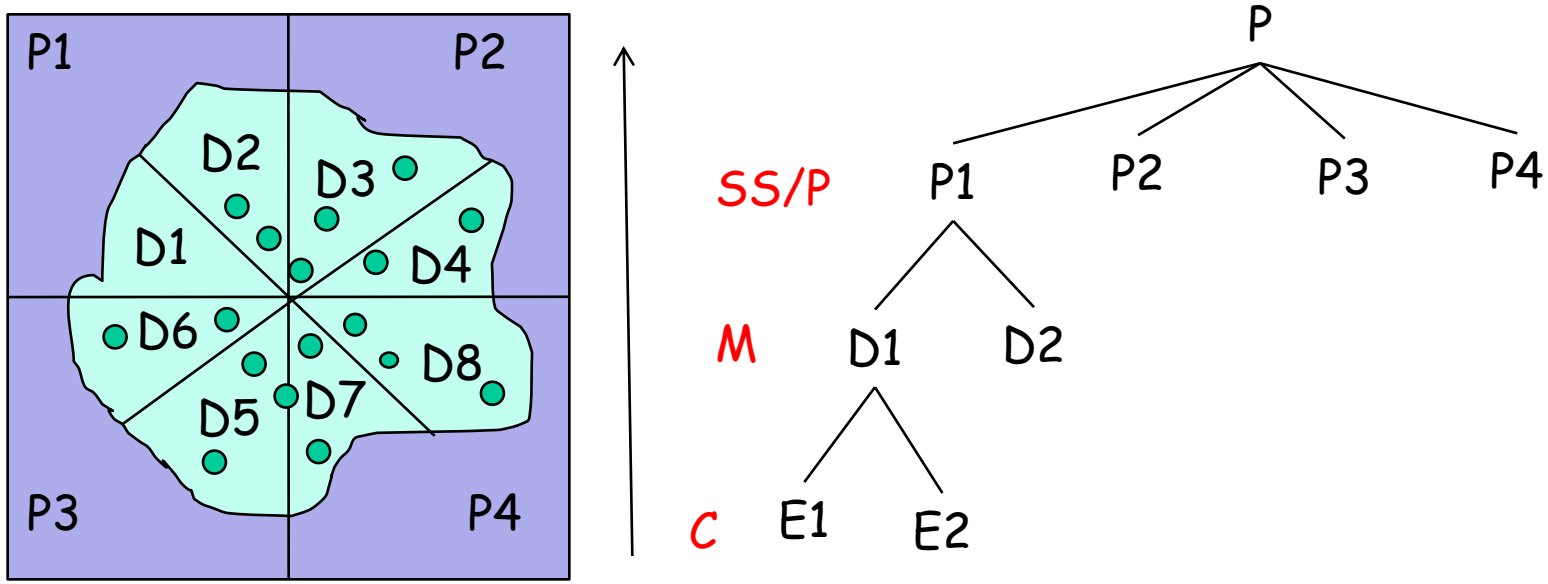
Accurate and precise: detailed assertions that are correct

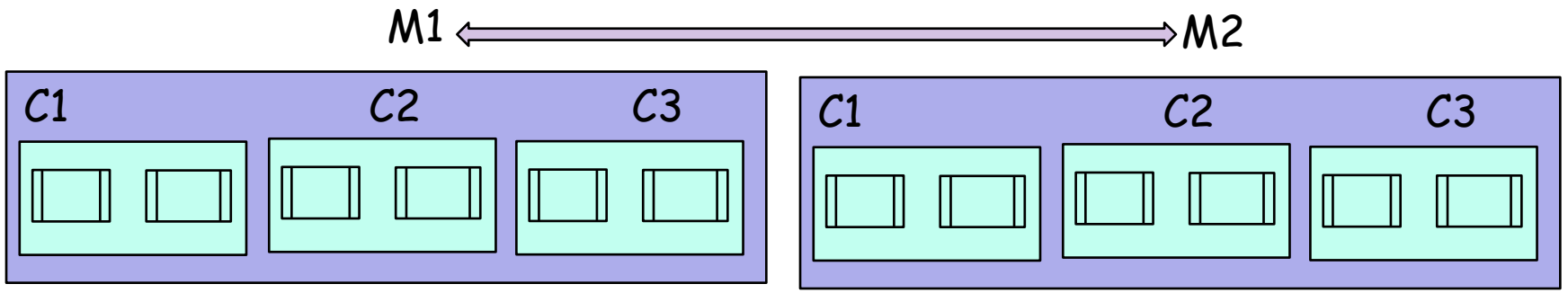
Views and Viewpoints

- Generally, it is not feasible to capture everything we want to model in a single model or document
  - The model would be too big, complex, and confusing
- So, we create several coordinated models, each capturing a subset of the design decisions
  - Generally, the subset is organized around a particular concern or other selection criteria
- We call the **subset-model** a 'view' and the **concern (or criteria)** a 'viewpoint'

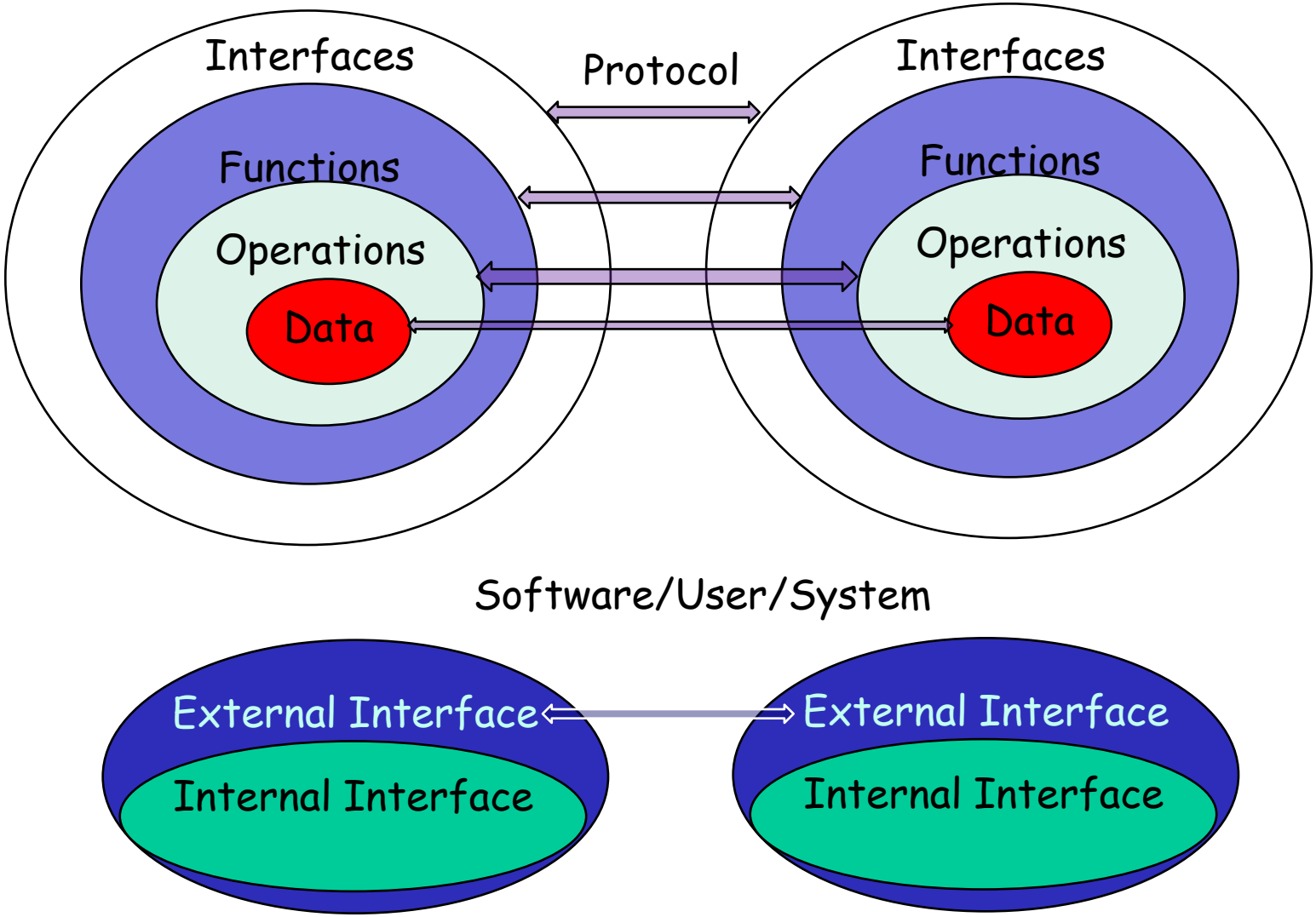
Concern is based on set of related and relevant ASPECTS.

Problem Structuring





Interface Objectives



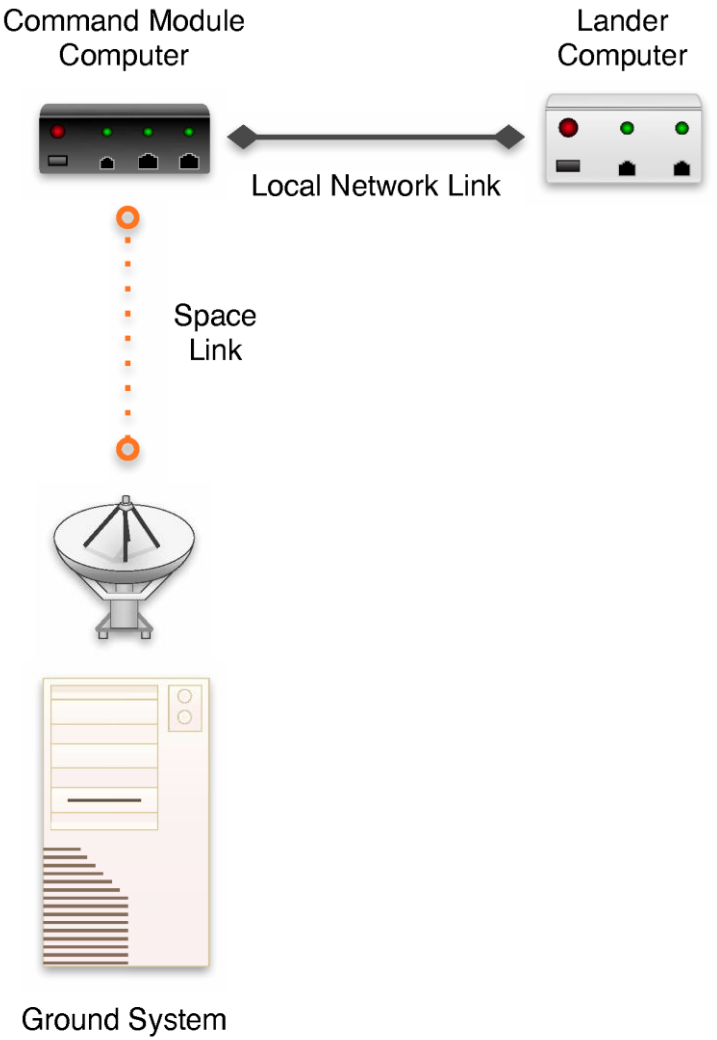
Commonly-Used Viewpoints

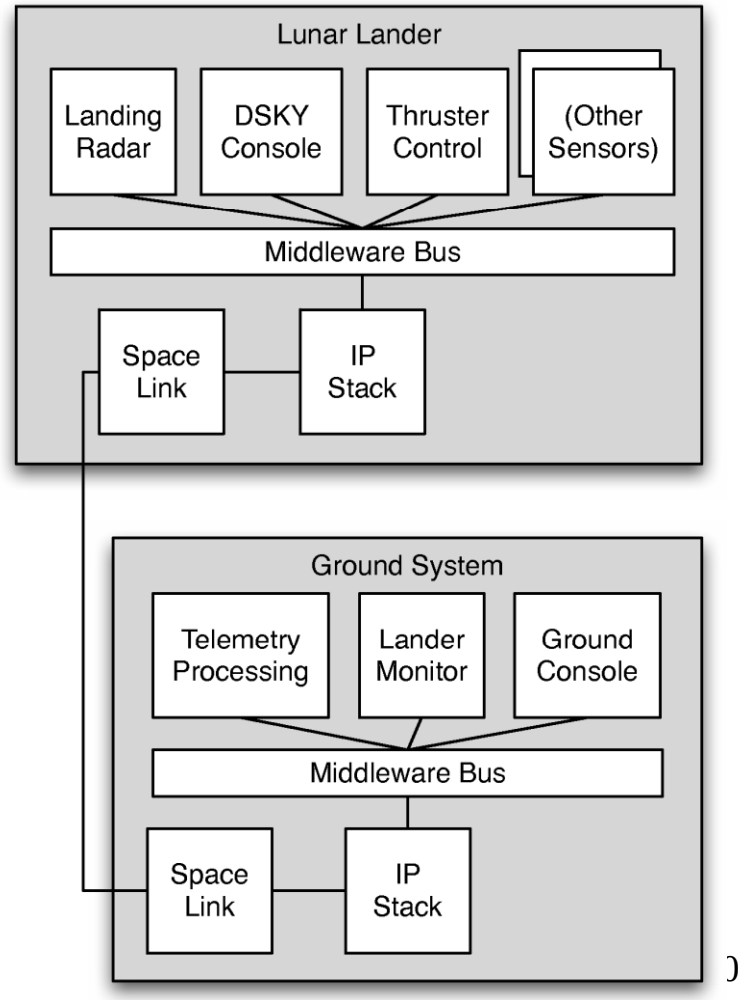
- Logical Viewpoints
  - Capture the logical (often software) entities in a system and how they are interconnected.
- Physical Viewpoints
  - Capture the physical (often hardware) entities in a system and how they are interconnected.
- Deployment Viewpoints
  - Capture how logical entities are mapped onto physical entities.
- Concurrency Viewpoints
  - Capture how concurrency and threading will be managed in a system.
- Behavioral Viewpoints
  - Capture the expected behavior of (parts of) a system.

Consistency Among Views

- Views can contain overlapping and related design decisions
  - There is the possibility that the views can thus become inconsistent with one another
- Views are **consistent** if the design decisions they contain are compatible
  - Views are **inconsistent** if two views assert design decisions that cannot simultaneously be true
- Inconsistency is usually but not always indicative of problems
  - Temporary inconsistencies are a natural part of exploratory design
  - Inconsistencies cannot always be fixed

Example of View Inconsistency





Common Types of Inconsistencies

- Direct inconsistencies
  - E.g., “The system runs on two hosts” and “the system runs on three hosts.”
- Refinement inconsistencies
  - High-level (more abstract) and low-level (more concrete) views of the same parts of a system conflict
- Static vs. dynamic aspect inconsistencies
  - Dynamic aspects (e.g., behavioral specifications) conflict with static aspects (e.g., topologies)
- Dynamic vs. dynamic aspect inconsistencies
  - Different descriptions of dynamic aspects of a system conflict
- Functional vs. non-functional inconsistencies

Evaluating Modeling Approaches

- Scope and purpose
  - What does the technique help you model? What does it *not* help you model?
- Basic elements
  - What are the basic elements (the 'atoms') that are modeled? How are they modeled?
- Style
  - To what extent does the approach help you model elements of the underlying architectural style? Is the technique bound to one particular style or family of styles?
- Static and dynamic aspects
  - What static and dynamic aspects of an architecture does the approach help you model?
- Dynamic modeling
  - To what extent does the approach support models that change as the system executes?
- Non-functional aspects
  - To what extent does the approach support (explicit) modeling of non-functional aspects of architecture?

Evaluating Modeling Approaches (cont'd)

- Ambiguity
  - How does the approach help you to avoid (or embrace) ambiguity?
- Accuracy
  - How does the approach help you to assess the correctness of models?
- Precision
  - At what level of detail can various aspects of the architecture be modeled?
- Viewpoints
  - Which viewpoints are supported by the approach?
- Viewpoint Consistency
  - How does the approach help you assess or maintain consistency among different viewpoints?

Natural Language

- Spoken/written languages such as English
- Advantages
  - Highly expressive
  - Accessible to all stakeholders
  - Good for capturing non-rigorous or informal architectural elements like rationale and non-functional requirements
  - Plentiful tools available (word processors and other text editors)
- Disadvantages
  - Ambiguous, non-rigorous, non-formal
  - Often verbose
  - Cannot be effectively processed or analyzed by machines/software

Natural Language Example

"The Lunar Lander application consists of three components: a **data store** *component, a* **calculation** *component, and a* **user interface** *component.*

The job of the **data store** *component is to store and allow other components access to the height, velocity, and fuel of the lander, as well as the current simulator time.*

The job of the **calculation** *component is to, upon receipt of a burn-rate quantity, retrieve current values of height, velocity, and fuel from the* data store *component, update them with respect to the input burn-rate, and store the new values back. It also retrieves, increments, and stores back the simulator time. It is also responsible for notifying the calling component of whether the simulator has terminated, and with what state (landed safely, crashed, and so on).*

The job of the **user interface** *component is to display the current status of the lander using information from both the* calculation *and the* data store *components. While the simulator is running, it retrieves the new burn-rate value from the user, and invokes the* calculation *component."*

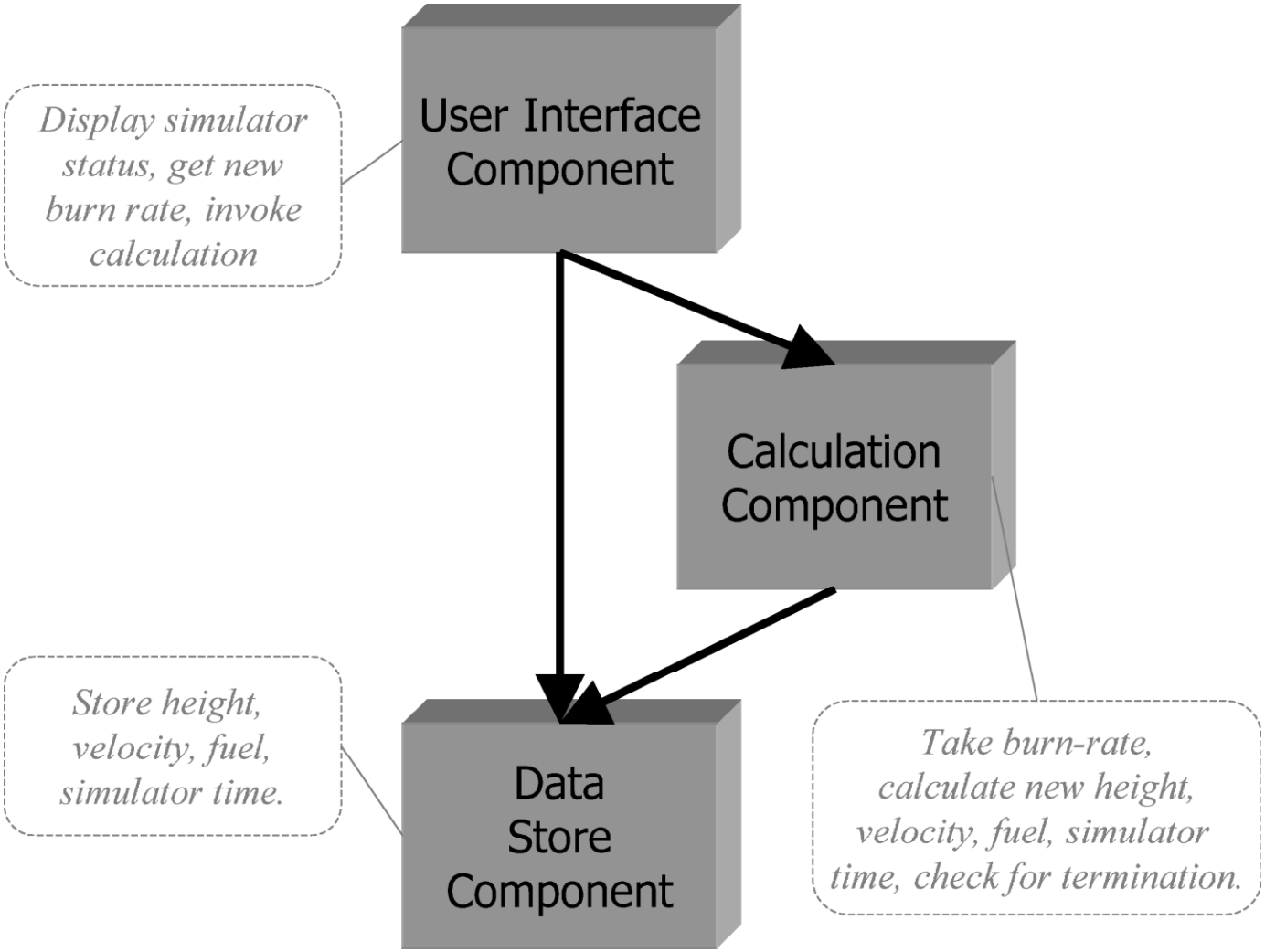
Natural Language Evaluation

- Scope and purpose
  - Capture design decisions in prose form
- Basic elements
  - Any concepts required
- Style
  - Can be described by using more general language
- Static & Dynamic Aspects
  - Any aspect can be modeled
- Dynamic Models
  - No direct tie to implemented/ running system
- Non-Functional Aspects
  - Expressive vocabulary available (but no way to verify)
- Ambiguity
  - Plain natural language tends to be ambiguous; statement templates and dictionaries help
- Accuracy
  - Manual reviews and inspection
- Precision
  - Can add text to describe any level of detail
- Viewpoints
  - Any viewpoint (but no specific support for any particular viewpoint)
- Viewpoint consistency
  - Manual reviews and inspection

Informal Graphical Modeling

- General diagrams produced in tools like PowerPoint and OmniGraffle
- Advantages
  - Can be aesthetically pleasing
  - Size limitations (e.g., one slide, one page) generally constrain complexity of diagrams
  - Extremely flexible due to large symbolic vocabulary
- Disadvantages
  - Ambiguous, non-rigorous, non-formal
    - But often treated otherwise
  - Cannot be effectively processed or analyzed by machines/software

Informal Graphical Model Example



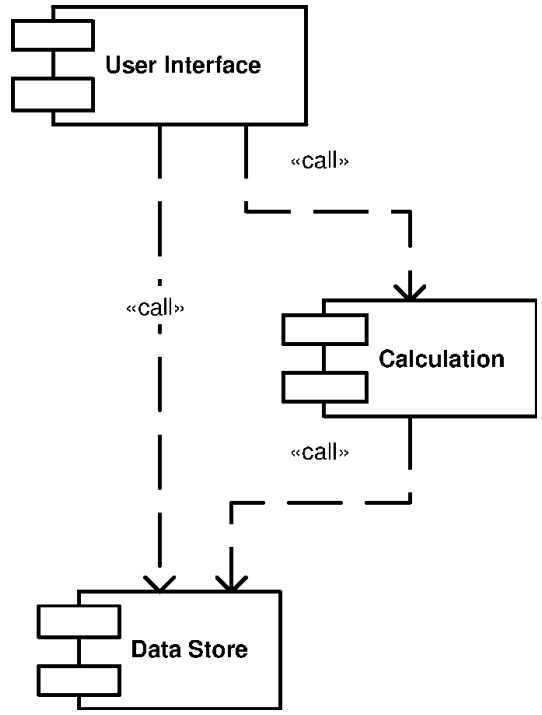
Informal Graphical Evaluation

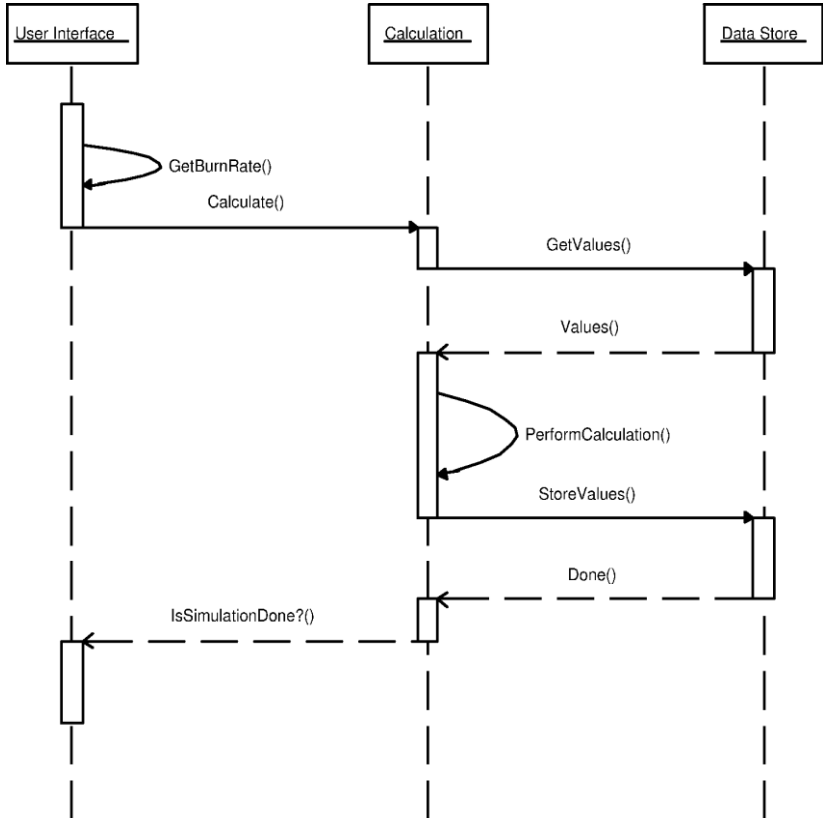
- Scope and purpose
  - Arbitrary diagrams consisting of symbols and text
- Basic elements
  - Geometric shapes, splines, clip-art, text segments
- Style
  - In general, no support
- Static & Dynamic Aspects
  - Any aspect can be modeled, but no semantics behind models
- Dynamic Models
  - Rare, although APIs to manipulate graphics exist
- Non-Functional Aspects
  - With natural language annotations
- Ambiguity
  - Can be reduced through use of rigorous symbolic vocabulary/dictionaries
- Accuracy
  - Manual reviews and inspection
- Precision
  - Up to modeler; generally canvas is limited in size (e.g., one 'slide')
- Viewpoints
  - Any viewpoint (but no specific support for any particular viewpoint)
- Viewpoint consistency
  - Manual reviews and inspection

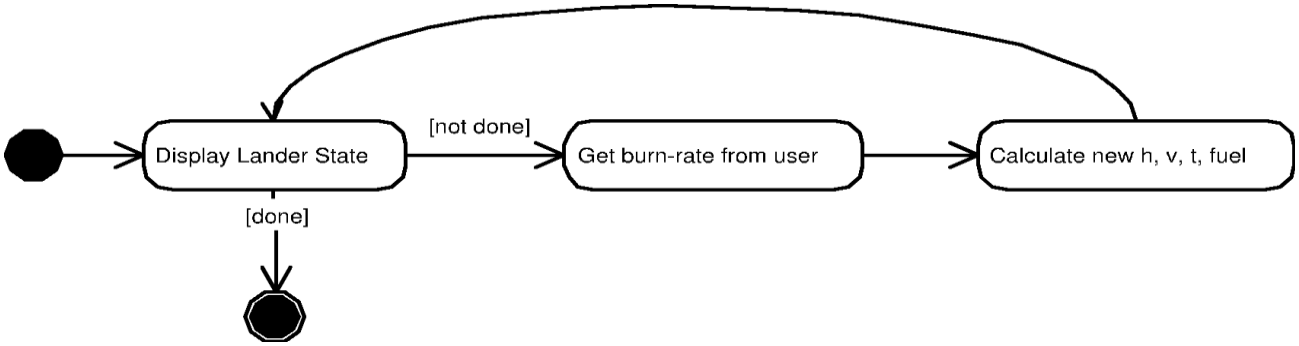
UML – the Unified Modeling Language

- 13 loosely-interconnected notations called diagrams that capture static and dynamic aspects of software-intensive systems
- Advantages
  - Support for a diverse array of viewpoints focused on many common software engineering concerns
  - Ubiquity improves comprehensibility
  - Extensive documentation and tool support from many vendors
- Disadvantages
  - Needs customization through profiles to reduce ambiguity
  - Difficult to assess consistency among views
  - Difficult to capture foreign concepts or views

UML Example


User Interface
«call»
«call»
Calculation
«call»
Data Store

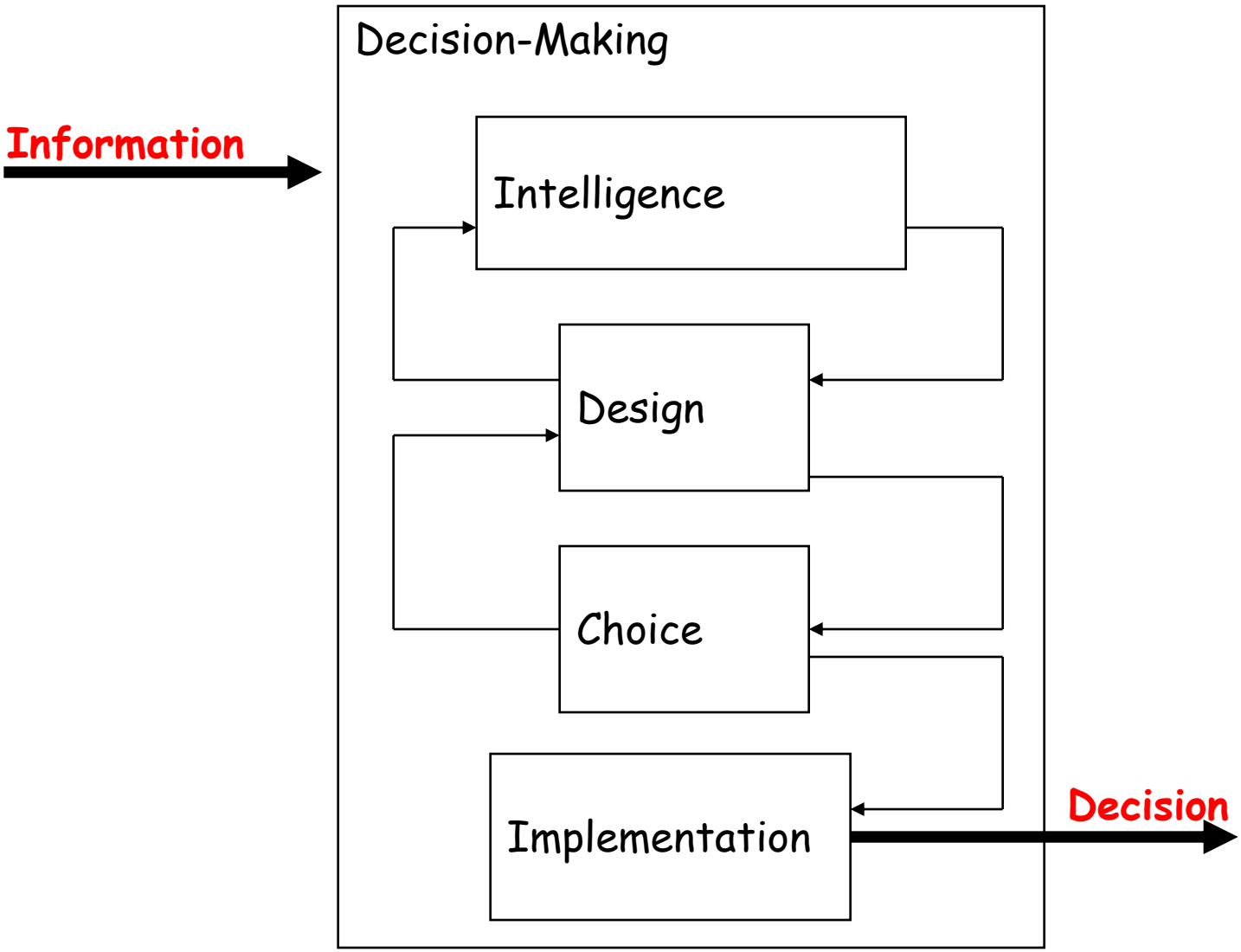

User Interface
Calculation
Data Store
GetBurnRate()
Calculate()
GetValues()
Values()
PerformCalculation()
StoreValues()
Done()
IsSimulationDone?()


Display Lander State
[not done]
Get burn-rate from user
Calculate new h, v, t, fuel
[done]

UML Evaluation

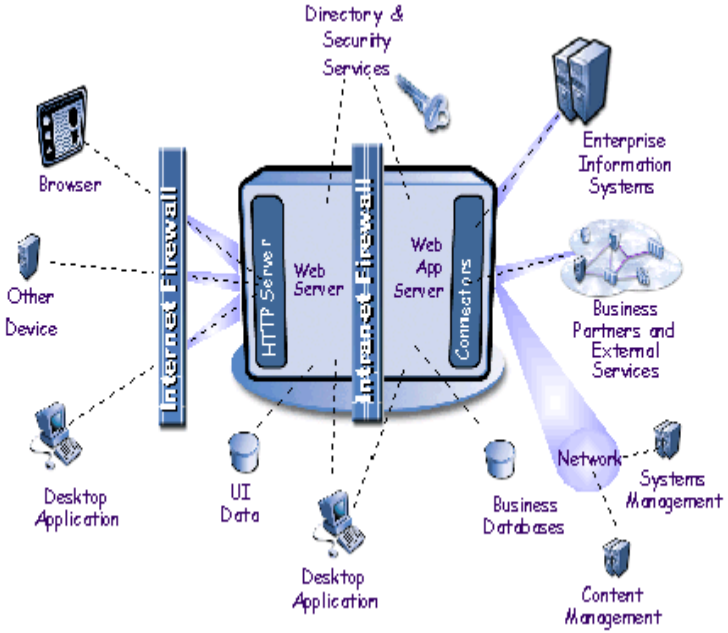
- Scope and purpose
  - Diverse array of design decisions in 13 viewpoints
- Basic elements
  - Multitude – states, classes, objects, composite nodes…
- Style
  - Through (OCL) constraints
- Static & Dynamic Aspects
  - Some static diagrams (class, package), some dynamic (state, activity)
- Dynamic Models
  - Rare; depends on the environment
- Non-Functional Aspects
  - No direct support; natural-language annotations
- Ambiguity
  - Many symbols are interpreted differently depending on context; profiles reduce ambiguity
- Accuracy
  - Well-formedness checks, automatic constraint checking, ersatz tool methods, manual
- Precision
  - Up to modeler; wide flexibility
- Viewpoints
  - Each diagram type represents a viewpoint; more can be added through overloading/profiles
- Viewpoint consistency
  - Constraint checking, ersatz tool methods, manual

Decision-Making Process



Modelling

- Any modelling needs three elements: constructs, notation and principles of construction
- Modelling is normally undertaken for three reasons: representation, communication and abstraction
- Modelling approaches can be distinguished in terms of levels of abstraction: conceptual models, logical models and physical models
- Modelling approaches can also be distinguished in terms of their coverage of particular aspects of an system:
- structural modelling, behavioural modelling and object modelling

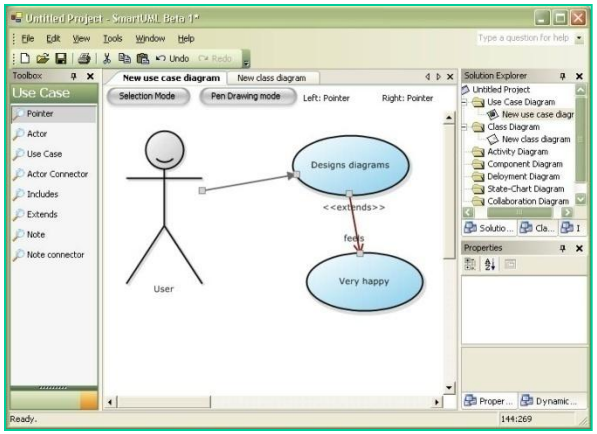


Elements of Modelling

Constructs. By constructs we mean the component elements of the modelling approach

Notation. By notation we mean the form of representation employed for the constructs within the modelling approach. Such a notation can be textual, graphical and/or mathematical. Graphical notations are probably the most frequently employed in information systems work because of ease of use

Principles of Construction. By principles we mean the formal and informal rules for correctly constructing models

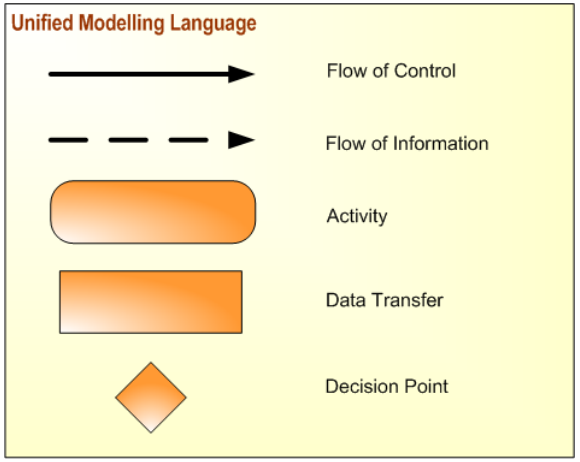


Purpose of Modelling

Communication. The primary use for a model is as a medium of communication between some group of persons

Representation. A model is used to represent common understandings about some real-world phenomena amongst this group of persons

Abstraction. Modelling generally implies some form of simplification of the real world. The modeller uses a model to focus on what are seen to be the important features of some real-world situation



θ a ח כ

ץ o ם