Chapter 5 System modeling is the process of developing abstract models of a system, with each model presenting a different view or perspective of that system. System modeling has now come to mean representing a system using some kind of graphical notation, which is now almost always based on notations in the Unified Modeling Language (UML). System modelling helps the analyst to understand the functionality of the system and models are used to communicate with customers.

Models of the existing system are used during requirements engineering. They help clarify what the existing system does

and can be used as a basis for discussing its strengths and weaknesses. These then lead to requirements for the new system. Models of the new system are used during requirements engineering to help explain the proposed requirements to other system stakeholders. Engineers use these models to discuss design proposals and to document the system for

implementation. In a model-driven engineering process, it is possible to generate a complete or partial system implementation from the system model.

An external perspective, where you model the context or environment of the system. An interaction perspective, where you model the interactions between a system and its environment, or between the components of a system. A structural perspective, where you model the organization of a system or the structure of the data that is processed by the system. A behavioral perspective, where you model the dynamic behavior of the system and how it responds to events.

Activity diagrams, which show the activities involved in a process or in data processing. Use case diagrams, which show the interactions between a system and its environment. Sequence diagrams, which show interactions between actors and the system and between system components. Class diagrams, which show the object classes in the system and the associations between these classes. State diagrams, which show how the system reacts to internal and external events.

As a means of facilitating discussion about an existing or proposed system Incomplete and incorrect models are OK as their role is to support discussion. As a way of documenting an existing system Models should be an accurate representation of the system but need not be complete. As a detailed system description that can be used to

generate a system implementation Models have to be both correct and complete.

Context models are used to illustrate the operational context of a system - they show what lies outside the system boundaries. Social and organizational concerns may affect the decision on where to position system boundaries. Architectural models show the system and its relationship with other systems. System boundaries are established to define what is inside and what is outside the system. They show other systems that are used or depend on the system

being developed. The position of the system boundary has a profound effect on the system requirements. Defining a system boundary is a political judgment There may be pressures to develop system boundaries that increase / decrease the influence or workload of different parts of an organization.

Process perspective Context models simply show the other systems in the environment, not how the system being developed is used in that environment. Process models reveal how the system being developed is used in broader business processes. UML activity diagrams may be used to define business process models. Interaction models Modeling user interaction is important as it helps to identify user requirements. Modeling system-to-system interaction highlights the communication problems that may arise. Use case diagrams and sequence diagrams may be used for interaction modeling.

Use case modeling Use cases were developed originally to support requirements elicitation and now incorporated into the UML. Each use case represents a discrete task that involves external interaction with a system.

Actors in a use case may be people or other systems. Represented diagrammatically to provide an overview of the use case and in a more detailed textual form. Sequence diagrams are part of the UML and are used to model the interactions between the actors and the objects within a system. A sequence diagram shows the sequence of interactions that take place during a particular use case or use case instance. The objects and actors involved are listed along the top of the diagram, with a dotted line drawn vertically from these. Interactions between objects are indicated by annotated arrows.

Structural models of software display the organization of a system in terms of the components that make up that system and their relationships. Structural models may be static models, which show the structure of the system design, or dynamic models, which show the organization of the system when it is executing. Class diagrams are used when developing an object oriented system model to show the classes in a system and the associations between these classes. An object class can be thought of as a general definition of one kind of system object. An association is a link between classes that indicates that there is some relationship between these classes. When you are developing models during the early stages of the software engineering process, objects represent something in the real world, such as a patient, a prescription, doctor, etc. A model is an abstract view of a system that ignores system details. Complementary system models can be developed to show the system’s context, interactions, structure and behavior. Context models show how a system that is being modeled is positioned in an environment with other systems and processes. Use case diagrams and sequence diagrams are used to describe the interactions between users and systems in the system being designed. Use cases describe interactions between a system and external actors; sequence diagrams add more information to these by showing interactions between system objects. Structural models show the organization and architecture of a system. Class diagrams are used to define the static structure of classes in a system and their associations. Generalization is used to manage complexity. Rather than learn the detailed characteristics of every entity that we experience, we place these entities in more general classes (animals, cars, houses, etc.) and learn the characteristics of these classes. This allows us to infer that different members of these classes have some common characteristics e.g. squirrels and rats are rodents. In modeling systems, it is often useful to examine the classes in a system to see if there is scope for generalization. If changes are proposed, then you do not have to look at all classes in the system to see if they are affected by the change. In object-oriented languages, such as Java, generalization is implemented using the class inheritance mechanisms built into the language. In a generalization, the attributes and operations associated with higher-level classes are also associated with the lower-level classes. The lower-level classes are subclasses inherit the attributes and operations from their super classes. These lower-level classes then add more specific attributes and operations. Object class aggregation models An aggregation model shows how classes that are collections are composed of other classes. Aggregation models are similar to the part-of relationship in semantic data models. Behavioral models are models of the dynamic behavior of a system as it is executing. They show what happens or what is supposed to happen when a system responds to a stimulus from its environment. You can think of these stimuli as being of two types: Data Some data arrives that has to be processed by the system. Events Some event happens that triggers system processing. Events may have associated data, although this is not always the case. Data-driven modeling Many business systems are data-processing systems that are primarily driven by data. They are controlled by the data input to the system, with relatively little external event processing. Data-driven models show the sequence of actions involved in processing input data and generating an associated output. They are particularly useful during the analysis of requirements as they can be used to show end-to-end processing in a system. Event-driven modeling Real-time systems are often event-driven, with minimal data processing. Event-driven modeling shows how a system responds to external and internal events. It is based on the assumption that a system has a finite number of states and that events (stimuli) may cause a transition from one state to another. State machine models These model the behavior of the system in response to external and internal events. State machine models show system states as nodes and events as arcs between these nodes. When an event occurs, the system moves from one state to another. State charts are an integral part of the UML and are used to represent state machine models. Model-driven engineering (MDE) is an approach to software development where models rather than programs are the principal outputs of the development process. The programs that execute on a hardware/software platform are then generated automatically from the models. Proponents of MDE argue that this raises the level of abstraction in software engineering so that engineers no longer have to be concerned with programming language details or the specifics of execution platforms. Model-driven engineering is still at an early stage of development, and it is unclear whether or not it will have a significant effect on software engineering practice. Pros Allows systems to be considered at higher levels of abstraction Generating code automatically means that it is cheaper to adapt systems to new platforms. Cons Models for abstraction and not necessarily right for implementation. Savings from generating code may be outweighed by the costs of developing translators for new platforms. Model-driven architecture (MDA) was the precursor of more general model-driven engineering MDA is a model-focused approach to software design and implementation that uses a subset of UML models to describe a system. Models at different levels of abstraction are created. From a high-level, platform independent model, it is possible, in principle, to generate a working program without manual intervention. Types of model A computation independent model (CIM) These model the important domain abstractions used in a system. CIMs are sometimes called domain models. A platform independent model (PIM) These model the operation of the system without reference to its implementation. The PIM is usually described using UML models that show the static system structure and how it responds to external and internal events. Platform specific models (PSM) These are transformations of the platform-independent model with a separate PSM for each application platform. In principle, there may be layers of PSM, with each layer adding some platform-specific detail. Agile methods and MDA The developers of MDA claim that it is intended to support an iterative approach to development and so can be used within agile methods. The notion of extensive up-front modeling contradicts the fundamental ideas in the agile manifesto and I suspect that few agile developers feel comfortable with model driven engineering. If transformations can be completely automated and a complete program generated from a PIM, then, in principle, MDA could be used in an agile development process as no separate coding would be required. Executable UML The fundamental notion behind model-driven engineering is that completely automated transformation of models to code should be possible. This is possible using a subset of UML 2, called Executable UML or xUML. Features of executable UML To create an executable subset of UML, the number of model types has therefore been dramatically reduced to these 3 key types: Domain models that identify the principal concerns in a system. They are defined using UML class diagrams and include objects, attributes and associations. Class models in which classes are defined, along with their attributes and operations. State models in which a state diagram is associated with each class and is used to describe the life cycle of the class. The dynamic behavior of the system may be specified declaratively using the object constraint language (OCL), or may be expressed using UML’s action language. Key points Behavioral models are used to describe the dynamic behavior of an executing system. This behavior can be modeled from the perspective of the data processed by the system, or by the events that stimulate responses from a system. Activity diagrams may be used to model the processing of data, where each activity represents one process step. State diagrams are used to model a system’s behavior in response to internal or external events. Model-driven engineering is an approach to software development in which a system is represented as a set of models that can be automatically transformed to executable code.

Chapter 6 The design process for identifying the sub-systems making up a system and the framework for sub-system control and communication is architectural design. The output of this design process is a description of the software architecture. An early stage of the system design process. Represents the link between specification and design processes. Often carried out in parallel with some specification activities. It involves identifying major system components and their communications. Architectural abstraction Architecture in the small is concerned with the architecture of individual programs. At this level, we are concerned with the way that an individual program is decomposed into components. Architecture in the large is concerned with the architecture of complex enterprise systems that include other systems, programs, and program components. These enterprise systems are distributed over different computers, which may be owned and managed by different companies. Advantages of explicit architecture Stakeholder communication Architecture may be used as a focus of discussion by system stakeholders. System analysis Means that analysis of whether the system can meet its nonfunctional requirements is possible. Large-scale reuse The architecture may be reusable across a range of systems Product-line architectures may be developed. Architectural representations Simple, informal block diagrams showing entities and relationships are the most frequently used method for documenting software architectures. But these have been criticized because they lack semantics, do not show the types of relationships between entities nor the visible properties of entities in the architecture. Depends on the use of architectural models. The requirements for model semantics depends on how the models are used. Box and line diagrams Very abstract - they do not show the nature of component relationships nor the externally visible properties of the sub-systems. However, useful for communication with stakeholders and for project planning. Use of architectural models As a way of facilitating discussion about the system design As a way of documenting an architecture that has been designed Architectural design is a creative process so the process differs depending on the type of system being developed. Architecture and system characteristics Performance Localize critical operations and minimize communications. Use large rather than fine-grain components. Security Use a layered architecture with critical assets in the inner layers. Safety Localize safety-critical features in a small number of subsystems. Availability Include redundant components and mechanisms for fault tolerance. Maintainability Use fine-grain, replaceable components. 4 + 1 view model of software architecture A logical view, which shows the key abstractions in the system as objects or object classes. A process view, which shows how, at run-time, the system is composed of interacting processes. A development view, which shows how the software is decomposed for development. A physical view, which shows the system hardware and how software components are distributed across the processors in the system. Architectural patterns Patterns are a means of representing, sharing and reusing knowledge. An architectural pattern is a stylized description of good design practice, which has been tried and tested in different environments. Patterns should include information about when they are and when the are not useful. Patterns may be represented using tabular and graphical descriptions. MVC (Model-View-Controller) Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model. Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown. " Advantages! Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them. " Disadvantages! Can involve additional code and code complexity when the data model and interactions are simple." Layered architecture Used to model the interfacing of sub-systems. Organizes the system into a set of layers (or abstract machines) each of which provide a set of services. Supports the incremental development of sub-systems in different layers. When a layer interface changes, only the adjacent layer is affected. However, often artificial to structure systems in this way. Layered architecture Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system. When used! Used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsibility for a layer of functionality; when there is a requirement for multi-level security." Advantages! Allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system." Disadvantages! In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer." Key points A software architecture is a description of how a software system is organized. Architectural design decisions include decisions on the type of application, the distribution of the system, the architectural styles to be used. Architectures may be documented from several different perspectives or views such as a conceptual view, a logical view, a process view, and a development view. Architectural patterns are a means of reusing knowledge about generic system architectures. They describe the architecture, explain when it may be used and describe its advantages and disadvantages. Repository architecture Sub-systems must exchange data. This may be done in two ways: Shared data is held in a central database or repository and may be accessed by all sub-systems; Each sub-system maintains its own database and passes data explicitly to other sub-systems. The Repository pattern All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository. Advantages" Components can be independent—they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently (e.g., backups done at the same time) as it is all in one place. " Disadvantages" The repository is a single point of failure so problems in the repository affect the whole system. May be inefficiencies in organizing all communication through the repository. Distributing the repository across several computers may be difficult." Client-server architecture Distributed system model which shows how data and processing is distributed across a range of components. Set of stand-alone servers which provide specific services such as printing, data management, etc. Set of clients which call on these services. Network which allows clients to access servers. The Client–server pattern In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them. Used when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable." Advantages" The principal advantage of this model is that servers can be distributed across a network. Disadvantages" Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the systemPipe and filter architecture Functional transformations process their inputs to produce outputs. May be referred to as a pipe and filter model (as in UNIX shell). Variants of this approach are very common. When transformations are sequential, this is a batch sequential model which is extensively used in data processing systems. The pipe and filter pattern The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing. Advantages" Easy to understand and supports transformation reuse. Workflow style matches the structure of many business processes. Evolution by adding transformations is straightforward. Disadvantages" The format for data transfer has to be agreed upon between communicating transformations. Each transformation must parse its input and unparse its output to the agreed form. This increases system overhead and may mean that it is impossible to reuse functional transformations that use incompatible data structures." Application architectures Application systems are designed to meet an organizational need. As businesses have much in common, their application systems also tend to have a common architecture that reflects the application requirements. A generic application architecture is an architecture for a type of software system that may be configured and adapted to create a system that meets specific requirements. Data processing applications Data driven applications that process data in batches without explicit user intervention during the processing. Transaction processing applications Data-centred applications that process user requests and update information in a system database. Event processing systems Applications where system actions depend on interpreting events from the system’s environment. Language processing systems where the users’ intentions are specified in a formal language that is processed and interpreted by the system. Information systems have a generic architecture that can be organized as a layered architecture. These are transaction-based systems as interaction with these systems generally involves database transactions. Layers include: The user interface User communications Information retrieval System database