DOWNLOAD PDF AN INTRODUCTION TO OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN WITH UML AND THE UNIFIED PROCESS

Chapter 1 : Stumpf & Teague, Object-Oriented Systems Analysis and Design With UML | Pearson

An introduction to object-oriented systems analysis and design with UML and the unified process. [Stephen R Schach] -- This text will be the first to present an object-oriented methodology from the outset for beginning Systems Analysis and Design students.

This may be due to a lack of training, time, budget or the complexity of the process itself. Bringing years of experience training students in object-oriented methodologies, Larman describes, step by step and building on a single, coherent project as an example, a sensible process for object-oriented analysis and design built around the UML. Besides being an excellent tutorial for learning the UML, this book also describes the GRASP design patterns, which explain how to assign responsibilities to classes. Management, Architects, Developers Style: A very concise, well organized book. Many clear UML diagrams are used to show a logical progression in the analysis and design process. Software professionals are frequently asking the question, "What is the single, best book for learning about object orientation? They want to become effective rapidly. With limited time to learn a new technology - one that forces programmers to make a major paradigm shift in their way of developing software - prospective readers are going for the big win. A good, solid introduction to the full UML notation. An excellent explanation of the object-oriented development process. An introduction to object-oriented software patterns. A single, coherent example application, methodically developed to show the application of the UML notation, object-oriented development process, and software patterns. In addition, the book is concisely written. This stands in contrast to many books on the subject which can lose their readers quickly, either because they are long-winded or wander from the point. Objects by Design has a very successful college course built largely around this text. In fact, we have proposed to the author that he promote his methodology in a number of different ways, with a view toward ultimately establishing a curriculum for certification. It does not, however, prescribe a definitive process or method; it provides a sample of common steps. We believe that object technology as it stands today has under emphasized the key role played by process in the development life-cycle. The UML notation, without a process, is just too confusing for most developers. The Dice Game The best way to jump start a description of a complex process is to provide a capsulized example at the outset. Students, being a generally impatient and skeptical breed, need to be convinced that object-oriented analysis and design is worth the effort. The dice game is ideal in this respect. Use Cases The analysis process commences with the use case. The UML includes a basic use case notation for depicting actors interacting with the system. However, the notation does not specify the format in which the narration of the use case should be captured, nor how it should be used. This is where the author steps in. The purpose of the high-level use case is to rapidly scope out the system, without getting bogged down in detail. This can be a highly effective tool for project planning: Using projections based on previous projects, a skilled manager may be able to produce reasonable time estimates for a new project. Also, the order in which the system is designed may be understood by mapping the dependencies between the full set of use cases. For one, the template makes a clear distinction between the Actor Action and System Response by separating them into two columns. Secondly, by using sequence numbers to order the events initiated by the actor and the system responses, the template provides a clear framework which feeds into subsequent stages in the process: This forms the foundation for the Larman process; subsequent stages always have well defined precedents from which they are derived. What immediately brings the use-case templates to life is their application to the example system which dominates the book - the point-of-sale terminal, or POST. Conceptual Model The conceptual model captures real-world concepts objects , their attributes, and the associations between these concepts. The emphasis throughout is that this is an analysis-level activity - a model of real-world objects - and not an attempt to design the actual software. The example application, the POST, is used to create a full-fledged model and the reader gets a true sense of accomplishment by the end when all the pieces come together in an intelligible diagram. From the process perspective, the author also describes the technique of

identifying nouns in the use cases as a strategy for finding concepts and their attributes. We particularly like this approach because it emphasizes the continuity of the process by deriving the outcome of each stage from previous stages. It also provides students with their first tangible proof of the value of doing good use cases. Ivar Jacobson and Bertrand Meyer. Meyer, in his highly regarded book Object-Oriented Software Construction , has roundly rejected a central role for use cases in object-oriented development. An online excerpt from the book clarifies this rejection. Essentially, he believes that an early emphasis on the sequencing of interactions between actors and the system being modeled shifts the focus away from creating well-abstracted, independent objects. The pitfall of focusing on the sequencing of interactions is inevitably to lock into the design artificial sequencing dependancies between objects which are hard to break later on. Instead, Meyer promotes the Design by Contract approach whereby a contract is formed for each object, precisely specifying the tasks it is obligated to perform on behalf of clients. This focus on abstraction allows interactions between objects to remain fluid; as long as the contracts are honored, the sequence of interactions may be easily modified as need arises. System Sequence Diagrams In order to clearly model the events which will be handled by a system, the system sequence diagram is used. In the Larman process, these are directly derived from the expanded use cases by identifying each of the events generated by the user to the system. Here the benefit of separating the actor actions from the system responses in the use case template using two columns becomes evident - it is much simpler to identify the actor initiated events with this visual partitioning. As we will soon see, identifying these system interactions forms a precise framework for subsequent stages of the process. Larman has switched over to the Bertrand Meyer approach! The emphasis that the author places is on one particular feature of contracts, the post-conditions contracts also include pre-conditions and class invariants. Object creation and deletion. Associations formed and broken. Actually, the UML 1. Larman does not utilize OCL and this could very well be for two reasons. Two, the context for post-conditions expressed by OCL is always a method on a class; at this stage of the process it would be premature to use OCL because methods have not yet been identified. We are still at the system as black-box stage. As Larman states, "A system operation contract describes changes in the state of the overall system when a system operation is invoked. However, what is of great value in the system contracts at this point in the process is their emphasis on object abstraction - the determination of how elements of the conceptual model take form. But note that it is the system sequence diagrams which provide the context for forming the system contracts because it is they that identify operations at the system level. The curtain is raised and we view the scenery for the next act. The system contracts have defined the demarcation between the analysis and design phases. Post-conditions describe all the assertions about the conceptual model which must prevail to satisfy the contracts for each system operation. We know the what, now we need to know the how. How are the post-conditions fulfilled? The answer unfolds with the design of interaction diagrams. The author identifies interaction diagrams as "one of the most important artifacts created in object-oriented analysis and design". The attention devoted to this stage of the process is proof of this conviction. These several chapters are pivotal in terms of defining a process for the UML. First the UML notation for interaction diagrams is introduced. While the treatment is thorough, Larman shows a distinct preference for collaboration diagrams over sequence diagrams. For the author, collaboration diagrams work well to convey the dynamic flow of messages to his reader. Our experience has often shown that, for complex software with many objects and messages, collaboration diagrams can quickly become more unwieldy and less clear than sequence diagrams, which follow a straight-forward top-down, left-right flow. The magic twist that Larman adds is the application of software patterns to the development of the collaboration diagrams. His GRASP patterns are essentially the key guidelines for assigning responsibilities to classes for fulfilling the system contracts. The patterns show the motivation for determining which objects should create other objects, build associations and modify attributes - the assertions contained in the contracts - by sending messages to the other objects. Each is explained in detail with application to the POST example. The details are beyond the scope of this review. Suffice it to say - the process makes a lot of sense and it really works. The proof is in the resulting design of the POST

example. Classes to Code While the production of the collaboration diagrams demands creativity, one gets the impression that the remaining stages of the process are almost mechanical. You know you have succeeded on an object-oriented project when this is in fact true. Class diagrams are easy to derive from the conceptual model and are completed by adding the newly discovered methods from the collaboration diagrams. In actuality these updates can be made concurrently with the discovery of the messages in the collaboration diagram. In fact, it may even be preferrable to create the correct method signature on the class diagram first and then create the message on the collaboration diagram by just picking the name from a pick list. The generation of the code may actually be mechanical if the UML tool provides forward engineering. This feature works for the creation of classes, attributes and method signatures. However, the really fun part is the concluding step of the process. The sequence of these messages translates to a series of statements in the method definition. End of Iteration The book continues with a second iteration which introduces some advanced concepts including inheritance, package diagrams, state diagrams, additional software patterns and persistence frameworks. These advanced concepts are all creatively applied to the POST example application. However, it is the first iteration that really stands out for its coherent presentation of the core aspects of analysis and design. In summary, we highly recommend this book because the author has achieved what few others have achieved before or after him. He has successfully demystified the object-oriented analysis and design process.

DOWNLOAD PDF AN INTRODUCTION TO OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN WITH UML AND THE UNIFIED PROCESS

Chapter 2 : Object Oriented Analysis and Design | GEOG GIS Analysis and Design

*An Introduction to Object-Oriented Systems Analysis and Design With Uml and the Unified Process [Stephen R. Schach] on calendrierdelascience.com *FREE* shipping on qualifying offers. An Introduction to Object-Oriented Systems Analysis and Design With Uml and the Unified Process: Stephen R. Schach: calendrierdelascience.com: Books.*

History of object-oriented methods and notation Before UML 1. The timeline see image shows the highlights of the history of object-oriented modeling methods and notation. It is originally based on the notations of the Booch method , the object-modeling technique OMT and object-oriented software engineering OOSE , which it has integrated into a single language. They were soon assisted in their efforts by Ivar Jacobson , the creator of the object-oriented software engineering OOSE method, who joined them at Rational in During the same month the UML Partners formed a group, designed to define the exact meaning of language constructs, chaired by Cris Kobryn and administered by Ed Eykholt, to finalize the specification and integrate it with other standardization efforts. The result of this work, UML 1. Recent researchers Feinerer, [13] Dullea et al. Hartmann [15] investigates this situation and shows how and why different transformations fail. The Superstructure that defines the notation and semantics for diagrams and their model elements The Infrastructure that defines the core metamodel on which the Superstructure is based The Object Constraint Language OCL for defining rules for model elements The UML Diagram Interchange that defines how UML 2 diagram layouts are exchanged The current versions of these standards are [19]: UML Superstructure version 2. It continues to be updated and improved by the revision task force, who resolve any issues with the language. Modeling[edit] It is important to distinguish between the UML model and the set of diagrams of a system. The set of diagrams need not completely cover the model and deleting a diagram does not change the model. The model may also contain documentation that drives the model elements and diagrams such as written use cases. UML diagrams represent two different views of a system model: It includes class diagrams and composite structure diagrams. Dynamic or behavioral view: This view includes sequence diagrams , activity diagrams and state machine diagrams. Use cases are a way of specifying required usages of a system. Typically, they are used to capture the requirements of a system, that is, what a system is supposed to do.

Chapter 3 : System Analysis and Design Object Oriented Approach

An Introduction to Object-Oriented Systems Analysis and Design with UML and the Unified Process McGraw-Hill, Stephen R. Schach srs@calendrierdelascience.com

Print There was a time when GIS data was simply represented as points, lines, and polygons with attached attributes. As GIS continues to build on an Object Oriented OO technology, advancements in database design have expanded the simple storage of geometry and attributes to include object states, enhanced relationships, validation rules, connectivity requirements, and intricate geometric networking. It is also iterative in nature, offering the flexibility of continuous or successive testing and revision throughout the entire process. This allows the user to test and register feedback as the functionality is explored or added rather than during the final testing phase, as found in many of the older development methodologies. Each phase contains specific tasks or activities that produce pieces of information, termed artifacts, which are created in incremental stages and refined through iterations or development cycles. UML Unified Modeling Language is a graphical notation language, used to visualize and store system objects, actions, and relationships in a sort of software blueprint fashion. The Unified Modeling Language UML is a simple and extensible modeling language that we will discuss in more detail in a later lesson. This discussion is just a brief introduction. UML does not dictate nor require that a particular process is followed, however, the developers of the language do encourage users to follow a use case driven, architecture-centric, iterative and incremental process that is object oriented and component based. It is a modeling language specification. Tools used to implement UML can range from pen and paper through to sophisticated off-the-shelf applications. The UML is not a visual programming language; it is a visual modeling language. It is not simply a notation but a language for capturing knowledge and expressing knowledge about the system under design. UML has become the industry-standard language for system modeling. UML diagrams provide many perspectives of a system during analysis and development. UML defines a set of graphical diagrams that are used for many planning, design and implementation tasks depending on the angle that you are viewing the system. The major views of the system that UML supports are: One or more diagrams for each view is defined by the UML, and each provides a unique window into the system. UML diagrams are to be treated as a set of resources for system modeling. Take care to utilize only those diagrams that provide a useful and beneficial view of your system. Some UML diagrams are more critical than others for system modeling. The general order is: UML diagrams are briefly described below. The user view provides a window into the system from the users perspective, in that the functionality of the system is modeled in terms of the user and what the user expects of the system. In UML-ese, the user of the system is called an actor, which can represent either a human user or users as other systems. The functionality of the system is defined by the use cases. The lines connecting the actors and the use cases show that the actors interact with the functionality provided by the use case. Use cases are usually described in greater detail in Functional Requirements. Business Use Case Diagram. The business use case diagram is an extension to the use case diagram and is defined in and supported by UML. The first step in business modeling using the UML is identifying the interactions between the business processes and those entities outside the business, such as customers and suppliers. The Structural View Class Diagram. Class diagrams describe the static structure of a system. The focus of the class diagram is to describe how a system is structured rather than how it behaves. Class diagrams are probably the most versatile of the UML diagrams. Data models, software components, software classes, and business objects are modeled using the class diagram, each in its own diagram. Sequence diagrams describe the behavior of a use case by diagramming the classes and the messages exchanged between them arranged in chronological order. Sequence diagrams do not describe object relationships; that view of the system is reserved for collaboration diagrams. Object and actor instances can be displayed in the sequence diagram along with how the objects communicate by sending messages to one another. The collaboration diagram is a view of the interactions of objects, and, unlike the sequence diagram that describes

the objects messaging over time, collaboration diagrams display the relationships between objects. Some UML tools can automatically generate collaboration diagrams from sequence diagrams, and vice versa. Collaboration and sequence diagrams express similar information, but they display it from different perspectives. The activity diagram is a specialization of the state diagram and it displays the dynamics of a system by showing the work flow. The activity diagram complements the business use case diagram in that it shows the process flow behind the use case. The state diagram describes the sequence of states that an object goes through during its lifetime. The state diagram displays the events that act upon the object that enables the transition from one state to the next. The Implementation View Component Diagram. The component diagram displays the static structure of the implementation view. The component diagram shows the organizations and dependencies of the components, subsystems, and their relationships. The diagram models the interface that can be used to show the externally visible operations of a class or component. The deployment diagram shows the configuration of run-time processing elements and the software components, processes and objects that execute on them. The deployment diagram can be used for business modeling where employees and organizations are displayed as run-time processing elements and the procedures and documents that they use are displayed as the software components.

Chapter 4 : Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design

Object-oriented analysis and design can offer an approach that facilitates logical, rapid, and thorough methods for creating new systems responsive to a changing business landscape. Object-oriented techniques work well in situations in which complicated information systems are undergoing continuous maintenance, adaptation, and redesign.

History[edit] In the early days of object-oriented technology before the mids, there were many different competing methodologies for software development and object-oriented modeling , often tied to specific Computer Aided Software Engineering CASE tool vendors. No standard notations, consistent terms and process guides were the major concerns at the time, which degraded communication efficiency and lengthened learning curves. Later, together with Philippe Kruchten and Walker Royce eldest son of Winston Royce , they have led a successful mission to merge their own methodologies, OMT , OOSE and Booch method , with various insights and experiences from other industry leaders into the Rational Unified Process RUP , a comprehensive iterative and incremental process guide and framework for learning industry best practices of software development and project management. The specific problem is: January Learn how and when to remove this template message The software life cycle is typically divided up into stages going from abstract descriptions of the problem to designs then to code and testing and finally to deployment. The earliest stages of this process are analysis and design. The analysis phase is also often called "requirements acquisition". In some approaches to software developmentâ€”known collectively as waterfall modelsâ€”the boundaries between each stage are meant to be fairly rigid and sequential. The term "waterfall" was coined for such methodologies to signify that progress went sequentially in one direction only, i. The alternative to waterfall models are iterative models. This distinction was popularized by Barry Boehm in a very influential paper on his Spiral Model for iterative software development. With iterative models it is possible to do work in various stages of the model in parallel. So for example it is possibleâ€”and not seen as a source of errorâ€”to work on analysis, design, and even code all on the same day and to have issues from one stage impact issues from another. As a result, in object-oriented processes "analysis and design" are often considered at the same time. The object-oriented paradigm emphasizes modularity and re-usability. The goal of an object-oriented approach is to satisfy the "open closed principle". A module is open if it supports extension. If the module provides standardized ways to add new behaviors or describe new states. In the object-oriented paradigm this is often accomplished by creating a new subclass of an existing class. A module is closed if it has a well defined stable interface that all other modules must use and that limits the interaction and potential errors that can be introduced into one module by changes in another. In the object-oriented paradigm this is accomplished by defining methods that invoke services on objects. Methods can be either public or private, i. This reduces a source of many common errors in computer programming. The distinction between analysis and design is often described as "what vs. In analysis developers work with users and domain experts to define what the system is supposed to do. Implementation details are supposed to be mostly or totally depending on the particular method ignored at this phase. The goal of the analysis phase is to create a functional model of the system regardless of constraints such as appropriate technology. In object-oriented analysis this is typically done via use cases and abstract definitions of the most important objects. The subsequent design phase refines the analysis model and makes the needed technology and other implementation choices. In object-oriented design the emphasis is on describing the various objects, their data, behavior, and interactions. The design model should have all the details required so that programmers can implement the design in code. The main difference between object-oriented analysis and other forms of analysis is that by the object-oriented approach we organize requirements around objects, which integrate both behaviors processes and states data modeled after real world objects that the system interacts with. In other or traditional analysis methodologies, the two aspects: For example, data may be modeled by ER diagrams , and behaviors by flow charts or structure charts. The primary tasks in object-oriented analysis OOA are: Find the objects Describe how the objects interact

Define the behavior of the objects Define the internals of the objects Common models used in OOA are use cases and object models. Use cases describe scenarios for standard domain functions that the system must accomplish. Object models describe the names, class relations e. Circle is a subclass of Shape , operations, and properties of the main objects. User-interface mockups or prototypes can also be created to help understanding. Object-oriented design During object-oriented design OOD , a developer applies implementation constraints to the conceptual model produced in object-oriented analysis. Such constraints could include the hardware and software platforms, the performance requirements, persistent storage and transaction, usability of the system, and limitations imposed by budgets and time. Concepts in the analysis model which is technology independent, are mapped onto implementing classes and interfaces resulting in a model of the solution domain, i. Object-oriented modeling[edit] Object-oriented modeling OOM is a common approach to modeling applications, systems, and business domains by using the object-oriented paradigm throughout the entire development life cycles. Object-oriented modeling typically divides into two aspects of work: Efficient and effective communication Users typically have difficulties in understanding comprehensive documents and programming language codes well. Visual model diagrams can be more understandable and can allow users and stakeholders to give developers feedback on the appropriate requirements and structure of the system. A key goal of the object-oriented approach is to decrease the "semantic gap" between the system and the real world, and to have the system be constructed using terminology that is almost the same as the stakeholders use in everyday business. Object-oriented modeling is an essential tool to facilitate this. Useful and stable abstraction Modeling helps coding. A goal of most modern software methodologies is to first address "what" questions and then address "how" questions, i. Object-oriented modeling enables this by producing abstract and accessible descriptions of both system requirements and designs, i.

DOWNLOAD PDF AN INTRODUCTION TO OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN WITH UML AND THE UNIFIED PROCESS

Chapter 5 : Object-oriented analysis and design - Wikipedia

An Introduction to Object-Oriented Systems Analysis and Design with UML and the Unified Process has 5 ratings and 1 review. Ali said: it was the first bo.

Object-oriented techniques work well in situations in which complicated information systems are undergoing continuous maintenance, adaptation, and redesign. In this chapter, we introduce the unified modeling language UML , the industry standard for modeling object-oriented systems. The UML toolset includes diagrams that allow you to visualize the construction of an object-oriented system. Each design iteration takes a successively more detailed look at the design of the system until the things and relationships in the system are clearly and precisely defined in UML documents. UML is a powerful tool that can greatly improve the quality of your systems analysis and design, and thereby help create higher-quality information systems. When the object-oriented approach was first introduced, advocates cited reusability of the objects as the main benefit of their approach. It makes intuitive sense that the recycling of program parts should reduce the costs of development in computer-based systems. It has proved to be very effective in the development of GUIs and databases. Although reusability is the main goal, maintaining systems is also very important, and because the object-oriented approach creates objects that contain both data and program code, a change in one object has a minimal impact on other objects. Chapter Summary Object-oriented systems describe entities as objects. Objects are part of a general concept called classes, the main unit of analysis in object-oriented analysis and design. Although reusability is the main goal, maintaining systems is also very important. Analysts can use CRC cards to begin the process of object modeling in an informal way. Object Think can be added to the CRC cards to assist the analyst in refining responsibilities into smaller and smaller tasks. CRC sessions can be held with a group of analysts to determine classes and responsibilities interactively. Unified modeling language UML provides a standardized set of tools to document the analysis and design of a software system. UML is fundamentally based on an object-oriented technique known as use case modeling. A use case model describes what a system does without describing how the system does it. A use case model partitions system functionality into behaviors called use cases that are significant to the users of the system called actors. Different scenarios are created for each different set of conditions of a use case. The main components of UML are things, relationships, and diagrams. Diagrams are related to one another. Structural things are most common; they include classes, interfaces, use cases, and many other elements that provide a way to create models. Structural things allow the user to describe relationships. Behavioral things describe how things work. Group things are used to define boundaries. Annotational things permit the analyst to add notes to the diagrams. Relationships are the glue that holds the things together. Structural relationships are used to tie the things together in structural diagrams. Structural relationships include dependencies, aggregations, associations, and generalizations. Behavioral diagrams use the four basic types of behavioral relationships: They include use case diagrams, activity diagrams, sequence diagrams, communication diagrams, class diagrams, and statechart diagrams. In addition to the diagrams, analysts can describe a use case using a use case scenario. By using UML iteratively in analysis and design, you can achieve a greater understanding between the business team and the IT team regarding the system requirements and the processes that need to occur in the system to meet those requirements. Understand what object-oriented systems analysis and design is and appreciate its usefulness. Comprehend the concepts of unified modeling language UML , the standard approach for modeling a system in the object-oriented world. Apply the steps used in UML to break down the system into a use case model and then a class model. Diagram systems with the UML toolset so they can be described and properly designed. Document and communicate the newly modeled object-oriented system to users and other analysts.

Chapter 6 : Object-Oriented Systems Analysis and Design Using UML Archives |

Unified modeling language (UML) provides a standardized set of tools to document the analysis and design of a software system. UML is fundamentally based on an object-oriented technique known as use case modeling.

Next Page In the object-oriented approach, the focus is on capturing the structure and behavior of information systems into small modules that combines both data and process. The main aim of Object Oriented Design OOD is to improve the quality and productivity of system analysis and design by making it more usable. In analysis phase, OO models are used to fill the gap between problem and solution. It performs well in situation where systems are undergoing continuous design, adaption, and maintenance. It identifies the objects in problem domain, classifying them in terms of data and behavior. It promotes the reuse of components. It simplifies the problem of integrating components to configure large system. It simplifies the design of distributed systems. All tangible entities student, patient and some intangible entities bank account are modeled as object. It defines the operation performed on objects. Objects with similar meaning and purpose grouped together as class. They are nothing more than an action that an object can perform. They are information sent to objects to trigger methods. Essentially, a message is a function or procedure call from one object to another. Features of Object-Oriented System An object-oriented system comes with several great features which are discussed below. Encapsulation Encapsulation is a process of information hiding. It is simply the combination of process and data into a single entity. Data of an object is hidden from the rest of the system and available only through the services of the class. It allows improvement or modification of methods used by objects without affecting other parts of a system. Abstraction It is a process of taking or selecting necessary method and attributes to specify the object. It focuses on essential characteristics of an object relative to perspective of user. Relationships All the classes in the system are related with each other. The objects do not exist in isolation, they exist in relationship with other objects. It indicates that two classes are similar but have some differences. Polymorphism and Dynamic Binding Polymorphism is the ability to take on many different forms. It applies to both objects and operations. A polymorphic object is one who true type hides within a super or parent class. In polymorphic operation, the operation may be carried out differently by different classes of objects. It allows us to manipulate objects of different classes by knowing only their common properties. It works with Bottom-up approach. Program is divided into number of submodules or functions. Program is organized by having number of classes and objects. Function call is used. Software reuse is not possible. Structured design programming usually left until end phases. Object oriented design programming done concurrently with other phases. Structured Design is more suitable for offshoring. It is suitable for in-house development. It shows clear transition from design to implementation. Not so clear transition from design to implementation. It is suitable for real time system, embedded system and projects where objects are not the most useful level of abstraction. It is suitable for most business applications, game development projects, which are expected to customize or extended. Class diagram, sequence diagram, state chart diagram, and use cases all contribute. In this, projects can be managed easily due to clearly identifiable phases. In this approach, projects can be difficult to manage due to uncertain transitions between phase. Unified Modeling Language UML UML is a visual language that lets you to model processes, software, and systems to express the design of system architecture. It is a standard language for designing and documenting a system in an object oriented manner that allow technical architects to communicate with developer. It is defined as set of specifications created and distributed by Object Management Group. UML is extensible and scalable. The objective of UML is to provide a common vocabulary of object-oriented terms and diagramming techniques that is rich enough to model any systems development project from analysis through implementation. For example, adding a new employee. For example, finding address of a particular employee.

Chapter 7 : Object-Oriented Systems Analysis and Design Using UML - Systems Analysis

Models are used during the analysis process to help to elicit the requirements, during the design process to describe the system to engineers, and after implementation to document the system.

Next Page UML is a standard language for specifying, visualizing, constructing, and documenting the artifacts of software systems. OMG is continuously making efforts to create a truly industry standard. UML is a pictorial language used to make software blueprints. UML can be described as a general purpose visual modeling language to visualize, specify, construct, and document software system. Although UML is generally used to model software systems, it is not limited within this boundary. It is also used to model non-software systems as well. For example, the process flow in a manufacturing unit, etc. UML is not a programming language but tools can be used to generate code in various languages using UML diagrams. UML has a direct relation with object oriented analysis and design. Object-oriented concepts were introduced much earlier than UML. At that point of time, there were no standard methodologies to organize and consolidate the object-oriented development. It was then that UML came into picture. There are a number of goals for developing UML but the most important is to define some general purpose modeling language, which all modelers can use and it also needs to be made simple to understand and use. UML diagrams are not only made for developers but also for business users, common people, and anybody interested to understand the system. The system can be a software or non-software system. Thus it must be clear that UML is not a development method rather it accompanies with processes to make it a successful system. A conceptual model can be defined as a model which is made of concepts and their relationships. A conceptual model is the first step before drawing a UML diagram. It helps to understand the entities in the real world and how they interact with each other. As UML describes the real-time systems, it is very important to make a conceptual model and then proceed gradually. An object contains both data and methods that control the data. The data represents the state of the object. A class describes an object and they also form a hierarchy to model the real-world system. The hierarchy is represented as inheritance and the classes can also be associated in different ways as per the requirement. Objects are the real-world entities that exist around us and the basic concepts such as abstraction, encapsulation, inheritance, and polymorphism all can be represented using UML. UML is powerful enough to represent all the concepts that exist in object-oriented analysis and design. UML diagrams are representation of object-oriented concepts only. OO Analysis and Design OO can be defined as an investigation and to be more specific, it is the investigation of objects. Design means collaboration of identified objects. Thus, it is important to understand the OO analysis and design concepts. The most important purpose of OO analysis is to identify objects of a system to be designed. This analysis is also done for an existing system. Now an efficient analysis is only possible when we are able to start thinking in a way where objects can be identified. After identifying the objects, their relationships are identified and finally the design is produced. Making a design, which can be converted to executables using OO languages. There are three basic steps where the OO concepts are applied and implemented. If these objects are identified efficiently, then the next job of design is easy. The objects should be identified with responsibilities. Responsibilities are the functions performed by the object. Each and every object has some type of responsibilities to be performed. When these responsibilities are collaborated, the purpose of the system is fulfilled. The second phase is OO design. During this phase, emphasis is placed on the requirements and their fulfilment. In this stage, the objects are collaborated according to their intended association. After the association is complete, the design is also complete. The third phase is OO implementation. Most of the UML diagrams discussed so far are used to model different aspects such as static, dynamic, etc. Now whatever be the aspect, the artifacts are nothing but objects. If we look into class diagram, object diagram, collaboration diagram, interaction diagrams all would basically be designed based on the objects. Once the OO analysis and design is done, the next step is very easy.

DOWNLOAD PDF AN INTRODUCTION TO OBJECT-ORIENTED SYSTEMS ANALYSIS AND DESIGN WITH UML AND THE UNIFIED PROCESS

Chapter 8 : Systems Analysis and Design with UML, 4th Edition [Book]

3 UML Applied - Object Oriented Analysis and Design using the UML Contents AN INTRODUCTION TO THE UML 7 What is the UML? 7 A Common Language 7 Summary 9 THE UML WITHIN A DEVELOPMENT PROCESS

Chapter 9 : Unified Modeling Language - Wikipedia

The process of object-oriented design is really just an extension of the object-oriented analysis process that preceded it, except with a critical caveat: the consideration and implementation of constraints.