CHAPTER ONE
INTRODUCTION

1.1 Background to the Study

At the core of every mature discipline, from the arts to the sciences and engineering is a common language and common approaches that enable practitioners to collaborate and the discipline to evolve (Alhir, 2010). Software engineering discipline is not left out. Modeling software visually is one of the six best practices that has become a paradigm in software engineering. A model is a complete description of a system from a particular perspective. For Visual Modeling, especially with the current Object oriented approach, Unified Modeling Language (UML) is one tool that can be used to make this task more feasible.

The early part of the 1990s saw a greatly heightened interest in the object paradigm and related technologies. New object-based programming languages, such as SmallTalk, Eiffel, C++, and Java, were devised and adopted. These were accompanied by a prodigious and confusing glut of object-oriented (OO) software design methods and modeling notations. Thus, in his very thorough overview of OO analysis and design methods (covering more than 800 pages), Graham (2000) lists more than 50 “seminal” methods. Given that the object paradigm consists of relatively few fundamental concepts, including encapsulation, inheritance, and polymorphism, there was clearly heavy overlap and conceptual alignment across these methods—much of which was obscured by notational and other differences of no consequence. This caused great confusion and needless market fragmentation, which, in turn, impeded the adoption of the useful new paradigm. Software developers had to make difficult and binding choices between mutually incompatible languages, tools, methods, and vendors (Selic, 2005).

For this reason, when Rational Software proposed the Unified Modeling Language (UML) initiative, led by Grady Booch, Ivar Jacobson, and Jim Rumbaugh, the reaction was immediate and positive. Rational did not intend to propose anything new, but—through collaboration among top industry thought leaders—consolidated the best features of the various OO approaches into one vendor-independent modeling language and notation (Selic, 2005). In the mid-1990s the Object Management Group (OMG) acted as forum for agreement between the thought-leaders in the nascent software modeling field. The time was exactly right for the emergence of a standard. Researchers and early-adopters had accumulated a great deal of modeling experience, but were being held back by the lack of a widely-used notation. UML quickly became the first de facto standard and, following its Object Management Group adoption in 1996, as a bona-fide industry standard (Object Management Group, 2003; Object Management Group, 2004; Rumbaugh, Jacobson, & Booch, 2005). Once UML eliminated this major obstacle to widespread use of visual modeling, its use grew spectacularly.

Watson (2010) observed that the history of visual modeling in the software industry divides cleanly into two eras - "Before UML" and "After UML". He noted that before the first Unified Modeling Language (UML) standards were published in the mid-1990s, visual software modeling was plagued by the incompatibility of different notations created by different modeling gurus. The absence of a standardized notation deterred potential users, and as an inevitable result the modeling tools market was tiny and fragmented. The few tools that were available suffered from a lack of investment; many only allowed sketching of software designs, lacking facilities for checking the diagrams' internal consistency or automatically processing the information they held. These early visual diagrams were useful as design aids or documentation, but were rarely integrated into the software development lifecycle.

The UML standard changed all that, and triggered the dramatic growth in visual modeling that has led to its widespread use not only in software design, but also in non-software disciplines such as systems engineering, and in the business domain. As UML use has grown, continuous feedback from the user community and investment by tool vendors has helped the standard evolve and mature. The original UML 1 standard of 1997 was backed by twenty one OMG member companies; feedback from dozens more submitted via OMG's issue-reporting system helped refine it, flushing out remaining inconsistencies. In 2005 OMG published UML 2, a major revision largely based on the same familiar diagram notations, but using a more rigorous underlying modeling infrastructure specified using OMG's Meta-Object Framework (MOF). While some designers still use UML merely for sketching designs to share with colleagues, UML 2's MOF foundation means that today's UML diagram is more than just a pretty picture. A MOF-aware modeling tool can capture the meaning of diagram elements and their relationships in machine-readable form, and use this to reason about the design, perform consistency checks, and even automatically generate parts of the application code. Creating, storing and transforming machine-readable models in this way puts modeling at the heart of the software production process, and forms the basis of OMG's Model Driven Architecture (MDA).

Selic (2005) noted that the language has become an essential part of the Computer Science and Engineering curricula in universities throughout the world and in various professional training programs stressing that academic and other researchers use it as a convenient lingua franca. Watson (2010) also noted that UML has become the lingua franca of software development, allowing engineers to exchange their designs freely. He observed that nowhere is this better illustrated than in the software for the new James Webb space telescope, scheduled for launch in 2013. To aid communication and help meet stringent reliability and performance goals, all the software being built for the telescope by NASA, the Canadian and European space agencies and all their subcontractors is being designed using UML. Organizations across the world are cooperating on guidance software, a command data handling system and software for the science module housing four different light-receiving instruments. All will be integrated in the telescope itself, destined for earth orbit at an altitude of 940,000 miles (four times the distance to the moon).

UML is supported by every major commercial IT vendor, as well as a flourishing selection of Open Source tools. UML books & training are widely available, and the OMG Certified UML Professional (OCUP) and OMG-Certified Real-time and Embedded Specialist (OCRES) certification programmes have allowed tens of thousands of engineers and architects to establish their UML credentials (Watson, 2010). UML has changed the software world.

It is also interesting to note that as at 2004 the British Computer Society Professional Examination on Object Oriented Programming using their New Syllabus has questions that require a good knowledge of UML. Question one to three of the six questions asked show that it was purely based on modeling object oriented problems using UML and parts of the remaining questions can still be answered well if the student have a good knowledge of UML. So anybody who wants to sit for the Object Oriented Programming examination without a good knowledge of the Unified Modeling Language is bound to fail the examination (British Computer Society, 2004).

The Unified Software Development Process or Unified Process (UP) for short is an emerging popular software development process framework. It divides project life cycle into four phases: Inception, Elaboration, Construction and Transition. (Alhir, 2010).

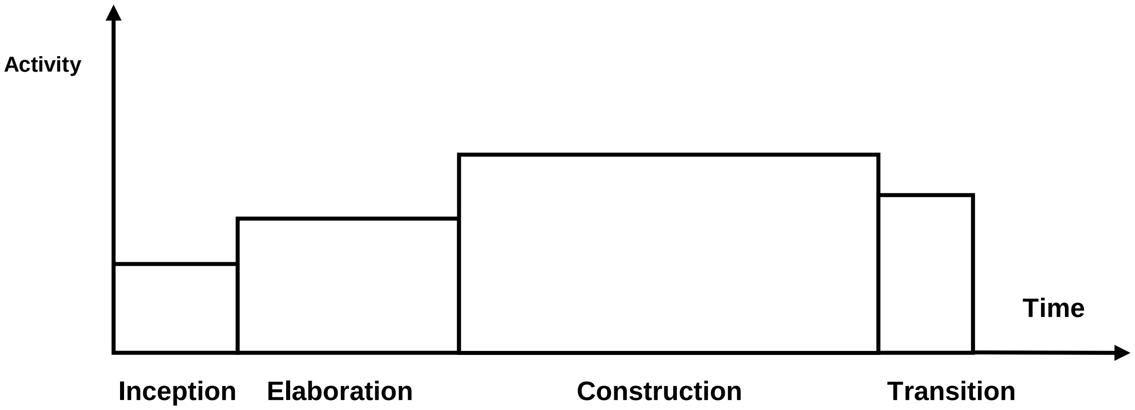


Figure 1.1 Profile of project showing the relative sizes of UP phases Source (Alhir, 2010).

They identified the major characteristic of the Unified Process as:

1. **Iterative and Incremental:** The unified process is an iterative and incremental development process. The Elaboration, Construction and Transition phases are divided into a series of time boxed iterations. (The Inception phase may also be divided into iterations for a large project.) Each iteration results in an *increment*, which is a release of the system that contains added or improved functionality compared with the previous release.
2. **Use Case Driven:** In the Unified Process, use cases (which can be modeled using UML use case diagrams) are used to capture the functional requirements and to define the contents of the iterations after which each iteration takes a set of use cases or scenarios from requirements all the way through implementation, test and deployment.
3. **Architecture Centric:** The Unified Process insists that architecture sit at the heart of the project team's efforts to shape the system and the UML is the major tool used for this purpose.
4. **Risk Focused:** The Unified Process requires the project team to focus on addressing the most critical risks early in the project life cycle. The deliverables of each iteration, especially in the Elaboration phase, must be selected in order to ensure that the greatest risks are addressed first.

The Unified Process is simply a software development process, that is based on the enlargement and refinement of a system through multiple iterations, with cyclic feedback and adaptation (Osis & Donins, 2017). The system is developed incrementally over time, iteration by iteration. The UP is broadly applicable to different types of software systems, including small-scale and large-scale projects having various degrees of managerial and technical complexity, across different application domains and organizational cultures (Alhir, 2010). From Hastie (2010) discussion on the various flavors of the Unified Process, the various refinements and variations of the Unified Process that are either a simplified subsets or expanded superset of the Unified Process are: Rational Unified Process, Agile Unified Process, Essential Unified Process (EssUP), OpenUP/Basic, The Unified Process for Education (UPEDU), Enterprise Unified Process, IBM Tivoli Unified Process (ITUP) and Oracle Unified Method.

As we have seen, the Unified Process employs the use of Unified modeling Language (UML) extensively in almost all phases of its software development process.

Here in Nigeria, it has not yet been clear the extent this Unified Modeling Language is being adopted by IT professionals for software modeling. Has it become an essential part of the computer science and engineering curricula as noted by Selic (2005)? Do we have over 70% of software development organizations using it as Recker (2008) observed. It is against this background that this researcher work of creating a model for evaluating the adaptation of The Unified Modeling Language and the Unified Process by Software Developers and indeed all IT professionals in Nigeria was conceived.

1.2 Statement of the Problem

Modeling software visually is one of the six best practices for software development. Unified Modeling Language (UML) has become a standard in software modeling. UML has also become an essential part of the Computer Science and Engineering curricula in universities throughout the world and in various professional training programs. In fact, academics and other researchers use it as a convenient lingua franca (Selic, 2005). This has also been proved by many of the literatures reviewed and its inclusion in object oriented professional examination questions.

The question is, how far is UML being adopted by IT students and professionals in software development. Till the time of this research, not much work has been done on UML Adoption.

A lot of studies have been carried out on different aspects of UML but very few work has been done on UML adoption and use. Budgen, Burn, Brereton, Kitchenham and Pretorius (2011) in their study on empirical evidence of the UML concluded that while there are many studies on different aspects of UML, there are relatively few for which the UML itself is the object of study.

Cabot (2013) in his paper, UML Adoption in practice: has anything changed in the last decade? noted that the results of ICSE 2013 paper on UML in practice, is not different from those reported in 2006 by Dobing and Parson on How UML is Used. This should not be the case and shows that more study is required to be conducted on UML adoption and to understand factors that may be slowing down its adoption.

1.3 Aim and Objectives of the Study

Aim: The aim of this work is to create an Evaluation Model that evaluates the level of adaptation to UML by software developers through their knowledge of UML content.

Objectives: The objectives of the study include:

1. Conduct a survey to capture UML diagrams usage in the industry and academy by IT professionals.
2. Model questions that will adequately evaluate software developer's level of knowledge and adaptation to the use of UML in software modeling based on the evaluation criteria determined from the survey.
3. Create developers knowledge evaluation model that plots the graph of UML content against the level of Knowledge.

1.4 Significance of the Study

Cernosek and Naiburg (2004) noted that software industry has adopted the Unified Modeling Language as its standard means for representing software models and related artifacts. Software architects, designers and developers use UML for specifying, visualizing, constructing and documenting all aspects of a software system and UML is supported by every major commercial IT vendor, as well as a flourishing selection of Open Source tools. Also, the OMG Certified UML Professional (OCUPTM) and OMG-Certified Real-time and Embedded Specialist (OCRESTM) certification programmes have allowed tens of thousands of engineers and architects to establish their UML credentials. UML has changed the software world.

Moreover, UML usage is an integral part of Object oriented Programming as a look at the British Computer Society professional examination on Object Oriented Programming (Version 2: New Syllabus) taken on 21st April 2004 between 2.30p.m and 4.30p.m reveals. The students were instructed to answer FOUR questions out of SIX. Of the six questions, three and half were fully UML based which means that anybody without a full knowledge of UML will not pass that examination. As UML use has grown, continuous feedback from the user community and investment by tool vendors has helped the standard evolve and mature.

It becomes therefore very necessary that we know the Nigerian situation. The major significance of the study is fivefold:

1. The evaluation model will help individual cooperate organizations in evaluating their knowledge of UML.
2. To ensure that software developers in Nigeria are part of this evolution in software modeling by adopting the industry standard paradigm.
3. To preserve our legacy software by promoting software re-engineering
4. Those willing to learn UML will know the modelling notations mostly used.
5. The findings will be used to provide more feedback from user community to stakeholders.
6. The findings will be valuable for policy making such as integrating the teaching of UML in Computer Science curriculum since it has become a standard to better equip our Computer Science and Computer Engineering tomorrow professionals.

1.5 Scope of the Study

This research will focus more on Unified modeling Language which has become a standard and the emphasis will be on commonly used UML Diagrams. It will also look briefly at the Unified process which is a software development process that works well with UML.

1.6 Motivation for the Study

From literature, it is evident that while there are many studies on different aspects of the UML, there are relatively few for which the UML itself is the object of study (Budgen, Burn, Brereton, Kitchenham and Pretorius, 2011). Also, as Cabot (2013) observed, not much has changed on UML Adoption in practice as the results of ICSE 2013 paper on UML in practice, is not different from those reported in 2006 by Dobing and Parson on How UML is Used. This should not be the case and shows that more study is required to be conducted on UML adoption and to understand factors that may be slowing down its adoption. The motivation for this study is to contribute towards filling this gap.

1.7 Definition of Terms

1. Semantic: The study of meaning in a Language. The Language this time is Unified Modeling Language.

2. Modeling: Modeling is an act of making models. A model serves as an abstraction—an approximate representation of the real item that is being built. Developers need a better understanding of what they are building, and modeling offers an effective way to do that.

3. Unified Process: *Unified Process* is a popular iterative and incremental software development process framework.

4. Unified Modeling Language (UML): UML is a software modeling language that has become a de facto standard.

5. Standard: Standard in relation to computers is a set of detailed technical guidelines used as a means of establishing uniformity in an area of hardware or software development.

6. Object Management Group (OMG): is a consortium, originally aimed at setting standards for distributed object-oriented systems, and is now focused on modeling (programs, systems and business processes) and model-based standards. Founded in 1989 by eleven leading computer companies, today, over 800 companies from both the computer industry and software-using companies from other industries are members of OMG. Since 2000 the OMG's International Headquarters are located in Needham, Massachusetts. Their website is www.omg.org.

CHAPTER TWO
LITERATURE REVIEW

2.1 The Value of Modeling

Modeling is the designing of software applications before coding (Object Management Group, 2009). For many years, business analysts, engineers, scientists and other professionals who build complex structures or systems have been creating models of what they build (Cernosek & Naiburg, 2004). Sometimes, the models are physical, such as scaled mock-ups of airplanes, houses or automobiles. Sometimes the models are less tangible, as seen in business financials models, market trading simulations and electrical circuit diagrams. In all cases, a model serves as an abstraction—an approximate representation of the real item that is being built.

Cernosek and Naiburg (2004) also noted that it is neither technically wise nor economically practical to build certain kinds of complex systems without first creating a design, a blueprint or another abstract representation and that while professional architects might build a dog house without a design diagram, they would never construct a 15-story office building without first developing an array of architectural plans, diagrams and some type of a mock-up for visualization. According to Coleman, Liebovitch and Fisher (2019) modeling help us to determine results of interaction between all interacting factors. This means that Modeling provides architects and others with the ability to visualize entire systems, assess different options and communicate designs more clearly before taking on the risks—technical, financial or otherwise—of actual construction.

According to Selic (2005), UML has helped raise general awareness about the value of modeling when dealing with software complexity. Before then the practice of software development was exempt from many of these modeling issues (Cernosek & Naiburg, 2004). They noted that by its very nature, software can be easily created and easily changed. Little capital equipment is required, and virtually no manufacturing costs are incurred. These attributes cultivated a do-it-yourself culture— imagine it, build it and change it as often as necessary. There is no “final” system anyway, so why even try to conceive of one before writing code?

Today however, software systems have become very complex. They must be integrated with other systems to run the items used in everyday life. Automobiles, for example, are now heavily equipped with computers and associated software to control everything from the engine and cruise control to all kinds of new on board navigation and communication systems. Software also is heavily used to automate business processes of all kinds, including those that are seen and experienced by customers and those that are in the back office. Some software systems support important health-related or property-related functions, making them necessarily complex to develop, test and maintain. And even those systems that are not critical to human health or property can be critical to businesses. Brian and John (2018) noted that developing and assuring safety and security critical real-time embedded systems is a challenging endeavour that requires many activities applied at multiple levels of abstraction and for such activities to be effective, they must be grounded in industry standard architecture. In many organizations, software development is no longer a cost-center overhead line item. It is an integral part of the company's strategic business processes. For those companies, software has become a key discriminator in competing in the marketplace. For these reasons and more, developers need a better understanding of what they are building, and modeling offers an effective way to do that. At the same time, modeling must not slow things down. Customers and business users still expect software to be delivered on time and to perform as expected on demand (Cernosek & Naiburg, 2004).

To achieve this "fast and good" goal, IBM sees four imperatives for software development: develop iteratively, focus on architecture, continuously ensure quality and manage change and assets. The same basic reasons why other complex, high-risk systems are modeled also apply to software—to manage the complexity and to understand the design and associated risks. More specifically, by modeling software, developers can: Create and communicate software designs before committing additional resources, Trace the design back to the requirements, helping to ensure that they are building the right system, Practice iterative development, in which models and other higher levels of abstraction facilitate quick and frequent changes (Cernosek & Naiburg, 2004).

Despite the many reasons and virtues behind modeling, a great majority of software developers still do not employ any form of abstraction higher than that of source code. Why? As described earlier, sometimes the actual complexity of the problem or solution does not warrant it. Again, if you are building a doghouse, you do not need to hire an architect or contract a builder to produce a set of design specifications. But in the world of software, systems often begin simple and well-understood and then—through the natural evolution of a successful implementation—become more and more complex. In other cases, developers choose not to model because they simply do not perceive a need for it until much too late.

Traditional programmers are very proficient at conventional coding techniques. Even when unexpected complexity begins to encroach, most developers are comfortable sticking to their integrated development environment (IDE) and debugger and simply working more hours on the problem. Because modeling requires additional training and tools, a corresponding investment in time, money and effort is needed—not at the time of toil, but early in a project's development life cycle. The reason traditional developers are not more proactive in this regard is that they believe modeling will slow them down.

UML has helped raise general awareness about the value of modeling when dealing with software complexity. Although this highly useful technique is almost as old as software itself (with flowcharts and finite state machines as early examples), most practitioners have generally been slow to accept it as anything more than a minor power assist. It is fair to say that this is still the dominant attitude, which is why so-called "model-driven" methods are encountering great resistance in this community (Selic, 2005). There are valid reasons for this situation:

1. The main one is that software models can often be inaccurate in unpredictable ways. Clearly, any model's practical value is directly proportional to its accuracy. If we cannot trust the model to tell us true things about the software system it represents, then the model is worse than useless—it can foster false conclusions. The key to increasing a software model's value then is to narrow the gap between it and the system it is modeling. Paradoxically, as we shall discuss later, this is easier to do in software than in any other engineering discipline.

You can blame some of this model inaccuracy on the extremely detailed and sensitive nature of current programming language technologies. Minor lapses and barely detectable coding errors, such as misaligned pointers or uninitialized variables, can have enormous consequences. For instance, a well-documented case noted that one missing break in one case of a nesting switch statement resulted in the loss of long-distance telephone service for a large part of the United States, causing immense economic losses (Lee, 1992). If such seemingly minute detail can have such dire consequences, how can we trust models to be accurate, since models, by definition, are supposed to hide or remove detail?

The solution to this conundrum is to formally link a model to its corresponding software implementation through one or more automated model transformations. Perhaps the best and most successful exemplar of that can be found in the concept of a compiler, which translates a high-level language program into an equivalent machine language implementation. Like all useful models, the model—in this case, a high-level language program—hides irrelevant detail, such as the idiosyncrasies of the underlying computing technology (internal word size, the number of accumulators and index registers, the type of ALU, etc.

Note that few, if any, other engineering media can provide such a tight coupling between a model and its corresponding engineering artifact. This is because the modeled artifact is software rather than hardware. A model of any kind of physical artifact (automobile, building, bridge, etc.) inevitably involves an informal step of abstracting the physical characteristics into a corresponding formal model, such as a mathematical or scale model. Similarly, implementing an abstract model using physical materials involves an informal transformation from the abstract into the concrete. The informal nature of this step can lead to inaccuracies that, as noted above, can render the models ineffective or even counterproductive. In software, however, this transformation can, in principle, be performed formally in either direction.

The potential behind this powerful combination of abstraction and automation has led to the emergence of new modeling technologies and corresponding development methods, collectively referred to as model-driven development (MDD) (Brown, 2004 & Booch, 2004). MDD’s defining feature is that models have become primary artifacts of software design, shifting the focus away from the corresponding program code. Models serve as blueprints from which programs and related models are derived by various automated and semi-automated processes. MDD’s degrees of automation today vary from simple skeleton code derivation to complete automatic code generation (which is comparable to traditional compilation). Clearly, the greater the levels of automation, the more accurate the models and greater the MDD benefits become.

Model-driven methods are not particularly new and have been used in software development with varying degrees of success. They are receiving much more attention today because the supporting technologies have matured to the point where you can automate much more than you could in the past. This is not just in terms of efficiency but also in terms of scalability, and the ability of such tools to be integrated with legacy tools and methods. The emergence of MDD standards that result in the commoditization of corresponding tools plus the obvious benefits to users reflect this maturation. One of these MDD standards is the Unified Modeling Language version 2.0.

Modeling complex applications has several general benefits. Some specific situations in which the modeling effort is worthwhile include: To better understand the business or engineering situation at hand ("as-is" model) and to craft a better system ("to-be" model), to build and design system architecture and create visualizations of code and other forms of implementation

2.2 Modeling before UML

Watson (2010) noted that the history of visual modeling in the software industry divides cleanly into two eras - "Before UML" and "After UML". Before the first Unified Modeling Language (UML) standards were published in the mid-1990s, visual software modeling was plagued by the incompatibility of different notations created by different modeling gurus. The absence of a standardized notation deterred potential users, and as an inevitable result the modeling tools market was tiny and fragmented. The few tools that were available suffered from a lack of investment; many only allowed sketching of software designs, lacking facilities for checking the diagrams' internal consistency or automatically processing the information they held. These early visual diagrams were useful as design aids or documentation, but were rarely integrated into the software development lifecycle. The UML standard according to him changed all that, and triggered the dramatic growth in visual modeling that has led to its widespread use not only in software design, but also in non-software disciplines such as systems engineering, and in the business domain.

Selic (2005) on the other hand noted that most early software modeling languages were defined informally with little attention paid to precision. More often than not, modeling concepts were explained using imprecise and informal natural language. This was deemed sufficient at the time, since most modeling languages were used either for documentation or for what Martin Fowler referred to as design “sketching” (Fowler, 2004). The idea was to convey a design’s essential properties, leaving developers to work out details during implementation. However, this often led to confusion because different individuals could—and often did—interpret models expressed in such languages quite differently. Further, unless these individuals explicitly discussed model interpretation up front, such differences could remain undetected, until later in the development stage when costs to fix resulting problems are much greater.

According to Cernosek and Naiburg (2004) Models can play a part in the software development process in many ways. Figure 2.1 illustrates the spectrum of ways to practice model-driven development.

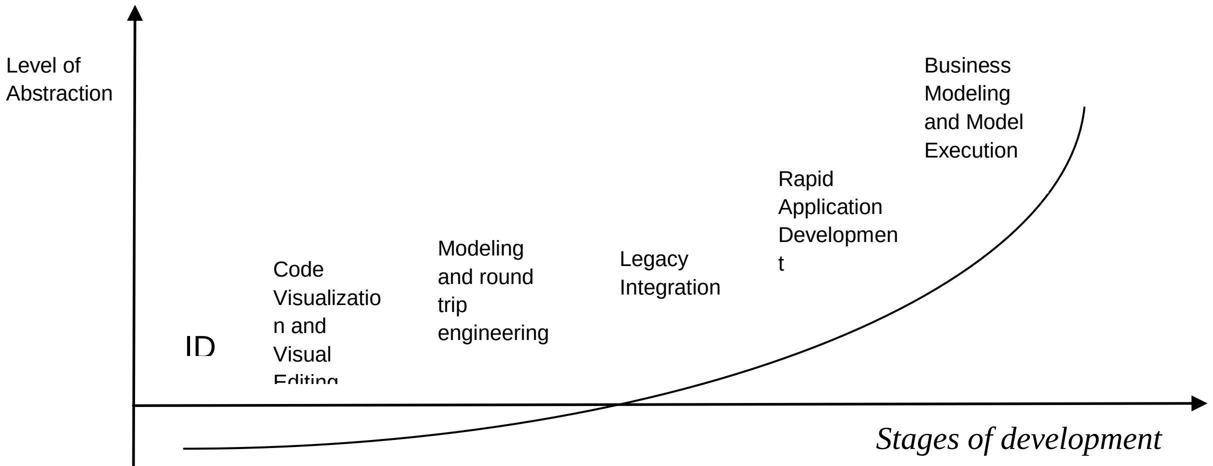

Level of Abstraction
Business Modeling and Model Execution
Rapid Application Developmen t
Modeling and round trip engineering
Legacy Integration
Code Visualizatio n and Visual Editing
ID
Stages of development

Figure 2.1: A spectrum of times, places and ways to model

Source: Cernosek and Naiburg (2004)

Integrated Development Environments: In the loosest notion of modeling, IDEs can be considered an entry point into the practice of model-driven development. Modern IDEs offer several mechanisms that raise the level of abstraction in creating and maintaining code. Tools such as language-sensitive editors, wizards, form builders and other GUI controls are not “models” in the more strict sense of the term. Nonetheless, they can raise the level of abstraction above source code, make developers more productive, help create more reliable code and enable a more effective maintenance process. All these attributes are the essence of model-driven development.

Code Visualization and Visual Editing: A step above the basic IDE functions is the ability to visualize source code in graphical form. Here, a picture is worth a thousand lines of code, in a sense. Developers have used graphical forms of abstraction above their code for many years. Traditional flow charts are a common method for depicting the algorithmic control flow of code. Structure charts, or even simple block diagrams with arrows, are often used on whiteboards—using boxes to represent functions and subprograms, arrows to indicate calling dependencies and so on. For object-oriented software, boxes typically denote classes and lines between boxes denote relationships between those classes. Coupled closely with code visualization is visual editing, in which developers edit code through the diagrams instead of through conventional IDE text windows. Visual editing is well suited for changes that have systematic effects on other pieces of code. For example, in an object-oriented system that has a set of classes related in an inheritance hierarchy, certain features of the classes (the field members, methods or functions) may need to be reorganized into different classes (a process called refactoring) as the application evolves. Using conventional code editors to enact such changes can be tedious and error-prone. But an effective visual editor allows developers, for example, to drag and drop a member function from one class to its base class and automatically adjust all code that is affected by such a change. In one sense, code visualization and visual editing are simply alternative methods for viewing and editing the code. Changes to the code are immediately reflected in corresponding diagrams and vice versa. Although some may argue that such depictions do not constitute a "model," the essence of modeling is abstraction and any visualization of code is indeed an abstraction—selectively exposing certain information while suppressing details deemed unnecessary or unwanted. Some practitioners prefer to use terms such as code model, implementation model or platform-specific model (PSM) to qualify such abstractions from other, higher-level forms of modeling that do not have such direct relationships to the code.

Modeling and round-trip engineering: The next step on the modeling spectrum represents the state of conventional model-driven development. Here, visual models are created from a methodological process that begins with requirements and delves into a high-level architectural design model. Developers then create a detailed design model from which skeletal code is generated to an IDE. The IDE is used to complete the detailed coding. Any changes made to the code that affect the design model are synchronized back into the model; any model changes are synchronized into the existing code.

Legacy integration: When developers are ready to integrate systems—whether all legacy or some new systems—they must understand the systems in place, know how the business intends for these systems to work together and prioritize those integrations. Modeling legacy systems does not necessarily mean that the entire system and all its components must be incorporated; however, developers should understand the legacy systems' architectures, how they work and how they interface with others. Understanding what the system does and what other software is dependent on it will help determine suitable steps moving forward. Several methods can be used to model legacy systems. Developers can reverse engineer code into models to understand them, manually model them or use some combination thereof.

Rapid Application Development (RAD): The practice of RAD has been around since the early 1980s. The premise is simply to provide highly productive ways to generate and maintain code. RAD is accomplished through easy-to-use, highly graphical features of an advanced IDE. RAD, distinct from both code-centric and model-driven development, raises the level of abstraction above the code, but does not use "models" per se. Business modeling and model execution. Before the need to develop software is even known, business and engineering analysts often find it useful to create "as-is" models of how their systems work today. From that model, they can analyze what works and what needs improvement. Special-purpose tools can simulate these models along several key variables, such as time, cost and resources. From the analysis, "to-be" models can be built to prescribe how new, improved processes should work. Generally, new software development is needed to implement the new processes, and the "to-be" models serve as key drivers for the ensuing development. For some application domains, the "to-be" models are specified to such rigor that complete applications can be generated from the models. Modeling at this level of abstraction offers the greatest potential for productivity and integration between the business or engineering problem domains and the technology or implementation domains.

2.3 The Unified Modeling Language (UML) 2.0

UML 2.0 is the standard's first major revision, following a series of lesser minor revisions (Object Management Group, 2004; Rumbaugh, Jacobson, & Booch, 2005). So why was it necessary to revise UML? The primary motivation came from the desire to better support

MDD tools and methods and the major highlights of UML 2.0 can be grouped into the following five major categories, listed in order of significance:

1. A significantly increased degree of precision in the language's definition. This addresses the need to support the higher levels of automation that MDD requires. Automation implies the elimination of model ambiguity and imprecision (and, hence, from the modeling language) so that computer programs can transform and manipulate models.
2. An improved language organization, characterized by a modularity that not only makes the language more approachable to new users but that also facilitates inter-working between tools.
3. Significant improvements in the ability to model large-scale software systems. Some modern software applications represent integrations of existing stand-alone applications into more complex systems of systems. This trend will likely continue, resulting in ever-more complex systems. To support such trends, the OMG added flexible new hierarchical capabilities to the language to support software modeling at arbitrary levels of complexity.
4. Improved support for domain-specific specialization. Practical experience with UML demonstrated the value of its so-called "extension" mechanisms. The OMG consolidated and refined these to allow simpler and more precise refinements of the base language.
5. Overall consolidation, rationalization, and clarifications of various modeling concepts resulting in a simplified and more consistent language. This involved, consolidating and, in a few cases, removing redundant concepts, refining numerous definitions, and adding textual clarifications and examples. Each of these is now looked at in more detail.

2.3.1 Degree of Precision

1. *To minimize ambiguity as well as in contrast to most other modeling languages of the time, the first standardized UML definition was specified using a metamodel.* This is a model that defines the characteristics of each UML modeling concept, and its relationships to other modeling concepts. The metamodel was defined using an elementary subset of UML[4] and was supplemented by a set of formal constraints written in the Object Constraint Language (OCL). This combination represented a formal specification of UML's abstract syntax[5]; that is, it defined the set of rules that you can use to determine whether a given model is well formed. For example, such rules would inform us not to connect two UML classes by a state machine transition.

2. *A major refactoring of the metamodel infrastructure*: UML 2.0's "infrastructure" comprises a set of low-level modeling concepts and patterns that are in most cases too rudimentary or too abstract to use directly in modeling software applications. However, their relative simplicity makes it easier to be precise about their semantics and their corresponding well-formedness rules. These finer-grained concepts are then combined in different ways to produce more complex user-level modeling concepts. For instance, in UML 1, the notion of ownership (i.e., elements owning other elements), the concept of namespaces (named collections of uniquely named elements), and the concept of classifier (elements that you can categorize according to their features), were all inextricably bound into one semantically complex notion. (Note that this also meant that you could not use any one of these without implying the other two.) In the UML 2.0 infrastructure, these concepts were separated and their syntax and semantics defined separately.

3. *Extended and more precise semantics descriptions:* The semantics definition of the UML 1 modeling concepts was problematic in a number of ways. The level of description was highly uneven, with some areas having extensive and detailed descriptions (e.g., state machines), while others had little or no explanations. The UML 2.0 specification puts more emphasis on the semantics and, in particular, in the key area of basic behavioral dynamics.

4. *A clearly defined dynamic semantic framework:* The UML 2.0 specification clarifies some of the critical semantic gaps in the original version. This framework is depicted in Figure 2.2 (Selic, 2004).

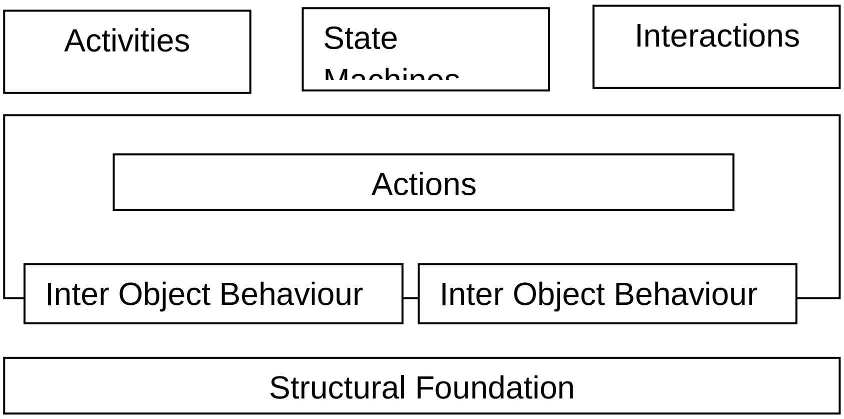


Figure 2.2: The UML 2.0 Semantic Framework
Source: Selic (2005)

In particular, this framework addresses explicitly the following issues:

1. The structural semantics of links and instances at runtime
2. The relationship between structure and behavior
3. The semantic underpinnings or causality model shared by all current high-level behavioral formalisms in UML (i.e., state machines, activities, interactions) as well as potential future ones. This also ensures that objects whose behaviors are expressed using different formalisms can interact with each other.

2.3.2 New Language Architecture

One immediate consequence of UML 2.0's increased level of precision is that the language definition has grown—even without accounting for the new modeling capabilities. This is a concern, especially given that the industry criticized the original UML for being too rich and, therefore, too cumbersome to learn and use. However, such criticisms typically ignore the fact that UML is intended to address some of today's most complex software problems and that such problems demand sufficiently powerful tools. (Successful technologies, such as automobiles and electronics, have not become simpler over time; it is a part of human nature to persistently demand more of our machinery, which, ultimately, implies more sophisticated tools. No one would even contemplate building a modern skyscraper using basic hand tools.)

To deal with the language-complexity problem, the OMG modularized UML 2.0 in a way that allows developers to selectively use language modules. Figure 2.3 shows the general form of this structure. It consists of a foundation comprising shared concepts, such as classes and associations, on top of which is a collection of vertical "sub-languages" or language units, each one suited to modeling a specific form or aspect. These vertical language units are generally independent of each other; therefore, you can use them independently. (Note that this was not the case in UML 1, where, for example, the activities formalism was based entirely on the state machine formalism.)

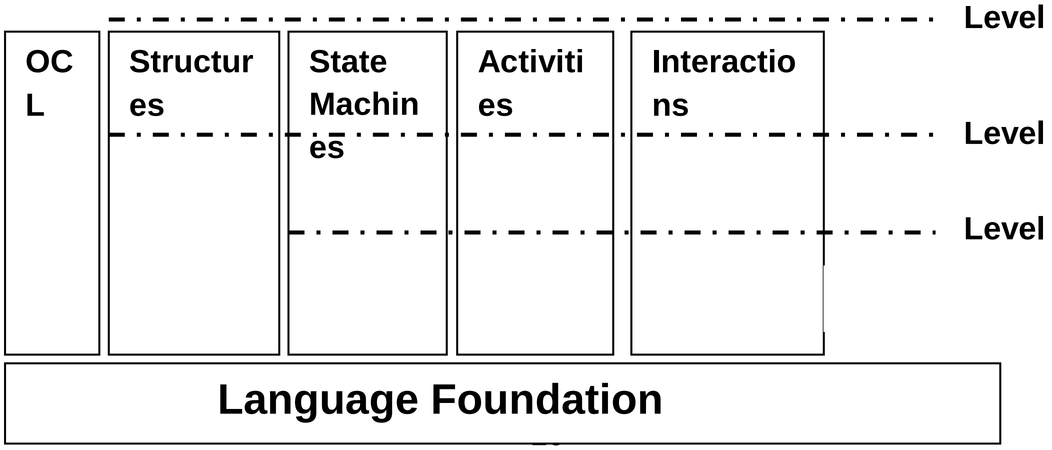


Figure 2.3: The language architecture of UML

Source: Selic (2004)

Further, the vertical language units are hierarchically organized into as many as three levels, with each successive level adding more modeling capabilities to those available in the levels below. This provides an additional dimension of modularity so that, even within a given language unit, you can only use specific subsets.

This architecture means that users can learn and use only the UML subset that suits them best. It is no more necessary to become familiar with the full extent of UML in order to use it effectively than it is to learn all of English to use it effectively. As you gain experience, you have the option of gradually introducing more powerful modeling concepts as necessary.

As part of the same architectural reorganization, the definition and structure of compliance has been significantly simplified in UML 2.0. In UML 1, the basic units of compliance were defined by the metamodel packages, with literally hundreds of possible combinations. This meant that it was highly unlikely to find two or more modeling tools that could interchange models, since each would likely support a different package combination.

In UML 2.0, only three levels of compliance are defined and those correspond to the hierarchical language unit levels already mentioned and depicted in Figure 2.3. These are defined in such a way that models at level (n) are compliant with models at any of the higher levels (n+1, etc.). That is, a tool compliant to a given level can import models, without loss of information, from tools that were compliant to any level equal to or below its own. Four types of compliance are defined:

- Compliance to the abstract syntax
- Compliance to the concrete syntax (i.e., the UML notation)
- Compliance to both abstract and concrete syntax
- Compliance to both the abstract and concrete syntax and the diagram interchange standard (OMG, 2004)

This means that there is a maximum of only 12 different compliance combinations with clear dependency relationships between them (e.g., abstract and concrete syntax compliance is compatible with only concrete syntax compliance or only abstract syntax compliance). Consequently, in UML 2.0, model interchange between compliant tools from multiple vendors becomes more than just a theoretical possibility.

2.3.3 Large-Scale System Modeling Capabilities

Relatively few features were added to UML 2.0. This was intentional to avoid the infamous "second system" effect (Brooks, 1995) whereby a language gets bloated by an excess of new features demanded by a highly diverse user community. In fact, the majority of new modeling capabilities are, in essence, simply extensions of existing features that allow you to use them to model large-scale software systems. Moreover, these extensions were all achieved using the same basic approach: recursive application of the same basic set of concepts at different levels of abstraction. This means that you could combine model elements of a given type into units that, in turn, you would use as the building blocks for the next level of abstraction and so on; this is analogous to the way that you could nest procedures in programming languages within other procedures to any desired depth. Specifically, the following modeling capabilities are extended in this way:

1. Complex structures
2. Activities
3. Interactions
4. State machines

The first three of these account for more than 90 percent of UML 2.0's new features.

2.3.4 Language Specialization Capabilities

Experience with UML 1 indicated that a very common way of applying UML was to first define a UML profile for a particular problem or domain and then to use that profile instead of or in addition to general UML. In essence, profiles are a way of producing what are now commonly referred to as domain-specific languages (DSLs). An alternative to using UML profiles is to define a new custom modeling language using the MOF standard and tools. The latter approach has the obvious advantage of providing a clean slate, enabling a language definition that is optimally suited to the problem at hand. At first glance, this may seem the preferred approach to a DSL definition, but closer scrutiny reveals that there can be serious drawbacks to it.

As noted earlier too much diversity leads to the kind of fragmentation problems that UML was designed to eliminate. In fact, this is one of the primary reasons why it was accepted so widely and so rapidly. Fortunately, the profile mechanism provides a convenient solution for many practical cases. This is because there is typically a lot of commonality even between diverse DSLs. For example, practically any object-oriented modeling language will need to define the concepts of classes, attributes, associations, interactions, etc. UML, which is a general-purpose modeling language, provides just such a convenient and carefully defined collection of useful concepts. This makes it a good starting point for a large number of possible DSLs.

However, there is more than just conceptual reuse at play here. Because a UML profile, by definition, has to be compatible with standard UML$_{1.0}$; (1) you can use any tool that supports standard UML to manipulate models based on that profile and (2) directly apply any knowledge of and experience with standard UML. Therefore, you can mitigate many of the fragmentation problems stemming from diversity or even avoid them altogether. This type of reasoning led the international standards body responsible for the SDL language (International Telecommunications Union, 2002)—a DSL widely used in telecommunications—to redefine SDL as a UML profile (International Telecommunications Union, 2000).

This is not to say that any DSL can and should be realized as a UML profile; there are indeed many cases where UML may lack the requisite foundational concepts that you can cast into corresponding DSL concepts. However, given UML’s generality, it may be more widely applicable than many people might think.

With these considerations in mind, the profiling mechanism in UML 2.0 has been rationalized and its capabilities extended. The conceptual connection between a stereotype and the UML concepts that it extends has been clarified. In effect, a UML 2.0 stereotype is defined as if it was simply a subclass of an existing UML metaclass, with associated attributes (representing tags for tagged values), operations, and constraints. The mechanisms for writing such constraints using a language such as OCL have been fully specified. In addition to constraining individual modeling concepts, a UML 2.0 profile can also explicitly hide UML concepts that make no sense or are unnecessary in a given DSL. This allows you to define minimal DSL profiles.

Finally, you can also use the UML 2.0 profiling mechanism to view a complex UML model from multiple, different domain-specific perspectives—something not generally possible with DSLs. That is, you can selectively "apply" or "de-apply" any profile without affecting the underlying UML model in any way. For example, a performance engineer may choose to apply a performance modeling interpretation over a model, attaching various performance-related measures to the model's elements. An automated performance analysis tool can then use these to determine a software design's fundamental performance properties. At the same time and independent of the performance modeler, a reliability engineer might overlay a reliability-specific view on the same model to determine its overall reliability characteristics.

2.3.5 General consolidation

This item covers several areas, including the removal of overlapping concepts as well as numerous editorial modifications, such as adding clarifications to confusing descriptions and the standardization of terminology and specification formats. The removal of overlapping concepts and the clarification of poorly defined concepts were two other important requirements for UML 2.0. The three major areas affected by this requirement were actions and activities, templates, and component-based design concepts.

Actions were introduced in UML 1.5. The conceptual model of actions was intentionally made general enough to accommodate both data-flow and control-flow computing models. This resulted in a significant conceptual similarity to the activities model. UML 2.0 exploits this similarity to provide a common syntactic and semantic foundation for actions and activities. From the user's point of view, these are formalisms that occur at different abstraction levels since they typically model phenomena at different granularity levels. However, the shared conceptual base results in overall simplification and greater clarity.

In UML 1, templates were defined very generally: you could make any UML concept into a template. Unfortunately, this generality impeded its application since it allowed for potentially meaningless template types and template substitutions. UML 2.0's template mechanism was restricted to cases that were well understood: classifiers, operations, and packages. The first two were modeled after template mechanisms found in popular programming languages. In the area of component-based design, UML 1 had a confusing abundance of concepts. You could use classes, components, or subsystems. These concepts had a lot in common but were subtly different in non-obvious ways. There was no clear delineation as to which to use in any given situation. Was a subsystem just a “big” component? If so, how big did a component have to be before it became a subsystem? Classes provided encapsulation and realized interfaces, but so did components and subsystems.

In UML 2.0, all these concepts were aligned, so that components were simply defined as a special case of the more general concept of a structured class, and, similarly, subsystems were merely a special case of the component concept. The qualitative differences between these were clearly identified so that you could decide when to use which concept on the basis of objective criteria. On the editorial side, the specification format was consolidated with the semantics and notation specifications for the modeling concepts combined for easier reference. Each metaclass specification was expanded with information that explicitly identifies semantic variation points, notational options, as well as its relationship to the UML 1 specification. Also, the terminology was made consistent so that a given term (e.g., type, instance, specification, and occurrence) has the same general connotation in all contexts in which it appears.

In summary, UML 2.0 was designed to allow a gradual introduction of model-driven methods. You can still use it in the same informal way as UML 1 if you prefer it as a “sketching” tool. Moreover, since the new modeling capabilities are non-intrusive, in most cases, you will not see any change in the language’s look and feel. However, the opportunity to move forward on the MDD scale is now available and standardized. The increased precision is also available for you to use, if desire, all the way through to completely automated code generation.

The standards body carefully reorganized the language structure to allow a modular and graduated approach to adoption: users only need to learn the parts of the language that are of interest to them and can safely ignore the rest. As your experience and knowledge increases, you can selectively add new capabilities. Along with this reorganization, the definition of compliance to facilitate interoperability between complementary tools as well as between tools from different vendors is greatly simplified. Only a small number of new features were added to avoid language bloat, and practically all of those are designed along the same recursive principle that enables modeling of large and complex systems. In particular, extensions were added to more directly model software architectures, complex system interactions, and flow-based models for applications, such as business process modeling and systems engineering. The language extension mechanisms were slightly restructured and simplified for a more direct way of defining UML-based domain-specific languages. These languages have the distinct advantage that they can directly take advantage of UML tools and expertise, both of which are abundantly available.

The overall result is a second-generation modeling language that will help us develop more sophisticated software systems faster and more reliably—but still using the same type of intuition and expertise that is every software developer's bread and butter. In essence, it is still program design, only at a higher level—comparable to the step that occurred in hardware design when discrete components gave way to large-scale integration.

2.4 UML Adoption and Usage

Cernosek and Naiburg (2004) noted that software industry has adopted the Unified Modeling Language as its standard means for representing software models and related artifacts. Software architects, designers and developers use UML for specifying, visualizing, constructing and documenting all aspects of a software system. Key leaders from IBM Rational led the original development of UML. Today, UML is managed by the Object Management Group (OMG), which consists of representatives throughout the world to help ensure that the specification continues to meet the dynamic needs of the software community. Adopting a standard notation such as UML is an important step in taking a model-driven approach to software development. UML is more than just a graphical notational standard—it is a modeling language. As with all languages, UML defines syntax (both graphical and textual, in this case) and semantics (the underlying meanings of the symbols and text). Having a true modeling language rather than just a standard notation is essential for standardizing the use of UML as well as for helping to ensure that automated tools can properly enforce the rules behind the symbols. UML— a true modeling language—has helped it become the software industry's most recognized and widely applicable modeling standard.

2.4.1 Space Telescope Software

Another example is the software for the new James Webb space telescope. To aid communication and help meet stringent reliability and performance goals, all the software being built for the telescope by NASA, the Canadian and European space agencies and all their subcontractors is being designed using UML. Organisations across the world are cooperating on guidance software, a command data handling system and software for the science module housing four different light-receiving instruments. All will be integrated in the telescope itself, destined for earth orbit at an altitude of 940,000 miles (four times the distance to the moon).

2.4.2 Certification Examinations

UML is supported by every major commercial IT vendor, as well as a flourishing selection of Open Source tools. UML books & training are widely available, and the OMG Certified UML Professional (OCUPTM) and OMG-Certified Real-time and Embedded Specialist (OCRESTM) certification programmes have allowed tens of thousands of engineers and architects to establish their UML credentials. UML has changed the software world.

As UML use has grown, continuous feedback from the user community and investment by tool vendors has helped the standard evolve and mature. The original UML 1 standard of 1997 was backed by 21 OMG member companies; feedback from dozens more submitted via OMG’s issue-reporting system helped refine it, flushing out remaining inconsistencies. In 2005 OMG published UML 2, a major revision largely based on the same familiar diagram notations, but using a more rigorous underlying modeling infrastructure specified using OMG's Meta-Object Framework (MOF). While some designers still use UML merely for sketching designs to share with colleagues, UML 2's MOF foundation means that today's UML diagram is more than just a pretty picture. A MOF-aware modeling tool can capture the meaning of diagram elements and their relationships in machine-readable form, and use this to reason about the design, perform consistency checks, and even automatically generate parts of the application code.

2.4.3 What People are Saying

Like any technology, UML had early adopters that led the charge in discovering its value. Here are just a few comments from IBM Rational® customers about the value that modeling contributed to their businesses:

“We are trying to reduce the overall cost of insurance to our members. One of the ways to do that is to reuse information and reuse the assets that we build as we go through our business modeling. Model-driven architecture is really at the core of what we’re doing from a business modeling perspective. When we begin projects from a software development perspective

without a clear business model, without a clear set of business objectives or business goals, we are finding that the customers don't get what they think they have asked for."
— *Sue Nelson, director of business modeling for Blue Cross and Blue Shield of Florida*

"I think visual modeling is just a key element in any developer's toolbox. It enables us to bring in specialized expertise, such as security analysis of a product. By having a common modeling technique that everyone knows how to read, we can bring in our company security expert and that person can very easily review the product and point out any potential holes."
— *Nanette Brown, director of applied architecture and quality assurance at Pitney Bowes*

"Enterprise architecture presents its own very unique modeling challenges. You are modeling at multiple levels. You are modeling with large groups of people and different teams. And the models at each level tend to have to be customized for the individual stakeholder types. [Modeling with UML] provided us with the flexibility [to meet] our unique needs and demands at each level of the enterprise architecture."
— *Frank Armour, president of ArmourIT, LLC*

Testimonies like these can show the reduced risks to others who are just getting started with modeling and can ultimately help position modeling closer to the mainstream of software development.

2.4.4 British Computer Society professional examination

A look at the British Computer Society professional examination on Object Oriented Programming (Version 2: New Syllabus) taken on 21st April 2004 between 2.30p.m and 4.30p.m also reveals that UML usage is an integral part of Object Oriented Programming. The students were instructed to answer FOUR questions out of SIX and all questions were UML based.

2.5 UML and the Unified Process

The OMG specification states that *The Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system. The UML offers a standard way to write a system's blueprints, including conceptual things such as business processes and system functions as well as concrete things such as programming language statements, database schemas, and reusable software components* (Sparx Systems, 2019).

The important point to note here is that UML is a 'language' for specifying and not a method or procedure. The UML is used to define a software system; to detail the artifacts in the system, to document and construct - it is the language that the blueprint is written in. The UML may be used in a variety of ways to support a software development methodology (such as the **Rational Unified Process)** - but in itself it does not specify that methodology or process.

Koichiro (2008) gave the relationship between method and UML as follows

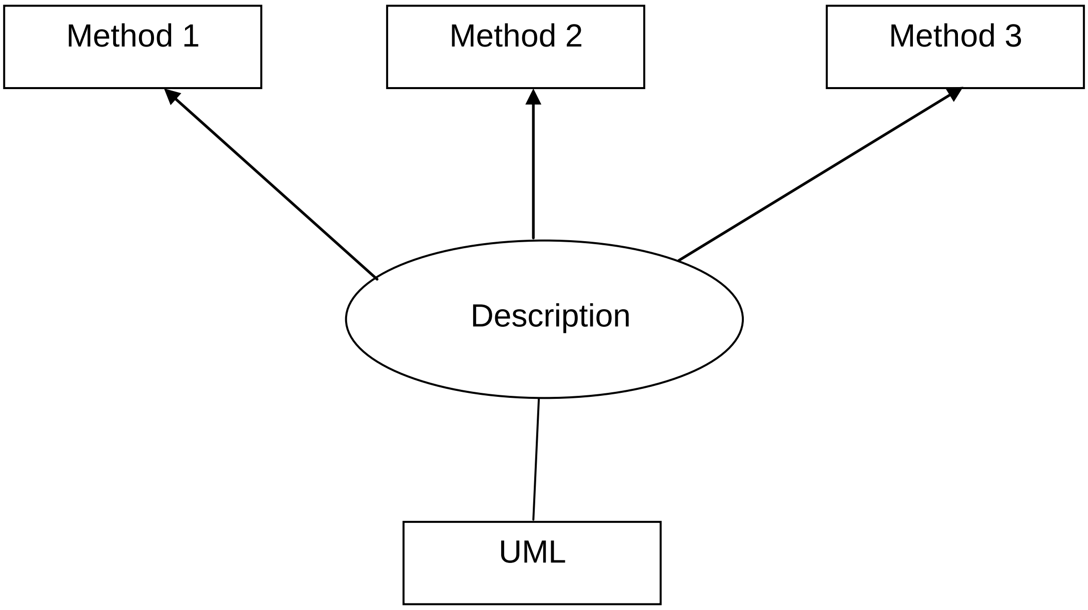


Figure 2.4 Relationship between method and UML
Source: Koichiro (2008)

The Unified Software Development Process or *Unified Process* is a popular iterative and incremental software development process framework. The best-known and extensively documented refinement of the Unified Process is the Rational Unified Process (RUP).

The Unified Modeling Language (UML) is an evolutionary general-purpose, broadly applicable, tool-supported, and industry-standardized modeling language or collection of modeling techniques for specifying, visualizing, constructing, and documenting the artifacts of a system-intensive process. The UML is broadly applicable to different types of systems (software and non-software), domains (business versus software), and methods and processes. The UML enables and promotes (but does not require nor mandate) a use-case-driven, architecture-centric, iterative and incremental process.

The Unified Modeling Language (UML) is an evolutionary general-purpose, broadly applicable, tool-supported, and industry-standardized modeling language or collection of modeling techniques for specifying, visualizing, constructing, and documenting the artifacts of a system-intensive process. The UML is broadly applicable to different types of systems (software and non-software), domains (business versus software), and methods and processes. The UML enables and promotes (but does not require nor mandate) a use-case-driven, architecture-centric, iterative and incremental process.

2.6 Models and Architectural views

Models are blueprints of systems used for system construction and renovation. They are used to understand and manage complexities in a system. Architectural views map models to a type of diagrams. Different architectural views include: user view, structural view, behavioural view and implementation view.

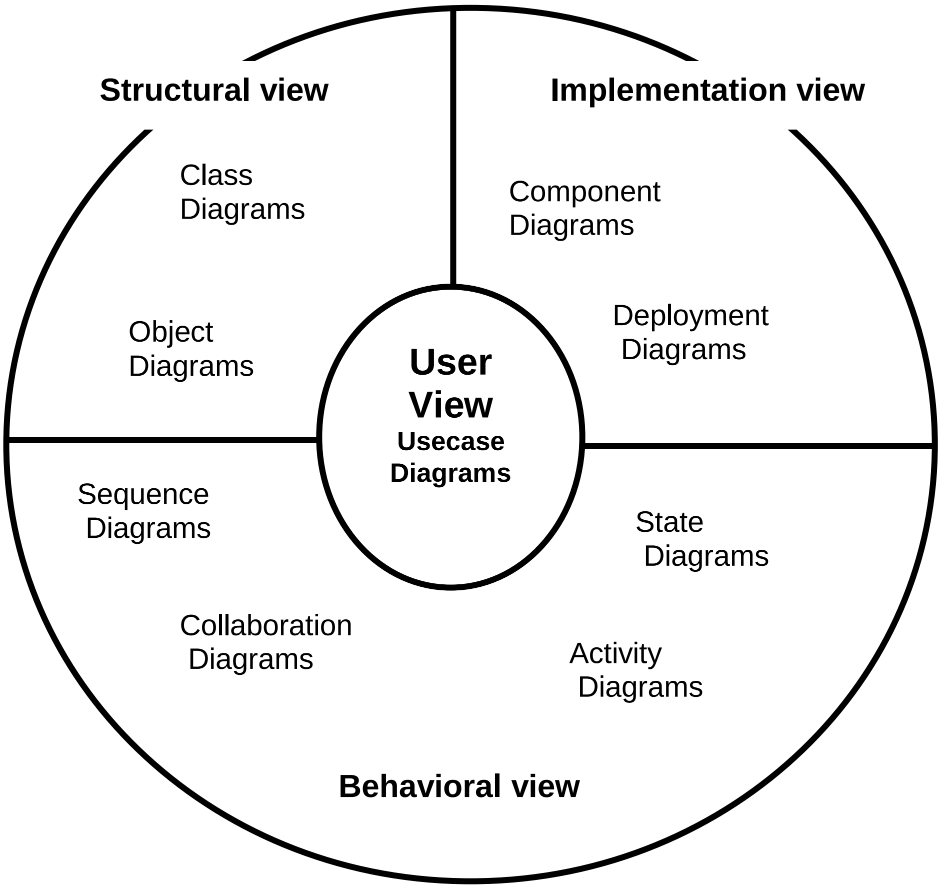


Figure 2.5 Architectural views of a Model
Source: Koichiro (2008)

Eriksson and Penker (2008) gave five views of UML as:

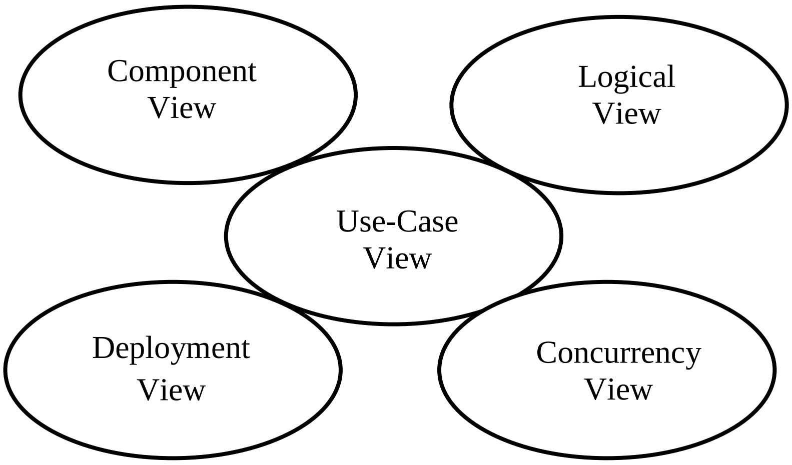


Figure 2.6 Views of UML

Source: Koichiro (2008)

And Koichiro (2008) showed the relationship between views and diagrams in UML as

Use-Case View

– Use-Case Diagram

Logical View

– Class Diagram, Object Diagram

– State Diagram, Sequence Diagram, Collaboration Diagram, Activity Diagram

Concurrency View

– State Diagram, Sequence Diagram, Collaboration Diagram, Activity Diagram

– Component Diagram, Deployment Diagram

Deployment View

– Deployment Diagram

Component View

– Component Diagram

Koichiro (2008) also showed where different UML diagrams are used in the Unified process models as

Use-Case Model

– Use-Case Diagram

Analysis Model

– describe "Realization ….a Use-Case" by a Collaboration Diagram and a Flow of Event Description

Design Model

– Class Diagram, Sequence Diagram, and Statechart Diagram

Deployment Model

– Deployment Diagram

Implementation Model

– Component Diagram

Test Model

– Test Case

2.7 UML Diagrams

UML 2.0 has 13 modeling diagrams: Activity Diagrams, Class Diagrams, Object Diagrams, Use Case Diagrams, State Machine Diagrams, Sequence Diagrams, Communication Diagrams, Deployment Diagrams, Timing Diagrams, Package Diagrams, Component Diagrams, Interaction Overview Diagrams and Composite Diagrams.

However, each of these diagrams is for different types of modeling and with UML 2.0 architecture users can learn and use only the UML subset that suits them best. It is no more necessary to become familiar with the full extent of UML in order to use it effectively just as you don’t need to know all of English language units to use it effectively. However, the discussion of UML diagram in this work will be grouped according to Koichiro (2008) classification of where different UML diagrams are used in the Unified process which is still summarised below.

1. **Use-Case Model**: Use-Case Diagrams are used for requirement capture.
2. **Analysis Model:** – describe “Realization of a Use-Case” by a Collaboration Diagram and a Flow of Event Description (with sequence diagram.)
3. **Design Model:** This will be achieved with Class Diagram, Sequence Diagram, and Statechart Diagram
4. **Implementation Model:** Component Diagram
5. **Deployment Model:** Deployment Diagram

The diagram for Use-case model which is central and the ones for Analysis model will be discussed in this chapter of system analysis while the ones for design, implementation and deployment will be discussed in their appropriate chapters.

2.8 How to use the UML

OMG (2004) explained that the UML is typically used as a part of a software development process, with the support of a suitable CASE tool, to define the requirements, the interactions and the elements of the proposed software system. The exact nature of the process depends on the development methodology used. An example process might look something like the following:

1. Capture a Business Process Model. This will be used to define the high level business activities and processes that occur in an organization and to provide a foundation for the Use Case model. The Business Process Model will typically capture more than a software system will implement (i.e. it includes manual and other processes).

2. Map a Use Case Model to the Business Process Model to define exactly what functionality you are intending to provide from the business user perspective. As each Use Case is added, create a traceable link from the appropriate business processes to the Use Case (i.e. a realization connection). This mapping clearly states what functionality the new system will provide to meet the business requirements outlined in the process model. It also ensures no Use Cases exist without a purpose.

3. Refine the Use Cases - include requirements, constraints, complexity rating, notes and scenarios. This information unambiguously describes what the Use Case does, how it is executed and the constraints on its execution. Make sure the Use Case still meets the business process requirements. Include the definition of system tests for each use case to define the acceptance criteria for each use case. Also include some user acceptance test scripts to define how the user will test this functionality and what the acceptance criteria are.

4. From the inputs and outputs of the Business Process Model and the details of the use cases, begin to construct a domain model (high level business objects), sequence diagrams, collaboration diagrams and user interface models. These describe the 'things' in the new system, the way those things interact and the interface a user will use to execute use case scenarios.

5. From the domain model, the user interface model and the scenario diagrams create the Class Model. This is a precise specification of the objects in the system, their data or

attributes and their behavior or operations. Domain objects may be abstracted into class hierarchies using inheritance. Scenario diagram messages will typically map to class operations. If an existing framework or design pattern is to be used, it may be possible to import existing model elements for use in the new system. For each class define unit tests and integration tests to thoroughly test i) that the class functions as specified internally and that ii) the class interacts with other related classes and components as expected.

6. As the Class Model develops it may be broken into discrete packages and components. A component represents a deployable chunk of software that collects the behavior and data of one or more classes and exposes a strict interface to other consumers of its services. So from the Class Model a Component Model is built to define the logical packaging of classes. For each component define integration tests to confirm that the component's interface meets the specification given it in relation to other software elements.

7. Concurrent with the work you have already done, additional requirements should have been captured and documented. For example - Non Functional requirements, Performance requirements, Security requirements, responsibilities, release plans & etc. Collect these within the model and keep up to date as the model matures.

8. The Deployment model defines the physical architecture of the system. This work can be begun early to capture the physical deployment characteristics - what hardware, operating systems, network capabilities, interfaces and support software will make up the new system, where it will be deployed and what parameters apply to disaster recovery, reliability, back-ups and support. As the model develops the physical architecture will be updated to reflect the actual system being proposed.

9. Build the system: Take discrete pieces of the model and assign to one or more developers. In a Use Case driven build this will mean assigning a Use Case to the development team, having them build the screens, business objects, database tables, and related components necessary to execute that Use Case. As each Use Case is built it should be accompanied by completed unit, integration and system tests. A Component driven build may see discrete software components assigned to development teams for construction.

10. Track defects that emerge in the testing phases against the related model elements - e.g. System test defects against Use Cases, Unit Test defects against classes & etc. Track any

changes against the related model elements to manage 'scope creep'.

11. Update and refine the model as work proceeds - always assessing the impact of changes and model refinements on later work. Use an iterative approach to work through the design in discrete chunks, always assessing the current build, the forward requirements and any discoveries that come to light during development.

12. Deliver the complete and tested software into a test then production environment. If a phased delivery is being undertaken, then this migration of built software from test to production may occur several times over the life of the project.

The above process is necessarily brief in description; it is just given as an example of how the UML may be used to support a software development project.

2.9 Review of related Work

UML is currently one of the most widely used modeling language Mohagheghi, Dehlen and Neple (2009) and it is often employed by companies in the software analysis and design phases. However, it is also perceived as a very complex notation. For this reason, in the last decade, several works have been presented with the aim of UML receiving wider adoption and Use in software modeling.

Dzidek (2008) in his dissertation on Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance explores the impact of UML with a focus on the maintenance of object-oriented software. In one of his papers on A Systematic Review on the Effects of UML during the Maintenance of Object-Oriented Software, The review presents a systematic literature review on the effects that the use of standard UML by developers has on the design and maintenance of object-oriented software. One of the findings is that few empirical studies exist that investigate the costs and evaluate the benefits of using UML in realistic contexts. Such studies are needed so that the software industry can make informed decisions regarding the extent to which they should adopt UML in their development practices. Adoption of UML is not without cost and risks. Costs include training of staff, purchase and integration of tools, and construction of the diagrams. Risk includes misinterpretations of inconsistent and incorrect models, though this can be mitigated with too support and model reviews. His findings however concluded that the benefits of UML adoption are well worth the costs and the risks.

Koivulahti-Ojala (2017) in his work: On UML Modeling Tool Evaluation, Use and Training, offered new knowledge about UML modeling tool use, evaluation, and training. The main research question was: How can a globally distributed product company where UML modeling activities are scattered across different locations and countries implement a UML modeling tool? The study provided information concerning how UML and UML modeling tools can be used in the context of product requirements and release management process.

The main conclusions from a study by Budgen, Burn, Brereton, Kitchenham, and Pretorius (2011) on empirical evidence of the UML is that while there are many studies that use the UML in some way, including to assess other topics, there are relatively few for which the UML is itself the object of study, that assess the UML in some way such as UML studies of adoption and use in the field.

Till the time of this research, not much work has been done on UML Adoption. Cabot (2013) in his work titled UML Adoption in practice: has anything changed in the last decade? noted that the results of International Conference on Software Engineering 2013 paper on UML in practice, is not different from those reported in 2006 by Dobing and Parson on How UML is Used. This should not be the case and shows that more studies need to be conducted on UML knowledge and adoption.

Petre (2013) in his study on UML in Practice conducted an empirical study which involves series of interviews conducted over 2 years with more than 50 practicing professional software developers. Informants were identified with an eye to gathering a broad range of perspectives, from corporate large-scale commercial software developers to independent consultants, and across a variety of application areas. Informants came primarily from countries in Europe and North America, but there were also informants from Brazil, India, and Japan, and many had worked in more than one country. Informants were identified opportunistically, via networks of collaborators, colleagues and contacts – people who could act as ‘brokers’ for introductions of various kinds: at meetings and conferences, via mailing lists, via social networks such as the Requirements Engineering Specialist Group (RESG) on LinkedIn, and via personal emails. All informants were practicing professional software developers in roles ranging from requirements engineering, to software architecture, software development, and quality assurance (and most identified themselves as fulfilling more than one role). Only one informant per company was included in the reported data, reducing the sample size to 50.

Simple, semi-structured interviews were conducted over the phone, on Skype, or in face-to-face meetings, as convenient. The protocol was straightforward, starting with background questions about the professional's experience, role, organizational context, and software projects. The key question was: 'Do you use UML?' Depending on the response, the second question was either: 'Can you tell me about how you use it?' or 'Why not?' Subsequent questions followed up responses and elicited examples of use of UML or other design representations. When appropriate, the informant was asked if his or her usage was typical of the organization. Hand-written or typed notes were captured for all interviews, and, subject to the informant's preference, some interviews were audio-recorded. Some informants provided actual examples of design representations, within confidentiality agreements. Discussions at times extended beyond the informants' current practice to past projects, past organizations, or other experiences. At times the discussion distinguished between the use by the informant and the use preferred or mandated by the organization. All accounts of UML use offered by the informants were collected, but a distinction was made in the data collection between the informants' own current use (identified in this paper as 'declared current use') and accounts of their own practice in the past or in other organizational contexts, accounts of organizational preferred practices, or accounts of their colleagues' practices which they have observed directly (identified as 'secondary reports'). The work focused on responses to do with current practice but includes, where relevant, discussion on 'secondary reports'. The analysis was inductive, allowing categories of use to emerge from the data. The initial sorting into 'use' and 'non- use' was obvious. Additional categories were identified in terms of what the informants presented as characteristic of their use. The categories, along with a representative selection of anonymized data, were presented to two experienced professional software developers for independent review, as a form of validation. In his overall results, five patterns of use were identified.

The research by Dos, Soares and Vrancken (2017) on Evaluation Of UML In Practice, Experiences in a Traffic Management Systems Company was on improving the Software Engineering process at a company that develops software-intensive systems. Their hypothesis was that UML has some difficulties/drawbacks in certain system development phases and activities. Many of these problems were reported in the literature normally after applying UML to one project and/or studying the language's formal specifications and comparing with other languages. However, they also reported that unfortunately, few publications are based on surveys and interviews with practitioners, i.e., the developers and project managers that are using UML in real projects and that frequently facing these problems.

The research methodology involved surveys, interviews and action research with a system developed in order to implement the recommendations and evaluate the proposed improvements. The recommendations were considered feasible, as they are not proposing to radically change the current situation, which would involve higher costs and risks. Their evaluation was done by applying some of the recommendations in a project in the company.

Erickson and Siau(2007) conducted a Delphi study with the goal of identifying a UML kernel for three well-known UML application areas: Real-Time, Web-based, and Enterprise systems. The participants to the study were asked to rate the relative importance of the various UML diagrams in building systems. UML overall results (i.e. non-domain specific) were: 100% for class and state machine diagrams, 95.5% for sequence diagrams, 90.9% for use case diagrams. All the others diagrams received a percentage lower than 50%, e.g. 27.3% for activity diagrams.

Previous study on UML by Grossman et al. (2005) confirmed the results of Erickson and Siau(2007). The results indicate that the three most important diagrams are use case diagram, class diagrams and sequence diagrams. Also, Wrycza and Marcinkowski (2007) in another UML survey, have tried to downsize the UML to find the most useful diagrams. The participants perceived use case, class, activity, and sequence diagrams as the most useful.

Dobing and Parsons (2006) pointed out another strong statement: "regular usage of UML components were lower than expected". Dobing and Parsons (2006) suggest that the difficulty of understanding many of the notations "support the argument that the UML may be too complex". In "Taking the temperature of UML" Jacobson (2009), he wrote: "Still, UML has become complex and clumsy. For 80% of all software only 20% of UML is needed. However, it is not easy to find the subset of UML which we would call the 'Essential' UML. We must make UML smarter to use". The need to simplify the UML is also shown by the recently released OMG draft proposal about this topic. Seidewitz (2012).

Adriana, Tayana and Igor (2019) in Analyzing students Perception of UML Diagrams: Instruments used in Evaluation conducted an exploratory study to investigate students perception regarding UML diagram acceptance. They investigated five UML diagrams taught

Federal University of Amazonas. In order to evaluate the students perception of UML diagrams, they applied Technology Acceptance Model (TAM) questionnaires.

Jonona and Milan (2019) In Evaluation of UML diagrams for test cases generation: Case study on depression of internet addiction observed that using UML diagrams for test cases generation and applying them in first stages of software development, leads to decrease in cost and effort. They applied UML diagrams for case study on depression on internet addiction estimation.

2.10 Research Methods

Every research aims at solving a problems or answering a question. In other words, the purpose of doing research is to add a new knowledge to the existing body of knowledge in an area of interest. Research Methodology is a set of systematic techniques used in research. It aims to describe and analyze methods and throw light on their limitations and resources. It is a believe about the way in which data about a phenomenon should be gathered, analyzed and used. (Igbokwe in Nnabude, Nkamnebe and Ezenwa 2009).

A large number of research methodologies exist. The choice of which method to employ is dependent upon the nature of the research problem. Research methodology for a particular research is chosen in such a way as to ensure that the evidence obtained enables the researcher to solve the problem or answer the research questions. Morgan and Smircich (1980) argue that the actual suitability of a research method derives from the nature of the social phenomena to be explored. There are basically two basic methodological traditions of research, namely positivism and postpositivism (phenomenology). Positivism is an approach to the creation of knowledge through research which emphasizes the model of natural science: the scientist adopts the position of objective researcher, who collects facts about the social world and then builds up an explanation of social life by arranging such facts in a chain of causality (Finch,1986). In contrast, post-positivism is about a reality which is socially constructed rather than objectively determined. Hence the task of social scientist should not be to gather facts and measure how often certain patterns occur, but to appreciate the different constructions and meanings that people place upon their experience (Easterby-Smith, Thorpe & Lowe 1991). Positivism, thus, which is based on the natural science model of dealing with facts, is more closely associated with quantitative method of analysis. On the other hand, post- positivism that deals with understanding the subjectivity of social phenomena requires a qualitative approach. Quantitative (Positivist) and qualitative (interpretivist also called anti – positivist) research designs are therefore commonly used to investigate research questions.

2.10.1 Quantitative Positivist Research

Quantitative Positivist Research is a set of methods and techniques that allow researchers to answer research questions about the interaction of humans and computers. There are two cornerstones in this approach to research. The first cornerstone is the emphasis on quantitative data. The second cornerstone is the emphasis on positivist philosophy. Regarding the first cornerstone, these methods and techniques tend to specialize in quantities in the sense that numbers come to represent values and levels of theoretical constructs and concepts and the interpretation of the numbers is viewed as strong scientific evidence of how a phenomenon works. The presence of quantities is so predominant in Quantitative Positivist Research that statistical tools and packages are an essential element in the researcher's toolkit. Sources of data are of less concern in identifying an approach as being Quantitative Positivist Research than the fact that empirically derived numbers lie at the core of the scientific evidence assembled. A Quantitative Positivist Research researcher may use archival data or gather it through structured interviews. In both cases, the researcher is motivated by the numerical outputs and how to derive meaning from them. This emphasis on numerical analysis is also a key to the second cornerstone, positivism, which defines a scientific theory as one that can be falsified.

2.10.2 Types of Quantitative Research

Ismail (2017) noted that fitting a research problem to a specific type of research is quite a task since one has to be crystal clear on the relationships among the variables in the research problem before deciding on the types of research to be adopted. He identified two major kinds of relationships

- With cause-and-effect (experimental Studies) or
- Without cause-and-effect (descriptive studies).

Experimental Studies

A cause-and-effect relationship may demands the following research types:

1. *A pure experiment:* This type enables us to manipulate an independent variable in order to see the effect on the dependent variable.

2. *A quasi experiment:* The only difference between a quasi experiment and a true experiment is that, in quasi experiment there is no randomization of subjects between levels of the independent variable, for instance between control and experimental groups.
3. *Ex-post-facto or causal-comparative:* A causal relation could also be established by causal-comparative method although not as strong as the experimental method.
4. *Time series design:* A cause-and-effect relationship could also be established using a time series design. A series of observations based on a defined duration between observations are recorded for a group of subjects before and after a treatment is given. If we find that the performance of the subjects are consistently higher after the treatment, than the effect has taken place and it is caused by the treatment.
5. *Survey research:* If the focus is not so much on A causes B, but rather the description of a phenomenon such as relationship among variables, survey research is appropriate.

Descriptive Studies

A non cause-and-effect relationship requires a plain descriptive research describing about the pattern of relationships among the variables (Ismail 2005). Iwueze in Nnabulue, Nkamnebe and Ezenwa (2009) explained that in descriptive study, no attempt is made to change behavior or conditions rather we measure things as they are. Descriptive studies are also called observational, because we observe the subject without otherwise intervening. Types of descriptive study are case, case series, cross sectional etc.

Alavi and Calson (1992) and Boudreau et el (2004) in their researches broadly categorized research methods in information system as:

1. Laboratory experiment
2. Field Experiment
3. Field study (Survey)
4. Case study

Laboratory experiments take place in a setting especially created by the researcher for the investigation of the phenomenon. With this research method, the researcher has control over the independent variable(s) and the random assignment of research participants to various treatment and non-treatment conditions.

Field experiments involve the experimental manipulation of one or more variables within a naturally occurring system and subsequent measurement of the impact of the manipulation on one or more dependent variables.

Field studies are non-experimental inquiries occurring in natural systems. Researchers using field studies cannot manipulate independent variables or control the influence of confounding variables. For data gathering technique, field studies can employ either questionnaires, administered in person, by mail or email, or over the Web, or they can use interview transcripts, coded for quantitative analysis or they can use a variety of other techniques. Sometimes, researchers will refer to “multiple” case studies which when they exceed a dozen or more sites are more than likely classified as field studies.

Finally, *case studies* involve the intense examination of a small number of entities by the researcher, where no independent variables are manipulated or confounding variables controlled. Like field studies, case studies typically utilize questionnaires, coded interviews, or systematic observation as their preferred techniques for gathering data. Unlike filed studies, the foremost concern in case studies is to generate knowledge of the particular (Stake, 1995), from which analytic generalization is possible, rather than statistical generalization (Stake, 1995; Yin, 1994). By intensively studying a small number of entities, a case researcher is likely to develop deep insight of a phenomenon, from which hypothesis may be generated (Yin, 1994).

2.10.3 Qualitative Research

In explaining qualitative research, Denzin and Lincoln (1994) state that, qualitative implies an emphasis on processes and meanings that are not rigorously examined, measured (if measured at all), in terms of quantity, amount, intensity, or frequency. Thus, there are instances, particularly in the social sciences, where researchers are interested in insight, discovery, and interpretation rather than hypothesis testing (Merriam, 1988). One of the methods in qualitative research is content analysis.

Content analysis has been defined by Weber (1990) as a systematic, replicable technique for compressing many words of text into fewer content categories based on explicit rules of coding. Content analysis enables researchers to sift through large volumes of data with relative ease in a systematic fashion. It can be a useful technique for allowing us to discover and describe the focus of individual, group, institutional, or social attention (Weber, 1990). It also allows inferences to be made which can then be corroborated using other methods of data collection. Content analysis research is motivated by the search for techniques to infer from symbolic data what would be too costly, no longer possible, or too obtrusive by the use of other techniques".

2.11 Summary and Gap in Literature

In summary, literature has shown that while there are many studies on different aspects of the UML, there are relatively few for which the UML itself is the object of study (Budgen, Burn, Brereton, Kitchenham and Pretorius, 2011). It was also observed that not much has changed on UML Adoption in practice as the results of ICSE 2013 paper on UML in practice, is not different from those reported in 2006 by Dobing and Parson on How UML is Used (Cabot, 2013). This should not be the case and shows that more study is required to be conducted on UML adoption and to understand factors that may be slowing down its adoption.

In all the literatures reviewed, none followed the approach of evaluating developers knowledge of UML to facilitate its evolution and adaptation. This is very important because since UML has become an industry modeling standard and a lot of studies are going on in different aspects of UML, there is need to study its adoption and use as recommended by various researchers such as Budgen, Burn, Brereton, Kitchenham,   and Pretorius (2011)

CHAPTER THREE

SYSTEM ANALYSIS AND METHODOLOGY

3.1 Overview

Methodology has to do with methods. That is the methods or organizing principles underlying a particular art, science, or other area of study (Microsoft, 2008). Research in general has several methodologies from which a researcher can make choice(s) depending on the nature of the research. If the research will lead to a software development, there are also several research methodologies for software development process.

This chapter analysed the present and the proposed system, it also discussed the advantages and disadvantages of the proposed system and justification of the work. Finally, the chapter also discussed research methodologies with emphasis on software development methodologies. The choice of methodology for this work was made and the chosen methodology was employed in the analyses of the present and proposed system.

3.2 Analysis of the Present System

The existing system studied was the work by Petre (2013). The study which was on UML in Practice conducted an empirical study that involved series of interviews conducted over 2 years with more than 50 practicing professional software developers.

3.2.1 Sample Used in the Existing System

Informants in the study were identified and gathered from a broad range of perspectives; from corporate large-scale commercial software developers to independent consultants, and across a variety of application areas. Informants came primarily from countries in Europe and North America, but there were also informants from Brazil, India, and Japan, and many had worked in more than one country. Informants were identified opportunistically, via networks of collaborators, colleagues and contacts – people who could act as ‘brokers’ for introductions of various kinds: at meetings and conferences, via mailing lists, via social networks such as the Requirements Engineering Specialist Group (RESG) on LinkedIn and via personal emails. All informants were practicing professional software developers in roles ranging from requirements engineering, to software architecture, software development, and quality assurance (and most identified themselves as fulfilling more than one role). Finally, only one informant per company was included in the reported data, reducing the sample size to 50.

3.2.2 Methodology Used in the Existing System

The work employed simple, semi-structured interviews conducted over the phone, on Skype, or in face-to-face meetings, as convenient. The protocol was straightforward; the starting point was with background questions about the professional's experience, role, organizational context, and software projects. Next was the key question which was: 'Do you use UML?' Depending on the informant's response to this question, the second question was either: 'Can you tell me about how you use it?' or 'Why not?' Subsequent questions simply followed up responses and elicited examples of use of UML or other design representations. When appropriate, the informant was asked if his or her usage was typical of the organization. Hand-written/typed notes or audio records were captured for all interviews subject to the informant's preference. Some informants provided actual examples of design representations within confidentiality agreements. Discussions at times extended beyond the informants' current practice to past projects, past organizations, or other experiences. The discussion also distinguished between the UML use preferred by the informant and the use preferred or mandated by the organization. All accounts of UML use offered by the informants were collected, but a distinction was made in the data collected between the informants' own current use (identified as 'declared current use') and the ones identified as secondary reports which include accounts of their own practice in the past or in other organizational contexts, accounts of organizational preferred practices, or accounts of their colleagues' practices which they have observed directly. The work focused on responses that had to do with current practice but included where relevant, discussion on 'secondary reports'. The analysis was inductive, allowing categories of use to emerge from the data. The initial categories was into 'use' and 'non- use'. Additional categories were identified in the use group in terms of what the informants presented as characteristic of their use. These categories were presented along with the results in the section 3.2.3.

3.2.3 The Result

In the overall results as given by Petre(2013), five categories of UML use were identified. The list following shows these five categories. The numbers in parentheses indicate the number of informants whose declared current usage fits within that category. The results were summarised in table 2.1.

1. No UML (35/50); Don't use UML.

2. Retrofit (1/50): Don’t really use UML but retrofit UML in order to satisfy management or comply with customer requirements.

3. Automated code generation (3/50): Don't use UML in design but use it to capture the design when it stabilizes in order to generate code automatically.

4. Selective (11/50): Use UML in design in a personal, selective and informal way and for as long as it is considered useful after which it is discarded.

5. Wholehearted (0/50) – Use UML wholeheartedly with organizational, top-down introduction of UML and investment in professionals, tools and culture change so that UML use is deeply embedded.(This though zero for the informants, is described in secondary reports).

TABLE 2.1: ***Declared Current UML Use among 50 Professional Software Developers From 50 Companies. Source: Petre (2013)***

Category of UML Use	**Instances of Declared Current Use**
No UML	35
Retrofit	1
Automated code generation	3
Selective	11
wholehearted	0

In summary, his result showed that out of the 50 professional software developers from different companies, 35 of them do not use UML. This represents 70% of the informants. For the remaining 30%, none uses it wholeheartedly.

3.2.4 Advantages of the Present System

The main advantages of the present system are:

1. The system added to the few literature available on the study of how UML are actually being used in Practice.
2. The sample used covered wide range of professional software developers.

3.2.5 Disadvantages of the Present System

The present system however possesses the following disadvantages:

1. The study used only one key question which was: ‘Do you use UML?’ and depending on the response, the second question was either: ‘Can you tell me how you use it?’ or ‘Why not?’
2. The study was based on oral interview and the documentation of answers was not done by the informant, this gives room for misrepresentation of the informants response.
3. The study was not in dept on the study of UML use.
4. There is also a possibility of Informant(s) supplying wrong responses to the interview questions since no structure exists to find out right and wrong responses.
5. The system does not provide an evaluation system that can be trusted.

3.2.6 Data Flow Diagram of the Existing System

The data flow diagram of the existing system is shown in figure. 3.1

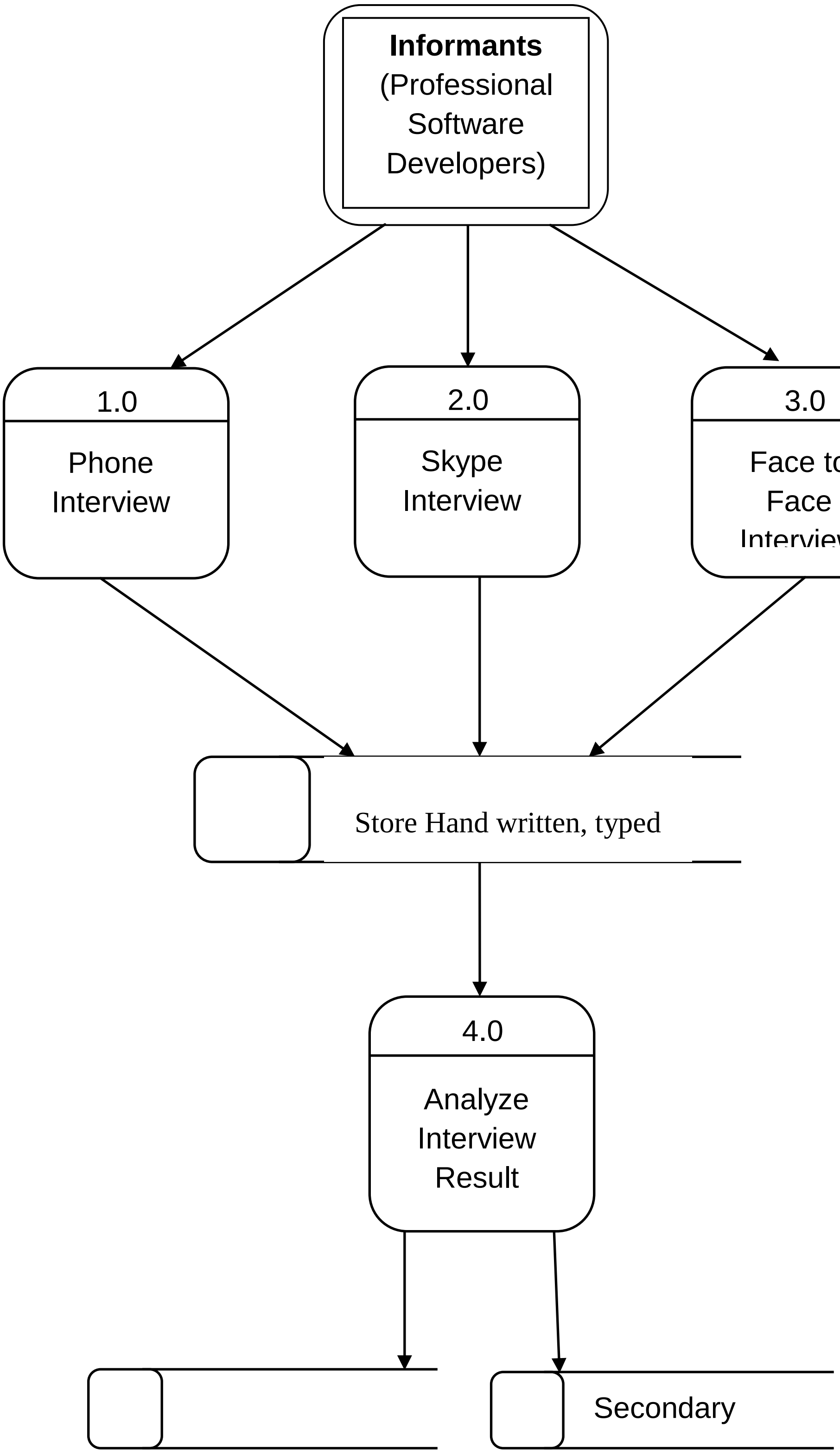

Informants
(Professional Software Developers)
1.0
Phone Interview
2.0
Skype Interview
3.0
Face to Face Interview
Store Hand written, typed
4.0
Analyze Interview Result
Secondary

Figure. 3.1 Data Flow Diagram of the Existing System

3.3 Analysis of the Proposed System

In analysing the proposed system, the functional requirements of the proposed system were first identified from the objectives.

3.3.1 Major Functional Activities Identified in this Requirement

Five major functional activities identified in this requirement are:

1. **UML Survey:** A preliminary survey used in determining the evaluation criteria.
2. **Evaluation Questions:** Responsible for creating and presenting evaluation questions.
3. **Evaluation Result:** Responsible for collating evaluation results from different criteria group results.
4. **Result Model:** Responsible for modeling UML content against developers level knowledge from the evaluation result.
5. **Model Reports:** Responsible for providing Stakeholders necessary information from the result models that will enhance decision making.

3.3.2 Use Case Diagram

Having established the scope of our system, we continue our analysis by studying several scenarios of its use. As mentioned earlier, the use case model captures the requirements of a system. Use cases are a means of communicating with users and other stakeholders what the system is intended to do. We begin by enumerating a number of primary use cases, as viewed from the various functional elements of the system.

1. The software developer fills login information
2. The system administrator authenticates the login information and grants access if correct.
3. The system administrator presents evaluation questions
4. The software developer responds to the evaluation questions
5. The system collates result
6. The system models result

We now start with initial use-case diagrams of figure 3.2. The actor here is a software developer and the actions are filling login information, editing login information, submitting login information, answering and submitting evaluation questions.

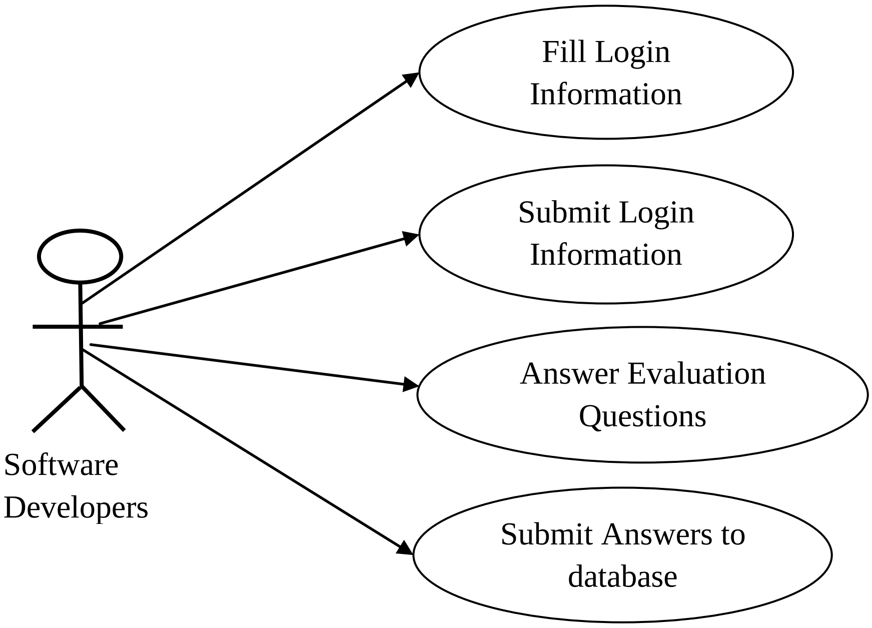


Figure 3.2 Initial Use Case Diagram for Software Developers

Next is the initial use case for system administrator. Here the actor is the system administrator and the actions are validating and updating the data supplied to the IT professional in the questionnaire, analysing the data to evaluate UML usage at specified intervals.

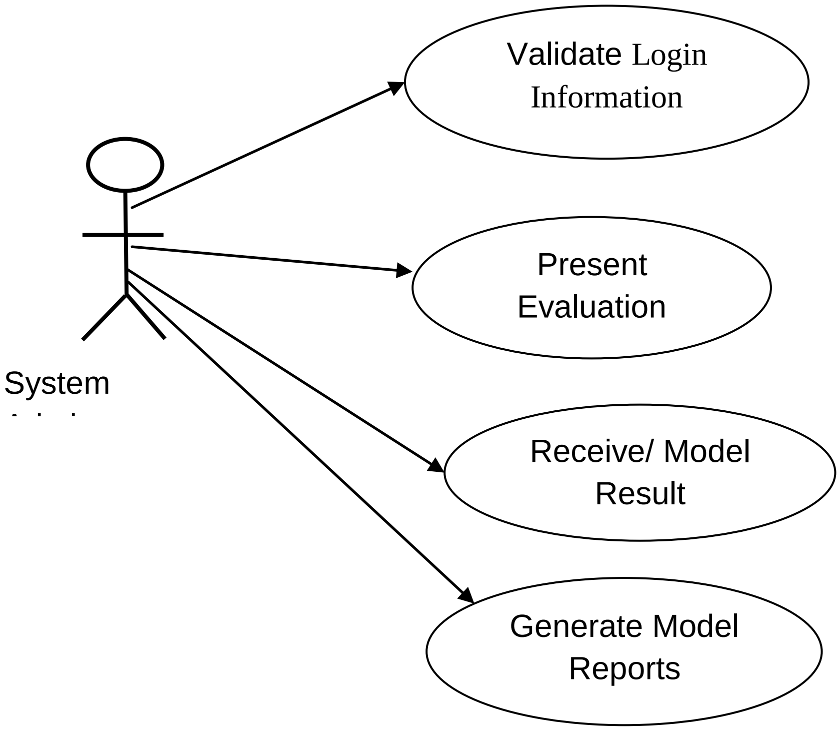


Figure 3.3 Initial Use Case Diagram for System Administrator

Having established the initial use cases for the different actors, the resulting system use case diagram is shown in figure 3.4.

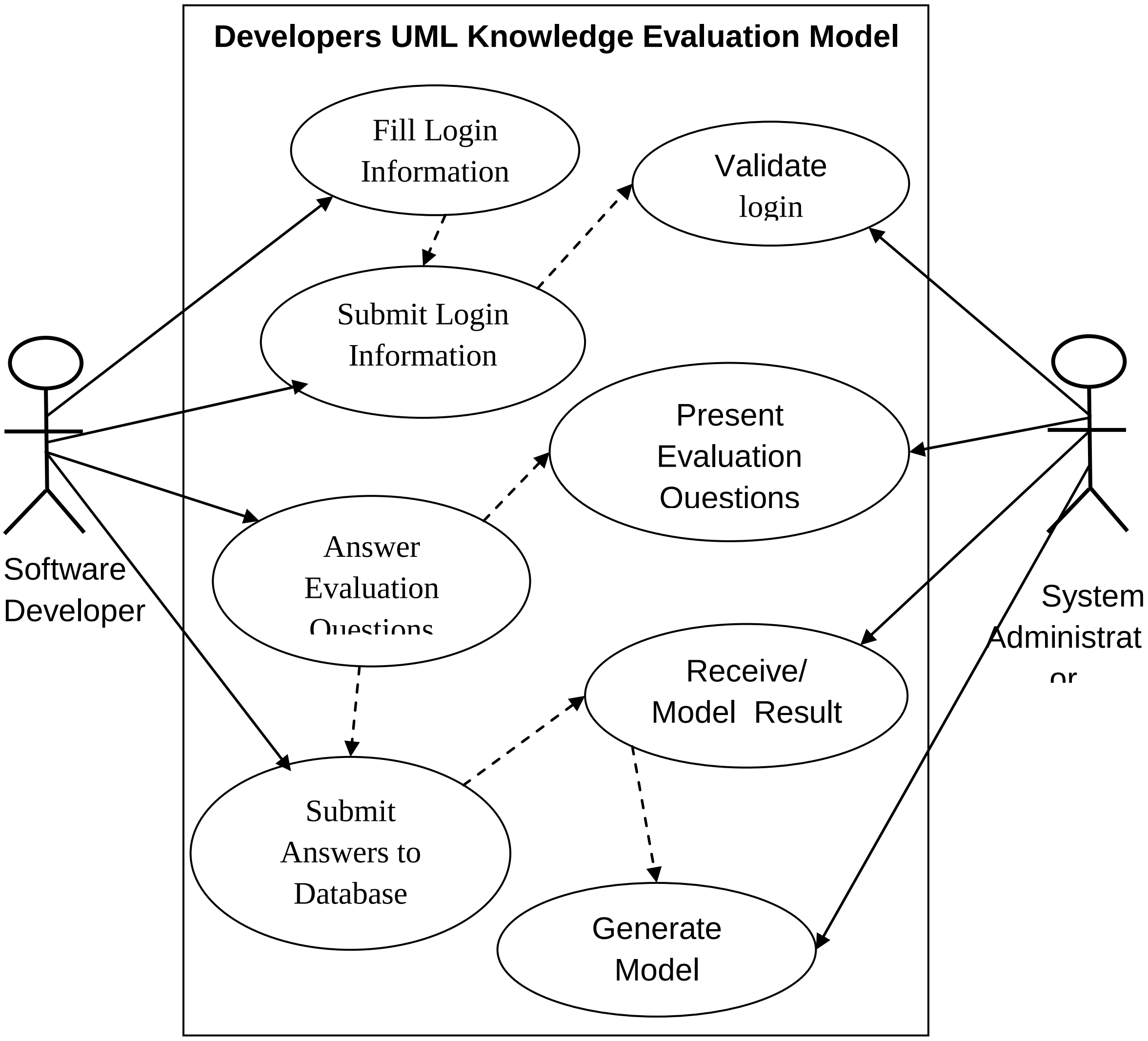


Figure 3.4 System Use Case Diagram for Developers UML Knowledge Evaluation Model (DUKEM)

3.3.3 Analysis Model

The analysis model analyzes the system specifications in the use case model. Hence from the use case model, we shall now try to identify the main classes necessary for the system to perform the different actions. In this analysis model, the interfaces, processes and databases are distinguished using different symbols as shown in figure 3.5.

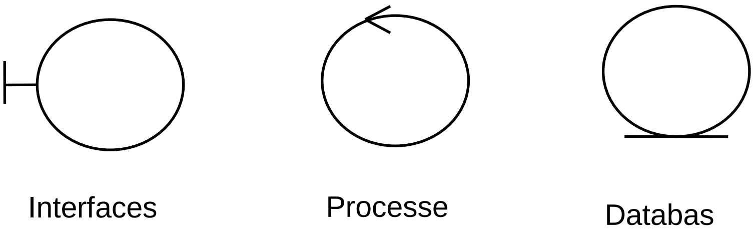


Figure 3.5 Symbols of Interface, Process and Database

Now, for the evaluation questions use cases, the main classes necessary for the system to perform the actions in the evaluation questions use cases were identified. The analysis model of these classes is shown in figure 3.6.

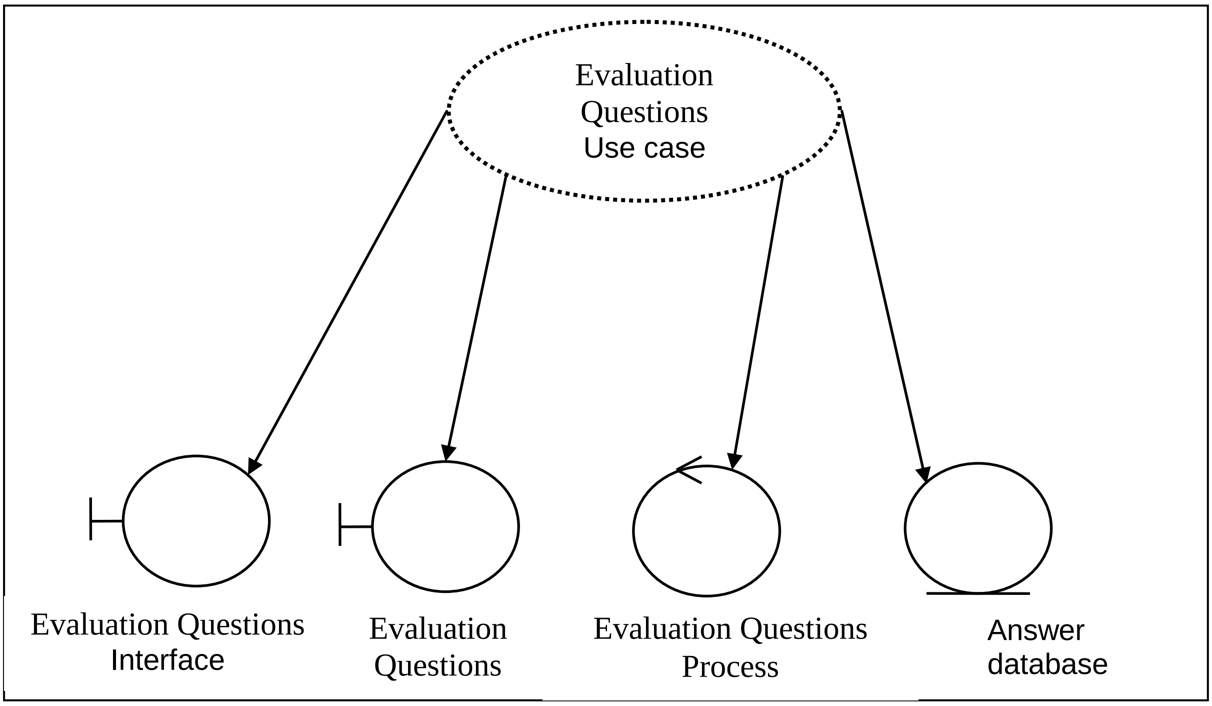

Evaluation Questions Use case
Evaluation Questions Interface
Evaluation Questions
Evaluation Questions Process
Answer database

Figure 3.6 Classes Identified for the Evaluation Questions Use Case

Now that the analysis classes for the evaluation questions use case have been identified, we can show how they interact with each other and with the actors with Sequence diagram. This is shown in figure 3.7.

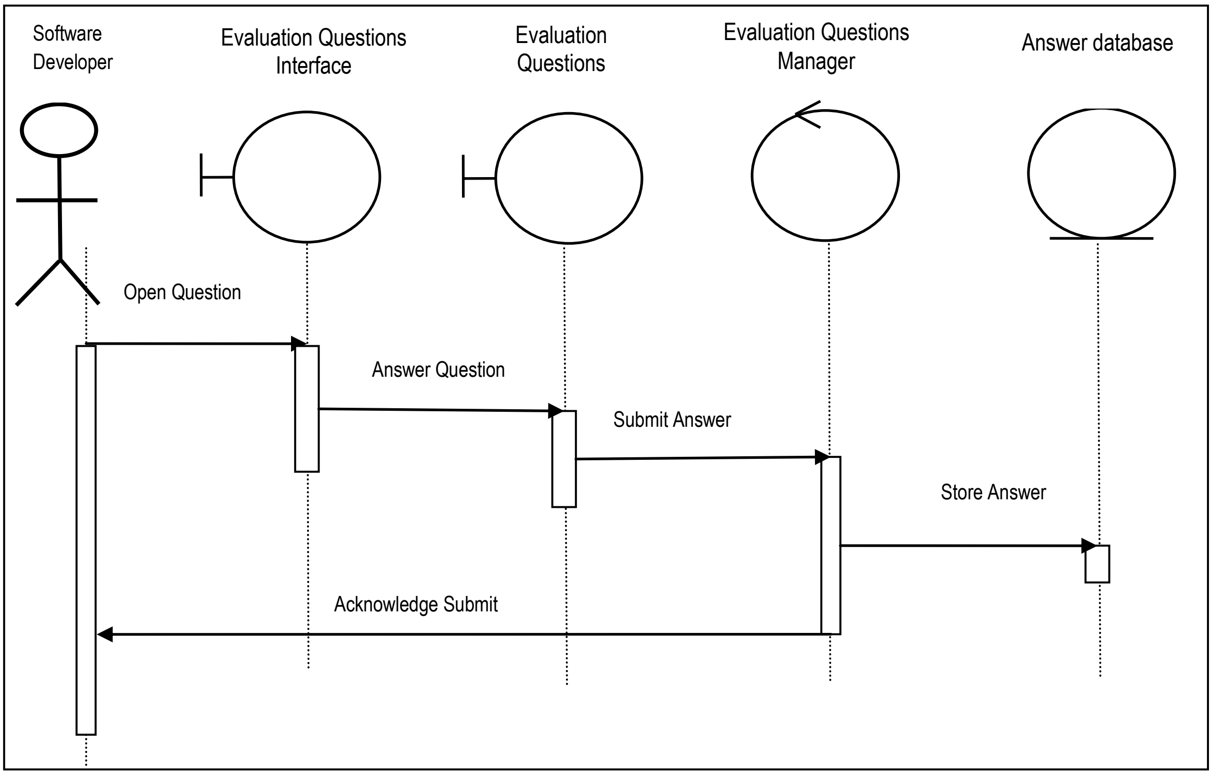


Figure 3.7 Sequence Diagram for Evaluation Questions Classes

Next, considering the evaluation model use cases, the analysis model of the main classes identified as necessary for the system to perform it is shown in figure 3.8

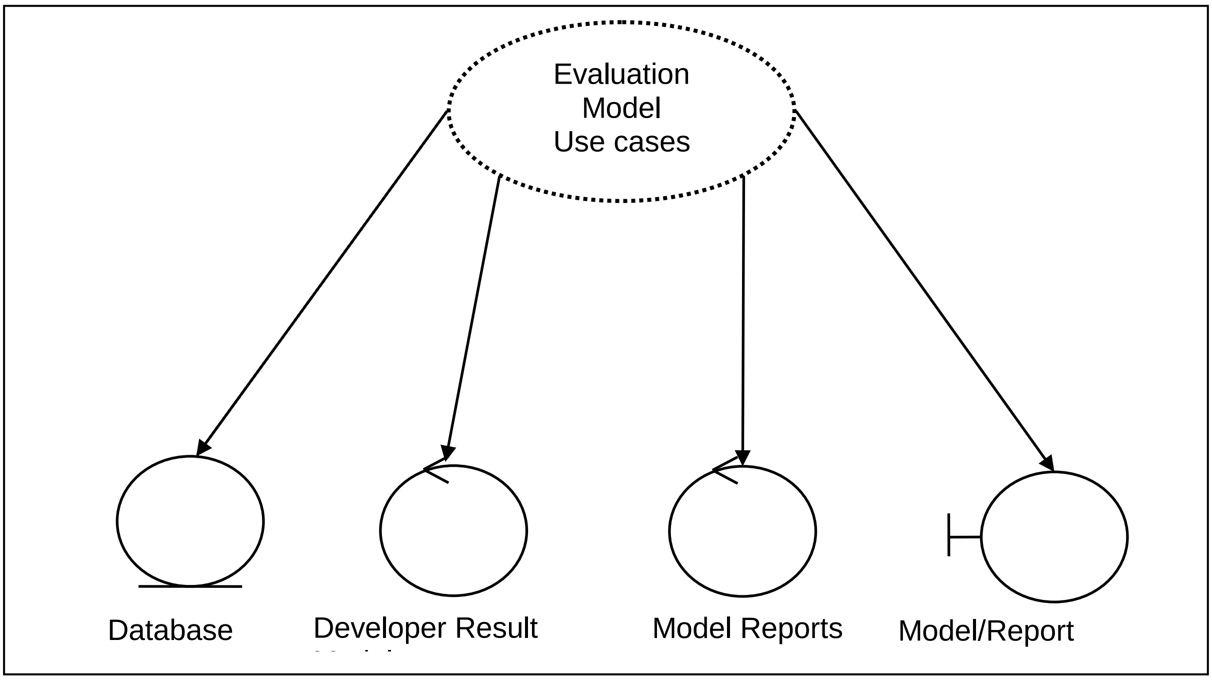


Figure 3.8 Classes Identified for the Evaluation Model/Reports Use Case

Now that the analysis classes for the fill questionnaire use case have been identified, we can show how they interact with each other and with the actors with sequence diagrams as shown i n figure 3.9.

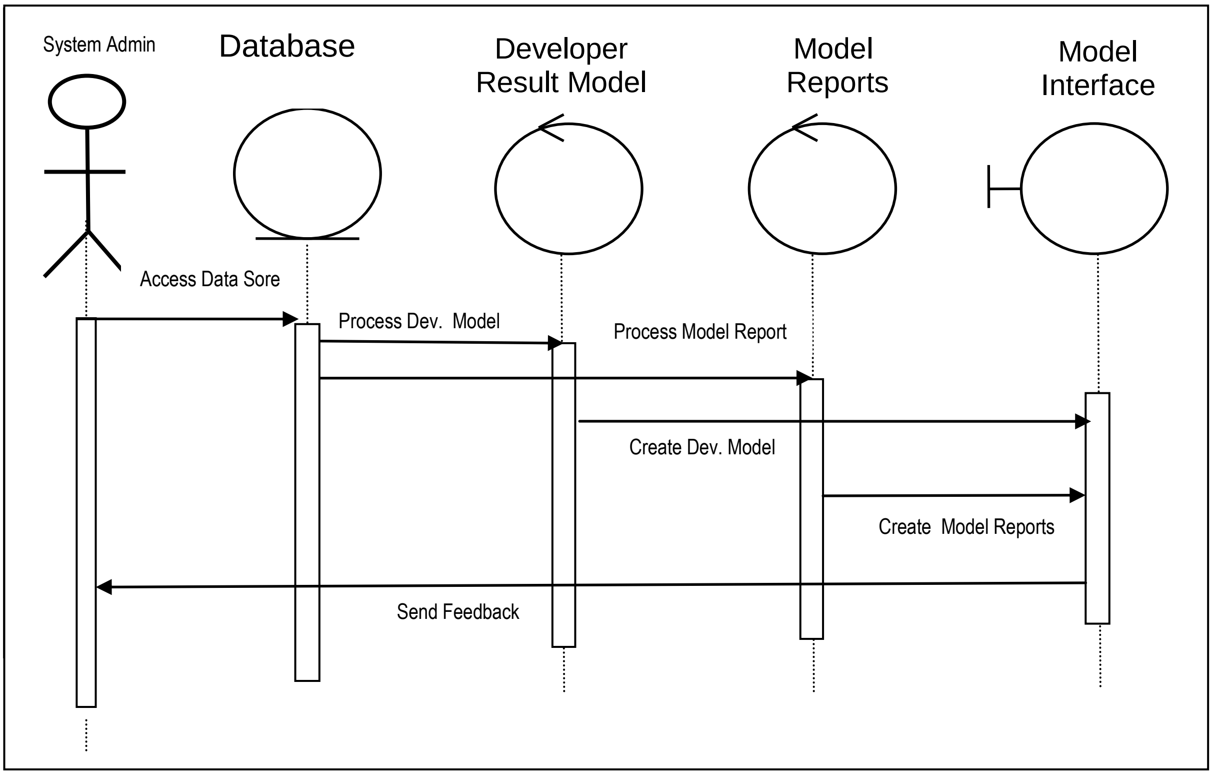


Figure 3.9 Sequence diagram for Evaluation Models/Reports Classes

3.4 Choice of Methodology for this Work

Because this work involves a research coupled with software development, the choice of methodology involves both the choice of research methodology and the choice of software development methodology.

3.4.1 Choice of Research Methodology

The research methodology chosen for this work is survey research. Survey is also an appropriate methodology for this work because survey research is most appropriate when the central questions of interest about the phenomena are "what is happening?", and "how and why is it happening?"

3.4.2 Choice of Software Development methodology

The choice of methodology for software development in this work is object oriented methodology using the Unified Process. This is because the unified process is a recent software development framework for object oriented designs and is purely UML based. Some of its key features that made it ideal for this work are:

1. **It is component based:** It is component based and is being commonly used to coordinate object oriented programming projects.
2. **It uses UML** – This is a diagrammatic notation for object oriented design - for all blueprints.
3. **It is user-centric:** The design process is anchored, and driven by, use-cases which help keep sight of the anticipated behaviours of the system. Analysts specify functionality with use cases, customers confirm use cases, designers and implementers realise use cases and testers verify the system with use cases.
4. **It is architecture centric**. This means that the system is partitioned into subsystems. Logical and physical views of the system are separated.
5. **Design is iterative:** Instead of trying to define all the details of the model at once, several passes are made and each iteration adds more details.
6. **It is also incremental** – The design system evolves through a set of increments. Each increment adds more functionality.

The iterations and increments are performed via a prescribed sequence of design phases within a cyclic process.

3.5 Phases of Design Cycles in the Unified Process

Design in the Unified Process proceeds through a series of cycles, each of which has the following phases:

1. **Inception:** The inception phase produces a commitment to go ahead. By the end of this phase a business case should have been made; feasibility of the project assessed; and the scope of the design should be known.
2. **Elaboration:** The Elaboration phase takes us to a working specification of the system. By the end of this phase a basic architecture should have been produced; a plan of construction agreed; all significant risks identified; and those risks considered to be major should have been addressed.
3. **Construction:** The construction phase produces beta-release of the system. By the end of this phase a working system should be available, sufficient for preliminary testing under realistic conditions.

4. **Transition:** The transition phase introduces the system to its intended users.

Within these phases we may go through a number of iterations, each involving the normal forms of workflow activity which are: requirements specification, analysis, design, implementation and testing.

A principal product of the Unified Process is a series of models, each appropriate to a key stage in system design. Since many different models are produced, each for a different design purpose but all related to the same system, we need some common point of anchorage. This is provided by a use case model. Figure 3.10 shows a typical arrangement, in which five models appropriate to specific design activities all are rooted in the same use case model.

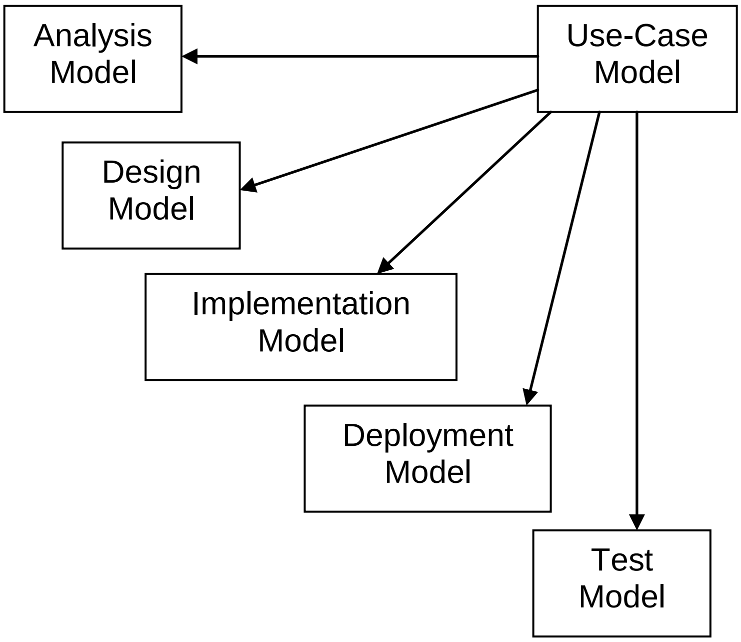


Figure 3.10 Series of Models in the Unified Process

The purpose of a use case is to describe the functionality required of the system from the point of view of those concerned with its operation. The way a use case does this is by specifying a sequence of actions, including variants, that the system can perform and that yield an observable result of value to some actor. In the Unified Process, this drives requirements capture, analysis and design of how system realises use cases, acceptance/system testing, planning of development tasks and traceability of design decisions back to use cases.

3.6 High Level Model of the New System

Figure 3.11 shows the high level block diagram model of the new system.

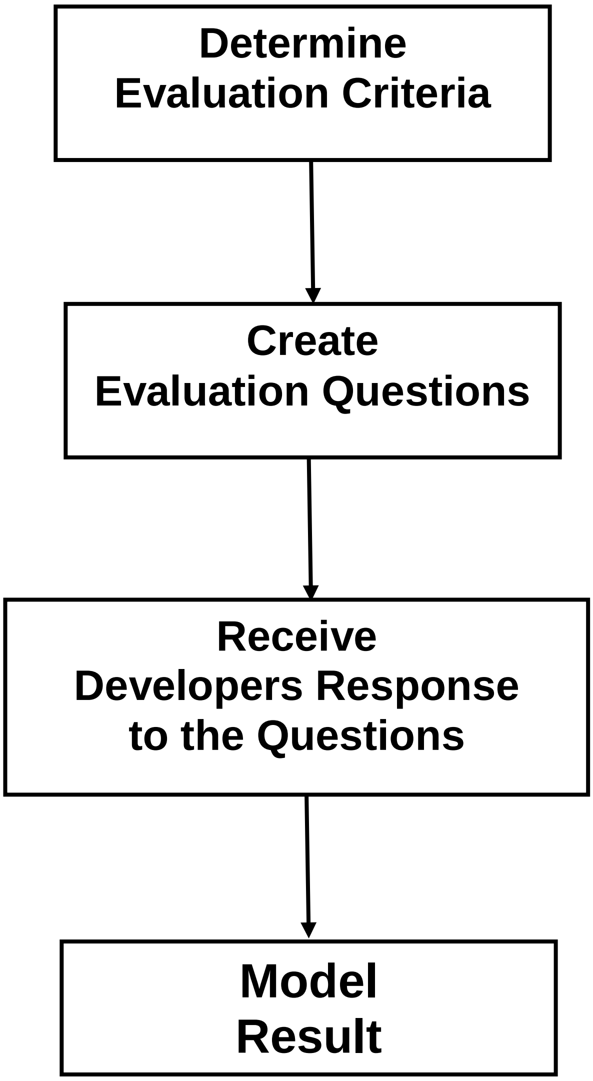

Determine Evaluation Criteria
Create Evaluation Questions
Receive Developers Response to the Questions
Model Result

Figure 3.11 High Level Model of the New System

3.7 Analysis and Methodology Applied in the Survey

To implement the survey, We also followed as much as possible the suggestions given in Kitchenham and Pfleeger (2008) and adopted the use of questionnaire to collect information.

The target population is the set of individuals to whom the survey applies. In this case, the population consisted of Nigerian IT Students and professionals. The IT students in this context include students of higher and post graduate studies in computer science, computer engineering and other related fields while the professionals include practicing professionals in these areas both in industry and academia who are involved in software development. The sample was obtained in two ways: (1) by convenience, i.e., relying on the network contacts of our research group and (2) by sending invitation messages through emails and professional groups.

In all, 158 completed responses was received from the survey. Unfortunately, it is not known exactly how many people have been reached by the invitation messages and advertisements, and therefore could not calculate the response rate. The same problem has been reported in other software engineering surveys as noted in Lethbridge (2008).

3.7.1 Data Collection and Validity

Data collection were designed to be in two ways. 1) through email and personal distribution and through the creation of an on line questionnaire. The online questionnaire had been developed and published using ASP.Net and the result automatically collected in a database. A pilot study was performed before the first execution of the survey to tune the questionnaire and to reduce the ambiguities contained in the questions. Two IT professionals carefully read all the documentation and provided their judgment on the questionnaire. Following the suggestions of the two contacted professionals, minor changes to the questionnaire were made. From the result of the pilot study we concluded that the survey was well suited for IT professionals and that the questions were clear enough.

CHAPTER FOUR
SYSTEM DESIGN AND IMPLEMENTATION

4.1 Overview

System design is the process of designing the systems components. The new system is designed with an object-oriented approach. This involves the use of many levels of abstraction to decompose the problem into manageable components, identify classes and interfaces and establish relationships among the classes and interfaces (Liang, 2001). By applying object-oriented design, we create software that is resilient to change and written with economy of expression. We achieve a greater level of confidence in the correctness of our software through an intelligent separation of its state space. Ultimately, we reduce the risks that are inherent in developing complex software systems and control complexity (Balin, 2016).

Object-oriented design is a method of design encompassing the process of object-oriented decomposition and a notation for depicting logical and physical as well as static and dynamic models of the system under design. There are two important parts to this definition: object-oriented design

1. Leads to an object oriented decomposition
2. Uses different notations to express different models of the logical (class and object structure) and physical (module and process architecture) design of a system, in addition to the static and dynamic aspects of the system.

High level views of the Inception and Elaboration phases are shown in figure 4.1.

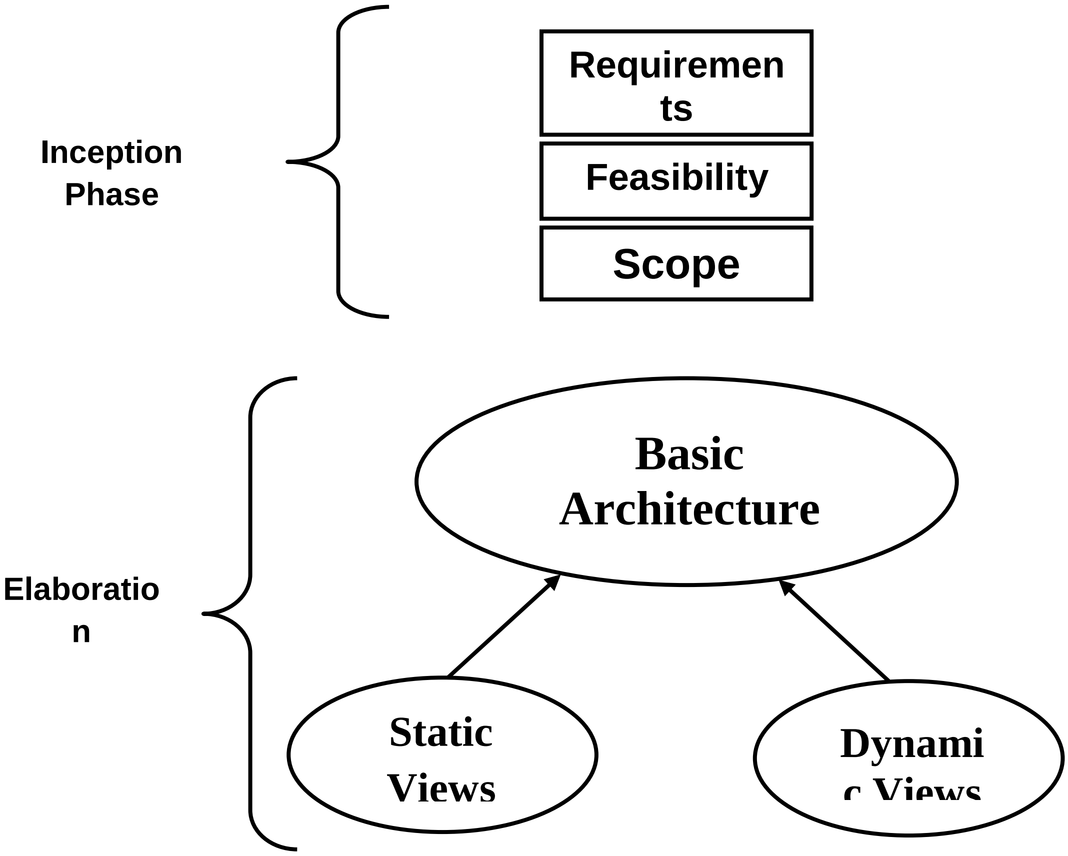


Figure 4.1 High Level Views of Inception and Elaboration Phases

Having gone through the inception stage in the previous chapters, this chapter moves more into the elaboration stage. By the end of this phase therefore, a basic architecture should have been produced along with a plan for construction.

4.2 Objective of the Design

The objective of the design is to integrate the different classes identified in the analysis model using object oriented technology based on unified process.

4.3 Design Models for the System

The design models for the system is shown in the following sections.

4.3.1 Detailed Use Case Description Package

Based on the Use Case diagram for the System given in Figure 3.5, a more detailed Use case description realised at the elaboration phase is in figure 4.2.

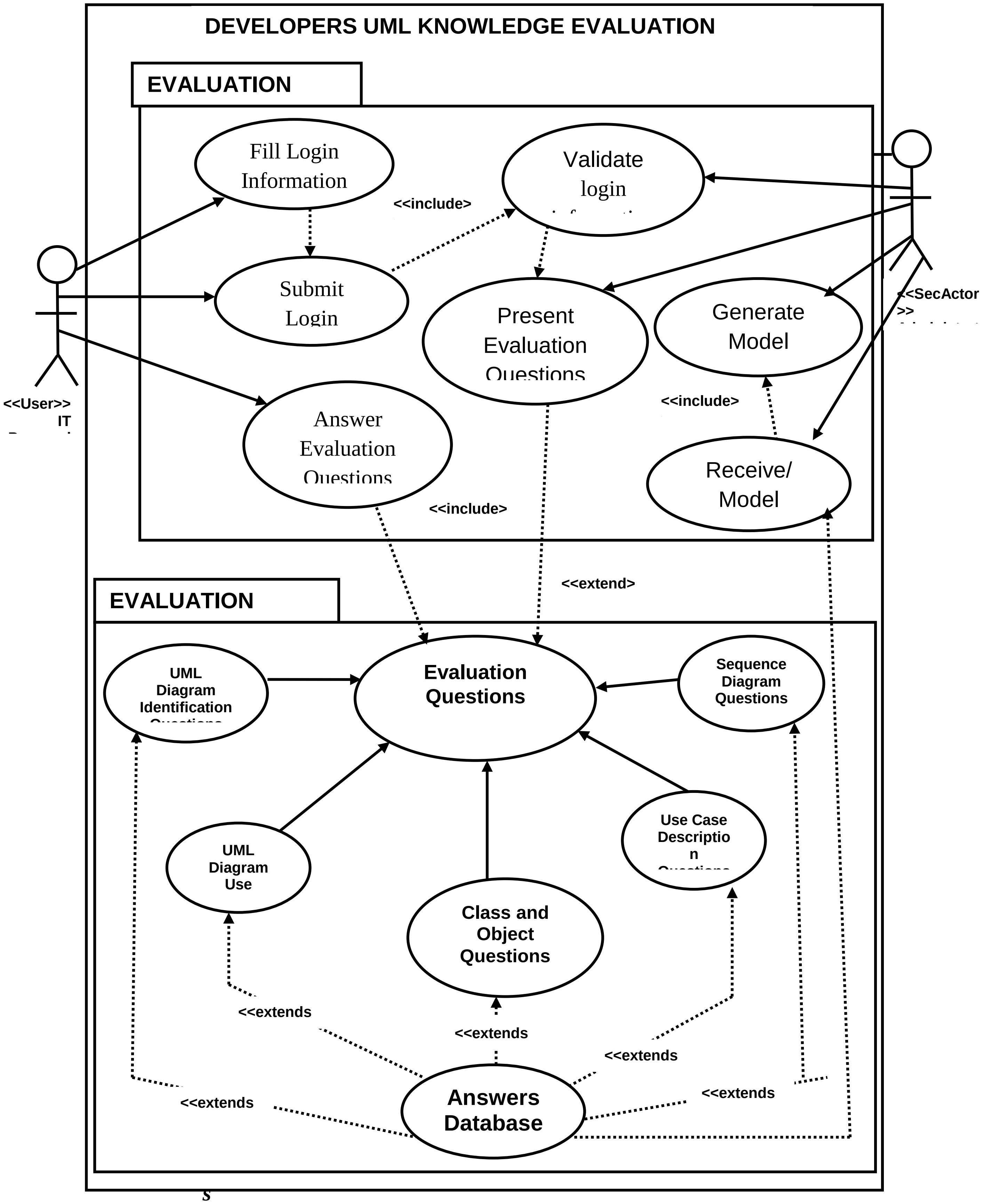
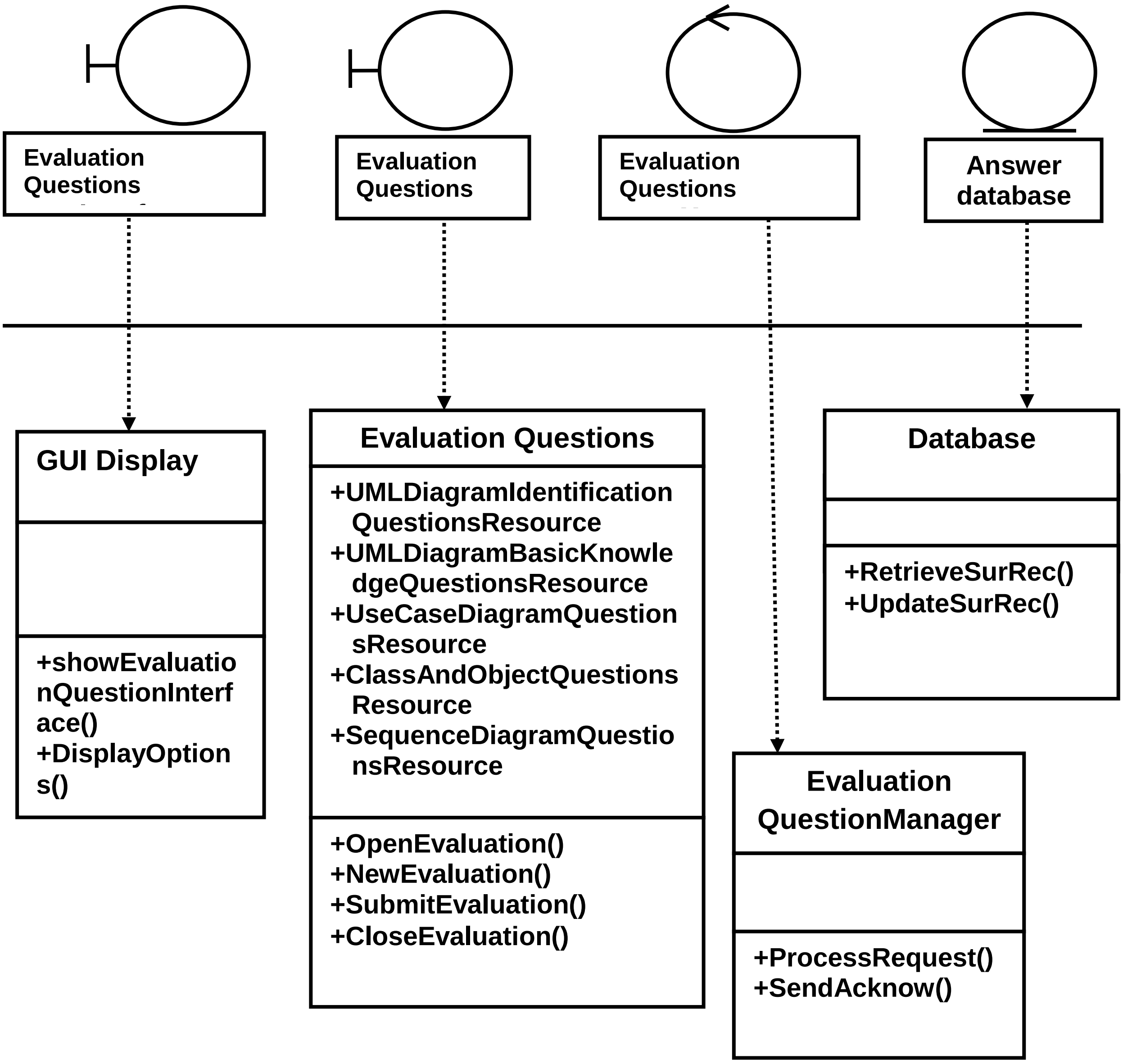


Figure 4.2 Detailed Use Case Description Package

4.3.2 Design Classes for the Evaluation Questions Use Cases

Figure 4.3 shows the design model classes introduced for the Evaluation Questions Use Case analysis model of figure 3.6. The relationships between these design classes were established in figure 4.4 that followed.

ANALYSIS MODEL



DESIGN MODEL

Figure 4.3 Design Model Classes Introduced for Evaluation Questions Use Case Analysis Model

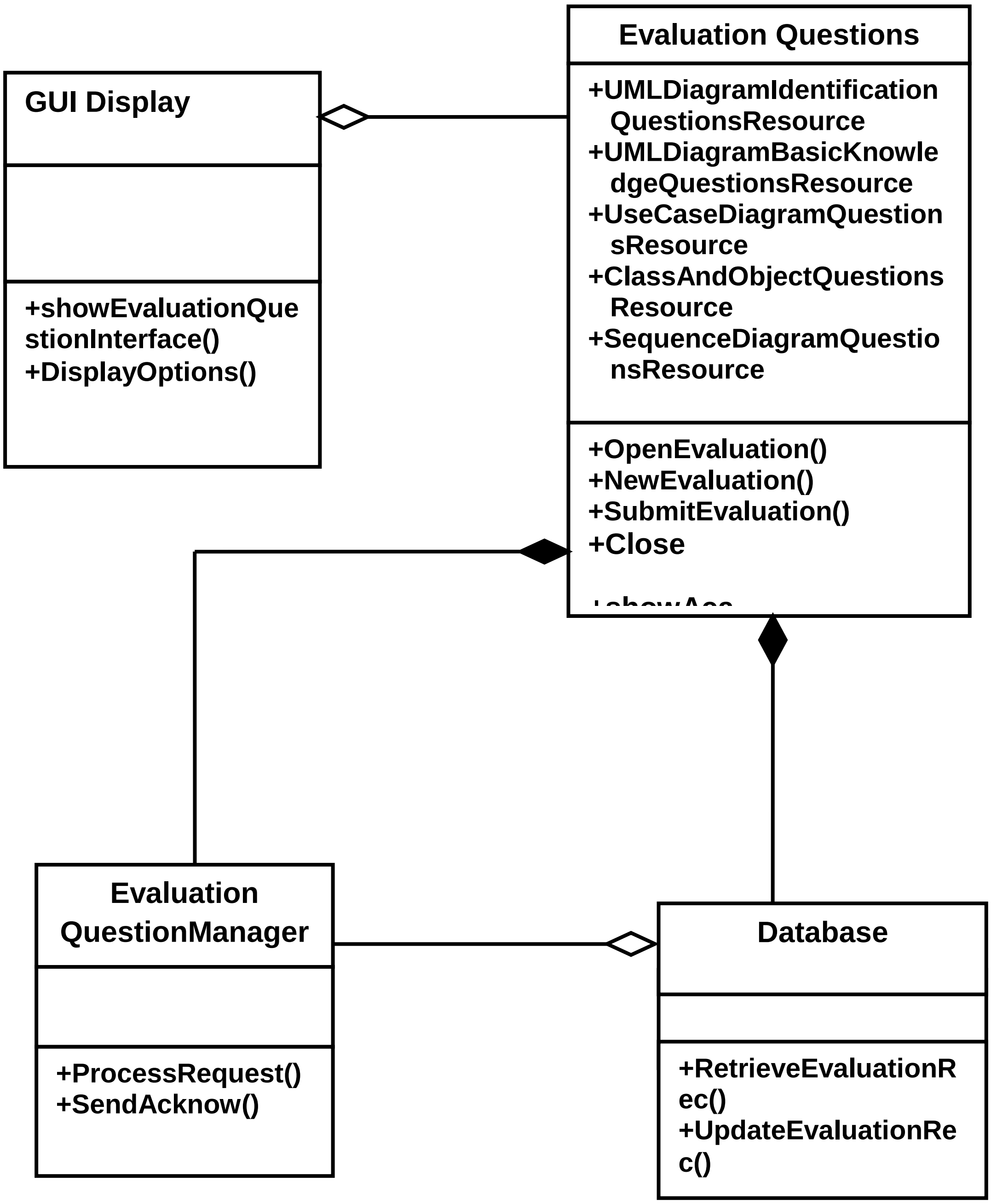
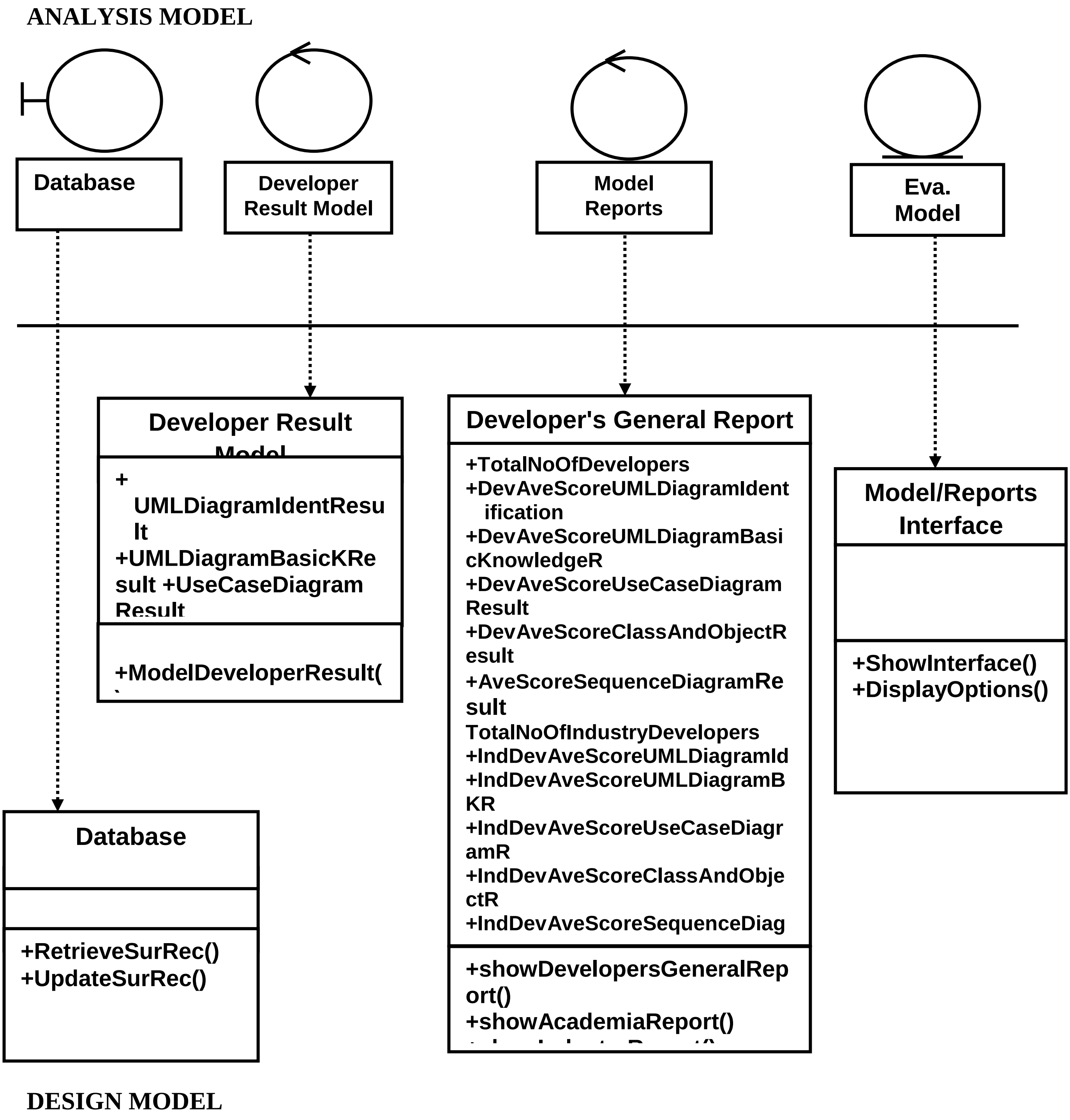

Evaluation Questions
+UMLDiagramIdentification QuestionsResource
+UMLDiagramBasicKnowledgeQuestionsResource
+UseCaseDiagramQuestionsResource
+ClassAndObjectQuestions Resource
+SequenceDiagramQuestionsResource
+OpenEvaluation()
+NewEvaluation()
+SubmitEvaluation()
+Close
+showAcc
GUI Display
+showEvaluationQuestionInterface()
+DisplayOptions()
Evaluation QuestionManager
+ProcessRequest()
+SendAcknow()
Database
+RetrieveEvaluationRec()
+UpdateEvaluationRec()

Figure 4.4 Relationships between the Evaluation Questions Design Model Classes

4.3.3 Design Model Classes for Evaluation Model Use Cases

Figure 4.5 shows the design model classes introduced for the Evaluation Model Use Case analysis model of figure 3.8. The relationships between these design classes were established in figure 4.6 that followed.



ANALYSIS MODEL
Database
Developer Result Model
Model Reports
Eva. Model
Developer Result Model
+ UMLDiagramIdentResult
+UMLDiagramBasicKResult +UseCaseDiagram Result
+ModelDeveloperResult(
Developer's General Report
+TotalNoOfDevelopers
+DevAveScoreUMLDiagramIdentification
+DevAveScoreUMLDiagramBasicKnowledgeR
+DevAveScoreUseCaseDiagram Result
+DevAveScoreClassAndObjectResult
+AveScoreSequenceDiagramResult
TotalNoOfIndustryDevelopers
+IndDevAveScoreUMLDiagramId
+IndDevAveScoreUMLDiagramBKR
+IndDevAveScoreUseCaseDiagramR
+IndDevAveScoreClassAndObjectR
+IndDevAveScoreSequenceDiag
+showDevelopersGeneralReport()
+showAcademiaReport()
Model/Reports Interface
+ShowInterface()
+DisplayOptions()
Database
+RetrieveSurRec()
+UpdateSurRec()
DESIGN MODEL

Figure 4.5 Design Model Classes introduced for the Evaluation Model Analysis

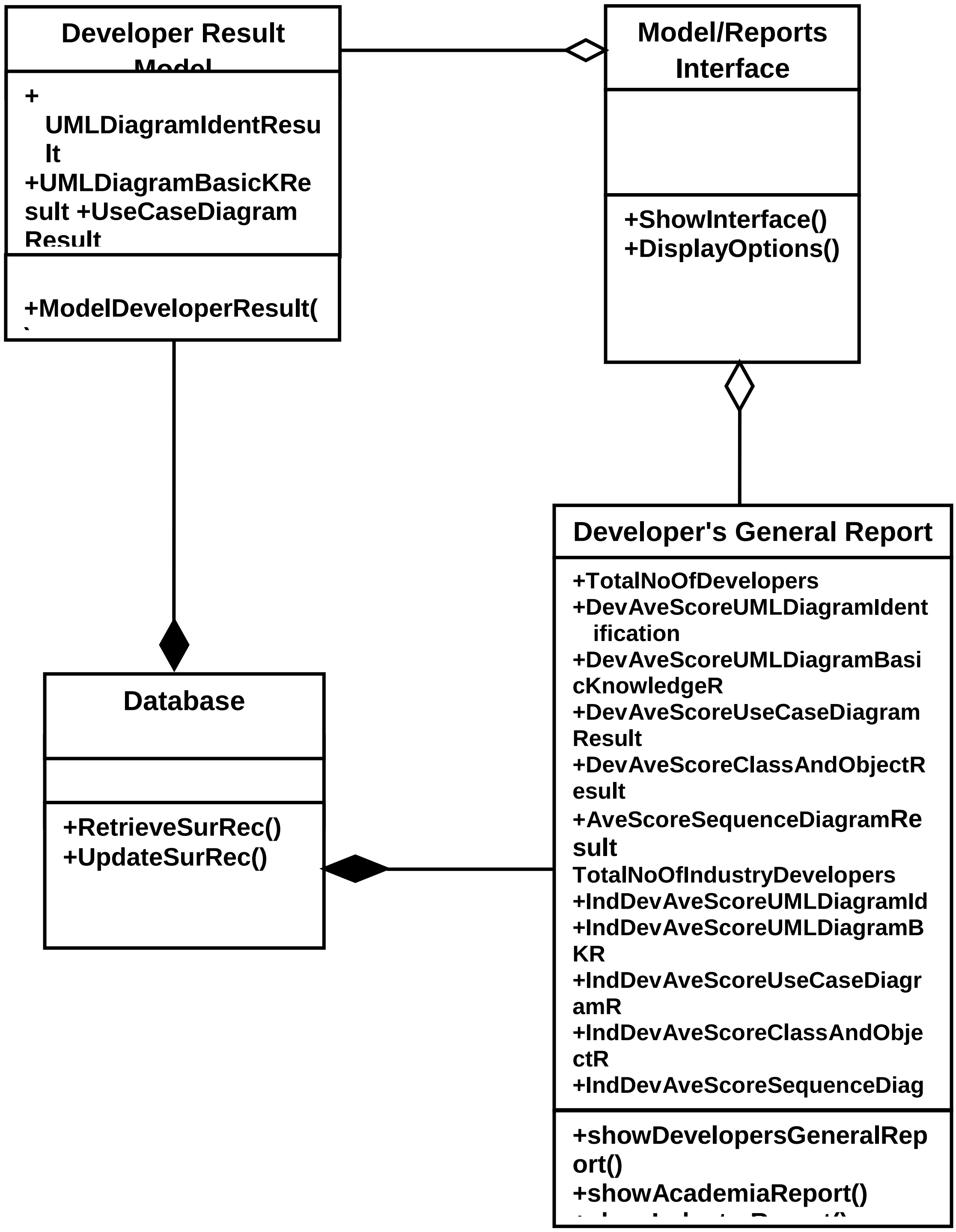

Developer Result Model
+ UMLDiagramIdentResult
+UMLDiagramBasicKResult +UseCaseDiagram Result
+ModelDeveloperResult(
Model/Reports Interface
+ShowInterface()
+DisplayOptions()
Database
+RetrieveSurRec()
+UpdateSurRec()
Developer's General Report
+TotalNoOfDevelopers
+DevAveScoreUMLDiagramIdentification
+DevAveScoreUMLDiagramBasicKnowledgeR
+DevAveScoreUseCaseDiagram Result
+DevAveScoreClassAndObjectResult
+AveScoreSequenceDiagramResult
TotalNoOfIndustryDevelopers
+IndDevAveScoreUMLDiagramId
+IndDevAveScoreUMLDiagramBKR
+IndDevAveScoreUseCaseDiagramR
+IndDevAveScoreClassAndObjectR
+IndDevAveScoreSequenceDiag
+showDevelopersGeneralReport()
+showAcademiaReport()

Figure 4.6 Relationships between the Evaluation Model Classes

4.5 The Control Center

The new systems control center is shown in figure 4.7.

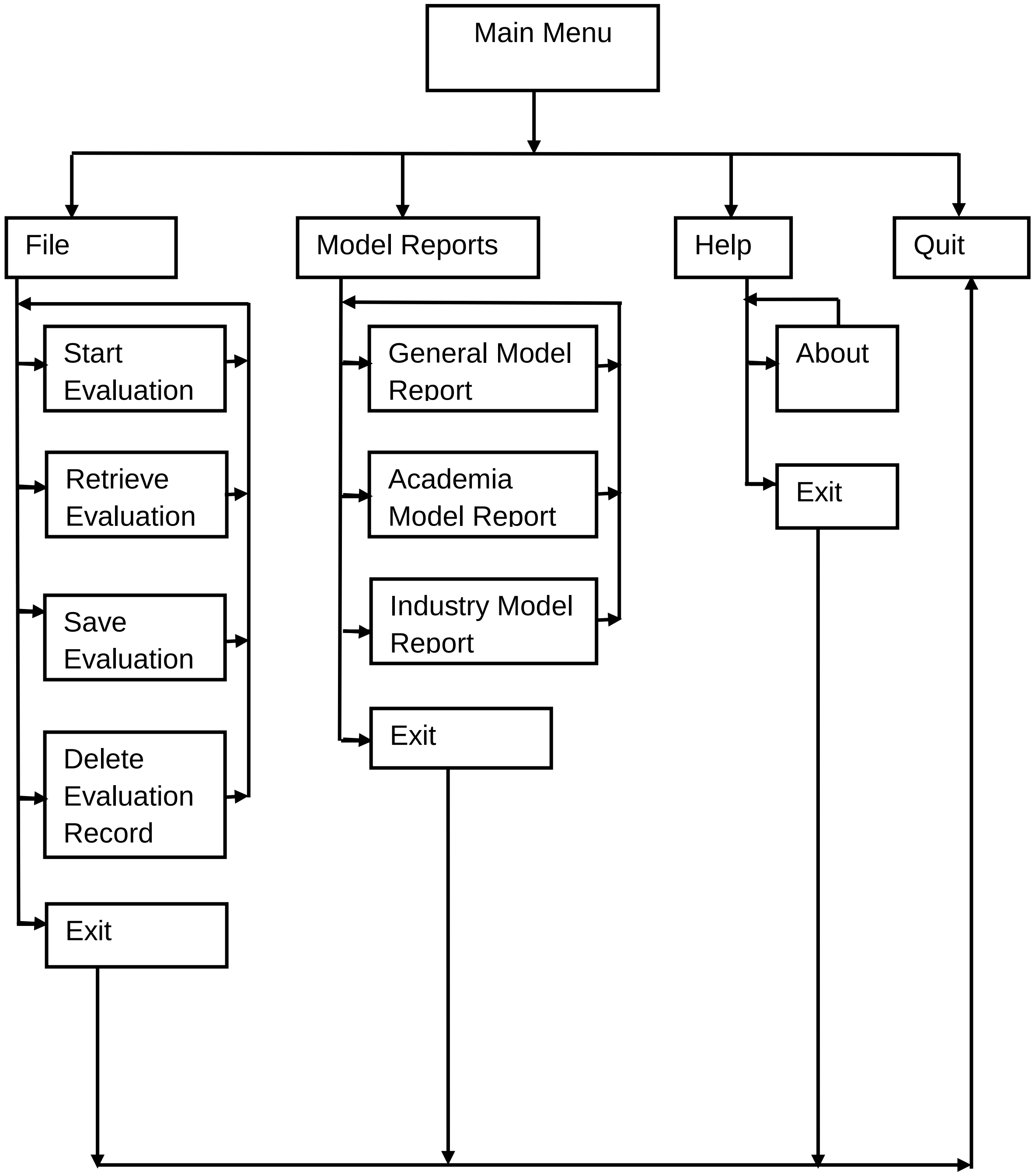


Figure 4.7 The Control Centre

File Submenu: The File Submenu controls record handling. It is responsible for entering new Evaluation records, saving records, removing and modifying existing records. Clicking on this menu option takes the user to the five submenu items shown in figure 4.8.

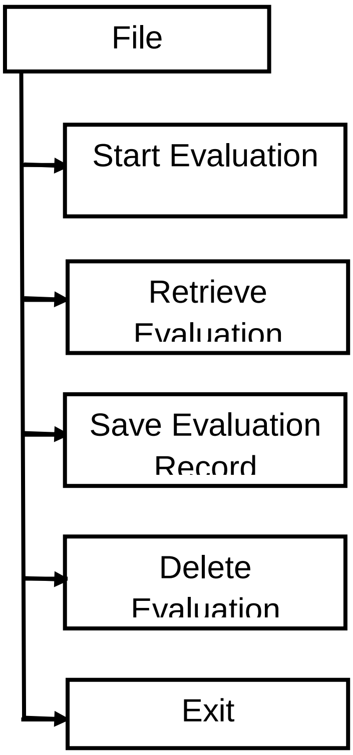


Figure 4.8 The File Submenu

The Models Reports Submenu: This menu activates the classes for generating the different reports expected from the system as shown in figure 4.9.

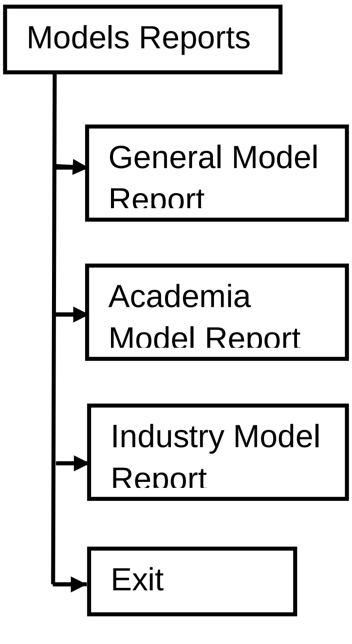


Figure 4.10 The Model Reports Submenu

The help Submenu: The menu provides the general knowledge about the application. From the About option in the Help Submenu, the users can get general information about the software and the importance of UML Knowledge.

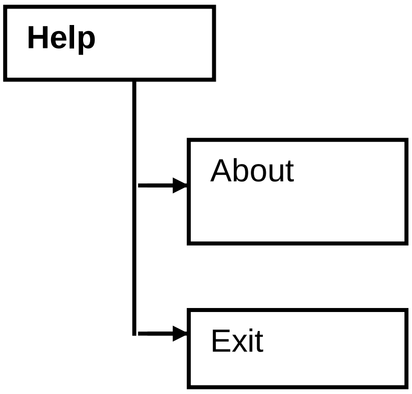


Figure 4.10 The Help Submenu

4.6 Input and Output Specification

The input and output specifications for the system are discussed in the following sections.

4.6.1 Output Format Specification

4.6.1.1 Output Format Specification for the Developers Result Model

The main output from the system are the models created by plotting a graph of UML content against level of the knowledge of it possessed by the software developers. The format is shown in figure 4.11.

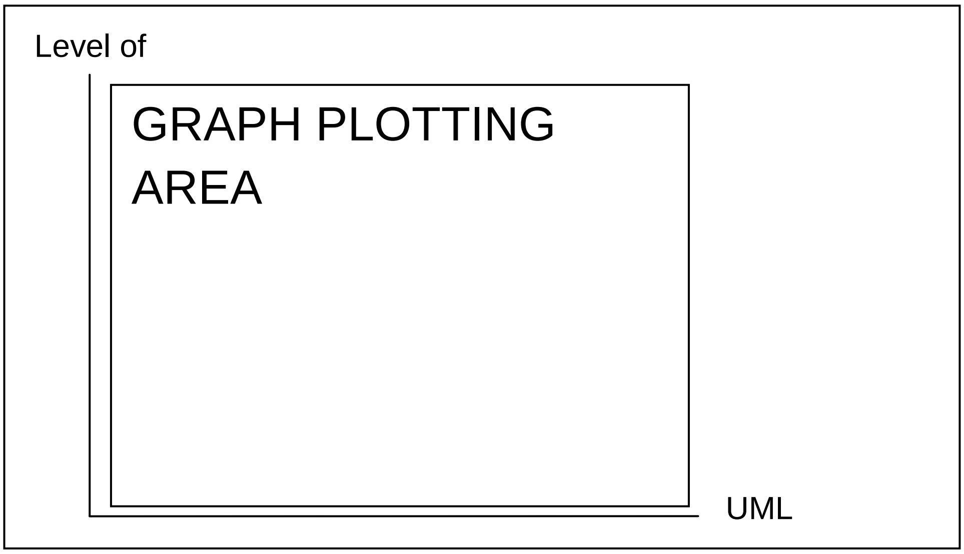


Figure 4.11 Evaluation Model Output Format Specification

4.6.1.2 Output Format Specification for the Model Reports

Another output from the system are the various summary reports of the developers evaluation results. These reports are: General Summary Reports, Academia Developers Summary Reports and Industry Summary Reports. These formats are shown in figure 4.12a, 4.12b and 4.12c.

General Summary Reports

Total Number of Developers Evaluated:
Average Score in UML Diagram Identification:
Average Score in General UML Diagram Basic Knowledge:
Average Score in UML Use Case Diagram Knowledge:
Average Score in UML Class and Object Diagram Knowledge:
Average Score in UML Sequence Diagram Knowledge:

Figure 4.12a General Summary Report Output Format Specification

Academia Developers Summary Reports

Total Number of Developers Evaluated:
Average Score in UML Diagram Identification:
Average Score in General UML Diagram Basic Knowledge:
Average Score in UML Use Case Diagram Knowledge:
Average Score in UML Class and Object Diagram Knowledge:
Average Score in UML Sequence Diagram Knowledge:

Figure 4.12b Academia Developers Summary Report Output Format Specification

Industry Developers Summary Reports

Total Number of Developers Evaluated:
Average Score in UML Diagram Identification:
Average Score in General UML Diagram Basic Knowledge:
Average Score in UML Use Case Diagram Knowledge:
Average Score in UML Class and Object Diagram Knowledge:
Average Score in UML Sequence Diagram Knowledge:

Figure 4.12c Industry Developers Summary Reports Output Format Specification

4.6.2 Input Format Specification

The input for the System include: developers personal data, evaluation pin, selecting of evaluation starting point and supplied answers to the evaluation questions presented. The specifications for these are shown in the following sections.

4.6.2.1 Developers Personal Data

Simple personal data of the software developer being evaluated is collected by providing the necessary fields for the data in an input form as shown if figure 4.13.

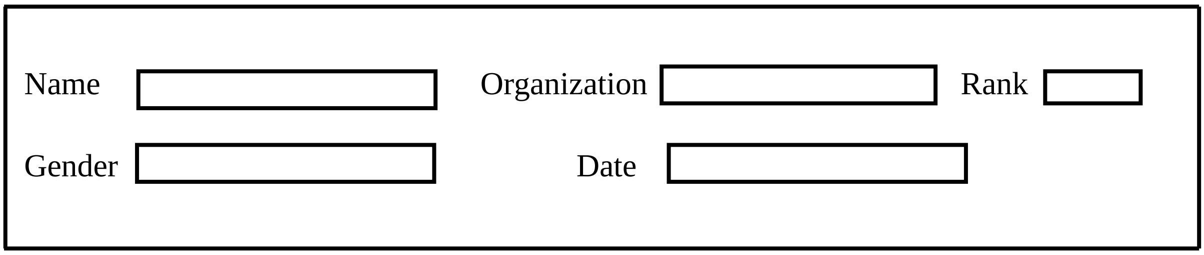


Figure 4.13 **Input Format Specification (Developers Personal Data)**

4.6.2.2 Evaluation Pin

Valid evaluation pin must be supplied before the developer will be granted access through the login button. this will collected as shown in figure 4.14.

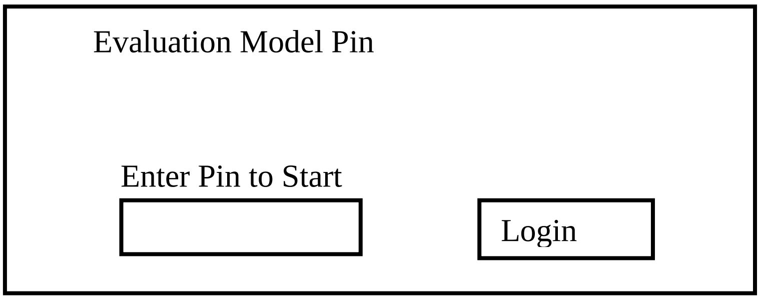
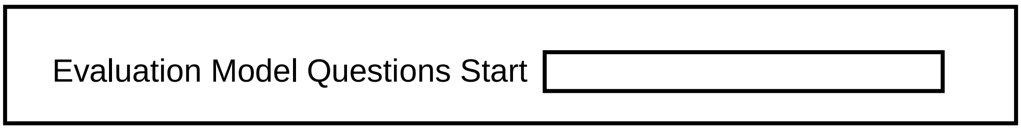


Figure 4.14 **Input Format Specification (Evaluation Pin)**

4.6.2.3 Evaluation Starting Point

The software developer being evaluated can choose the starting point of his or her evaluation from the five evaluation criteria. The evaluation criteria are presented as a drop down list from where the developer can choose. This is shown in figure 4.15.



Evaluation Model Questions Start □

Figure 4.15 **Input Format Specification (Developers Personal Data)**

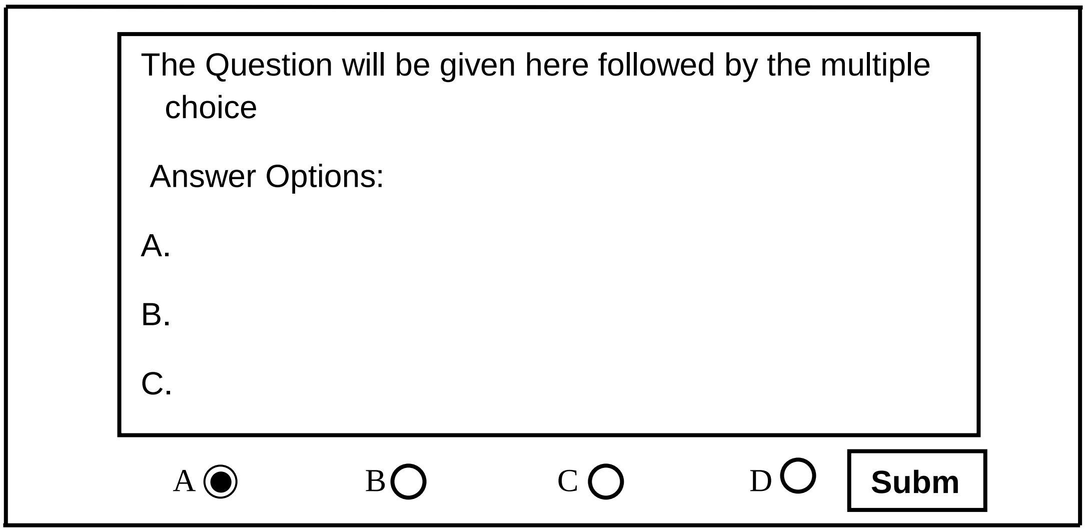
The evaluation criteria to choose starting point from include:

1. Class and Object Diagrams Questions
2. Sequence Diagram Questions
3. UML Diagrams Basic Knowledge
4. UML Diagram Identification
5. Use Case Questions

Ten questions will be presented on each of the given criteria. This gives a total of fifty questions and all the questions must be answered before the evaluation model will be created.

4.6.2.4 Answers To The Evaluation Questions Presented

Another important input to the system are the answers to the various evaluation questions presented to the developer. These are multiple choice questions with one of the options being the most correct answer. The question will be presented with the options A, B, C and D. Radio button with these letters A to D will be presented. The button whose label represent the most correct answer will be clicked by the user. The answer selected will be received when the Submit button is clicked. This is shown in figure 4.16



The Question will be given here followed by the multiple choice

Answer Options:

A.

B.

C.

A ● B ○ C ○ D ○ Subm

Figure 4.16 **Input Format Specification (Developers Answers to Evaluation Questions)**

4.7 Application Algorithm

The following is the Pseudo code that implements the new application.

```
Display Login Window
Prompt user for necessary information and evaluation pin

Do while (user information and evaluation pin is OK i.e. successful login)
        Display Main window
        Check evaluation start point selected
        Present question 1 of evaluation start point selected
        the user selects submit to submit result
        call submit subroutine
        Check for successful submit
        When submit is successful
        Save result
        Present next question
        Continue question presentation until all questions are answered
        If all questions has been answered
        Summarise result
        Present result
        Prompt to click on Model button
        Call model subroutine
End
If Submit is clicked
        Check if a an answer  is selected
        If not, insist on selecting an answer
        Else receive result
        Process result
 End
 If model button is clicked
        Access summarised result
        Plot the model
 End
```

4.8 System Activity Flows

System flows in object oriented methodologies are modeled with object models. In UML, activity diagrams are used to model both signal and data flows. Figure 4.17 shows the systems activity flow diagram. The Activity diagram for the Evaluation Questions is shown in Figure 4.18.

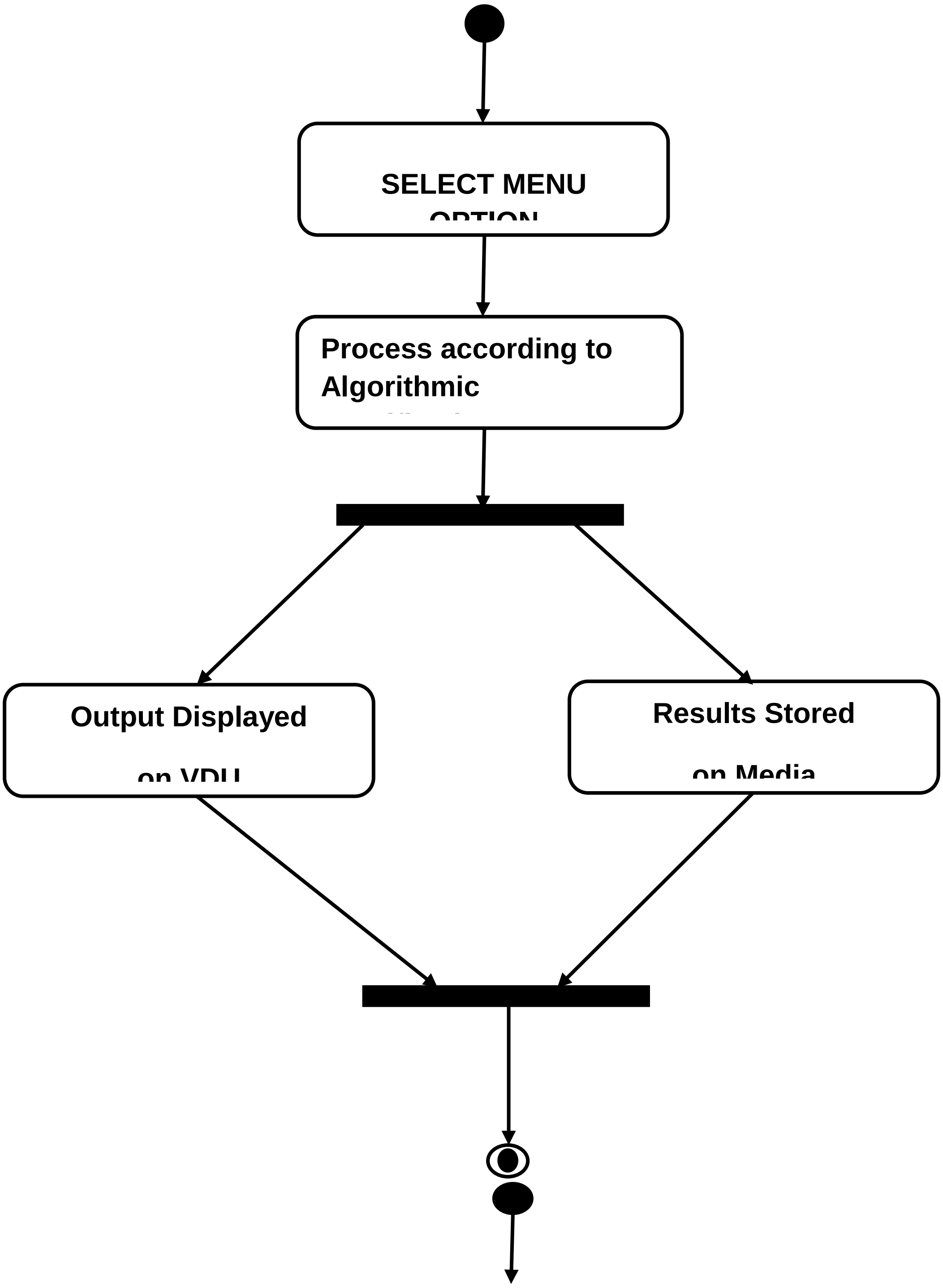


Figure 4.17 Activity Diagram for the System Flow

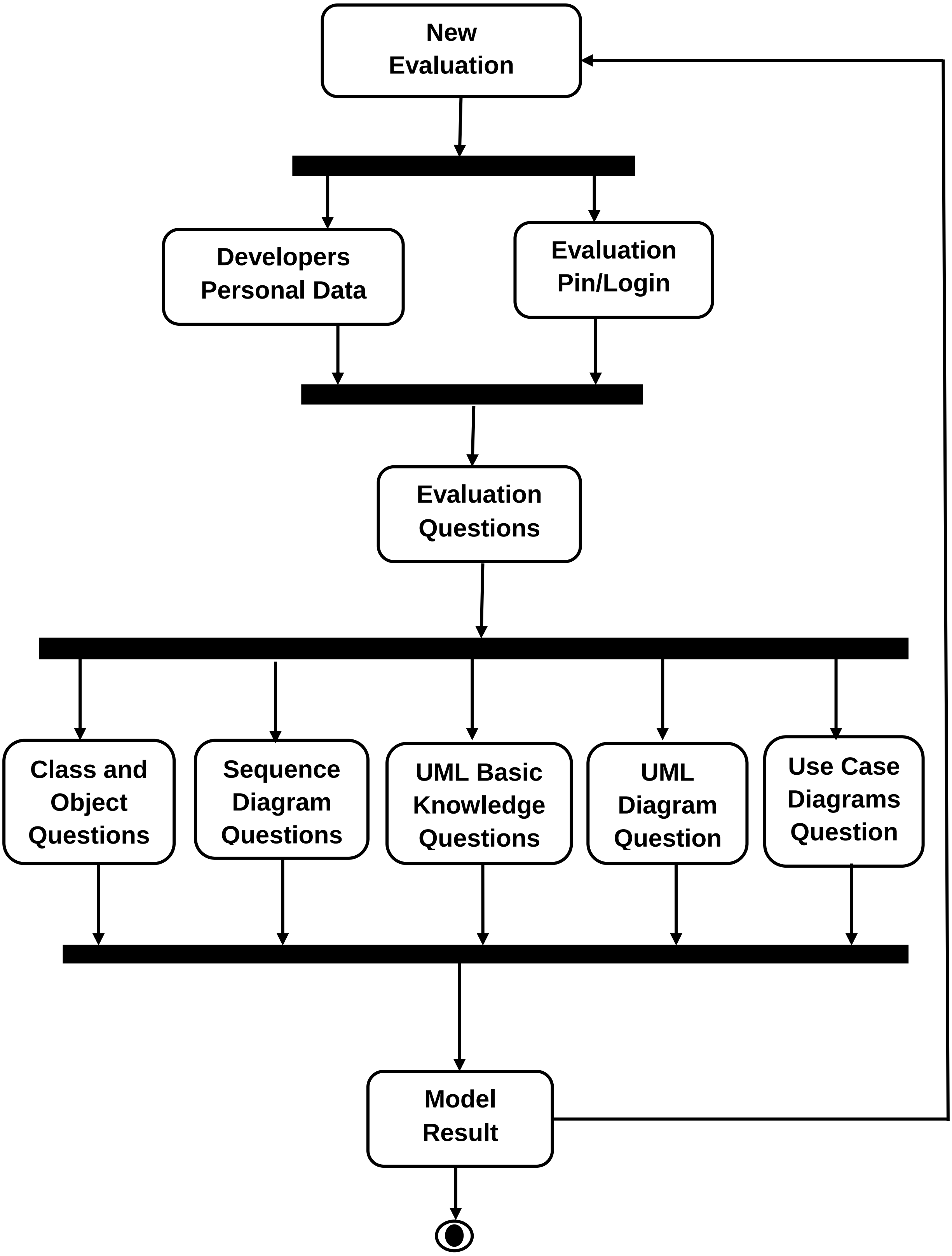


Figure 4.18 Activity Diagram for Evaluation Questions

4.9 Program Specifications

The program specification for different components of the new system is given in the sections that follows.

4.9.1 Program Specification for the Evaluation Questions

The classes identified in the Evaluation Questions component are outlined and explained in the following sections. They include:

1. Class and Object Questions class
2. Sequence Diagram Questions class
3. UML Basic Diagram class
4. UML Diagram identification class
5. UML Use Case Diagram class

4.9.2 Program Specification for the Evaluation Model

The classes identified in the Evaluation component are outlined and explained. They include:

1. Analyze Result class
2. Individual Developer Model class
3. Organization Model class

4.10 Development Increments

The software will be developed in five increments:

1. The Software Interface
2. The Evaluation Questions Designs
3. The Evaluation Questions addition as Resource
4. Result collation
5. Result Modeling

4.11 Database Specification

This system involves a database which will be used to save the users responses to the survey questions. This data is what will be used in generating the different reports required of this system. A relational database will be used. The database contains the following tables: Evaluation Registration table and Evaluation Result table shown in Table 4.1.

Table 4.1 Database Specification

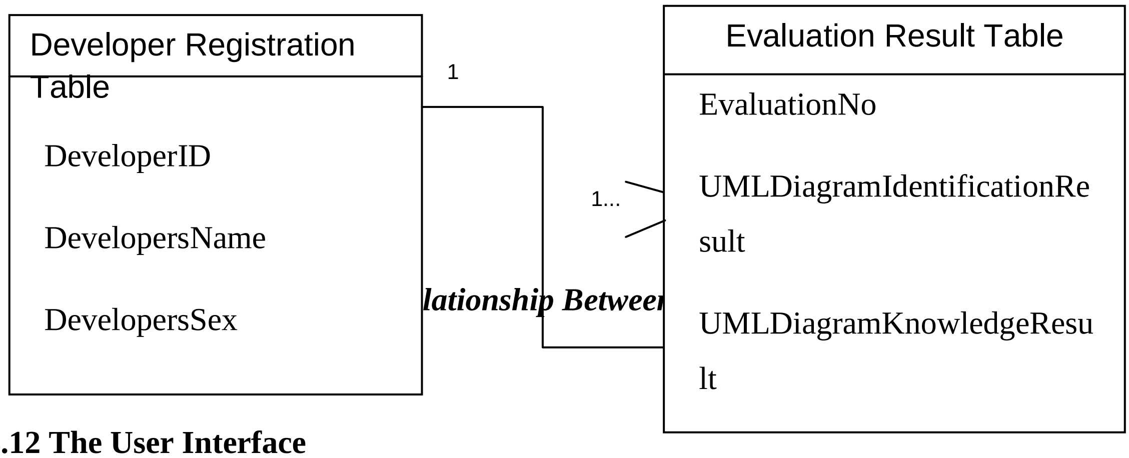
S/No	**Table Name**
1.	Evaluation Registration Table
2.	Evaluation Result Table

Table 4.2 Evaluation Registration Data description Table

S/No	**Field Name**	**Data Type**	**Size**
1.	DeveloperID	Integer	10
2.	DevelopersName	String	20
3.	DevelopersSex	String	20
4.	HighestEducationQualification	Boolean	5
5.	Organization	String	20

Table 4.3 Evaluation Result Data Description Table

S/No	**Field Name**	**Data Type**	**Size**
1.	EvaluationNo	String	10
2.	UMLDiagramIdentificationResult	Integer	20
3.	UMLDiagramKnowledgeResult	Integer	20
4.	ClassAndObjectDiagramResult	Integer	20
5.	UseCaseDiagramResult	Integer	20
6.	SequenceDiagramResult	Integer	20


Developer Registration Table
DeveloperID
DevelopersName
DevelopersSex
1
1...
Evaluation Result Table
EvaluationNo
UMLDiagramIdentificationResult
UMLDiagramKnowledgeResult

lationship Betweer

4.12 The User Interface

The User Interface of the application represents the part of the software the user sees and works with. From this interface, the user can view and access everything that the software has to offer in an interactive manner. Such intuitive interface is designed in a way that it will be easy for the user to move about. Figure 4.19 depicts a ketch of the user interface.

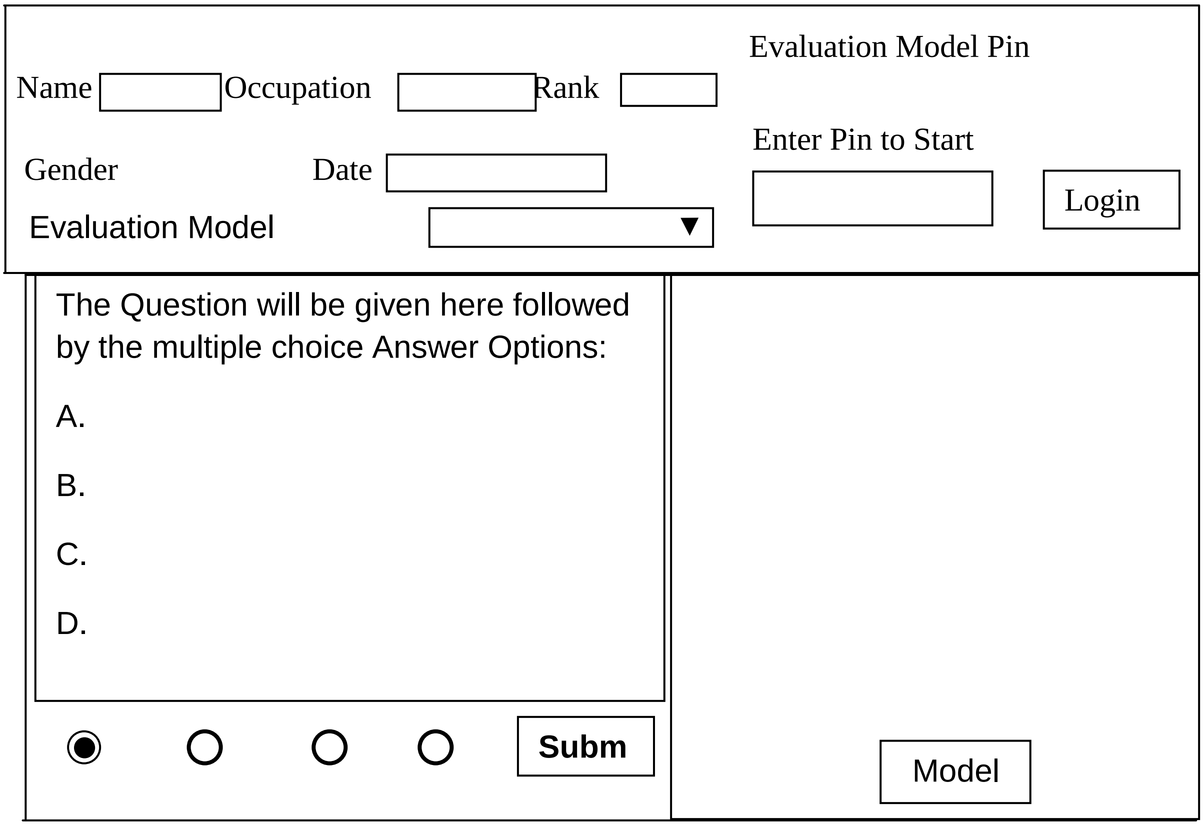


Figure 4.20 Home Screen of the user interface Design

4.13 Overall Data Flow in the New System

The overall data flow in the new system is presented in figure 4.20 using Data flow diagrams notation.

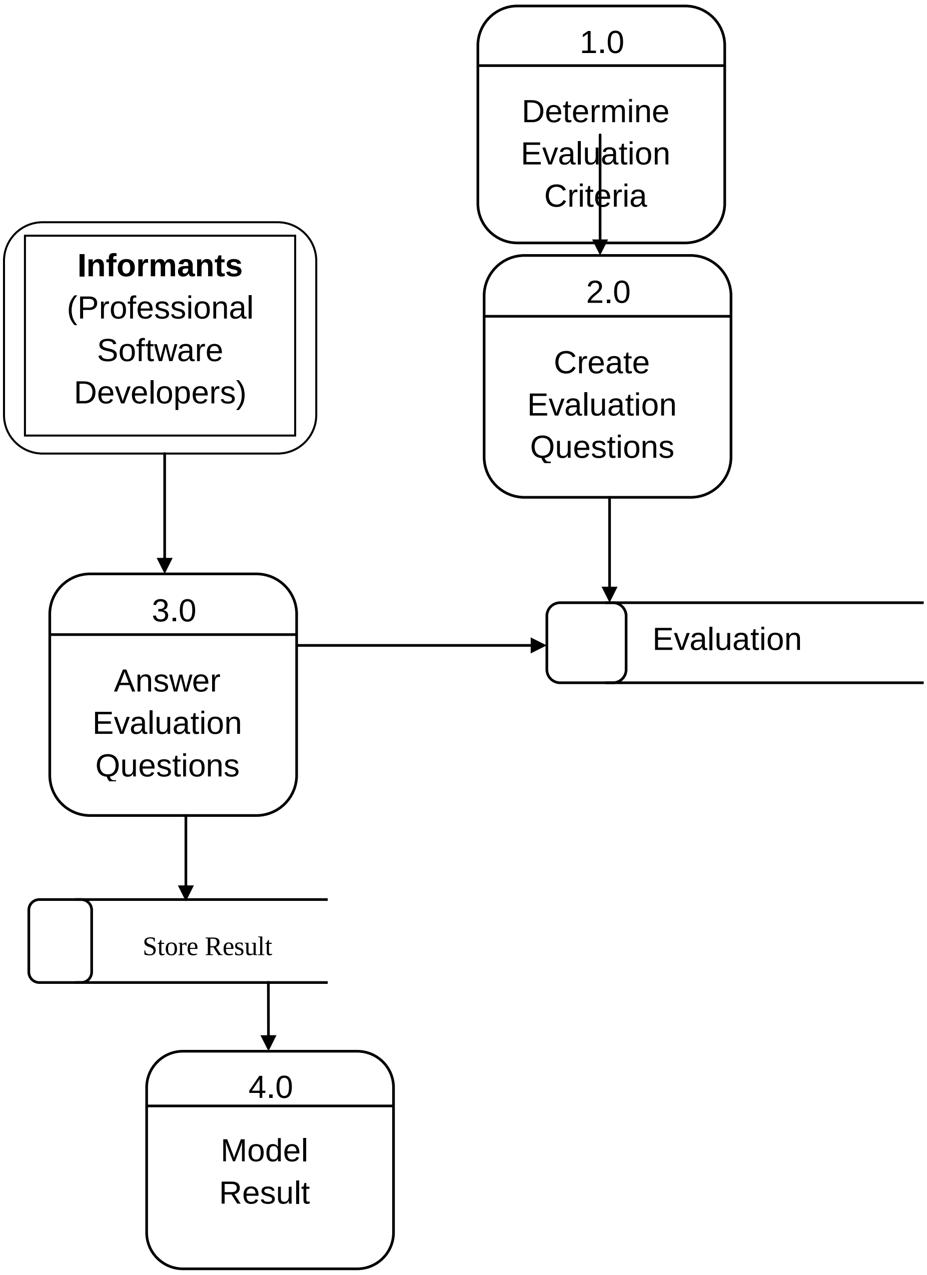

1.0
Determine Evaluation Criteria
Informants (Professional Software Developers)
2.0
Create Evaluation Questions
3.0
Answer Evaluation Questions
Evaluation
Store Result
4.0
Model Result

Figure 4.20 Overall Data Flow in the New System

4.14 The Survey Questionnaire Design

The questionnaire contained series of questions designed to get information in three areas:

1. About the software developers participating in the survey.

2. Assessing the knowledge and use of UML

3. Obtain information about which UML diagram is used more/less by software developers. The questionnaire shown in Table 4. contains both multiple choice questions (mutually exclusive and non-exclusive) and open- ended. To harvest more answers, it was decided that the questionnaire should not take more than approximately 10 minutes to complete. This is in line with the observation of Reggio, Leotta, Ricca (2014) that long questionnaires get less response than short questionnaires. The questionnaire was introduced with a brief motivation statement about the purpose of our research in line with the work of Kitchenham and Pfleeger (2008) and we added a sentence to clarify that all the collected information had to be considered highly confidential. All the participants were informed that the data collected will be used only for research purposes and they will be revealed only in aggregated form.

Table 4.4: Questionnaire. ME: means mutually exclusive multiple choice questions; NE: means non-mutually exclusive multiple choice questions; OP: means Open Question

ID	KIND	QUESTION
1.1	ME	Your Highest Educational Qualification in IT field is*:* *[PhD, M.sc, B.sc, HND]*
1.2	ME	How Many Software Projects Have you been Involved in Developing? *[Less than 3, 3 to 6, 7 and above]*
1.3	ME	What Software Development Methodology did you employ for it? *[Functional Decomposition, Structured Analysis and Design, Information Modeling, Object oriented, None]*
2.1	ME	Have you ever heard of Unified Modeling Language? *[Yes, No]*
2.2	ME	Have you ever modeled your software before developing them? *[Yes, No]*
2.3	ME	Which Modeling notation did you employ? *[None, UML, Others (Specify)]*
2.4	NE/OP	Which of UML diagrams have you used? *[Use Case, Activity, Collaboration, Sequence, Class, Object, State charts, Package, Component, Others (specify)]*
3.1	OP	How did you acquire your knowledge of UML? *[In school, Through Other tutorials (specify), Through Textbook (specify)]*
4.1	ME	What Level of impact did UML make to your software development experience? *[No Impact, Little Impact, Good Impact]*
4.2	ME	In which aspect of your software development stages do you mostly perceive the impact of UML on productivity? *[Requirement specification, Analysis, Design, Implementation, Testing]*

5.1 ME What is your overall rating of UML
[Excellent, Very Good, Good, Satisfactory]

4.15 System Implementation

4.15.1 Overview

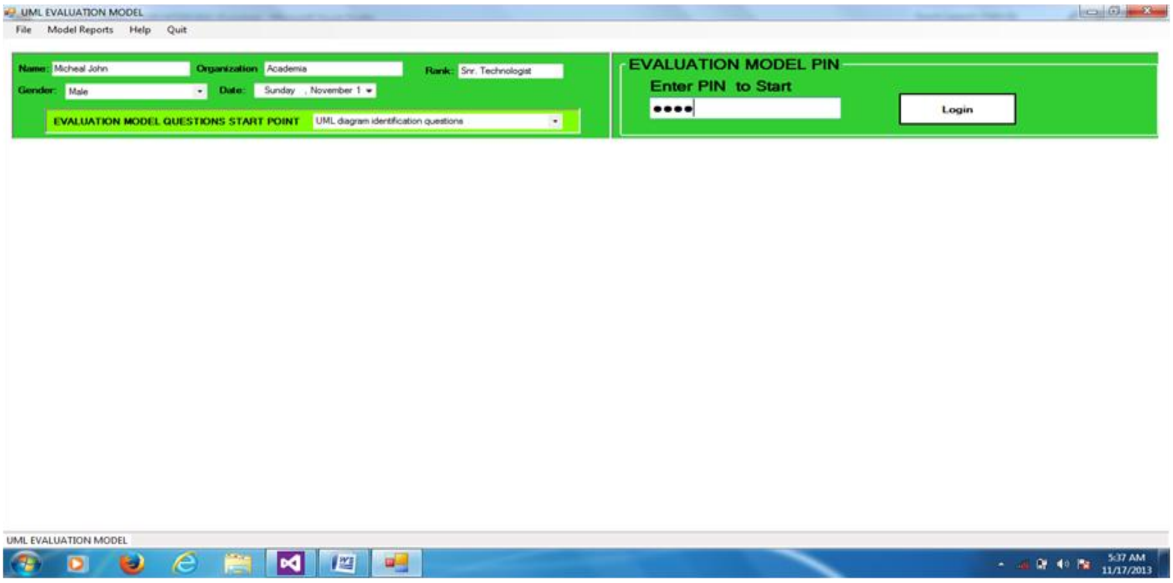
Implementation is a process of translating the system design into programs. Separate programs were written for each component, and they are put to work together. The implementation involved coding, testing and debugging of the program until the requirement specification is met.

In the unified Process, this is part of the Construction Phase. The construction phase is a manufacturing phase in which the product is designed and implemented. The emphasis is on managing resources and controlling operations to optimize cost and quality. The construction phase is broken down into several iterations focusing initially on determining the core architecture and then on designing and developing the components delivering the various use cases iteratively.

Reports of the scientific procedure followed in determining evaluation criteria and the creation of the Developers UML Knowledge Evaluation Model (DUKEM) is given in section 4.25 and 4.26 respectively. The following section discussed the implementation of the software designed based on DUKEM.

4.15.2 Implementation of the Software Designed based on DUKEM

The construction of the software was done with VB 2012. The listing of the program developed is shown in appendix A. The home page of DUKEM is shown in figure 4.21.



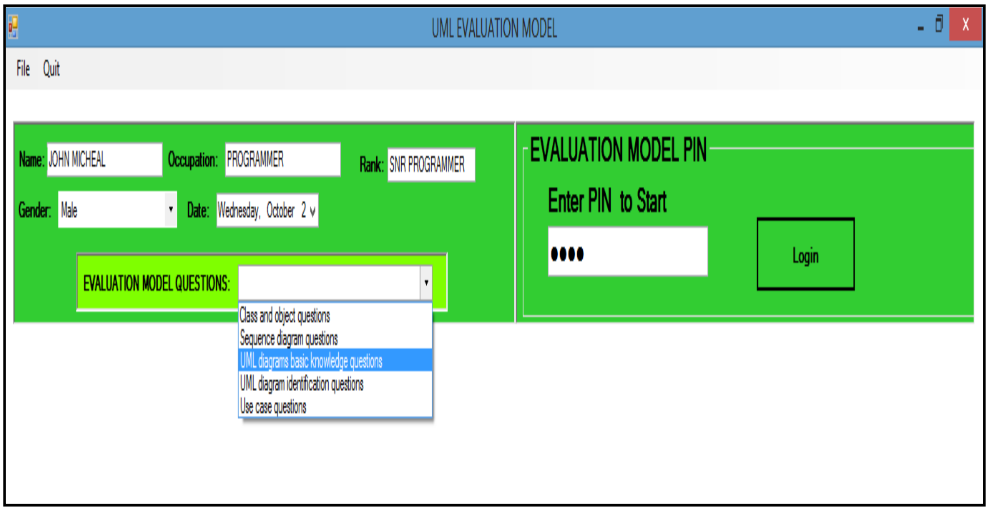
UML EVALUATION MODEL

Name: Micheal John
Organization: Academia
Rank: Snr. Technologist
Gender: Male
Date: Sunday , November 1
EVALUATION MODEL QUESTIONS START POINT: UML diagram identification questions

EVALUATION MODEL PIN
Enter PIN to Start: ●●●●

Figure 4.21 Home Page of DUKEM

From the home page, the developer fills registration information and enters evaluation pin before access to the evaluation questions will be granted. The developer also has the option of choosing a starting point by choosing a criterion to start with as shown in figure 4.22.


UML EVALUATION MODEL
File
Quit
EVALUATION MODEL PIN
Enter PIN to Start
Login
Class and object questions
Sequence diagram questions
UML diagrams basic knowledge questions
UML diagram identification questions
Use case questions

Name: JOHN MICHEAL

Occupation: PROGRAMMER

Rank: SNR PROGRAMMER

Gender: Male

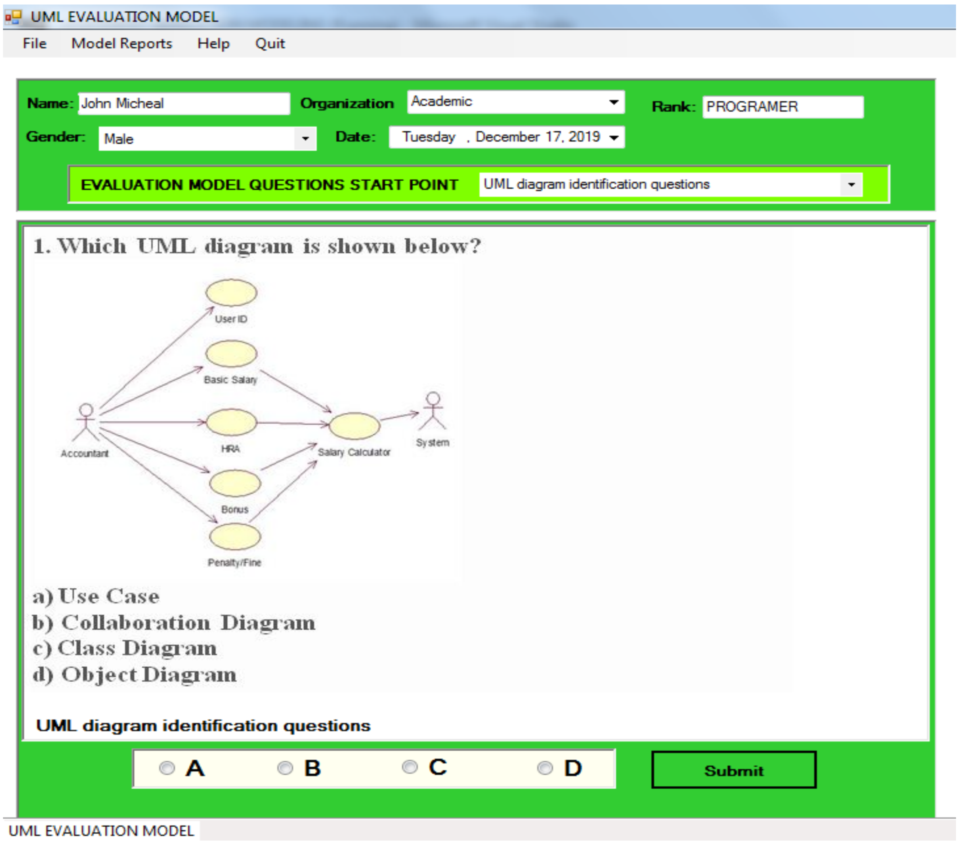
Date: Wednesday, October 2

EVALUATION MODEL QUESTIONS:

●●●●

Figure 4.22 Evaluation Criteria in DUKEM

There are five evaluation criteria and enough questions were created for each criterion. Sample question in each of the five criteria are shown in the figures that follows.



Name: John Micheal

Organization: Academic

Rank: PROGRAMER

Gender: Male

Date: Tuesday , December 17, 2019

EVALUATION MODEL QUESTIONS START POINT: UML diagram identification questions

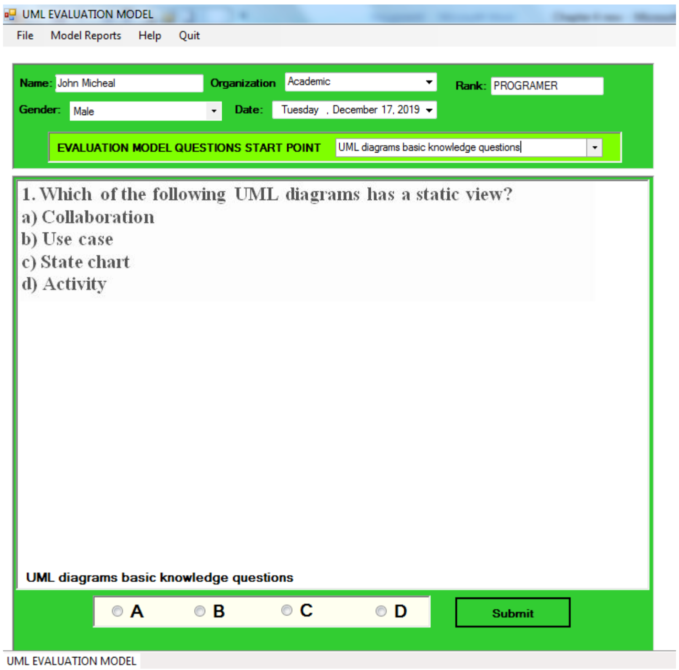
1. Which UML diagram is shown below?

a) Use Case
b) Collaboration Diagram
c) Class Diagram
d) Object Diagram

UML diagram identification questions

○ A ○ B ○ C ○ D

Figure 4.23 Sample Question for **UML Diagram Identification**



Name: John Micheal

Organization: Academic

Rank: PROGRAMER

Gender: Male

Date: Tuesday , December 17, 2019

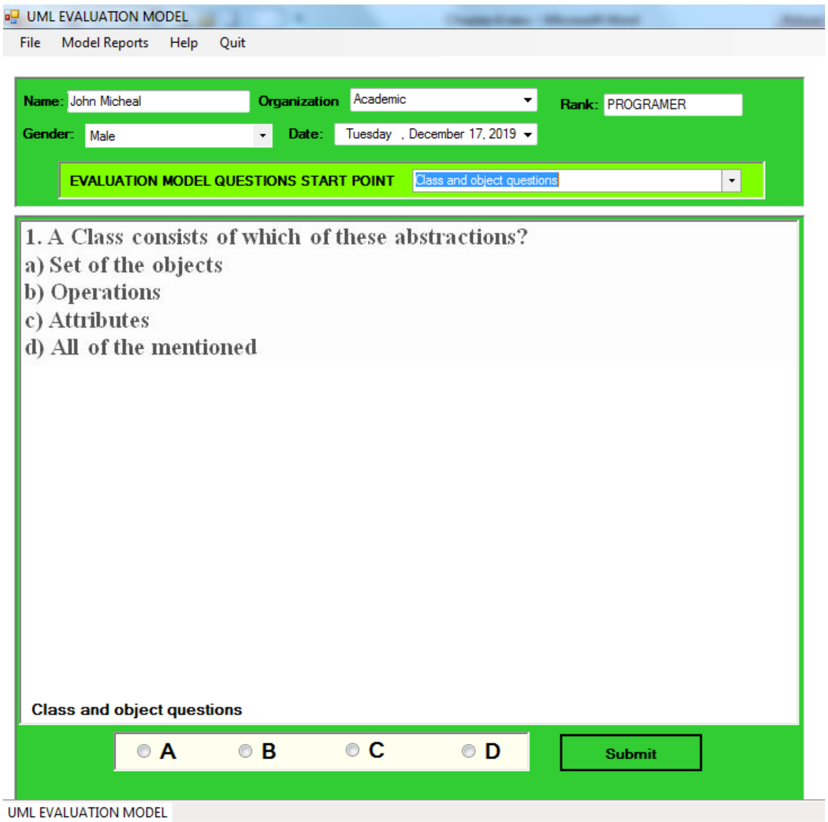
EVALUATION MODEL QUESTIONS START POINT: UML diagrams basic knowledge questions

1. Which of the following UML diagrams has a static view?
a) Collaboration
b) Use case
c) State chart
d) Activity

UML diagrams basic knowledge questions

○ A ○ B ○ C ○ D

Figure 4.24 Sample Question for UML Basic Knowledge



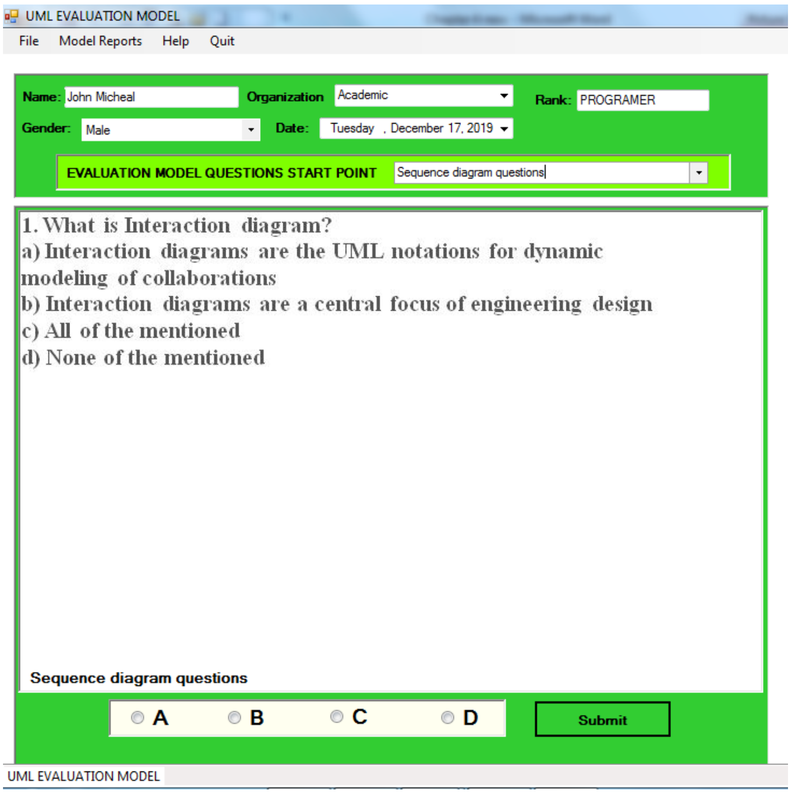
Name: John Micheal
Organization: Academic
Rank: PROGRAMER
Gender: Male
Date: Tuesday , December 17, 2019
EVALUATION MODEL QUESTIONS START POINT: Class and object questions

1. A Class consists of which of these abstractions?
a) Set of the objects
b) Operations
c) Attributes
d) All of the mentioned

Class and object questions

○ A ○ B ○ C ○ D

Figure 4.25 Sample Question for Class Diagrams



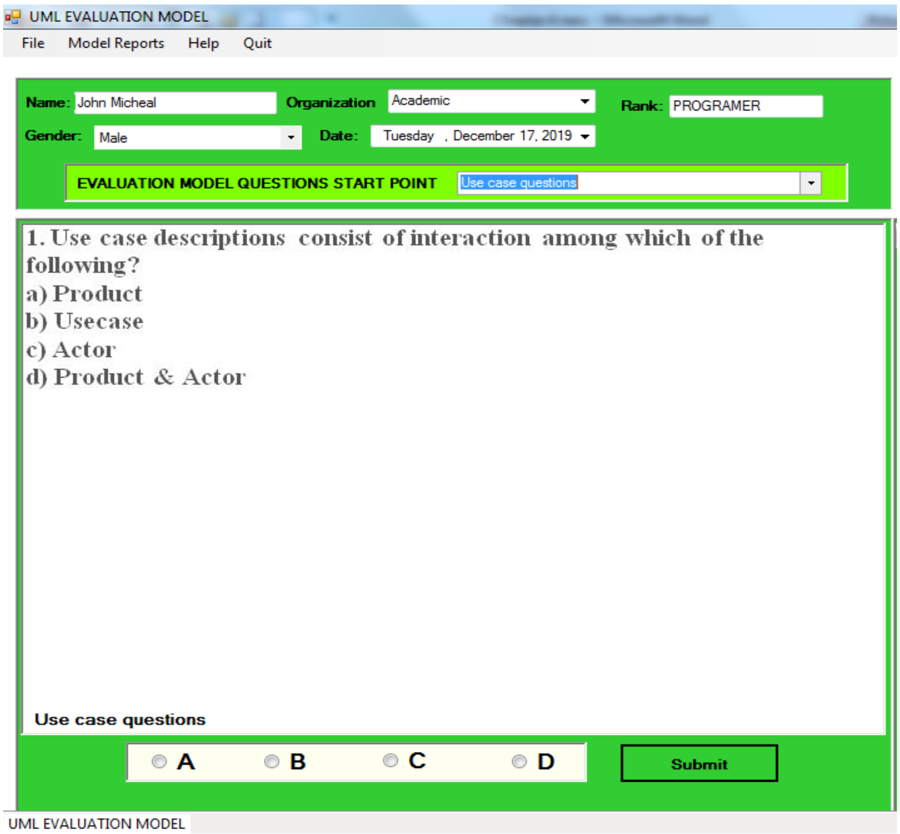
Name: John Micheal
Organization: Academic
Rank: PROGRAMER
Gender: Male
Date: Tuesday , December 17, 2019
EVALUATION MODEL QUESTIONS START POINT: Sequence diagram questions

1. What is Interaction diagram?
a) Interaction diagrams are the UML notations for dynamic modeling of collaborations
b) Interaction diagrams are a central focus of engineering design
c) All of the mentioned
d) None of the mentioned

Sequence diagram questions

○ A ○ B ○ C ○ D

Figure 4.26 Sample Question for Sequence Diagram



Name: John Micheal
Organization: Academic
Rank: PROGRAMER
Gender: Male
Date: Tuesday , December 17, 2019
EVALUATION MODEL QUESTIONS START POINT: Use case questions

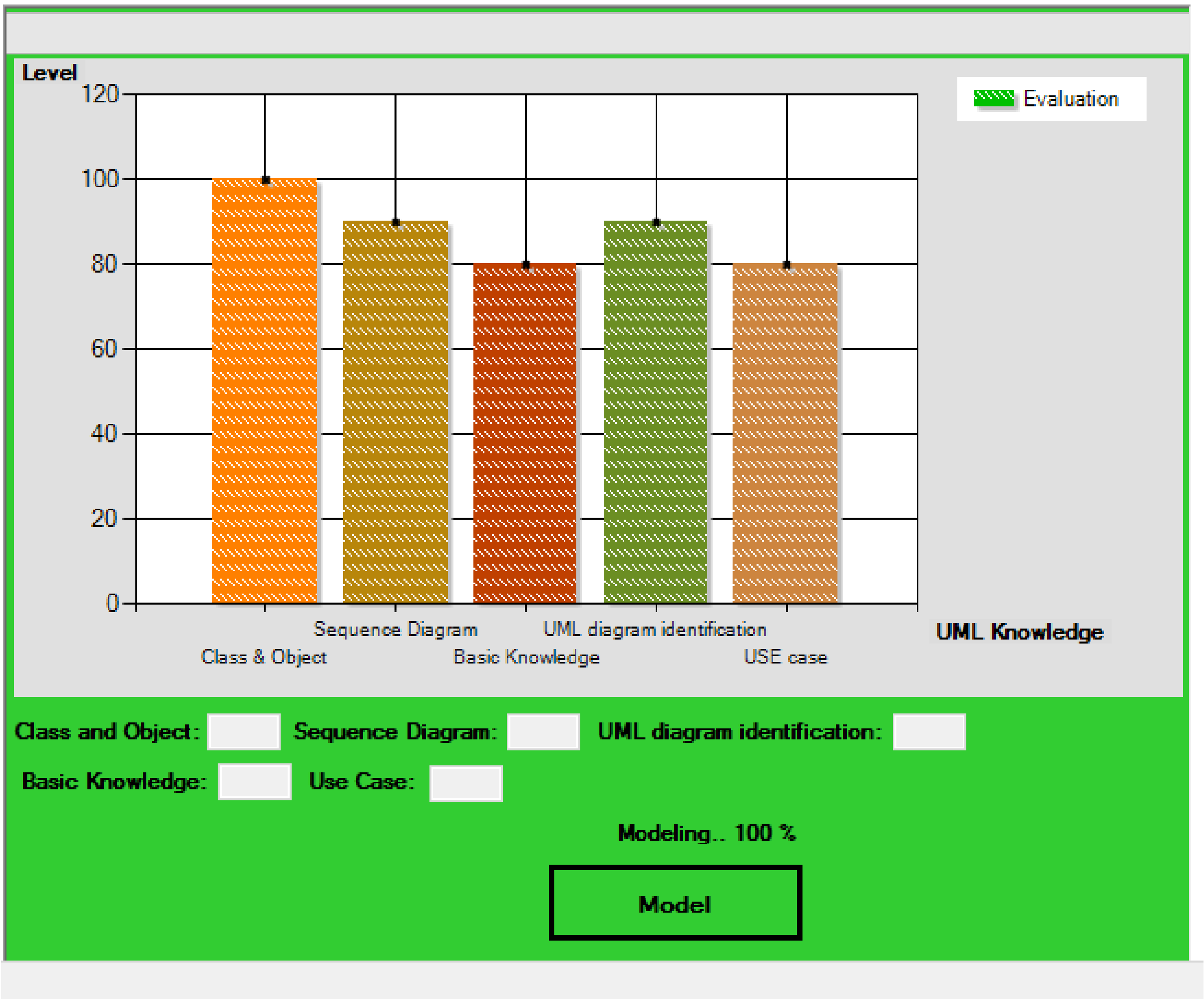
1. Use case descriptions consist of interaction among which of the following?
a) Product
b) Usecase
c) Actor
d) Product & Actor

Use case questions

○ A ○ B ○ C ○ D

Figure 4.27 Sample Question for Use Case Diagram

All the questions presented from all the evaluation criteria must be answered before the system will model the developers UML knowledge based on the result of the evaluation. Sample of the final model created is shown in figure 4.28.



Modeling.. 100 %

Figure 4.28 Sample of Developers UML Knowledge Modelled

Result Discussion

The evaluation was based on the evaluation Developers Knowledge evaluation model defined in figure 4. Scores of 60 and above in all five evaluation criteria shows that the developer is following the evolution trend of UML and can adequately adapt to its use in software modeling. Score below 60 in any of the five evaluation criteria shows that the developer is not fully following the evolution trend of UML and cannot yet adequately adapt to its use in software modeling. Figure 4.29 shows the pass level and the fail level of the sample model.

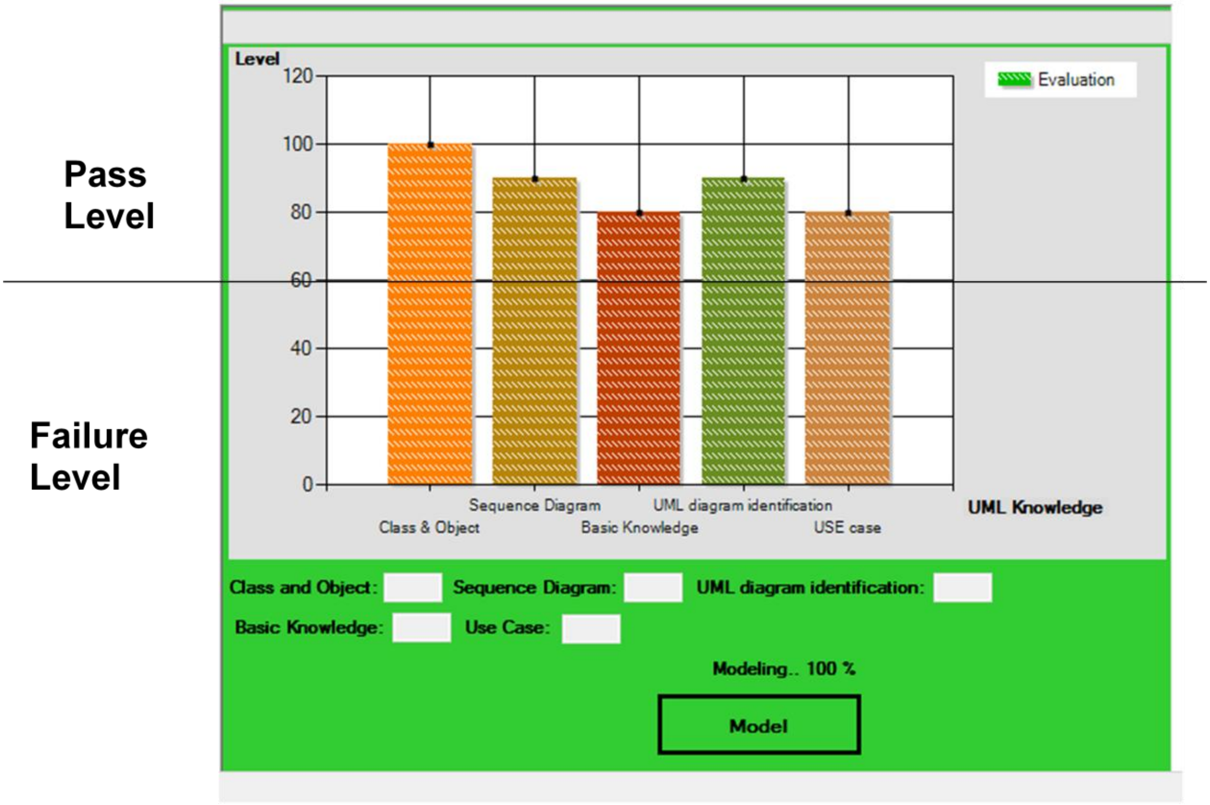


Figure 4.29 Pass Level and Fail Level in DUKEM

4.15.3 Hardware and Software Requirements

The recommended hardware and software requirements for this system are:

i. Microsoft Windows Server 2003, Windows XP (with Service Pack 2), Windows 2000 professional, Windows 7, Windows 8 or above.

ii. The processor recommended is Intel Pentium Dual CPU T3200 @ 2.00GHz

iii. 2.00GB RAM (memory) or above.

iv. Microsoft Visual Studio 2012 Professional edition and above or Visual Web Developer which contains the ASP.NET

v. At least I50GB Hard Disk

vi. Screen Resolution 600 Colour @ 1024 X 768 (minimum)

vii. CD-RW/DVD drive

viii. Optional External Hard Drive

ix. Internet Connection

x. Internet Browser

4.16 Program Development

4.16.1 Choice of programming Environment

The Language chosen for the implementation of this work was Visual Basic 2012. This is because Visual Basic 2012 is an Object Oriented Language and also has language compatibility with other languages in the visual studio.

4.16.2 Language Justification

The language deployed for the development of this application is ASP.Net. The language was chosen over other languages that can be used to implement object oriented designs for the reasons of the wonderful components of its development environment.

4.17 Program Testing

The new application that was developed was tested against the use-cases developed and the design architecture. High level of conformance was observed. The test was finally extended to the original objectives of the new system and it was observed that these objectives were satisfied.

4.17.1 Test Plan

The unified process test plan involves:

1. The user testing phase called “beta testing”. This may require some user training. The system is tested in its real environment and against user expectations.
2. If some or all of the use cases were previously delivered by some legacy system (manual or computerized), the new system usually runs for some time in simulation mode next to the legacy system. Any difference in behaviour and results should be understood.

Because none of the use cases were previously delievered by any legacy system, the system is simply tested in its real environment against the user expectations that is the system requirements. The frame work used for the testing is shown in figure 5.21.

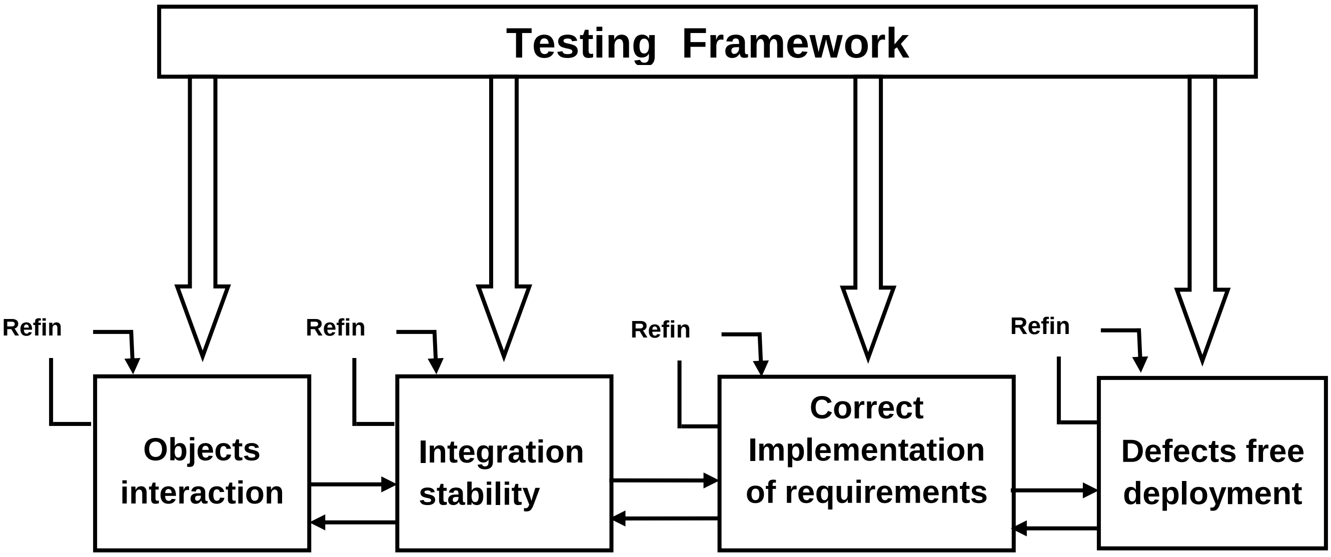


Figure 4.30 The Testing Framework Used

4.17.2 Testing

In addition to unit testing done for individual objects, the testing framework in figure 5.21 which was derived from the unified Process prescription for workflow testing was also used to test the system. The results shown in Table 4.5 showed a high level of conformity.

Table 4.5 Elements of the systems workflow tested

S/No	Tests Performed	Level of Conformity
1.	That the objects interact correctly.	High
2.	That the integration of higher level components and subsystems results in a stable system.	High
3.	That the requirements have been implemented correctly.	High
4.	That any failures are fed back to the development team and that the system is not deployed with any defects.	High

4.17.3 Test Data

To test the new system, data that was collected from Students and IT professionals within Anambra state. This enabled the researcher to test the work with real data.

4.17.4 Actual Versus Expected Test Results

There was a strong agreement between the actual test and expected results. The results of all the test components such as Object interactions, Integration stability, correct implementation of requirements and defect free deployment all showed a high level of conformity with the expected result.

4.18 Changeover Procedure

Since this research is a novelty work, no changeover is required. Direct installation and use of the software is therefore recommended. Subsequent researchers in this area will have to recommend a changeover procedure when this work will serve as the old system.

4.19 System Security

Security is a vital aspect of ASP.NET Web applications. To ensure the security of this application,

1. Membership was used to validate and store user credentials. This helps to manage user authentication in the Web sites. Membership was also used with the login controls to create a complete system for authenticating users.
2. Role management was used to manage authorization, which enables you to specify the resources that users in your application are allowed to access. Role management lets you treat groups of users as a unit by assigning users to roles such as manager, stakeholders, Student member, Professional member, and so on.

4.20 System Documentation

The software is very easy to install and use. It is packaged in an Auto play manner. To install and use this software,

1. Make sure you have the minimum hardware and software requirements for its installation and use (see section
2. Insert the disk in the CD-RW/DVD drive and follow the on screen instructions to install the software.

3. After the installation process is complete, double click on the program icon to run.
4. When the software is finally deployed to the internet, you can run it by entering its web address in any browser.

4.21 Maintenance

Maintenance means upkeep which is the general condition of something with respect to repairs. Software Maintenance involves either the removal of residual faults after a software project has been tested or enhancement carried out. This research presents the first version of this software. It is expected that the maintenance of the software will result in subsequent its subsequent versions.

4.22 Evaluation

4.22.1 Evaluation Criteria in the Unified Process

In the unified process, each phase of the development has its own evaluation criteria. The system was evaluated using these criteria. The sections that followed outlined the evaluation criteria for each of the four phases followed by the result of the evaluation using those criteria. The results were presented in a three point scale namely:

1. ***High (HI):*** This stands for highly agreeable and a YES condition with the evaluation criteria being considered.
2. ***Low (LO):*** This stands for not agreeable and a NO condition with the evaluation criteria being considered.
3. ***Adequate (AD):*** This represents a value that though not highly agreeable but is still very much satisfactory with the evaluation criteria being considered.

4.22.2 Evaluation of the Inception Phase

The evaluation criteria for this phase are:

1. Is there agreement among stake holders on the project's scope and cost/schedule estimates?
2. Credibility of the cost and schedule estimates and risks of the development process.
3. The depth and breadth of the architectural prototype.
4. Actual expenditures for this phase versus the planned expenditure.

Inception Phase Evaluation result is presented in table 4.6.

Table 4.6: Inception Phase Evaluation Result

S/No.	Evaluation Criteria	HI	AD	LO
1.	Is there agreement among stake holders on the project's scope?	✓		
2.	Credibility of the cost and schedule estimates.		✓	
3.	The depth and breadth of the architectural prototype.	✓		
4.	Actual expenditures for this phase versus the planned expenditure.		✓	

4.22.3 Evaluation of the *Elaboration* Phase

The evaluation criteria for the elaboration phase are:

1. Is the vision of the product stable?
2. Is the architecture stable?
3. Is there an executable prototype which demonstrates how the major risk elements have been resolved?
4. Is the plan for the construction phase sufficiently detailed and accurate and are the estimates credible?
5. Do the actual expenditures for this phase compare well with the planned expenditure?

Elaboration Phase Evaluation result is presented in table 4.7.

Table 4.7 Elaboration Phase Evaluation Result

S/No.	Evaluation Criteria	HI	AD	LO
1.	Is the vision of the product stable?	✓		
2.	Is the architecture stable?	✓		
3.	Is there an executable prototype which demonstrates how the major risk elements have been resolved?	✓		
4.	Is the plan for the construction phase sufficiently detailed and accurate and are the estimates credible?	✓		
5.	Do the actual expenditures for this phase compare well with the planned expenditure?	✓		

4.22.4 Evaluation of the Construction Phase

The evaluation criteria for the construction phase are:

1. Is the product release stable and mature enough for deployment?
2. Are all stake holders ready for project transition into the user community?
3. Are the actual expenditures versus the planned expenditures still acceptable?

Construction Phase Evaluation result is presented in table 4.8.

Table 4.8 Construction Phase Evaluation Result

S/No.	Evaluation Criteria	HI	AD	LO
1.	Is the product release stable and mature enough for deployment?	✓		
2.	Are all stake holders ready for project transition into the user community?		✓	
3.	Are the actual expenditures versus the planned expenditures still acceptable?	✓		

4.22.5 Evaluation of the Transition Phase

The evaluation criteria for the transition phase are

1. Is the user satisfied with the product?
2. Are the actual expenditures versus the planned expenditures still acceptable?

Transition Phase Evaluation result is presented in table 4.9.

Table 4.9 Transition Phase Evaluation Result

S/No.	Evaluation Criteria	HI	AD	LO
1.	Is the user satisfied with the product?	✓		

2.	Are the actual expenditures versus the planned expenditures still acceptable?	✓

4.23 Implementation of the Survey

Five main procedures were followed to prepare, administer, and collect the questionnaire data.

1. The questions for the Questionnaire were designed to meet the research goal and answer the research questions.

2. Pilot Study. A pilot study was performed had earlier been conducted

3. Deployment. The questionnaires were deployed to the targeted population after the pilot study.

4. Monitoring. During the data capture phase, our research group monitored the progress of the questionnaire submission. Few persons that reported difficulties about the questions were clarified.

5. Data Analysis. Analysis of the responses on the questions were done with the aim of finding answers to the research question.

4.24 Survey Results/Discussions

4.24.1 Respondents Background

From the answers to the first section of the questionnaire, we have found that: in Q1.1, the majority (110 respondents representing 70.5%) of the respondents are Higher degree Holders 64% M.Sc. and 5.6% PhD. This is shown in figure 4.31. Also majority (119 respondents representing 75%) are actually professional software developers having been involved in developing more than seven software projects Q1.2. Responding to Q1.3 on methodology employed, we discovered that two main methodology employed by these professionals are Structured System Analysis and Design methodology (38%) and Object Oriented Analysis and Design (58%). Only 3 respondents used Functional Decomposition and only 2 used information modeling as shown in figure 4.32.

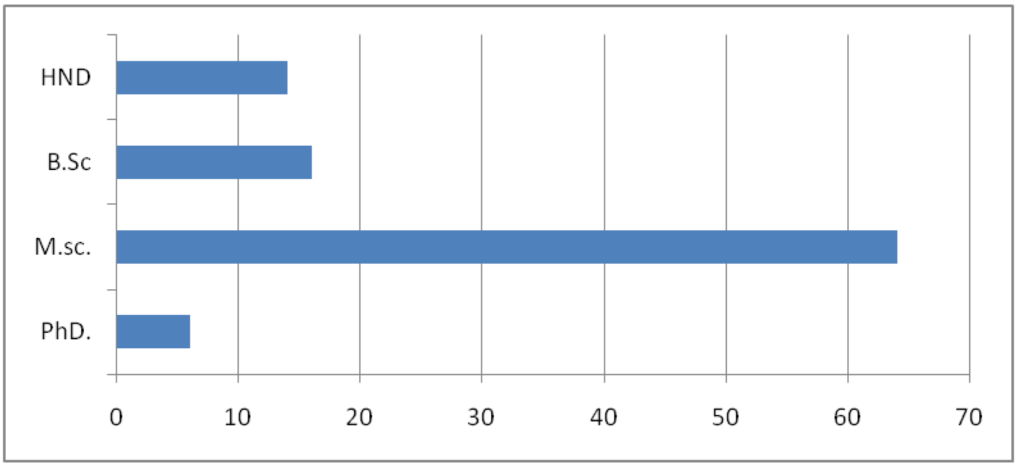


Figure 4.31. Respondents Highest Education Qualification Q1.1

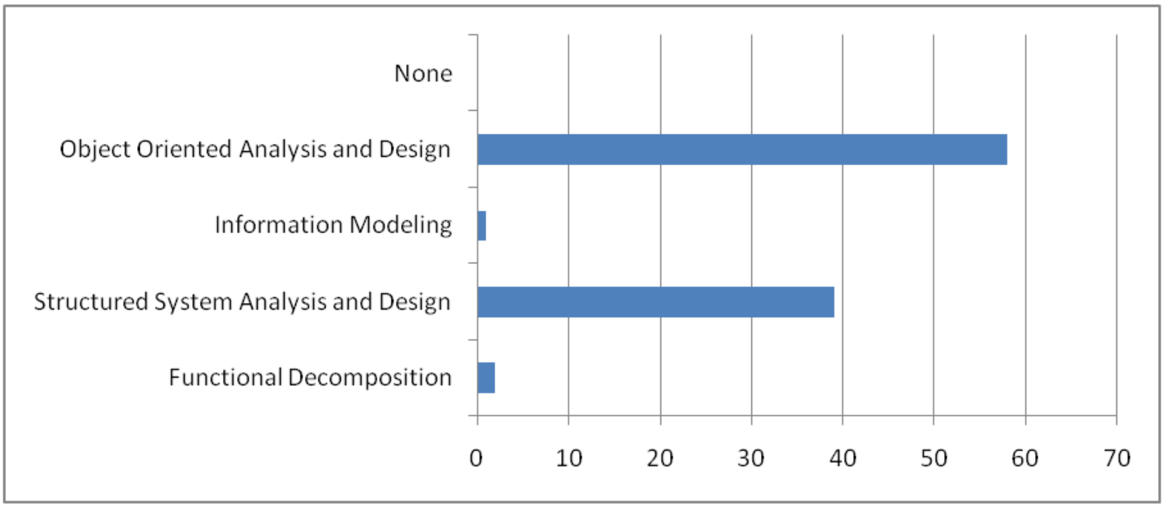


Figure 4.32. Software Methodology as used by Respondents Q1.3

4.24.2 UML Diagrams Knowledge

The results about the knowledge of the UML diagrams of all the respondents can be seen represented in Fig. 4.33. The chart shows that of all the software developers, 126 (68%) have heard of UML while the remaining 51 respondents representing (32%) have not heard about UML (Q2.1). Again, of the 126 respondents who have heard of UML, only 42 have used UML (Q2.2). The answer to RQ1 is that Software developers in Africa knew very well of the existence of UML as industry standard modeling tool, but many of them do not employ UML in their software modeling activities.

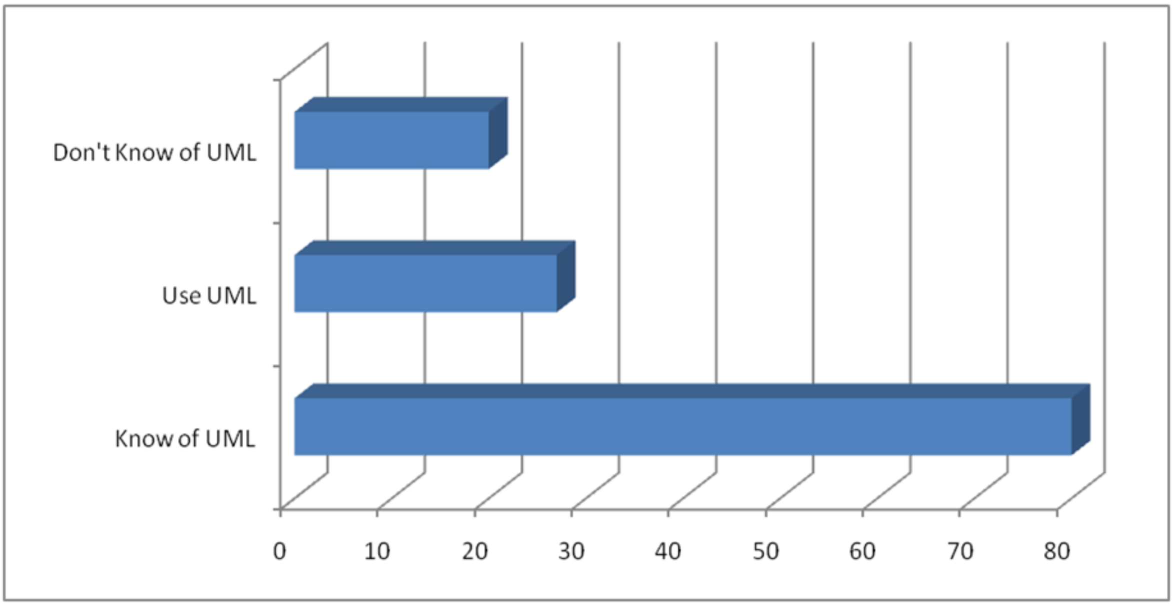


Figure 4.33. Knowledge and Usage of UML by Respondents (Q2.1, Q2.2)

4.24.3 UML Diagrams Usage

The level of usage of the various UML diagrams is as shown Figure 4.34. The chart shows that the level of usage is quite different. The diagrams usage level can be distributed in three main groups: G1, G2 and G3. G1 are those diagrams that are without any doubt widely used. These include the use-case diagram (98%), class diagram (97%), and sequence diagram (95%). The most known one is the use case diagram, and this is not surprising, since this diagram may be used without any other part of the UML, and it is truly useful to complement classical textual use case based requirements specifications, offering a nice way to visually summarize use cases, actors and relationships among them.

G2 diagrams are used with averagely good percentage. They are state-chart diagrams (52%), package diagram (61%), component diagram (71%), object diagram (81%), deployment diagram (66%), and collaboration diagram (73%). Lastly G3 are the remaining diagrams which are scarcely used. They are: composite structure diagram (45%), profile diagram (36%), interaction overview diagram (53%), and timing diagram (38%). The answer to RQ2 is that some UML diagrams are very widely used (G1), others are averagely used (G2), while the remaining ones are scarcely used (G3). The least used among them is the profile diagram followed by the timing diagram.

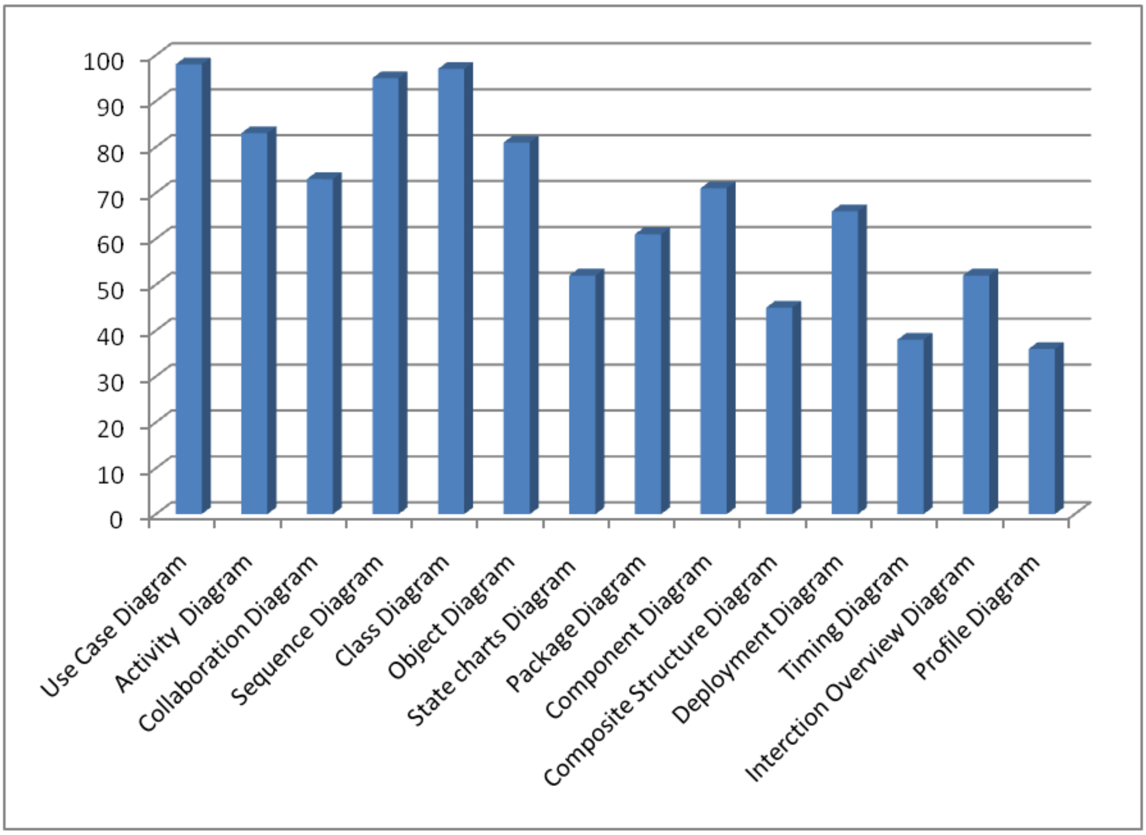


Figure 4.34. Usage Level of UML Diagrams by Respondents that use UML

4.25 More detailed Analysis to Determine Evaluation Criteria

For proper understanding and evaluation of research question raised and to ultimately achieve the research objectives, different techniques of analysis were employed. Majorly, the statistical tools used by the researcher to analyze the data were percentages, mean and One Way ANOVA for test of research hypothesis.

Table 4.10: Percentage and Mean usage of each UML diagram Type

4.25.1 Percentage Analysis

Percentage was used to answer research questions1and 2.

The formula is mathematically stated as:

Percentage **(%)** = (Frequency/Total Frequency) x 100

4.25.2 Descriptive Statistics (Mean)

Descriptive statistics (mean) was utilized in answering research question 2. The rating scale used was 4 points attitudinal rating scale, often referred to as “Likert Scale” (Brown, 2010).

The scale was quantified as follows:

Often = 4, Not Often)= 3, Sometimes = 2, Rarely =1

The formula for mean is given as (x-bar) = $\frac{\sum fx}{n}$

Where : x = Each of the rating scale point

f = Frequency of the Responses

n = Total number of respondents

Cut off mean = $\frac{(4+3+2+1)}{4}$ = 2.5 and above (Accept).

S/N		UML TYPE	PERCENTAGE (%) USAGE	MEAN USAGE
1	Often	Case Diagram	98	2.82
2		Activity Diagram	81	2.70
3		Sequence Diagram	95	2.74
4		Class Diagram	97	2.81
5	Not Often	Collaboration Diagram	70	2.50
6		Object Diagram	79	2.54
7		Package Diagram	60	2.44
8		Component Diagram	69	2.51
9		Deployment Diagram	64	2.12
10		Compute Structure Diagram	44	2.11
11	Sometime	Interaction Overview Diagram	50	2.20
12		Scale Chart Diagram	49	1.82
13	Rarely	Profile Diagram	35	1.54
14		Timing Diagram	36	1.33

Result Interpretation

From table 1, based on the analysis in percentage and descriptive statistics (mean), it's evident that Case Diagram has the highest percentage and mean usage among the UML types that are often used by the respondents with 98% (2.82), followed by Class Diagram, Sequence Diagram and Activity Diagram with percentage and mean usage of 97% (2.81), 95% (2.74) and 81% (2.70) respectively. Profile diagram and Timing diagram was observed to have the less usage in practice.

4.25.3 Test of Research Hypothesis at 5% Level of Significance.

4.25.3.1 Statement of Hypothesis

The statement of hypothesis is given as:

H_0: There is no significance difference in mean usage of UML by types.

***Table 4.11: One Way ANOVA test on the difference in mean usage of UML by types*.**

ANOVA

MEAN_USAGE

	Sum of Squares	Df	Mean Square	F	Sig.
Between Groups	2.597	3	.866	55.342	.000
Within Groups	.156	10	.016		
Total	2.753	13			

*Significant at 0.05; df = 3&10; F – critical 4.00.

4.25.3.2 Decision Rule

The decision rule is given as:

Reject the hypothesis if P-value is < 0.05, otherwise accept.

4.25.3.3 Results Discussion

From the table 4.11, F (3,10) = 55.342; P = 0.000 < 0.05. Following the decision rule, the above hypothesis is rejected hence we conclude that there is a significant difference in mean usage of UML by types. However, it implies that the result obtained through the percentage and mean was not by chance. Based on this investigation one can infer from the result that Use Case diagram, Class diagram and Sequence diagram has the most usage in practice while profile diagram and Timing has the least usage.

These are employed as evaluation criteria in DUKEM.

4.26 The Evaluation Criteria in DUKEM

Developers Knowledge Evaluation Model (DUKEM) suggests that since from the statistical investigation given in section 4.25, Use Case diagram, Class diagram and Sequence diagram has the most usage in practice while profile diagram and Timing has the least usage. Good working knowledge of these three diagrams and a general basic knowledge of other UML diagrams (including identification of the diagrams and basic usage of the diagrams) will provide adequate evaluation criteria for the model.

The evaluation criteria used in DUKEM is therefore has two major components: General Knowledge (GK) and Detailed Knowledge (DK).

DUKEM formula used in measuring Developers Adequate Adaptation to UML (DAA_{UML}) therefore states that:

$$DAA_{UML} = GK_{AD} + DK_{OUD}$$

Where

DAA_{UML} is Developers Adaptation Adequacy to UML

GK_{UML} is the General Knowledge of UML diagrams

DK_{OUD} is the Detailed Knowledge of Often Used UML diagrams

GK_{UML} is given as:

$$GK_{UML} = GA_{DK} + GA_{DU}$$

and

DK_{OUD} is given as:

$$DK_{OUD} = DA_{KUC} + DA_{KUU} + DA_{KUS}$$

Where

GA_{DK} = General Adequate Diagram Knowledge

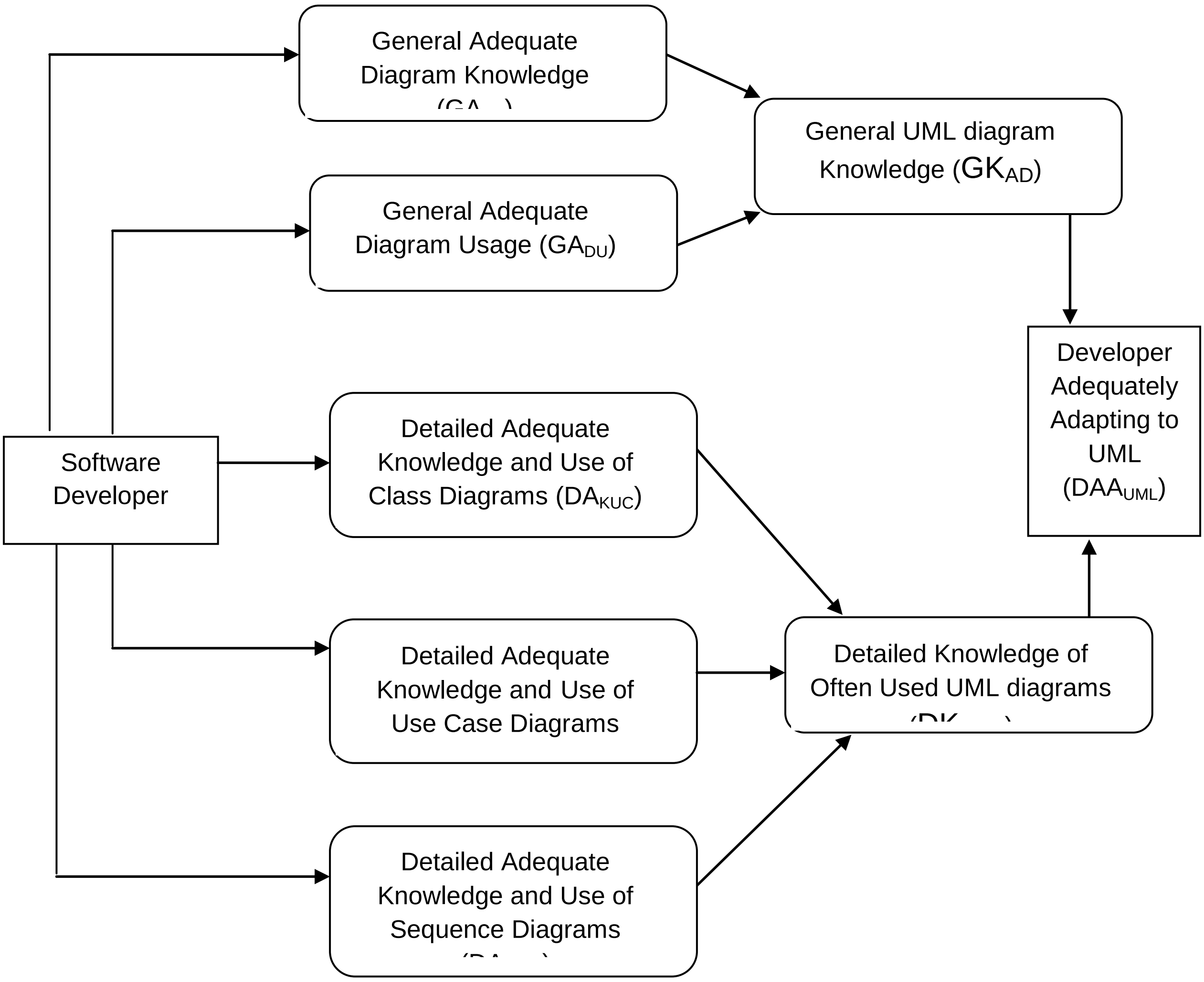
GA_{DU} = General Adequate Diagram Usage

DA_{KUC} = Adequate Detailed Knowledge and Use of Class Diagrams

DA_{KUU} = Adequate Detailed Knowledge and Use of Use Case Diagrams

DA_{KUS} = Adequate Detailed Knowledge and Use of Sequence Diagrams

Figure 4.35 shows the Developers UML Knowledge Evaluation Model (DUKEM)



General Adequate Diagram Knowledge (GA)
General UML diagram Knowledge (GK$_{AD}$)
General Adequate Diagram Usage (GA$_{DU}$)
Developer Adequately Adapting to UML (DAA$_{UML}$)
Software Developer
Detailed Adequate Knowledge and Use of Class Diagrams (DA$_{KUC}$)
Detailed Adequate Knowledge and Use of Use Case Diagrams
Detailed Knowledge of Often Used UML diagrams (DK)
Detailed Adequate Knowledge and Use of Sequence Diagrams (DA)

Fig 4.35 Developers UML Knowledge Evaluation Model (DUKEM)

CHAPTER FIVE

SUMMARY AND CONCLUSION

5.1 Summary

Modeling software visually is one of the best practices in software engineering and UML has emerged as a bonafide industry standard for software modeling since 1996. This dissertation created Developers UML Knowledge Evaluation Model (DUKEM) for evaluating software developers adaptation to industry standard modeling tool. The goal is fill literature gap of very few study on actual adaptation and use of this industry standard tool by software developers.

The first chapter of this work presented the background to the study, the Statement of the Problem and Objectives of the Study. The Significance of the Study and the Scope of the Study were also presented.

The second chapter reviewed related literatures on the Unified Modeling Language and the Unified Process. The discussions in this chapter was organised under the following headings: The Value of Modeling, Modeling before UML, The Unified Modeling Language (UML) 2.0, UML adoption and usage, UML and the Unified Process, Models and Architectural views, UML Diagrams, How to use the UML, Research Methods, Current researches on UML Usage. The chapter in looking at the Value of Modeling noted its importance in design software development. It also saw what modeling was like before the introduction of UML as the de facto standard for software development. New features in UML 2.0 were discussed and some of the usage of UML round the globe was sampled. The relationship between UML and the Unified process was also discussed and finally, the chapter discussed some current post graduate works on UML usage.

The third chapter analysed the present system noting its advantages and disadvantages. It also analysed the proposed system identifying the major functional activities from the requirements. Both the use case model and the analysis model of this proposed system were also presented. The chapter also show how the classes identified in the analysis model interact with each other and with the actors using sequence diagrams.

In chapter four, the researcher created the design models. The design models created includes; detailed use case description package of the system, the design classes for the different use cases, and the relationship between these design classes, state chart model for password control, high level model of the new application, and the activity diagrams for various activities such as the system flow, dynamic models tutorials and the unified process tutorial. Other designs carried out in this chapter are: output format specification, input format specification, application algorithm, program specifications, database specification, the user interface and the overall data flow in the new system with UML's interaction overview diagram.

Chapter four also described the evolution of the new application using Visual Studio 2012. Program development was discussed in details, the justification for the hardware and software platforms were discussed as well as testing and implementation considerations. Here, comprehensive testing and evaluation were also carried out. The testing framework used was derived from the unified Process prescription for workflow testing. The testing showed a high level of conformity with the elements of the system workflow tested. The system was evaluated using the unified process evaluation criteria for each of the four phases - inception, elaboration, construction and transition - and the results were presented in a three point scale.

Finally in chapter five, the researcher, provided a summary of the entire dissertation, made recommendations and suggested areas for further research work.

5.2 Review of Achievements

In this work, the researcher has modeled and implemented UML knowledge Evaluation System for evaluating software developer's knowledge and adaptation to the use of Unified Modeling Language which has became the de facto standard for modeling in software Industry.

Other achievements include:

1. Conduct a survey to capture UML diagrams usage in the industry and academy by IT professionals.
2. Model questions that will adequately evaluate software developers level of knowledge and adaptation to the use of UML in software modeling based on the evaluation criteria determined from the survey.
3. Creation of a novelty developer evaluation model that plots a graph of UML content against the level of knowledge possessed. A picture speaks a thousand words.

5.3 Areas of Application

The software can be applied in the following areas:

1. Any organization whether business or academics that is a stakeholder in software development project since UML provides a lingua franca for communication between stakeholders in software development project.
2. Any organization whether business or academics that needs a UML evaluation system.
3. For software developers who are involved in team work, it will help them for effective communication and division of labour.
4. Educational institutions can deploy it in their computer science, computer engineering and other IT related fields as a model for teaching object oriented system analysis and design.
5. The work will also assist researchers who are carrying out UML related researches in several ways.

5.4 Major Contributions to Knowledge

1. This work is has made special contribution to knowledge since it addressed the problem of standard in software modeling.
2. Provision of new and unique evaluation model
3. It has provided the Nigerian IT students and professionals with a unique learning tool that will empower them for more effective practice.

5.5 Suggestions for Further Work

The following are suggestions for future research.

- Development of an automated software architecture tool that will make creation of the models easier.
- Implementation of this work with other object oriented languages like Java so that a comparative analysis of the implementation on Visual Studio and Java can be carried out.
- Since this is a novelty work, there must be room for improvement. Other researchers can also analyse this work and design a new system that will be an improvement on this work.

5.6 Recommendations

This research work discovered that most software developers do not model their software thereby losing all the values derived from modeling. It was also discovered that this is due to inadequate knowledge of appropriate modeling tools. The researcher therefore recommends the following:

1. That the Nigerian universities commission, National board for technical education and other education governing bodies in Nigeria should introduce the course object oriented analysis and design with Unified Modeling Language into computer science curriculum or expand the present course titled system analysis and design to include it so that Nigerian computer science graduates will be properly equipped.
2. That the computer professionals Registration Council of Nigeria should make the working knowledge of unified modeling Language a requirement for the award of its certificates.
3. That both IT students and professional should cultivate the habit of modeling their software projects before implementation because of the several implication of not modeling as discussed in literature.
4. That all stakeholders should make maximum use of the information system provided by this work.

5.7 Conclusion

The goal of standards in relation to computers is to establish uniformity in area of hardware or software development. Modeling software visually is one of the six best practices for software development and Unified Modeling Language (UML) has emerged as the de facto standard in software modeling.

Every IT students/professionals should therefore be equipped with a working knowledge of how to employ UML in object oriented system analysis and design along with a compatible software development process like the unified process. Also, all stakeholders in computer industry or software using companies from other industries in Nigeria should possess a handy evaluation tool

This dissertation has achieved that by creating UML evaluation model that will help developers to key in into its adaptation This Model will help to move software development forward if the recommendations are implemented.

REFERENCES

Adolph, S. (2002). *Patterns for effective use cases*. Addison-Wesley. from http://books.google.com/books?id=FGdXBs5uCxMC&pg=PA2

Adriana, D., Tayana, C., and Igor, S. (2019) Analyzing Students Perception of UML Diagrams Instruments used in Evaluation. https://figshare.com/articles/Analyzing_Students_Perception_of_UML_Diagrams_Instruments_Used_in_Evaluation/9118949.

Alavi, M. and Carlson, P. (1992) A Review of MIS Research and Disciplinary Development. Journal of Management Information Systems. Vol. 8, No. 4. Pp. 45-62.

Alexander, C. (1979) Christopher Alexander. *The Timeless Way of Building*. Oxford University Press. 1979.

Alhir, S.S. (2010) Understanding the Unified Process (UP) retrieved on 15th July 2010 from http://www.methodsandtools.com/archive/archive.php?id=32

Anderson, K.M. (2003) *Object Design: Roles, Responsibilities and Collaborations*. Addison-Wesley/Pearsons Education. 2003.

Bailin, S.C. (1989) "An Object-Oriented Requirements Specification Method." *Commun. ACM* 32(5), May 1989, pp. 608−623.

Balin, V. (2016) Software: Managing Complexity from https://medium.com/@gaperton/software-managing-the-comlexity-caff5c4964cf

Bass, L., Clements, P. and Kazman, R (2003) *Software Architecture in Practice*. second edition. Addison Wesley. 2003.

Bea, B. (2003) Business Process Execution Language for Web Services 1(1) May 2003, (ftp://www6.software.ibm.com/software/developer/library/ws-bpel.pdf).

Beck, K. and Cunningham, W (1989) "A Laboratory for Teaching Object-Oriented Thinking." *Proc. Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '89)* in *ACM SIGPLAN Notices* **24**(10), New Orleans, LA, Oct. 1989, pp. 1−6.

Ben-Abdallah, H., Bouassida, N, Gargouri, F., and Ben-Hamadou,A.(2004) "A UML Based Framework Design Method". Journal of Object Technology. 3. 8. 2004.

Berard, E. (1991) "Object-Oriented Semantic Networks." Berard Software Engineering Inc., Gaithersburg, MA, 1991.

Bittner, K. & Spence, I. (2003). Use case modeling. Addison-Wesley

Booch, G. (1994) *Object-Oriented Analysis and Design with Applications*. 2nd edn, Benjamin/Cummings, Redwood City, CA, 1994.

Booch, G. et al. (2004) "An MDA Manifesto," with Frankel, D., and Parodi, J. (eds.), The MDA Journal, Meghan-Kiffer Press.

Budgen, D., Burn, A.J. Brereton, O.P. , Kitchenham, B. A. and Pretorius. R. (2011) Empirical evidence about the UML: a systematic literature review. Software Practice and Experience, 41(4):363–392, Apr. 2011.

Boudreau, M.C., Gegen, D. and Straub, D. (2001) "Validation in IS Research: A State of the Art Assessment," MIS quaterly, Vol. 25, No. 1, pp. 1-16, 2001

Brian Henderson-Sellers, B. (1994) *Book Two of Object-Oriented Knowledge: The Working Object*. Prentice Hall, Englewood Cliffs, NJ, 1994.

Brian, R.L. and John, H (2018) A Unified Approach for Modeling, Developing and Assuring Critical Systems. https://www.researchgate.net/publicatio/328569106 A Unified Approach for Modeling, Developing and Assuring Critical Systems 8th International Symposium ISoLA 2018 Limassol Cyprus November 5-9 2018 Procedings Part Is 5

British Computer Society (2004) BCS Professional Examination Object Oriented Programming Question (Version 2 Sylabus).

Brooks Jr., F.(1995) The Mythical Man-Month (1995 edition), Addison-Wesley.

Brown, A. (2004) "An Introduction to Model-Driven Architecture," IBM Rational developerWorks(http://www-106.ibm.com/developerworks/rational/library/3100.html).

Cantor, M.R. (1998) *Object-Oriented Software Management*. Wiley. 1998.

Cernosek, G. and Naiburg, E. (2004) A technical discussion of software modeling. Rationale Software. Copyright IBM Corporation 2004 IBM U.S.A. IBM Software Group Route 100 Somers, NY 10589 U.S.A.

Clements, P. Kazman, R. Klein, M. (2002) *Evaluating Software Architectures*. Methods and case studies. Addison-Wesley. 2002.

Coad, P. and Yourdon, E.(1991a) *Object-Oriented Analysis*. 2nd edn, Prentice Hall, Englewood Cliffs, NJ, 1991.

Coad, P. and Yourdon, E.(1991b) *Object-Oriented Design*. Prentice Hall, Englewood Cliffs, NJ, 1991.

Cockburn, A. (2001) *Writing Effective Use Cases*. Addison-Wesley. 2001.

Colbert, E. (1989) “The Object-Oriented Software Development Method: A practical approach to object-oriented development.” *Proc. Tri-Ada*, New York, 1989.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C. Gilchrist, H., Hayes, F., and Jeremes, P. (1994) *Object-Oriented Development:The Fusion Method*. Prentice Hall, Englewood Cliffs, NJ, 1994.

Coleman, P.T, Liebovitch, L.S and Fisher (2019) Taking Complex systems seriously: Visualizing and Modeling the Dynamics of Sustainable Peace. https://doi.org/10.1111/1758-5899.12680

Cook, S. and Daniels, J. (1994) *Designing Object Systems*. Prentice Hall, Hemel Hempstead, England, 1994

Denzin, N., and Lincoln, Y. (1994). Handbook of Qualitative Research. Sage Publicatio, California,pp: 3-5.

Desfray, P. (1992). *Ingénerie des Objets: Approche class-relation application à C++*. Editions Masson, Paris, 1992.

Dobing, B. and Parson, J. (2008) Dimensions of UML Diagram Use: A Survey of Practitioners. Journal of Database Management, 19(1), March 2008, pp.1-18.

Dobing, B. and Parsons, J. (2006) How UML is Used communications of the acm May 2006/Vol. 49, No. 5, 109-113

Dobing, B. and Parsons, J. (2007) impact of the UML on systems development. http://www.acmqueue.org/modules.php?name=Content&pa=showpage&pid=461

Dos, S., Soares, M. and Vrancken, J. (2017) Evaluation of UML in Practice – Experiences in a Traffic Managent Systems Company. https://www.semanticscholar.org.

Dzidek, W.G. (2008) Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance from http://simulase581pdf/download&ved

Easterby-Smith, M., Thorpe, R. and Lowe, A. (1991). Management Research: An Introduction. Sage Publication, London, pp: 23-25.

Embley, D.W., Kurtz, B.D. and Scott N. Woodfield, S.N. (1992) *Object-Oriented Systems Analysis: A Model-Driven Approach*. Prentice Hall, Englewood Cliffs, NJ, 1992.

Erickson, J and Siau, K (2007) Can UML be simplified? Practitioner use of UML in separate domains. In Proceedings of 12th International Workshop on Exploring Modeling Methods for Systems Analysis and Design, volume 365 of EMMSAD 2007, pages 81–90. CEUR Workshop Proceedings, 2007.

Eriksson H.E. and Penker, M. “UML Toolkit” John Wiley & Sons, Inc.

Finch, J., (1986). Research and Policy. The Falmer Press, London, pp: 6-10.

Firesmith, D.G. (1993) *Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach*. John Wiley, New York, 1993.

Forrester (2008) "Modernising Software Development Through Model-Driven Development", A commissioned study conducted by Forrester Consulting on behalf of Unisys, 13 August 2008, http://tinyurl.com/5nrfss

Fowler, M. (2004) UML Distilled (3rd edition), Addison-Wesley.

Frankel, D.S. (2003) *Model Driven Architecture: Applying MDA to enterprise computing*. John Wiley & Sons. 2003.

Fritz, S. (2012) Object Oriented Analysis and Design Using UML, retrieved 2012 from http:www.solms.co.za.

Galbraith, J., Downy, D. and Kates, A. (2001) *Designing Dynamic Organizations: A Hands-On Guide for Leaders at All Levels* . American Management Association. November 2001.

Graham, I. (1994) *Object-Oriented Methods*. Addison-Wesley, Wokingham, England, 1994.

Graham, I. (2001) Object-Oriented Methods: Principles and Practice (3rd edition), Addison-Wesley.

Grossman, M. Aronson, J. E. and McCarthy, R. V. (2005) *Does UML make the grade? Insights from the software development community.* Information and Software Technology, 47(6):383–397, 2005.

Gummesson, E., (1991). Qualitative Methods in Management Research. Sage Publication, California, pp: 83-156.

Hartley, J., (1994). Case Studies in Organizational Research in Casell and Symon 1994 Qualitative Methods in Organizational Research. Sage Publication, London, pp: 208-229.

International Telecommunications Union (2000), ITU Recommendation Z.109: SDL Combined with UML, ITU-T, 2000.

International Telecommunications Union(2002) ITU Recommendation Z.100: Specification and Description Language (SDL), (08/02), ITU-T, 2002.

Ismail, S. (2017) Qualitative Research for beginners. From http:// www.kobo.com/ww/en/ebook/qualitative-research-for-beginners

Jacobson, I. (2009) Taking the temperature of UML. from http://blog.ivarjacobson.com/taking-the-temperature-of-uml/, 2009.

Jacobson, I. (2001) Appling UML in the Unifies process. Retrieved from www.researchgate.net/publication/237035406_ Appling UML in the Unifies process

Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process* Addison Wesley. 1999.

Jacobson, I., Christerson, M., Jonsson, P. and Övergaard, G. (1992) *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, Wokingham, England, 1992.

Johnson, D., (1994) Research Methods in Educational Management. Longman Group, Essex

Jonova, C. and Milian, C. (2019) Evaluation of UML diagrams for test cases generation: Case study on depression of internet addiction https://www.sciencedirect.com/science/article/pii/SO37843711930336X.

Katzman, R. (1994) Rick K. *Toward Deriving Software Architectures from Quality Attributes*. 1994.

Kevin, L. and Howard, H. (1994). *Object-Oriented Specification Case Studies*. Prentice Hall, Hemel Hempstead, England, 1994.

Khairul, B. and Mohd, N. (2008) Case Study: A Strategic Research Methodology, American Journal of Applied Sciences 5 (11):

Kitchenham, B.A. and Pfleeger, S.L. (2008) Personal opinion surveys. In F. Shull, J. Singer, and D. I. K. Sjoberg, editors, Guide to Advanced Empirical Software Engineering, pages 63–92. Springer London, 2008.

Koichiro, O. (2008) Outline of UML and Unified Process School of Information Science JAIST

Koivulahti-Ojala, M (2017) On UML Modeling Tool Evaluation, Use and Training. University Library of Jyväskylä. http://urn.fi/URN:ISBN:978-951-39-7273-8

Kruchten, P. (2000)*The Rational Unified Process*. An Introduction. second. Addison Wesley. 2000.

Lano, K. (2009) UML 2 Semantics and Applications November 2009 http://www.wiley-vch.de/publish/dt/books/ISBN978-0-470-40908-4

Lee, L. (1992) The Day the Phones Stopped Ringing, Plume Publishing, 1992.

Leffingwell, D. and Widrig, D. (2000) *Managing Software Requirements*. Addison-Wesley. 2000.

Lethbridge, T.C. and Forward, A. (2008) Problems and opportunities for model-centric versus code-centric software development: a survey of software professionals. Models in Software Engineering workshop (MiSE '08) at ICSE, ACM, 27-32.

Martin, J. and Odell, J.J. (1992) *Object-Oriented Analysis and Design*. Prentice Hall, Englewood Cliffs, NJ, 1992.

Merriam, S., (1988). Case Study Research in Education: A Qualitative Approach. Jossey-Bass Publishers, California, pp: 4-25.

Meyer, B. (1985) "On Formalism in Specifications." *IEEE Software* **2**(1), Jan. 1985, pp. 6−26.

Microsoft Encarta Encyclopedia, (2009). "Standard (computer)." Redmond, WA: Microsoft Corporation, 2008.

Microsoft (2019) The Simple Guide to UML Diagramming and Database Modeling from https://www.microsoft.com/en-us/microsoft-365/growth-center/resources/guide-to-uml diagramming and database modeling

Mohagheghi, P. Dehlen, V. and Neple,T. (2009) Definitions and approaches to model quality in model-based software development - a review of literature. Information and Software Technology, 51(12):1646–1669, Dec. 2009.

Morgan, G., and L. Smircich, (1980). The Case for Qualitative Research. Acad. Manag. Rev. 5 (4): 491-500.

Norton, D. (2006) "View DSLs and UML as 'Fraternal Twins', Not Competitors", Gartner Research, 29 September 2006

Nugroho, A. (2010) The Effects of UML Modeling on the Quality of Software PHD Thesis.

Nugroho, A. and Chaudron, M.R.V. (2008) A survey into the rigor of UML use and its perceived impact on quality and productivity. In Proceedings of Empirical software engineering and measurement (ESEM). 2008, 90-99.

Nugroho, A., Flaton, B, and Chaudron, M.R (2008) Empirical Analysis of the Relation between Level of Detail in UML Models and Defect Density. Proceedings of the 11th International Conference on Model Driven Engineering Languages and Systems (MODELS), September 2008.

Object Management Group (2003) Unified Modeling Language (UML), Version 1.5, OMG document formal/03-03-01 (http://www.omg.org/cgi-bin/doc?formal/03-03-01).

Object Management Group (2004) UML 2.0 Superstructure, Available Specification, OMG document ptc/04-10-02 (http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-02.zip),

Page-Jones, M., Constantine, L. L. and Weiss, S. (1990) "Modeling Object-Oriented Systems: The Uniform Object Notation." *Computer Language* **7**(10), Oct. 1990, pp. 69−87.

Petre, M (2013) UML in practice. In Proceedings of 35th International Conference on Software Engineering, ICSE 2013, pages 722–731. IEEE, 2013.

Pinsonneault , A. and Kraemer, K. (1991) Survey Research Methodology in Management Information Systems: An assessment. *www.crito.uci.edu/papers/1993/urb-022.pdf* -

Recker, J. (2008) "BPMN Modeling – Who, Where, How and Why", BP Trends, retrieved May 2010, from http://tinyurl.com/5f2fh8

Reenskaug, T., Andersen, E.P., Berre, A.J., Hurlen, A., Landmark, A.,Odd Arild Lehne, O. A., Nordhagen,E., Nêss-Ulseth,E. (1992) Gro Oftedal, Anne Lise Skaar, and

Pål Stenslet. "OORASS: Seamless support for the creation and maintenance of object oriented systems." *Journal of Object-Oriented Programming* **5**(6), Oct. 1992, pp. 27−41.

Reggio, G., Leotta, M. and Ricca, F. (2014) Who Knows/Uses What of the UML: A Personal Opinion Survey. Springer International Publishing Switzerland http://dx.doi.org/10.1007/978-3-319-11653-2_10

Robinson, P. (1992) *Hierarchical Object-Oriented Design*. Prentice Hall, Englewood Cliffs, NJ, 1992.

Rosenberg, D. and Scot, K. (2000 "Driving design with use cases". Software Development. 2000.

Rubin, K.S. and Goldberg, A. (1992) "Object Behavior Analysis." *Commun. ACM* **35**(9), Sept. 1992, pp. 48−62

Rumbaugh, J. Blaha, M., Premerlani, W. Eddy, F. and Lorensen, W.(1991) *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, NJ, 1991.

Rumbaugh, J., Jacobson, I., and Booch, G., (2005) The Unified Modeling Language Reference Manual (2nd edition), Addison-Wesley.

Ruud Lemmers(2008) Using UML and the Unified Process retrieved on 15/07/2010 from http://www.Logicacmd.com

Seidewitz, E and Stark, M.(1987) "Towards a General Object-Oriented Design Methodology." *Ada Letters* **7**(4), July/Aug. 1987.

Seidewitz, E. (2012) UML 2.5: Specification simplification. Presented at "Third Biannual Workshop on Eclipse Open Source Software and OMG Open Specifications", May 2012.

Selic, B. (2004) "On the Semantic Foundations of Standard UML 2.0," with Bernardo, M., and Corradini, F. (eds.), Formal Methods for the Design of Real-Time Systems, Lecture Notes in Computer Science vol. 3185, Springer-Verlag, 2004.

Selic, B. (2005). What's new in UML 2.0? http://www-01.ibm.com/software/rational/uml/

Shlaer, S. and Mellor, S. J. (1992) *Object Lifecycles: Modeling the World in States*. Prentice Hall, Englewood Cliffs, NJ, 1992.

Siemens Nixdorf Informationssysteme AG.(1993) “MooD V1.0—Methodology for Object-Oriented Development: Introduction.” Frankfurt / Main, Germany, 1993.

Sparx Systems (2019) UML Modeling tool for Business, Software, Systems and Architechture. https://sparxsystems.com.

Stake, R.E.,(1995) The Art of Case Study. Thousand Oaks, CA. Sage.

Stevens, P.(2002) On the Interpretation of Binary Associations in the Unified Modeling Language,” Journal of Software and Systems Modeling, vol.1, no.1, Springer-Verlag

Straub, D., David G., and Marie-Claude, B. (2005). "The ISWorld Quantitative, Positivist Research Methods Website," (Ed) Dennis Galletta, http://www.dstraub.cis.gsu.edu:88/quant/.

UML Revision Taskforce. (2001) *OMG UML Specification v. 1.4*. Object Management Group. OMG Document Number formal/01-09-67. Available at http://www.omg.org.

Velho, A. V. and Carapuça, R.(1992) “SOM—A Semantic Object Model: Towards an Abstract, Complete and Unifying Way to Model the Real World.” *Proc. Third Int. Conference on Dynamic Modeling of Information Systems*, Noordwijkerhout, The Netherlands, 1992, pp. 65−93.

Velho, A. V. and Carapuça, R.(1994) “Attribute: A Semantic and Seamless Construct.” *Proc. Technology of Object-Oriented Languages and Systems* (TOOLS 13, Versailles, France, Mar. 1994), Prentice Hall, Hemel Hempstead, England, 1994.

Waldén, K. and Nerson, J. (1994) *Seamless Object-Oriented Software Architecture Analysis and Design of Reliable Systems* from http//www.bon-method.com

Wasserman,A. Pircher, P. A. and Muller, R. J. (1990) “The Object-Oriented Structured Design Notation for Software Design Representation.” *IEEE Computer* **23**(3), Mar. 1990, pp. 50−63.

Watson, A. (2010) Visual Modeling: past, present and future retrieved from http://www.omg.org/

Weber, R.P. (1990) Basic Content Analysis. Htpps://www.amazon.com/ ISBN-13: 978-0803938632.

Wirfs-Brock, R. and McKean, A. (2002) *Object Design: Roles, Responsibilities and Collaborations*. Addison Wesley Professional. 2002.

Wirfs-Brock, R. J., Wilkerson, B. and Wiener, L. (1990) *Designing Object-Oriented Software*. Prentice Hall, Englewood Cliffs, NJ, 1990.

Wirfs-Brock,R.,Wilkerson, B. and Wiener, L (1990). *Designing Object-Oriented Software*. Prentice Hall. 1990.

Workflow Management Coalition (1999) Workflow management coalition specification — terminology & glossary, 1999. WFMC document WFMC-TC-1011. Available at http://www.wfmc.org.

Wrycza, S. and Marcinkowski, B. (2007) A light version of UML 2: Survey and outcomes. In Proceedings of the Computer Science and IT Education Conference, CSITEd 2007Yin R.K. (1994) Case Study Research: Design and Methods, Sage Publications, Thousand Oaks, CA.

Yin, R., (1993). Application of Case Study Research. Sage Publication, California.

Zeichick, A. (2004) "UML adoption making strong progress", Software Development Times, 15 August 2004.